А. И. КИТОВ

ПРОГРАММИРОВАНИЕ ИНФОРМАЦИОННО-ЛОГИЧЕСКИХ ЗАДАЧ

«СОВЕТСКОЕ РАДИО»
МОСКВА — 1967

УДК. 681.142.2

Настоящая книга представляет собой монографию, посвященную программированию информационно-логических задач (обработка экономических данных, поиск научно-технической информации и т. д.).

Рассматривается расширенный алгоритмический язык, построенный на базе АЛГОЛа и включающий в себя средства для обработки составных величин, списков и некоторые другие средства.

Описываются способы построения в памяти машины и обработки списков различных типов, обеспечивающие быстрый поиск данных в больших объемах информации.

Книга предназначена для лиц, работающих в области программирования и составления алгоритмов решения информационно-логических задач, студентов высших учебных заведений по специальностям вычислительная техника и вычислительная математика.

3-3-14 35-66

ПРЕДИСЛОВИЕ

Настоящая книга представляет собой монографию, посвященную вопросам программирования информационно-логических задач.

В настоящее время электронные цифровые машины получают все более широкое применение для решения информационно-логических задач в экономике, науке, технике, медицине и других областях человеческой деятельности. Сюда относятся задачи обработки данных в системах материально-технического снабжения, поиск справочных и патентных данных, поиск библиографии, обработка историй болезней в клиниках, машинная медицинская диагностика и т. д. Общим для всех этих задач является наличие больших объемов обрабатываемой информации и логический характер процесса обработки.

При решении подобных задач применяются различные приемы и способы расположения данных в запоминающих устройствах машин, повышающие эффективность поиска данных, и различные методы представления самих процессов поиска и обработки данных. Для этой цели разработан целый ряд так называемых алгоритмических языков, т. е. формальных правил, позволяющих описывать процессы решения задач на машинах без детального знания устройства самих машин.

В настоящей книге, предназначенной в основном для лиц, занимающихся применением электронных цифровых вычислительных машин для решения информационно-логических задач, рассматриваются два основных вопроса:

а) алгоритмический язык для описания информационно-логических задач;

б) ряд приемов и способов расположения данных в памяти машины и их обработки, обеспечивающих быстрый поиск и обработку этих данных; эти приемы составляют сущность ассоциативного или спискового программирования.

Оба указанных вопроса относятся к новой, быстро развивающейся области науки.

Упитывая, что данная книга предназначается для лиц, занимающихся постановкой и формально-логическим описанием задач для их решения на машинах, во введении излагаются основные черты кибернетики и элементы алгебры логики.

Важной задачей в настоящее время является создание общего разветвленного алгоритмического языка, включающего в себя ряд подъязыков, предназначенных для решения задач различных классов.

Создавать такой язык необходимо, очевидно, путем широкого обсуждения и практического апробирования различных вариантов языков.

Автор надеется, что ознакомление с данной работой будет интересным для лиц, занимающихся разработкой алгоритмических языков и вопросами автоматизации программирования.

Хотя книга предназначена для лиц, работающих в области программирования и составления алгоритмов решения информационно-логических задач, она доступна и для лиц, впервые сталкивающихся с этими вопросами. В § 4 и 5 введения читатель может получить минимально необходимые сведения об устройстве машин и программировании.

Описываемый в настоящей работе алгоритмический язык для программирования информационно-логических задач относится к классу так называемых процедурно-ориентированных языков, т. е. языков, основу построения которых составляет набор операций, характерных для автоматического решения задач.

Эти операции, с одной стороны, должны быть достаточно универсальными, чтобы из них можно было строить любые алгоритмические программы для данного класса задач, и, с другой стороны, должны быть достаточно элементарными, чтобы допускать их простую машинную интерпретацию с помощью команд реальных вычислительных машин.

Рассматриваемый алгоритмический язык получен путем наращивания АЛГОЛ-60 сначала средствами, необходимыми для описания процессов обработки больших массивов информации с фиксированным составом и структурой членов, последовательно размещаемых в памяти машины. Таким путем получается язык АЛГЭМ, предназначенный в основном для обработки экономической информации. Затем к полученному языку добавляются средства, необходимые для обработки списковой информации, характеризующейся тем, что количество членов в списковых массивах и их расположение в памяти машины заранее не фиксируются.

При изложении данного алгоритмического языка, представляющего собой, по существу, объединение трех языков (АЛГОЛ, АЛГЭМ и язык ассоциативного программирования), используется метод синтаксических определений Бэкуса. В частности, описание АЛГОЛ-60 и основное содержание АЛГЭМа даны с помощью формальных синтаксических определений.

Изложение остальных разделов языка построено частично с использованием синтаксических определений, а частично на основе словесных описаний. При этом синтаксические определения используются только как средство для более четкого пояснения некоторых понятий языка.

Естественно, что данная книга далеко не исчерпывает всех вопросов, связанных с программированием информационно-логических задач. Например, здесь не рассматриваются вопросы построения трансляторов, различные методы сортировки и упорядочения данных, вопросы программирования задач машинного перевода и др.

Достаточно полное представление о круге, вопросов, рассмотренных в книге, можно получить из оглавления.

Язык АЛГЭМ разработан автором совместно с Ф. Ф. Шиллер. Этот язык и методика ассоциативного программирования в течение двух лет читались автором в Московском энергетическом институте.

Автор выражает признательность д-ру техн. наук А. А. Папернову и инженеру А. М. Бухтиярову за рецензирование рукописи и сделанные замечания.

I. ВВЕДЕНИЕ

1. НЕКОТОРЫЕ СВЕДЕНИЯ ИЗ КИБЕРНЕТИКИ И МАТЕМАТИЧЕСКОЙ ЛОГИКИ

Изучение принципов построения и функционирования информационно-логических систем, в том числе методов программирования и решения информационно-логических задач, составляет одну из основных задач кибернетики. В связи с этим целесообразно предварительно познакомиться с общим содержанием этой науки и основными направлениями ее развития. Это позволит яснее представить назначение и особенности информационно-логических систем с точки зрения общих методов и принципов кибернетики.

§ 1 СОДЕРЖАНИЕ И ОСНОВНЫЕ ЧЕРТЫ КИБЕРНЕТИКИ

Кибернетика — наука об общих закономерностях процессов управления и связи в организованных системах (машинах, живых организмах и их объединениях). Кибернетика изучает процессы управления в основном с информационной стороны, поэтому кибернетику определяют также как науку о способах восприятия, передачи, хранения, переработки и использования информации в машинах, живых организмах и их объединениях.

Систематическое изложение идей и методов кибернетики было дано в 1948 г. Н. Винером. Большую роль в создании кибернетики сыграли работы К. Шеннона по теории релейно-контактных схем, теории передачи информации и теории автоматов, а также работы Дж. Неймана по теории электронных вычислительных машин, теории автоматов и математической теории игр.

Возникновение кибернетики как общей теории процессов управления обусловлено потребностями практики в создании сложных систем автоматического управления и связано с появлением электронных вычислительных машин, являющихся мощным средством автоматизации различных процессов переработки информации и управления. Характерной особенностью кибернетики является то, что она возникла в результате процесса интеграции и взаимного проникновения методов и достижений ряда точных и биологических наук. Развитие теории автоматического регулирования, основанной в значительной мере А. М. Ляпуновым и И. В. Вышнеградским, и создание И. П. Павловым объективных методов изучения высшей нервной деятельности дали большой фактический материал для глубоких обобщений и способствовали выявлению аналогий и общих принципов управления в живых организмах и машинах. Рефлекторная теория, объясняющая процессы регулирования, происходящие внутри организмов, а также механизмы приспособления организмов к внешней среде, пользуется, в сущности, теми же принципами передачи информации и обратной связи, на которых основана и теория автоматического регулирования. В подготовке кибернетики значительную роль сыграли такие науки, как эволюционное учение и теоретическая генетика, изучающие закономерности развития биологических видов и процессы передачи наследственной информации. Непосредственное участие в формировании кибернетики приняли, с одной стороны, такие науки, как математическая экономика и методика исследования военных операций, возникшие в годы второй мировой войны и изучающие процессы передачи информации и управления в общественной жизни, и, с другой стороны, теория связи и теория вычислительных машин, занимающиеся закономерностями передачи и переработки информации в технических устройствах.

В отличие от указанных наук, изучающих процессы управления в различных конкретных областях, кибернетика изучает общие закономерности, относящиеся к любым процессам управления, независимо от их природы. В качестве первых конкретно разработанных разделов кибернетики можно указать теорию фильтрации случайных процессов Н. Винера и теорию интерполяции и экстраполяции случайных процессов А. Н. Колмогорова.

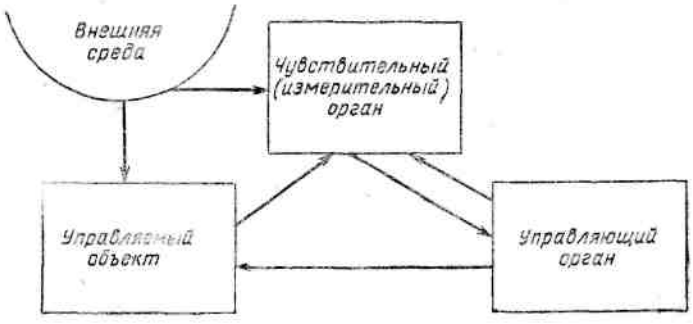


РИС. 1. Общая схема системы управления

Значение кибернетики состоит, прежде всего, в построении единой теории процессов управления и в выработке единой методологии их изучения. Несмотря на чрезвычайное многообразие конкретных проявлений процессов управления, оказывается, что они имеют универсальный характер и осуществляются по общей схеме. Любой процесс управления всегда связан с некоторой организованной системой, включающей в себя собственно управляющую систему, органы, воспринимающие внешнюю информацию, и управляемые или исполнительные органы, объединенные каналами связи (рис. 1). Естественными, созданными природой, организованными системами являются живые организмы. Кроме живых организмов пока нам известны только искусственные

организованные системы, созданные человеком.

Характерной чертой любого процесса управления является наличие цели. Управление — это организация целенаправленного (целесообразного) поведения. Целью процесса управления в общем случае является приспособление организованной системы к внешним условиям, необходимое для ее существования или выполнения свойственных ей функций.

Управление всегда осуществляется на основе приема, передачи и переработки информации в условиях взаимодействия данной организованной системы с внешней средой. Информацией обычно называют новые сведения о каких-либо событиях или явлениях; более точное определение понятия информации может быть дано в связи с понятием памяти. Память — это способность организованной материи селективно фиксировать и сохранять во времени внешние воздействия, а также при определенных условиях селективно воспроизводить (полностью или частично) их следы. Памятью обладают только живые организмы и искусственные управляющие системы. Их функционирование в пространстве и во времени обусловлено всегда не только характером текущих воздействий внешней среды, но и информацией, хранящейся в их памяти. Таким образом, информация может быть определена как опосредствованное через память восприятие управляющей системой внешних воздействий. Любое внешнее воздействие только в том случае несет информацию для управляющей системы, если оно, хоть в какой-то степени, связано со следами прошлых воздействий, хранящимися в памяти этой системы, т. е. если это воздействие опознается системой. Этот вид воздействия принципиально отличен от непосредственных физических воздействий, результаты которых полностью определяются самими воздействиями. Принципиально важно то, что опознавание информации возможно не только в случаях полного совпадения поступившего воздействия (сигнала) с одним из прошлых. Информация опознается и извлекается управляющей системой из тех воздействий, которые с прошлыми воздействиями связаны сложными зависимостями (частичные совпадения, совпадения по комбинациям и др.).

Методы опознавания информации живыми организмами весьма сложны и многообразны, и их изучение находится еще в самой начальной стадии. Эти методы в сильной степени изменяются в зависимости от уровня организации живых организмов.

Процессы управления в общем случае протекают по следующей схеме.

Управляющая система выдает исполнительным органам по каналам прямой связи командную информацию, по каналам обратной связи она получает от исполнительных органов информацию о действительном состоянии этих органов и об исполнении команд управления; управляющая система получает также информацию о состоянии внешней среды от специальных чувствительных или измерительных органов. На основе полученной информации управляющая система вырабатывает команды управления, определяющие действия исполнительных органов и будущее состояние всей организованной системы.

В отличие от указанной общей схемы в простейших случаях в технике иногда применяется так называемое жесткое управление по заранее заданной программе без использования обратных связей; обратные связи заменяются предварительным расчетом ожидаемых реакций внешней среды на тех или иных этапах управления, т. е. участвуют в неявной форме.

Для любого процесса управления характерно наличие алгоритма управления. Под алгоритмом понимают систему формальных правил, четко определяющих процесс реализации определенной цели, в частности порядок решения задач определенного класса. Для алгоритма характерны следующие черты: а) определенность алгоритма, состоящая в четкости образующих его указаний, их полной понятности для исполнителя, даже не знающего существа задачи; б) массовость алгоритма, состоящая в его применимости не к одной единственной, а к некоторому множеству вариантов исходных данных, т. е. к целому классу задач; в) результативность алгоритма, заключающаяся в том, что для всякой допустимой системы исходных данных число операций, приводящих к определенному результату, конечно.

Теория алгоритмов зародилась в 30-х годах в недрах математической логики в связи с изучением теоретических вопросов природы некоторых математических задач. С развитием кибернетики выяснилось, что алгоритмы играют важную роль при описании и изучении различных процессов переработки информации и управления. Понятия цели и алгоритма в кибернетике имеют весьма широкий смысл. В случае искусственных управляющих систем (технических, административных и т. п.) цель и алгоритм управления вкладываются в системы извне при создании этих систем. Например, в системе автоматического управления движением самолета цель управления — обеспечение выполнения заданных программой параметров движения самолета (координат, скоростей, ускорений), а алгоритм управления — уравнения, определяющие положения органов управления рулями и двигателем самолета по отклонениям действительных значений параметров от заданных. В качестве примера административно управляющей системы можно привести систему управления производственным предприятием, где целью управления является обеспечение выпуска заданной продукции, а алгоритм управления определяется совокупностью технической, технологической и планово-экономической документации, регламентирующей работу данного предприятия. В случае управляющих систем живой природы цель и алгоритм управления формируются естественно-причинным способом в результате длительной эволюции. Например, сам процесс биологической эволюции является управляющим процессом, целью которого является приспособление организмов к внешним условиям, а алгоритмом — закономерности естественного отбора. Таким образом, кибернетическое понятие цели включает и такое свойство, присущее управляющим системам живой природы, как сохранение устойчивости своей организации (гомеостазис).

Проблема изучения механизмов естественно-причинного возникновения целесообразных управляющих систем живой природы — одна из важнейших задач кибернетики. Разработка этой проблемы позволит глубже подойти к пониманию соотношений причинности и целесообразности в природе на основе выяснения конкретных механизмов и математических зависимостей, определяющих переход от причинно-детерминированного поведения отдельных элементов системы к целесообразным формам поведения системы в целом. По-видимому, основной особенностью причинно-следственных связей в целесообразно функционирующих системах является наличие обратных влияний следствий на внутренние причины, определяющие ход процесса.

Помимо теоретического значения кибернетика имеет большое практическое значение как теория, обеспечивающая единый подход к изучению искусственных и естественных управляющих систем и

использование принципов построения систем живой природы в технике. Специальный раздел прикладной кибернетики — бионика — занимается изучением и применением принципов действия чувствительных и управляющих элементов живых существ при построении различных технических устройств, а также изучением принципов переработки информации живыми организмами. Важной практической задачей кибернетики является проблема информационного симбиоза человека и машины в процессе решения различных научных задач. В отличие от существующих способов автономного использования машин для решения отдельных задач информационный симбиоз предусматривает тесное взаимодействие человека и машины непосредственно в процессе творческого мышления; при этом за человеком сохраняются функции постановки задач, формулировки вопросов и гипотез, анализа данных, а на машину возлагается работа по сбору и обработке материалов, выполнению вычислений, выдаче справок и представлению данных в виде, удобном для анализа.

Кибернетика как единая теория процессов управления включает в себя три основных раздела: теорию информации, теорию программирования (или методов управления) и теорию управляющих систем.

Теория информации занимается изучением способов кодирования (преобразования), передачи и восприятия информации. Передача информации осуществляется при помощи сигналов — физических процессов, у которых определенные параметры находятся в однозначном соответствии с передаваемой информацией. Установление такого соответствия называется кодированием. Хотя передача любых сигналов требует затраты энергии, количество затрачиваемой энергии в общем случае не связано с количеством, а тем более с качеством передаваемой информации. В этом состоит одна из принципиальных особенностей процессов управления: управление большими потоками энергии может осуществляться при помощи сигналов, требующих для своей передачи незначительных количеств энергии.

Наиболее разработан в теории информации статистический подход, основанный на учете вероятностных характеристик передаваемых сообщений. Центральным понятием теории информации является мера количества информации, определяемая как изменение в результате получения сообщения степени неопределенности в ожидании некоторого события, о котором говорится в сообщении. Эта мера позволяет измерять количество информации в сообщениях подобно тому, как в физике измеряется количество энергии, и оценивать эффективность различных способов кодирования информации, а также пропускную способность и помехоустойчивость различных каналов связи. Математическое определение понятия количества информации получается следующим образом. В теории вероятностей полной системой событий называют такую группу событий A_1 A_2, ..., $A_п$, в которой при каждом испытании обязательно наступает одно и только одно из этих событий, например, выпадение 1, 2, 3, 4, 5 или 6 при бросании игральной кости; выпадение герба или надписи при бросании монеты. В последнем случае имеется простая альтернатива, т. е. пара противоположных событий.

Конечной схемой называется полная система событий A_1, A_2, ..., $A_п$, заданная вместе с их вероятностями $P_1 P_2$, ..., $P_п$:

$$A = \begin{pmatrix} A_1 & A_2 & \cdots & A_n \\ P_1 & P_2 & \cdots & P_n \end{pmatrix}, \tag{1}$$

где $$\sum_{k=}^{n} P_k = 1; \quad P_k \geq 0.$$

Всякой конечной схеме свойственна некоторая неопределенность, т. е. известны только вероятности возможных событий, но какое именно произойдет событие в действительности, остается неопределенным.

Теория информации вводит следующую характеристику для оценки степени неопределенности любой конечной схемы событий:

$$H(P_1, P_2, \cdots, P_n) = -\sum_{k=1}^{n} P_k \log P_k \tag{2}$$

где логарифмы имеют произвольное, но всегда одно и то же основание, при $P_k \to 0$ принимается, что $P_k \log P_k \to 0$. Величина H называется энтропией данной конечной схемы событий. Она обладает следующими свойствами:

1. величина H (P_1, P_2, ..., $P_п$) непрерывна относительно P_k
2. величина H (P_1, P_2, ..., $P_п$)=0 только в том случае, когда из чисел P_1, P_2, ..., $P_п$ одно какое-либо равно единице, а остальные равны 0, т. е. энтропия равна нулю, когда отсутствует какая-либо неопределенность в конечной схеме;
3. величина H (P_1, P_2, ..., $P_п$) при фиксированном n имеет максимальное значение, когда все P_k равны между собой, т. е. когда конечная схема имеет наибольшую неопределенность. В этом случае

$$H(P_1, P_2, \cdots, P_n) = -\sum_{k=1}^{n} P_k \log P_k = \log n . \tag{3}$$

Кроме того, энтропия обладает свойством аддитивности, т. е. энтропия двух независимых конечных схем равна сумме энтропий этих конечных схем. Выбранное выражение энтропии достаточно удобно и полно характеризует степень неопределенности той или иной конечной схемы событий. В теории информации утверждается, что единственной формой, удовлетворяющей трем указанным свойствам, является принятая форма для выражения энтропии.

Данные о результатах испытания, возможные исходы которого определялись заданной конечной схемой А, представляют собой некоторую информацию, снимающую ту неопределенность, которая была до испытания. Чем больше неопределенность конечной схемы, тем большее количество информации получается в результате проведения испытания и снятия неопределенности. Так как характеристикой степени неопределенности любой конечной схемы является энтропия этой конечной схемы, то количество информации, полученное при испытании, целесообразно измерять той же величиной. Таким образом, в общем случае количество информации,

описывающей состояние какой-либо системы, имеющей различные вероятности возможных исходов, определяется энтропией конечной схемы, характеризующей состояние этой системы.

Так как за единицу количества информации принят наиболее простой и единый вид информации, а именно сообщение о результате выбора между двумя одинаково вероятными вариантами, то и основание логарифмов в выражении для энтропии обычно принимается равным двум.

Исключительный интерес представляет изучение созданных природой естественных способов кодирования наследственной информации, обеспечивающих сохранение в ничтожных объемах наследственного вещества огромных количеств информации, содержащих в зародышевой клетке признаки взрослого организма.

Методы статистической теории информации могут быть применены при общем планировании и организации физических экспериментов, оценке их результатов, анализе эффективности различных способов наблюдения.

Помимо вопросов оценки количества информации и надежности ее передачи и хранения большое значение имеет изучение различных форм представления информации. Одно и то же количество информации в зависимости от формы ее представления может быть доступным для использования в большей или меньшей степени, а преобразование информации из одной формы в другую зачастую оказывается достаточно сложным. Например, одна и та же функция может быть представлена в аналитической, графической или табличной форме; все три формы содержат формально одинаковую информацию, но отличаются с точки зрения удобства пользования ею. Развивающееся новое семантическое направление в теории информации занимается вопросами количественного выражения содержания информации, что сводится, по существу, к изучению форм представления информации и способов ее сжатия, т. е. преобразования в более экономную форму. Одной из основных задач этого направления является исследование процессов опознавания образов живыми организмами и моделирование этих процессов в искусственных системах; процесс выработки образов представляет собой не что иное, как преобразование информации, заданной в форме большого количества конкретных примеров, в форму совокупности характерных признаков и их связей. Разрабатывается количественный подход к вопросу о ценности информации для получателя. При допущении, что информация собирается для достижения некоторой определенной цели, ценность ее может быть измерена как разность вероятностей достижения цели до и после получения информации.

Теория программирования в широком смысле представляет собой науку, занимающуюся изучением и разработкой методов описания и моделирования любых процессов переработки информации и управления. Сюда относится, прежде всего, теория программирования задач для решения их на электронных цифровых машинах, теория алгоритмизации различных процессов управления, различные математические методы нахождения оптимальных решений. Большое теоретическое и практическое значение имеет развитие автоматизации программирования задач для электронно-цифровых машин, для передачи машинам все более сложных видов работы по подготовке задач. На основе операторного метода, предложенного в 1953 г. А. А. Ляпуновым, построен ряд программирующих программ (трансляторов), составляющих рабочие программы для машин. Развивается международный язык программирования (АЛГОЛ), который предназначается для облегчения обмена алгоритмами решения различных задач в международном масштабе.

Методы автоматического программирования обеспечивают в настоящее время возможность постановки задач на машины сразу в виде математических формул или словесных выражений, определяющих методы решения без детального описания процессов решения в виде последовательностей элементарных машинных операций. В перспективе развитие автоматического программирования и структуры машин должно позволить решать на машинах вычислительные, логические и технические задачи, задаваемые только своими условиями. Машины должны автоматически выбирать оптимальные методы решения, определять последовательность и методы, требуемые варианты расчетов, анализировать результаты решений и выдавать их в наглядном обобщенном виде. К общей теории программирования процессов переработки информации могут быть отнесены также математические методы выбора оптимальных решений при управлении в сложной обстановке. Процесс нахождения решения в общем случае включает оценку информации об обстановке, определение линии поведения (стратегии), отвечающей цели управления, и выработку команд управления, определяющих конкретные действия исполнительных органов. Крут процессов, связанных с нахождением решений, весьма широк и объединяет всевозможные процессы переработки информации, начиная от элементарных реакций рефлекторного типа, свойственных простейшим управляющим системам, и кончая процессами творческого мышления человека. Целесообразное функционирование управляющих систем (искусственных и естественных) с математической точки зрения представляет собой процесс минимизации на каждом шаге управления некоторой функции от состояния системы (включая информацию, хранимую в ее памяти) и информации, поступающей из внешней среды.

Значительное развитие в прикладной кибернетике получили математические методы нахождения оптимальных решений, такие, как линейное и динамическое программирование, а также методы теории массового обслуживания, теории игр и др. Этими методами пользуются не только в области военного и экономического управления, но и при планировании и анализе результатов физических экспериментов, построении математических моделей и исследовании различных физических процессов и систем.

Одним из важных достижений кибернетики является выработка единого подхода к изучению различных процессов переработки информации путем расчленения этих процессов на элементарные акты, представляющие собой, как правило, альтернативные выборы («да» или «нет»). Систематичное применение этого подхода позволяет последовательно формализовывать все более сложные процессы умственной деятельности людей, а также детально и однозначно описывать (алгоритмизировать) процессы функционирования различных управляющих систем.

Важной задачей этого раздела кибернетики является изучение сущности процессов обучения и творческого мышления человека и воспроизведение подобных процессов на электронных машинах. В этом направлении проводятся эксперименты по исследованию и моделированию на машинах процессов эвристического решения задач, сводящегося к расчленению процесса решения задачи на ряд последовательных этапов и применению метода попыток с анализом ошибок на каждом этапе. Большой интерес для решения этой проблемы

имеют работы по созданию различных систем ассоциативной памяти, позволяющей автоматически находить и объединять информацию по содержанию, о чем более подробно будет сказано дальше.

Теория управляющих систем изучает общие информационные и физические принципы построения управляющих систем различной природы и назначения. Управляющей системой в общем случае называется любой физический объект, осуществляющий целенаправленную переработку информации.

Можно выделить следующие основные классы управляющих систем: а) биологические системы хранения и передачи наследственной информации; б) управляющие системы живых организмов, обеспечивающие их рефлекторную деятельность; в) мозг как орган мышления; г) автоматические системы переработки информации в технике; д) экономические и другие общественные системы переработки информации; е) человечество как единую систему, осуществляющую получение и переработку информации в процессе развития науки.

Специальный раздел кибернетики — теория автоматов — изучает абстрактные управляющие системы дискретного действия, отражающие информационные свойства различных классов реальных систем. Большое место отводится в теории автоматов построению математических моделей нейронных сетей мозга и изучению на этой основе механизмов мышления и структуры мозга, обеспечивающих возможность восприятия и переработки огромных количеств информации в органах малого объема с ничтожной затратой энергии и с исключительно высокой надежностью.

Анализ различных управляющих систем показывает, что в основе их строения лежат два общих принципа: принцип обратной связи и принцип многоступенчатости (иерархичности) управления. Наличие обратных связей от исполнительных органов к управляющему обеспечивает постоянный учет управляющей системой действительного состояния системы и воздействий внешней среды. Принцип иерархичности управления обеспечивает экономичность структуры и устойчивость функционирования системы. Он заключается в построении многоярусной системы, в которой непосредственное управление исполнительными органами осуществляют механизмы низшего уровня, контролируемые механизмами 2-го уровня, которые сами контролируются механизмами 3-го уровня и т. д.

В реальных управляющих системах указанные принципы проявляются весьма сложно, образуя большое количество взаимосвязанных и перекрещивающихся контуров и уровней управления, в которых сама иерархичность управления является относительной. При этом отдельные элементы, принадлежащие к более низким уровням управления, сами могут оказывать управляющие воздействия на элементы более высоких уровней управления, а связи между элементами носят не однозначный, строго определенный, а вероятностный характер. Специфическое качество подобных систем — сложность, исключающая возможность их описания и анализа только на основе знания поведения отдельных элементов. Задача кибернетики — создать для описания сложных систем специальные методы, основанные на применении интегральных характеристик (например, степень организации) и структурных свойств (иерархия управления, система обратных связей).

Органическое сочетание принципов обратной связи и иерархичности управления придает управляющим системам свойство «ультраустойчивости», позволяющее им автоматически находить оптимальные режимы функционирования и приспосабливаться к различным изменениям внешней обстановки. Эти принципы являются основой развития, обучения и приобретения опыта живыми организмами в процессе их жизни. Постепенная выработка условных рефлексов и их наслаивание повышают и усложняют уровни управления в нервной системе животного.

Указанными принципами обратной связи и иерархичности управления пользуются также при построении сложных управляющих систем в технике и организации процессов управления в общественной жизни.

Опыт по изучению реальных управляющих систем может быть обобщен в виде принципиальной схемы, описывающей порядок изучения этих систем в самых различных областях. Процесс исследования управляющих систем разделяется на два основных этапа: первый называется макроподходом, или макроскопическим изучением этой системы; второй — микроподходом, или микроскопическим изучением управляющей системы. Первый подход характеризуется тем, что управляющая система рассматривается с чисто функциональной точки зрения, а именно изучаются потоки информации, входящей и выходящей из управляющей системы, способы кодирования этой информации, а также закономерности действия управляющей системы при тех или иных условиях. После того как проведено такое макроскопическое (функциональное) исследование, возникают задачи детального изучения внутреннего строения этой системы: выясняют, из каких частей или элементов она состоит, как связаны между собой различные ее части, каковы закономерности работы отдельных элементов системы и т. д. На основе полученных результатов можно полностью описать процесс функционирования системы, т. е. составить алгоритм работы системы. Алгоритмизация самой управляющей системы заключается в описании порядка ее работы, причем акты действия, осуществляемые частями этой управляющей системы, считаются элементарными. При этом часто возникает своеобразное преобразование микроподхода в макроподход. Именно микроподход к изучению управляющей системы в целом содержит в себе макроподход к изучению функционирования составляющих частей этой управляющей системы. Нередко первоначально выделенные части сами представляют сложную управляющую систему и микроподход к изучению всей системы в целом естественным образом перемежается с макроподходом к изучению ее звеньев.

Алгоритмизацию самой управляющей системы, т. е. детальное описание порядка ее работы, следует отличать от раскрытия алгоритма, реализуемого этой управляющей системой. Последний определяет, каким образом данная система перерабатывает поступающую в нее внешнюю информацию и что она выдает в тех или других случаях. Следовательно, приходится различать два алгоритма, связанных с управляющей системой: алгоритм, которому система подчиняется, и алгоритм, который она сама осуществляет.

При изучении управляющих систем возникают два рода вопросов: анализ структуры управляющей системы и синтез из заданных элементов системы, обеспечивающей выполнение заданного алгоритма. Общим требованием является обеспечение заданного быстродействия, точности работы, минимального количества элементов и надежности функционирования. Для оценки степени организации сложных управляющих систем вводится специальная количественная мера, характеризуемая количеством информации, которую требуется ввести в систему, чтобы обеспечить переход системы из начального беспорядочного состояния в требуемое организованное состояние.

Особый интерес представляют самоорганизующиеся системы, обладающие свойством самостоятельно переходить из произвольных начальных состояний в определенные устойчивые состояния, соответствующие характеру внешних воздействий. В общем случае состояние таких систем изменяется под влиянием внешних воздействий произвольно. Благодаря наличию иерархичности управления и обратных связей эти системы осуществляют целенаправленный отбор устойчивых состояний, соответствующих характеру внешних воздействий.

Свойство организации может проявляться только у систем, обладающих избыточностью структурных элементов и случайным характером связей между ними, изменяющихся в результате взаимодействия системы с внешней средой. К таким системам относятся сети нейронов мозга, различные типы колоний живых организмов, некоторые сложные самоорганизующиеся экономические или административные системы, а также технические самоорганизующиеся устройства типа перцептрон и др.

В задачу кибернетики входит наряду с информационными аспектами также изучение общих физических принципов построения управляющих систем с точки зрения их способности воспринимать и перерабатывать информацию. Сюда относится изучение соотношений между размерами и предельным быстродействием управляющей системы, обусловленных конечностью скорости распространения света, изучение ограничений в способности управляющих систем очень малых размеров однозначно воспринимать информацию в связи с проявлением в этих масштабах законов квантовой физики и т. д.

По своим методам кибернетика — математическая наука, широко пользующаяся для описания и исследования управляющих систем разнообразным математическим аппаратом. Для кибернетики характерно применение методов математического моделирования различных управляющих систем с помощью электронных программно-управляемых машин универсального назначения. Этот метод, основанный на математическом описании изучаемых процессов, позволяет учитывать также влияние случайных факторов путем представления их последовательностями случайных чисел, подчиняющихся соответствующим законам.

Кроме математического моделирования применяются также различные виды физического моделирования, заключающегося в замене изучаемых явлений другими изоморфными (т. е. подобными) явлениями, которые легче воспроизводятся и наблюдаются в лабораторных условиях. Метод моделирования, основанный на кибернетическом принципе единства законов управления в любых управляющих системах, имеет большое теоретическое значение.

Кибернетика, изучающая общие законы процессов управления в машинах, живых организмах и человеческом обществе, открывает новые перспективы в познании явлений жизни, в том числе сущности человеческого интеллекта, и в создании на этой основе новых все более совершенных кибернетических машин, которые, в свою очередь, будут способствовать дальнейшему проникновению человека в тайны природы.

§ 2 ОСНОВНЫЕ ТИПЫ И ОСОБЕННОСТИ ИНФОРМАЦИОННО-ЛОГИЧЕСКИХ ЗАДАЧ

Ознакомившись с общим содержанием кибернетики, рассмотрим теперь основные типы информационно-логических задач и особенности их решения.

Информационно-логическими системами принято называть достаточно широкий круг автоматических устройств, служащих для обработки, хранения и поиска информации. Основой таких устройств являются электронные цифровые машины с программным управлением. Особенность этих машин — наличие запоминающих устройств большого объема и специальная структура, приспособленная для реализации логических операций поиска и сортировки данных. Различают следующие основные классы процессов переработки информации с помощью автоматических цифровых машин с программным управлением.

а) **Решение математических и научно-технических задач.** Эти процессы являются детерминированными как в отношении состава исходной информации, так и в отношении алгоритмов решения. Методика постановки, программирования и решения научно-технических задач на электронных цифровых машинах является наиболее отработанной в настоящее время.

В научно-технических и математических задачах, как правило, объем исходной информации невелик, а ее структура является сравнительно простой, в то время как алгоритмы решения этих задач во многих случаях являются достаточно сложными.

б) **Задачи математического планирования, теории игр и математического моделирования.** Эти задачи характеризуются достаточно сложными алгоритмами и большими объемами исходной информации. Процессы ввода и вывода данных сравнительно просты и не связаны непосредственно с процессом решения.

в) **Обработка информации (в основном экономической).** Алгоритмы обработки являются сложными, объем перерабатываемой информации весьма велик и ее структура достаточно сложна. Большой удельный вес в общем процессе обработки занимают операции ввода **и** вывода данных. Характерным является использование операций сортировки и перегруппировки данных. Структура информации и ее расположение в памяти машины, как и в случае научно-технических задач, полностью определяются заранее, так же как и алгоритмы переработки.

г) **Накопление и поиск библиографической информации.** Суть задачи состоит в накоплении названий документов с некоторыми дополнительными признаками, используемыми для поиска этих документов; фактическое содержание документов в этих системах непосредственно не учитывается. В данном случае объем информации заранее не может быть определен, так как в процессе работы системы происходит непрерывное пополнение ее новыми библиографическими данными. Наряду с этим происходит периодическое исключение устаревших данных или перенос их с одного уровня хранения на другой. Структура информации и алгоритмы включения и исключения данных, а также поиск сведений по запросам являются заранее определенными.

д) **Накопление и анализ фактографических данных.** В отличие от предыдущего класса задач в данном случае происходит накопление, поиск и выдача не только названий документов, но и фактических сведений по определенным областям знаний, которые накапливаются и систематизируются в памяти машины, независимо от того, из каких источников (статей, книг и т. п.) они получены. Характерным для этого класса задач является

невозможность точно определить заранее как объем, так и структуру информации, поэтому система организации памяти машины должна обладать большой гибкостью, обеспечивающей возможность ее перестроения, в зависимости от поступающей информации. Эта информация должна автоматически классифицироваться машиной и размещаться в определенных местах памяти или снабжаться признаками, обеспечивающими возможность быстрого ее поиска.

Фактографические системы в отличие от библиографических систем оперируют непосредственно с различного рода конкретными сведениями (фактами) и по своему назначению и характеру процессов обработки информации в наибольшей степени приближаются к так называемым «интеллектуальным кибернетическим машинам», которые в будущем должны помочь человеку в решении творческих научных задач. В этих системах возможно широкое варьирование характера алгоритмов обработки от полностью детерминированных до самосовершенствующихся (самообучающихся), в которых заранее заложены только некоторые общие принципы обработки, а детальный алгоритм процессов поиска и обработки данных формируется в процессе работы в соответствии с характером поступающей информации и характером внешних запросов, на которые должна давать ответы машина. Нужно сказать, что резких границ между этими четырьмя классами систем нет; возможны различные варианты промежуточных типов. Особенно это относится к трем последним классам процессов: обработка информации, библиографический поиск и фактографический анализ. Эти три класса процессов мы и будем объединять общим названием информационно-логические процессы; во всех трех указанных классах задач имеет место хранение и логическая обработка больших объемов информации; информация представляется не только в количественной числовой форме, но и в качественной форме при помощи слов и предложений естественных человеческих языков (с определенной формализацией). Указанные процессы в настоящее время имеют исключительно большое значение в различных областях экономики, науки и техники.

В первую очередь здесь следует указать на применение различного рода электронных автоматических систем обработки информации для планирования, учета и статистики, для оперативного управления предприятиями и их объединениями, а также для управления народным хозяйством страны в целом. В современных условиях в связи с колоссальным расширением масштабов и увеличением темпов производства, а также значительным усложнением связей между отраслями хозяйства и предприятиями невозможно сколько-нибудь рациональное управление хозяйством без применения электронных вычислительных машин. Эти машины обеспечивают не только сокращение административно-управленческого персонала, но и, что самое главное, быстрый, полный и точный сбор данных, их точную обработку и выдачу решений, позволяющих оперативно управлять сложным производством.

В качестве другого характерного примера массовых процессов обработки данных может служить задача обработки медицинских данных (историй болезней, статистических отчетов и сводок); подобная работа, будучи поставлена в должном масштабе, позволит выявить важные объективные закономерности распространения и течения различных болезней, оценивать эффективность различных средств лечения и профилактики и рационально организовать медицинскую службу.

Применение автоматических библиографических поисковых систем имеет исключительное значение для развития всех областей науки и техники. В настоящее время имеет место колоссальный рост числа публикаций па всем отраслям знаний и специалисты не в состоянии следить за литературой даже в своих узких областях. Положение усугубляется тем, что весьма важные сведения по некоторым вопросам помещаются в статьях, которые по своей основной тематике относятся к другим областям знаний.

В связи с трудностями поиска нужной литературы зачастую легче произвести заново тот или иной эксперимент, чем найти его описание в литературе, даже если известно, что такие эксперименты уже делались. Все это приводит к тому, что огромные запасы знаний и достижений человеческой мысли используются далеко не полностью. Сейчас широко ведутся работы по созданию и применению различного рода информационных поисковых библиографических систем, и имеется ряд практически действующих систем.

Фактографические информационные системы будут представлять собой новый уровень автоматизации умственных процессов. Они должны позволить не только находить соответствующую литературу по тому или иному вопросу, но и производить логический анализ и сопоставление содержания различных статей и книг, выявлять противоречие как внутри данного документа, так и между вновь вводимыми в машину сведениями и сведениями, уже имеющимися в машине; производить обобщение сведений, поступающих из многих источников, и выявлять новые факты и закономерности явлений и процессов.

Такие машины в отличие от человека, не являясь смертными, будут непрерывно накапливать сведения, добываемые человечеством в процессе познания внешнего мира и самого человека, будут анализировать и обобщать эти сведения и выдавать человеку новые факты и закономерности, в том числе такие, которые невозможно открыть одному человеку в силу ограниченности его жизни и возможностей индивидуального мозга.

Примеры типовых информационно-логических задач

Для того чтобы яснее представить назначение алгоритмических языков и особенности программирования информационно-логических процессов, мы рассмотрим
кратко некоторые типовые примеры подобных задач, а именно:

— обработку массивов записей;
— накопление и поиск данных в иерархических классификационных системах;
— библиографический поиск;
— фактографический поиск.

Обработка массивов записей

Этот тип процессов является основным при решении экономических задач. Записью называется точно установленный набор данных, характеризующих некоторый объект или процесс. Примерами записей могут

служить товарные чеки, бланки для расчета заработной платы, наряды на выполнение работ, накладные для получения каких-либо материалов и т. д. В области учета кадров примерами записей могут служить анкеты или личные листки, в здравоохранении — истории болезней, статистические отчеты и т. п. Обычно различного рода записи используются большими группами — массивами, и обработка их носит массовый характер. Весь процесс обработки массивов записей складывается из ряда этапов.

1. Фиксация записей на первичном носителе (перфокарты, перфоленты, бланки со специальными магнитным и стилизованным шрифтом или бланки с графитовыми отметками, воспринимаемые непосредственно читающими устройствами машин и др.).

2. Ввод записей в машину и размещение их на магнитных лентах. Как правило, для больших объемов данных (сотни миллионов знаков) используются магнитные ленты; магнитные диски и барабаны используются для средних (десятки миллионов знаков) и небольших массивов. Поэтому в качестве типового случая следует рассматривать использование магнитных лент (МЛ).

3. Обработка массивов записей, находящихся на магнитных лентах. Записи по одной или группами последовательно переписываются с МЛ в оперативную память машины, там обрабатываются и выводятся на магнитные ленты. Основной особенностью обработки является то, что один и тот же алгоритм обработки применяется ко всем однотипным записям массива, т. е. обработка носит *параллельно-циклический* характер.

В оперативкой памяти машины обычно при подобных процессах обработки выделяются пять областей: область для размещения программы (программа может тоже по частям вводиться в оперативную память), область для размещения констант, область для приема группы обрабатываемых записей, рабочая область для хранения промежуточных результатов и выходная область для размещения обработанных записей перед выдачей их на МЛ.

4. Вывод обработанных данных с МЛ на печать в форме, удобной для чтения или дальнейшего использования на машинах.

Обработка записей включает в себя не только преобразование данных, находящихся в записях, и выдачу их в виде записей другой формы, но и получение каких-либо общих показателей по всем записям массива или группам записей (например, подсчет общего количества и стоимости проданных товаров, если записями являются товарные чеки, или подсчет общей потребности в материалах различных видов, если записями являются заявки на материально-техническое снабжение, поступившие от многих предприятий, и т. д.).

В отличие от процессов математических вычислений в подобных процессах обработки информации наибольшую трудность представляет не построение алгоритмов вычислений, а организация обращений к данным (их опознавание, выборка нужных величин из записей, перестроение записей, ввод и вывод и т. д.). Поэтому в алгоритмических языках, предназначенных для обработки информации, важное место занимает методика описания данных как исходных, так и результирующих. Рассмотрим, например, следующую запись, представляющую собой отчет некоторого предприятия о наличии кадров:

Номер предприятия: 2462;
Дата составления отчета: 25.05.66;
Общее число работников: 698;
Наличное число работников: 650;
Больных: 24;
В отпусках: 15;
В командировках: 6;
Отсутствующих по другим причинам: 3.

Предположим, что подобные отчеты представляются многими предприятиями, и они должны обрабатываться с помощью ЭЦМ. Ясно, что, так как все эти отчеты имеют одинаковый формат, то на магнитную ленту достаточно записывать только сами числа без соответствующих названий данных, т. е. можно перечисление названий данных с указанием их разрядности и вида задать один раз в самой программе обработки в виде специального описания данных. Пользуясь этим описанием данных, машина по специальной программе сможет выделить из сплошной последовательности чисел на магнитной ленте запись, относящуюся к любому интересующему нас предприятию, а также выделить из выбранной записи любые требуемые данные (например, число больных или число командированных).

Таким образом, для каждого массива, содержащего однотипные записи, должно быть сделано точное описание формата записей, указано общее их число и порядок расположения на магнитной ленте. Подобные описания, сделанные для всех типов записей, участвующих в задаче, являются неотъемлемой составной частью программы. Методика составления таких описаний будет подробно рассмотрена в дальнейшем.

Накопление и поиск данных в иерархических классификационных системах

Весьма распространенным способом организации информации о больших количествах различных объектов является использование иерархических (т. е. многоуровневых, древовидных) классификационных систем. Примерами таких систем могут служить универсальная десятичная классификация литературы (УДК), единая десятичная классификация материальных продуктов, различные частные десятичные классификаторы материальных продуктов (классификатор проката черных металлов, классификатор полупроводниковых приборов и т. д.). Все иерархические классификационные системы строятся по общему принципу: классифицируемая совокупность объектов делится сначала по основному признаку (например, по назначению) на несколько крупных классов, затем каждый класс делится по определенному признаку на ряд подклассов, которые, в свою очередь, делятся на другие более мелкие подразделения (типы, виды и т. д.). В результате образуется классификационное дерево, отражающее выбранную схему деления совокупности предметов. Деревья не обязательно должны быть симметричными и иметь одинаковое число уровней во всех ветвях. Не обязательно также, чтобы было одинаковое количество разветвлений в каждой точке деления. В наиболее простых случаях

основание классификационной системы выбирается постоянным для всех уровней дерева (например, число 10), практически же не все возможности деления используются сразу (т. е. при создании системы). На каждом уровне оставляют запасные подразделения классификации, обеспечивающие возможность включения новых разновидностей объектов.

В процессе работы классификационные системы могут дополняться и развиваться как в направлении заполнения свободных классификационных рубрик на различных уже имеющихся уровнях, так и в направлении наращивания новых уровней в тех или иных ветвях. Для того чтобы внести конкретный объект в заданную классификационную систему, ему необходимо приписать многопозиционный код, каждая из позиций которого определяет номер классификационного подразделения на соответствующем уровне, причем номер позиции в коде (слева направо) указывает номер уровня дерева сверху вниз. Так, например, трехзначное десятичное число соответствует трехуровневому дереву в десятичной системе классификации, причем старший разряд указывает номер классификационного подразделения на верхнем уровне, второй разряд — на среднем уровне и младший разряд — на нижнем уровне.

Примером иерархической десятичной классификационной системы может служить следующая система учета кадров. Допустим, что весь личный состав сначала делится на основные классы в зависимости от производственной категории (неквалифицированные рабочие, квалифицированные рабочие, средний технический персонал, инженерный состав, научные работники, административный состав, вспомогательный состав). Затем каждая из этих категорий делится на более узкие категории, например, рабочие — по разрядам, научные работники — по ученым степеням (без ученых степеней, кандидаты наук, доктора наук, члены-корреспонденты и академики). Далее можно представить себе деление на третьем уровне по специальности (например, для научных работников: математики, физики, механики, электрики, радисты), на четвертом уровне — по стажу работы (до 5, от 5 до 10, от 10 до 15, от 15 до 20, свыше 20 лет), на пятом — по возрасту (до 25, от 25 до 35, от 35 до 50, от 50 до 65, свыше 65 лет).

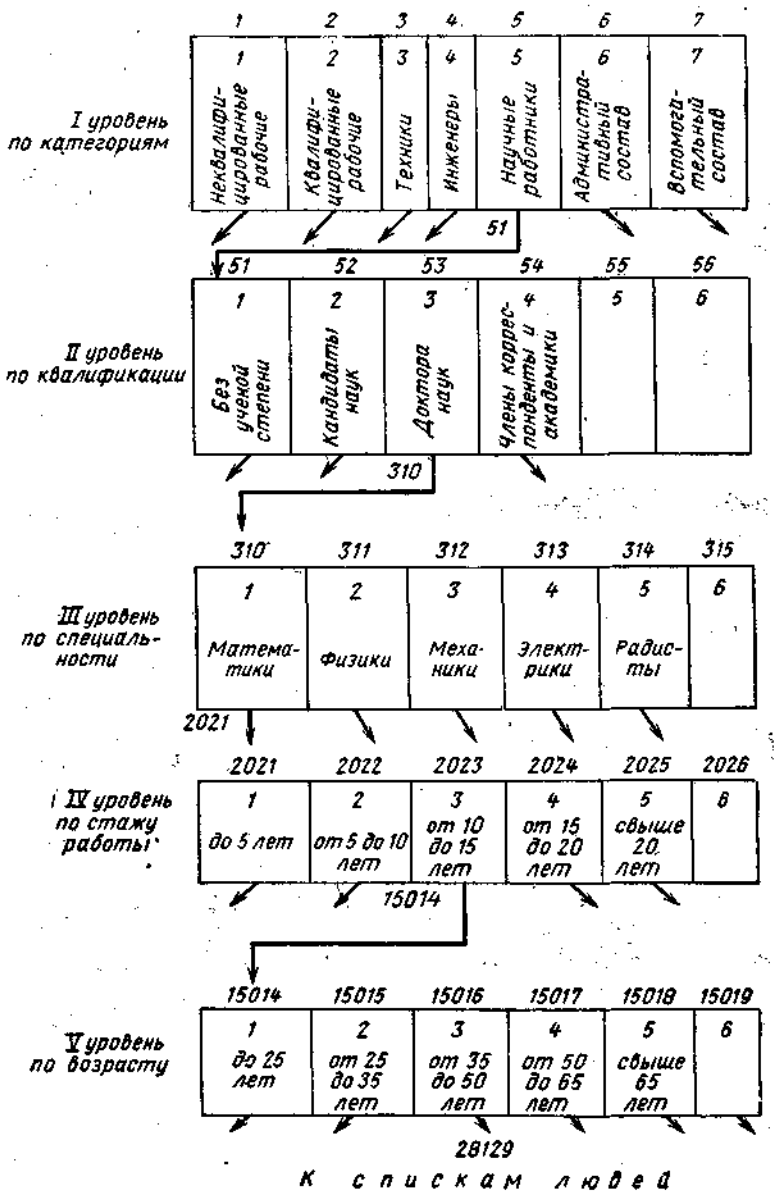


РИС. 2. Схема иерархической классификационной системы для учета кадров.

Для поиска человека, отвечающего определенному набору признаков, необходимо задать соответствующий многопозиционный код, каждая из позиций которого указывает номер рубрики на соответствующем уровне.

Например, код 53133 будет обозначать: научный работник, доктор наук, математик, стаж работы от 10 до 15 лет, возраст между 35 и 50 годами. Поиск объектов по заданным признакам осуществляется путем прослеживания дерева сверху вниз. Сначала просматривается список подразделений верхнего уровня и там находится признаковый код, соответствующий цифре старшего разряда заданного для поиска кода (53133). Там же указывается адрес, где находится в памяти машины список, представляющий собой второй уровень классификации, ответвляющийся от данного подразделения. Аналогичным образом просматривается этот список и происходит переход к списку третьего уровня. Этот процесс продолжается до тех пор, пока не будет достигнут список самого нижнего уровня, который указывает уже на список объектов, обладающих всеми заданными признаками.

На рис. 2 приведен пример подобной классификационной системы. Над клетками указаны адреса соответствующих ячеек (взятые ориентировочно). Внутри клеток сверху указаны значения кодов признаков по соответствующим позициям, причем если ячейки расположены подряд, то эти значения могут в явном виде не храниться. Снизу под клетками, соответствующими коду запроса 53133, указаны адреса начальных ячеек подсписков, отходящих от данных клеток. Например, адрес 28129 взят условно как адрес ячейки, в которой хранятся данные о специалисте, обладающем признаками, отвечающими этому коду.

Основные трудности, которые возникают при реализации иерархических классификационных систем на электронных цифровых машинах, связаны с тем, что в процессе их эксплуатации происходит изменение классификационного дерева и состава объектов, информация о которых накапливается. Поэтому в таких системах, как правило, не представляется возможным заранее предусмотреть полностью структуру дерева и особенно количество объектов, которые будут относиться к разным рубрикам классификации. Это обстоятельство не позволяет заранее произвести точное распределение всего объема памяти машины (в основном, магнитных лент) и в процессе работы часто приходится производить полную или частичную перезапись заполненных МЛ.

В настоящее время разрабатываются специальные способы гибкой адресации и записи на МЛ таких систем, исключающие необходимость перезаписи и обеспечивающие возможность эффективного использования объема МЛ. Одна из таких методик, называемая ассоциативным программированием, будет подробно рассмотрена нами в дальнейшем. Недостатком иерархических классификационных систем является то, что с их помощью трудно производить поиск объектов, относящихся одновременно ко многим классификационным подразделениям (многоаспектный поиск), а также чрезвычайная сложность и громоздкость этих систем, затрудняющая их эксплуатацию. Кроме того, иерархические системы при всей своей сложности все же не отражают полностью всего многообразия классификации объектов. Указанные недостатки присущи, в частности, и универсальной десятичной системе классификации литературы, в связи с чем сейчас применяются другие более гибкие системы, например системы, основанные на дескрипторном принципе.

Библиографический дескрипторный поиск

Дескрипторные информационно-логические системы накопления, хранения, обработки и поиска библиографической информации по способу организации информации в памяти машины можно условно разделить на два основных типа: простейшие дескрипторные системы без грамматики и дескрипторные системы с грамматикой.

Рассмотрим сущность простейшего дескрипторного метода поиска литературы. Содержание каждого литературного источника (книги, журнала, статьи, сборника и т. д.), называемого для краткости документом, может быть представлено в общих чертах при помощи набора характерных для данного текста слов. Эти слова называются ключевыми словами для данного текста. В различных текстах, посвященных одной и той же тематике, набор ключевых слов может сильно отличаться не только из-за различий в содержании текстов, но и благодаря наличию синонимов, а также возможности описывать одни и те же понятия различными формулировками. Для построения поисковой системы из всего многообразия ключевых слов, выбранных из ряда характерных для данной тематики текстов, составляется стандартный набор терминов со строго фиксированными значениями, в котором устранены синонимы. Эти термины, представляемые отдельными словами или группами слов, называются дескрипторами, а набор таких терминов — словарем дескрипторов. Например, статья об алгоритмических языках, предназначенных для программирования информационно-логических задач, может быть описана набором дескрипторов: *программирование, алгоритмический язык, информация, логическая обработка, вычислительная машина.* Для каждого из документов, образующих массив, в котором должен производиться поиск, составляется набор дескрипторов, называемый поисковым образом данного документа.

Для поиска литературы по определенному вопросу заказчик должен сформулировать свое требование в виде поискового образа запроса, который также должен представлять собой набор дескрипторов, характеризующих этот вопрос. Обычно заказчик не знает точно словарь дескрипторов и в запросах часто употребляет синонимы дескрипторов или близкие выражения. Поэтому первым этапом поиска является перевод запроса заказчика на язык дескрипторов, имеющихся в словаре. Затем производится поиск документов, целью которого является определение документов, наборы дескрипторов у которых соответствуют набору дескрипторов запроса.

Вопрос о критерии такого соответствия является достаточно сложным вопросом. В простейшем случае в качестве критерия соответствия принимается условие наличия всех дескрипторов запроса среди дескрипторов документа. Массив дескрипторных наборов документов может быть построен двумя различными способами: прямым и инверсным. При прямом способе в памяти машины последовательно записываются номера документов (или другие данные, показывающие их местонахождение) и за каждым номером документа указываются все коды дескрипторов, относящихся к нему. Процесс поиска заключается в последовательном сравнении для каждого документа всех дескрипторов запроса с дескрипторами этого документа и выделении тех документов, для которых выполняется критерий соответствия. Обычно эту простейшую схему поиска усовершенствуют путем разделения всего массива документов на некоторые разделы с тем, чтобы производить прямой дескрипторный поиск не по всем документам, а по их части, относящейся к требуемой рубрике.

Дескрипторы		Номера документов
Наименования	Коды	
Вычислительная машина	0101	0031, 0034, 0038, 0045, 0049, 0101
Логическая обработка	0102	0012, 0026, 0031, 0046, 0049, 0082
Алгоритмический язык	0105	0003, 0022, 0027, 0031, 0049
Экономика	0205	0031, 0168, 1342
.....	.....	.....
ПЕРТ	0340	0010, 0031
Оптимизация	0520	0612, 0831, 1342

РИС. 3. Пример дескрипторного массива документов, построенного по инверсному принципу.

При инверсном способе за основные позиции хранения информации принимают не документы, а дескрипторы. Для каждого дескриптора записываются все номера документов, которые имеют в своих поисковых образах этот дескриптор. Поиск нужных документов, отвечающих заданному набору дескрипторов запроса, осуществляется путем обращения в словаре дескрипторов к тем дескрипторам, которые имеются в запросе, и выписывания всех номеров документов, которые относятся к каждому из этих дескрипторов. Затем производится отбор нужных документов путем последовательного сравнения номеров документов, относящихся к разным дескрипторам, указанным в запросе. Отобранными считаются те документы, номера которых оказались общими для всех дескрипторов, указанных в запросе.

На рис. 3 показан пример построения дескрипторного словаря. Если, например, требуется найти литературу по алгоритмическим языкам для программирования экономических задач, решаемых методом ПЕРТ, то запрос может быть сделан в таком виде:

0105, 0205, 0340.

Этому запросу отвечает документ с номером 0031. Как при прямом, так и при инверсном способе поиска возможны другие критерии соответствия документов запросу, а не только полное совпадение дескрипторов. Иногда дескрипторы запросов делятся на категории, указывающие их важность, что учитывается при поиске. Иногда алгоритм поиска предусматривает в случае отсутствия подходящих документов выдачу других, близких по смыслу дескрипторов, что позволяет заказчику уточнить или изменить запрос.

В этом простейшем виде дескрипторного поиска, дескрипторы, относящиеся к документам, и дескрипторы, входящие в состав запроса, представляют собой простые наборы, не связанные какой-либо грамматикой; в частности, порядок их расположения роли не играет.

Такой простейший способ, однако, приводит либо к выдаче излишних документов, не относящихся к интересующему вопросу, либо, наоборот, к пропуску (потере) нужных документов.

Например, по вопросу, содержащему 3 дескриптора: устройство, программа, выработка — могут быть выданы документы об устройствах для выработки программы, либо информация о программе выработки устройств, либо о выработке программы для устройств.

Более эффективными являются дескрипторные поисковые системы с грамматикой, в которых дескрипторы, относящиеся к документам, и дескрипторы, входящие в состав запросов, снабжаются дополнительными символами, указывающими на семантическую роль этих дескрипторов (субъект процесса, объект процесса, атрибут, процесс, причина и т. д.).

Иногда роль дескрипторов определяется их положением в наборе, а также некоторыми символами, указывающими связи между отдельными дескрипторами. В этом случае дескрипторные поисковые системы приближаются по своему принципу действия к поисковым системам, построенным на основе смыслового кодирования (например, система Перри — Кента (США), система информационного поиска Института кибернетики АН УССР и др.).

Принцип смыслового кодирования является основным для построения фактографических информационных систем, к рассмотрению которых мы и переходим.

Фактографические системы

Основными принципами построения фактографических систем является принцип смыслового кодирования, т. е. построение специального формализованного информационного языка, позволяющего записывать фактические сведения, относящиеся к тем или иным областям знаний.

Основой подобных информационных языков является некоторая обобщенная абстрагированная модель естественных языков, освобожденная от двусмысленностей и различных дополнительных эмоциональных средств, сильно усложняющих и обогащающих язык как средство общения людей, но не нужных для информационного накопления научных данных. Информационный язык как средство отображения закономерностей и связей внешнего мира должен отражать и его основную схему, а именно, наличие двух основных категорий: объектов и отношений между ними.

Информационные языки, построенные на основе смыслового кодирования, включают в себя обычно четыре основные части;

а) набор базисных терминов, обозначающих основные предметы данной области знаний. Например, для вычислительной техники такими терминами могут быть двоичный разряд, машинное слово, адрес, код операции и т. д.

Ясно, что выбор базисных терминов не является строго однозначным и может осуществляться в известной мере произвольно;

б) набор смысловых отношений между предметами. В отличие от базисных терминов, которые сильно связаны с конкретными областями знаний, наборы базисных отношений более универсальны и сходны для различных областей знаний (во всяком случае, для многих областей науки).

Примеры отношений: один предмет является элементом класса, представляющего другой предмет; один предмет является частью предмета; предмет является субъектом процесса; предмет является объектом процесса и т. д.;

в) набор формальных правил (синтаксис), позволяющих из основных терминов и отношений языка строить более сложные термины и отношения, а также осуществлять переход между информационным и естественным языком;

г) систему кодирования понятий, отношений и синтаксических правил языка, позволяющую осуществлять преобразование и хранение информации, представленной на этом языке, с помощью цифровых вычислительных машин. Эта часть информационного языка включает из себя и алгоритмический язык для описания алгоритмов преобразования информации в машине.

По своим возможностям анализа и логической переработки информации фактографические информационные системы могут различаться в сильной степени. В качестве простейшего варианта таких систем можно представить себе систему, осуществляющую накопление информации и ее проверку на непротиворечивость. В такую систему должны вводиться некоторые утверждения, и она должна выдавать один из трех ответов: «да», «нет», «неизвестно».

Ответ «да» свидетельствует о справедливости введенного для проверки утверждения, ответ «нет» показывает, что это утверждение противоречит той информации, которая имеется в памяти машины, и ответ «неизвестно» выдается в тех случаях, когда информации, имеющейся в машине, недостаточно для проверки поставленного утверждения.

Такие системы реализуют накопление и классификацию данных и проверку вводимых утверждений, используя основные аксиомы логики. В частности, подобные справочные фактографические системы могут быть созданы для химии для проверки возможности новых реакций, свойств новых веществ и т. д.

Подобную систему можно представить себе и в области законодательства, когда каждый новый закон или поправка проверяются на непротиворечивость всем ранее принятым законам.

В более сложном варианте фактографические системы должны не только проверять правильность утверждений, вводимых извне, но и выдавать фактический материал на различные вопросы (определенного круга и характера).

§ 3 АЛГЕБРА ЛОГИКИ

При программировании информационно-логических задач часто приходится пользоваться разделом математической логики, называемым алгеброй логики. Этот раздел, основанный на применении алгебраических методов в логике, широко используется также в теории ЭЦМ и дискретных автоматов.

Алгебра логики представляет собой, прежде всего, алгебру высказываний. Под высказыванием в алгебре логики понимают всякое предложение, которое либо истинно, либо ложно; при этом может иметь место только одно из двух указанных значений (например, «Москва — столица СССР», «снег — черен», «9 — нечетное число»). Отдельные высказывания в алгебре логики обозначаются буквами какого-либо алфавита, например: *A, B, C*... Истинность или ложность высказываний называется их значениями истинности. В алгебре логики принято отождествлять истинность высказывания с числом 1, а ложность высказывания — с числом 0. Запись $A = 1$ и $C = 0$ означает, что *A* истинно и что *C* ложно. Каждое конкретное высказывание имеет вполне определенное значение истинности: это постоянная, равная 0 или 1. От конкретных (постоянных) высказываний следует отличать так называемые переменные высказывания. Переменное высказывание не есть высказывание в подлинном смысле, так как вопрос о его истинности или ложности не имеет смысла; это переменная для высказываний (пропозициональная переменная), т. е. символ, на место которого можно подставить постоянные высказывания (или их наименования) и который может принимать лишь два значения: «истинно» и «ложно», или соответственно 1 и 0 (двоичная переменная). Переменные высказывания (т. е. пропозициональные переменные) обозначаются буквами, отличными от тех букв, которыми обозначаются постоянные высказывания. Применение переменных высказываний в алгебре логики служит для выражения всеобщности; оно позволяет формулировать законы алгебры логики для любых высказываний.

Предметом изучения в алгебре логики являются двоичные (или двузначные) функции, т. е. функции, которые принимают лишь два значения («истинно», «ложно»; 0 или 1), и зависят от одной или нескольких двоичных переменных. Это так называемые функции алгебры логики.

Из одного или нескольких высказываний, принимаемых за простые, можно составлять сложные высказывания, которые будут двоичными функциями простых высказываний. Объединение простых высказываний в сложные в алгебре логики производится без учета внутреннего содержания (смысла) этих высказываний. Используются определенные логические операции (или логические связи), позволяющие объединять некоторые данные высказывания (постоянные или переменные) в более сложные (постоянные или переменные).

К числу основных логических операций относятся операции отрицания, конъюнкции, дизъюнкции, эквивалентности и импликации. Логические операции задаются таблично как функции простых высказываний.

Отрицание высказывания A — это высказывание, которое истинно, когда A ложно, и ложно, когда A истинно; обозначается через $\overline{A}$ и читается «не A». Операция отрицания задается табл. 1.

Таблица 1

A	$\overline{A}$
1	0
0	1

Конъюнкция двух высказываний — сложное высказывание, которое истинно в случае истинности обоих высказываний, его образующих, и ложно в остальных случаях; обозначается через $A \wedge B$ и читается «A и B»; знак логической операции $\wedge$ имеет смысл союза «и» и называется знаком конъюнкции (другое обозначение &, другое название — логическое умножение). Операция конъюнкции задается табл. 2.

Таблица 2

A	B	$A \wedge B$
1	1	1
0	1	0
1	0	0
0	0	0

Дизъюнкция двух высказываний — сложное высказывание, которое ложно в случае ложности обоих составляющих его высказываний и истинно в остальных случаях; обозначается $A \vee B$ и читается «A или B» (другое обозначение $A+B$; другое название — логическое сложение). Знак логической связи $\vee$ имеет смысл союза «или» и называется знаком дизъюнкции. Союз «или» вообще может употребляться в нескольких различных смыслах. Знак $\vee$ может иметь смысл «или», употребленного, например, в фразе: «При звоне будильника Петр или Иван проснется» (здесь «или» не исключает возможности того, что проснутся оба), т. е. смысл так называемого неразделительного «или». Существует еще исключающее «или» (пример: «Выбирай: он или я»), которое тоже может быть принято за один из видов логических операций, но его не следует смешивать с дизъюнкцией. Дизъюнкция задается табл. 3

. Таблица 3

A	B	A∨B
1	1	1
0	1	1
1	0	1
0	0	0

Эквивалентность двух высказываний — сложное высказывание, истинное тогда, когда значения истинности составляющих высказываний одинаковы, и ложное в противном случае; обозначается $A \equiv B$ и читается: «A эквивалентно B». Задается табл. 4.

Таблица 4

A	B	$A \equiv B$
1	1	1
0	1	0
1	0	0
0	0	1

Для эквивалентности справедливо, что $A \equiv 1 = A$ и $A \equiv 0 = \overline{A}$.

Применив операцию отрицания к высказыванию, представляющему собой эквивалентность двух высказываний, получаем новое сложное высказывание $\overline{A \equiv B}$, называющееся отрицанием эквивалентности. Используя специальный знак $\not\equiv$ для выражения отрицания эквивалентности, можно записать его в виде $A \not\equiv B$ (читается: «A неэквивалентно B»). Нетрудно видеть, что знак $\not\equiv$ имеет смысл исключающего «или». Операция отрицания эквивалентности задается табл. 5.

Эта операция имеет важное значение в теории ЭЦМ, так как она представляет собой так называемое сложение двоичных чисел по модулю два.

Таблица 5

A	B	$A \not\equiv B$
1	1	0
0	1	1
1	0	1
0	0	0

Импликация двух высказываний (обозначается $A \supset B$ и читается: «если A, то B») — такое сложное

высказывание, которое ложно в том и только в том случае, когда A истинно, а B ложно. Задается табл. 6.

Таблица 6

A	B	$A \supset B$
1	1	1
0	1	1
1	0	0
0	0	1

Импликация не предполагает обязательно связь по смыслу между условием A и следствием B (хотя и не исключает такую связь). Смысл импликации $A \supset B$ можно передать словами: «A ложно или B истинно» (здесь «или» — неисключающее).

Любое сложное выражение, полученное из простых высказываний посредством указанных выше логических операций, называется формулой алгебры логики. Две формулы алгебры логики, образованные из простых высказываний A_1, A_2, ..., A_n называются равносильными в том случае, если для каждой комбинации значений истинности высказываний A_1, A_2, ..., A_n, обе формулы алгебры логики будут иметь одинаковые значения истинности. Так как существует в точности 2^n различных комбинаций значений истинности n простых высказываний и для каждой из этих 2^n комбинаций сложное выражение будет либо истинным, либо ложным, то может быть 2^{2^m} различных функций алгебры логики, построенных из n данных простых высказываний. В частном случае при двух двоичных переменных A, B (т. е. для двух переменных высказываний) можно построить $2^{2^2} = 16$ различных функций алгебры логики, т. е. составить 16 неравносильных друг другу сложных логических выражений. Среди этих выражений содержатся все описанные выше логические связи (исключая отрицание, являющееся функцией одной переменной).

Важнейшую роль в алгебре логики играют следующие равносильные формулы, выражающие собой основные законы алгебры логики:

$$1)\ \overline{\overline{A}} = A;\ 2)\ A \wedge B = B \wedge A;\ 3)\ (A \wedge B) \wedge C = A \wedge (B \wedge C);$$
$$4)\ A \vee B = B \vee A;\ \ 5)\ (A \vee B) \vee C = A \vee (B \vee C);$$
$$6)\ A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C);\ 7)\ A \vee (B \wedge C) =$$
$$= (A \vee B) \wedge (A \vee C);\ 8)\ \overline{(A \vee B)} = \overline{A} \wedge \overline{B};\ 9)\ \overline{(A \wedge B)} = \overline{A} \vee \overline{B};$$
$$10)\ A \wedge A = A;\ 11)\ A \vee A = A;\ 12)\ A \wedge 1 = A;\ 13)\ A \vee 0 = A.$$

Проверка справедливости указанных соотношений может быть произведена на основании определений и таблиц логических операций — конъюнкции, дизъюнкции и отрицания. Соотношения 1 — 13 используются для преобразования сложных логических выражений к более удобному или простому виду. Соотношения 2, 3, 4, 5 показывают, что для операций конъюнкции и дизъюнкции справедливы переместительный и сочетательный законы, в силу чего многочленные конъюнкции и дизъюнкции можно писать без скобок. Например, вместо $[(A \wedge B) \wedge C] \wedge D$ можно просто написать $A \wedge B \wedge C \wedge D$. Для дальнейшего уменьшения количества скобок в логических формулах соглашаются считать связь с помощью знака $\wedge$ более тесной, чем с помощью знака $\vee$, а последнюю более тесной, чем связь с помощью знаков $\equiv$, $\not\equiv$ и $\supset$ Выражения вида $A \wedge B \wedge C \wedge ...$ часто называются произведением, а члены его — множителями.

Выражения вида $A \vee B \vee C \vee ...$ называют суммой, а члены его — слагаемыми. Соотношение 6 показывает, что в алгебре логики справедлив закон распределительности конъюнкции относительно дизъюнкции. Запись его в виде $A \cdot (B + C) = A \cdot B + A \cdot C$ (где точка означает логическое умножение, а плюс — логическое сложение) наглядно показывает аналогию между этим законом и законом распределительности умножения относительно сложения в обычной арифметике. Но в отличие от арифметики в алгебре логики имеет место еще закон распределительности дизъюнкции относительно конъюнкции, выражаемый соотношением 7. Оба распределительных закона позволяют производить над формулами алгебры логики преобразования раскрытия скобок и вынесения общих множителей подобно тому, как это делается в обычной алгебре (а также вынесение общих слагаемых).

Соотношения 8 и 9 называются законами де Моргана, вместе с соотношением 1 позволяют преобразовывать логические выражения к такому виду, что знаки отрицания будут относиться только к простым высказываниям.

Помимо соотношений 1 — 13 весьма полезными для преобразования логических выражений являются следующие равносильные формулы:

$$14)\ A \vee \overline{A} \wedge B = A \vee B;\ 15)\ A \wedge (\overline{A} \vee B) = A \wedge B;$$
$$16)\ A \vee A \wedge B = A;\ 17)\ A \wedge B \vee \overline{A} \wedge C =$$
$$= A \wedge B \vee \overline{A} \wedge C \vee B \wedge C;\ 18)\ A \wedge (A \vee B) = A;$$
$$19)\ (A \vee B) \wedge (\overline{A} \vee C) = (A \vee B) \wedge (\overline{A} \vee C) \wedge (B \vee C);$$
$$20)\ \overline{A} \vee A \wedge B = \overline{A} \vee B;\ 21)\ \overline{A} \wedge (A \vee B) = \overline{A} \wedge B;$$
$$22)\ A \supset B = \overline{A} \vee B;\ 23)\ A \equiv B = A \wedge B \vee \overline{A} \wedge \overline{B}.$$

Использование последних двух соотношений позволяет любые выражения, содержащие знаки $\supset$ и $\equiv$, приводить к выражениям, содержащим только знаки $\wedge$, $\vee$, $\overline{\ }$.

Если под переменными A, B, C и т. д. понимать не только высказывания, а вообще любую систему элементов,

для которой определены действия сложения, умножения и отрицания и которая удовлетворяет соотношениям 1 — 13, то получим абстрактную алгебру, называемую алгеброй Буля. В частности, высказывания и основные логические операции $\wedge$, $\vee$, $^-$ представляют собой частный случай (или, как говорят, интерпретацию) алгебры Буля. Другим примером алгебры Буля является алгебра классов, которая дает наглядное геометрическое истолкование для основных логических операций.

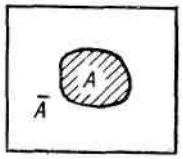


РИС. 4. Операция отрицания $\overline{A}$

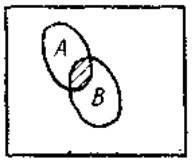


РИС. 5. Конъюнкция двух высказываний $A \wedge B$.

Рассмотрим высказывание *A,* в котором идет речь о принадлежности некоторого свойства α предметам какой-то области. Представим себе, что предметы нашей области изображаются точками части плоскости, ограниченной некоторым квадратом (рис. 4 — 9), которую мы обозначим через Q. Ясно, что точки плоскости Q разбиваются на два класса (множества): на класс точек, имеющих свойство α, т. е. таких, для которых $A = 1$, и на класс точек, не имеющих этого свойства, т. е. таких, для которых $A = 0$, причем каждая точка плоскости Q обязательно принадлежит одному (и только одному) из этих классов.

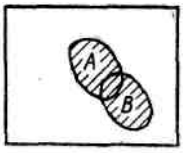


РИС. 6. Дизъюнкция двух высказываний $A \vee B$.

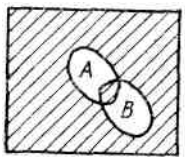


РИС. 7 Эквивалентность двух высказываний $A \equiv B$

Первый класс можно считать геометрическим изображением высказывания A и для удобства называть множеством A. При этом может получиться, например, картина, приведенная на рис. 4: высказывание A изображено в виде некоторой области, ограниченной замкнутым контуром и покрытой штриховкой. Очевидно, что высказывание $\overline{A}$ («не *A*») будет тогда изображаться множеством всех остальных точек квадрата Q. При такой интерпретации конъюнкция двух высказываний будет представляться пересечением двух множеств (рис. 5). Действительно, $A \wedge B = 1$ только тогда, когда $A = 1$ и $B = 1$, а это имеет место лишь для точек, одновременно принадлежащих множеству A и множеству B (их пересечению). Дизъюнкция $A \vee B$ будет изображаться множеством, которое получается путем объединения множеств A и B (рис. 6). Высказывание $A \equiv B$ изобразится так, как показано на рис. 7, ибо истинность $A \equiv B$ равна 1 либо при $A = 1$, $B = 1$, либо при $A = 0$, $B = 0$. Высказывание $A \not\equiv B$ показано на рис. 8. Его изображение без труда получается, если учесть, что $A \not\equiv B$ равно $\overline{B \equiv A}$. Подобные диаграммы, называемые диаграммами Венна, могут быть использованы для наглядного представления логических формул с целью их анализа и упрощения.

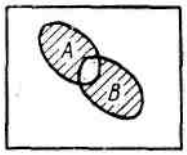


РИС. 8. Отрицание эквивалентности двух высказываний $A \not\equiv B$

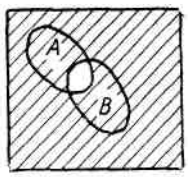


РИС. 9. Операция Шеффера (отрицание конъюнкции $\overline{A \wedge B}$).

Рассмотренные логические операции $\wedge$, $\vee$, $^-$, $\supset$, $\not\equiv$ не являются независимыми, а могут быть выражены друг через друга. В частности, из них можно выделить системы логических операций и с их помощью представить вообще все функции алгебры логики. Такие системы логических операций (иногда вместе с константами 1 или 0) называются функционально полными (например, системы операций: $^-$, $\wedge$, $\vee$, или $^-$, $\wedge$ или $^-$, $\vee$). Кроме того, существуют две операции: операция отрицания конъюнкции $\overline{A \wedge B}$ и операция отрицания дизъюнкции $\overline{A \vee B}$, через каждую из которых (одну) может быть выражена любая функция алгебры логики. Например, операция отрицания конъюнкции, утверждающая несовместимость двух высказываний, обозначается через A/B и называется операцией Шеффера или штрихом Шеффера. Через операцию Шеффера отрицание, конъюнкция и дизъюнкция выражаются так: $\overline{A} = A/A$, $A \wedge B = (A/B)/(A/B)$, $A \vee B = (A/A)/(B/B)$. Эта операция играет важную роль в теории логических схем и в теории ЭЦМ, поскольку электронная схема, реализующая операцию Шеффера, является универсальным функциональным элементом, при помощи которого в принципе могут быть построены любые функциональные схемы автоматов. Графическое представление операции A/B приведено на рис. 9.

Операции конъюнкции и дизъюнкции называются двойственными, и формулы алгебры логики, получаемые одна из другой заменой $\wedge$ на $\vee$ и $\vee$ на $\wedge$, также называются двойственными. Для двойственных формул F и F^* справедлива равносильность:

$$F(A_1, A_2, \cdots, A_n) = \overline{F} * (\overline{A}_1, \overline{A}_2, \cdots, \overline{A}_n)$$

В алгебре логики устанавливается следующий принцип двойственности: если формулы F и Φ равносильны, то и двойственные им формулы F^* и Φ^* также равносильны.

Наиболее наглядно структура формул алгебры логики видна тогда, когда они приведены к одной из двух так называемых нормальных форм. Первая из них — конъюнктивная нормальная форма (КНФ) — представляет собой некоторую конъюнкцию дизъюнкций, причем в каждой дизъюнкции отдельные члены представляют собой либо простые высказывания (т. е. высказывания, которые не включают в себя других высказываний), либо отрицания простых высказываний. Вторая нормальная форма — дизъюнктивная (ДНФ); она представляет собой некоторую дизъюнкцию конъюнкций; в каждой конъюнкции отдельные члены являются либо простыми высказываниями, либо их отрицаниями. Преобразование логических формул к той или иной нормальной форме осуществляется по следующим основным правилам: 1) со знаками $\wedge$ и $\vee$ можно оперировать так же, как в алгебре оперируют со знаками $\times$ (умножение) и $+$ (сложение), пользуясь переместительным, сочетательным и распределительным законами; 2) выражение с двойным (и вообще четным) количеством отрицаний можно заменить исходным выражением: $A = \bar{\bar{A}} = \bar{\bar{\bar{\bar{A}}}} = \cdots$; 3) отрицание конъюнкции двух высказываний можно заменить дизъюнкцией отрицаний этих высказываний, а отрицание дизъюнкции — конъюнкцией отрицаний $\overline{A \wedge B} = \bar{A} \vee \bar{B}$, $\overline{A \vee B} = \bar{A} \wedge \bar{B}$; 4) выражение $A \supset B$ можно заменить на $\bar{A} \vee B$, а выражение $A \equiv B$ – на $(\bar{A} \vee B) \wedge (A \vee \bar{B})$.

Порядок пользования этими правилами следующий: сначала, применяя правило 4, устраняют имеющиеся в формуле импликации и эквивалентности. Затем (по правилу 3) формула приводится к такому виду, когда знаки отрицания относятся к отдельным дизъюнктивным или конъюнктивным членам; наконец (применяя правила 1 и 2), раскрывают скобки и устраняют двукратные знаки отрицания.

Нормальные формы удобны для выделения двух важных классов формул: класса постоянно-истинных (т. е. совпадающих с константой 1) формул и класса постоянно-ложных (т. е. совпадающих с константой 0) формул алгебры логики, играющих существенную роль при упрощении логических выражений. При упрощении сложных формул, используя равносильности $A \wedge 1 = A$ и $A \vee 0 = A$, можно отбрасывать постоянно-истинные и постоянно-ложные высказывания, а, используя равносильности $A \wedge 0 = 0$ и $A \vee 1 = 1$, можно отбрасывать высказывания, конъюнктивно присоединенные к постоянно-ложному высказыванию и дизъюнктивно присоединенные к постоянно-истинному высказыванию.

Суждение о постоянной истинности сложной формулы может быть получено на основе применения правил: 1) формула $A \vee \bar{A}$ постоянно-истинная; 2) если A истинно, а B — произвольное высказывание, то формула $A \vee B$ истинна; 3) если A и B истинны, то и формула $A \wedge B$ тоже истинна. Применение этих правил позволяет вывести следующий критерий постоянной истинности сложной формулы. Постоянно-истинными являются такие формулы, в КНФ которых в каждую дизъюнкцию входит одно (по меньшей мере) основное высказывание вместе со своим отрицанием. Действительно, в каждой дизъюнкции в этом случае будет, по меньшей мере, один истинный член, а значит, истинны и все дизъюнкции, являющиеся членами КНФ, т. е. будет истинна вся КНФ, представляющая данную формулу алгебры логики.

Аналогичным образом посредством приведения к ДНФ можно определить, является ли данная формула алгебры логики постоянно-ложной или нет. Формула будет постоянно-ложной, если в каждой из конъюнкций, дизъюнктивно соединенных в ДНФ этой формулы, имеется, по крайней мере, одно высказывание вместе со своим отрицанием.

Формулы алгебры логики, являющиеся истинными при некоторых значениях входящих в них переменных, называются выполнимыми. Выполнимой будет любая формула, не являющаяся постоянно-ложной.

Существует универсальный способ представления любой функции алгебры логики $F(A_1, A_2, \ldots, A_n)$ в виде дизъюнкции всех конъюнкций вида

$$F(\alpha_1, \alpha_2, \cdots, \alpha_n) \wedge A_1' \wedge A_2' \wedge \cdots \wedge A_n' \qquad \text{(а)}$$

где $\alpha_1, \alpha_2, \ldots, \alpha_n$ — набор из значений 0 и 1, а $A'_i = A_i$ при $\alpha_i = 1$ и $A'_i = \bar{A}_i$ при $\alpha_i = 0$.

Действительно, для любого набора $\alpha_1, \alpha_2, \ldots, \alpha_n$ найдется одна и только одна конъюнкция вида (а), в которой постоянный множитель $F(\alpha_1, \alpha_2, \ldots, \alpha_n)$ будет иметь то же значение истинности, что и данная функция при данном наборе, а остальные множители $A'_1, A'_2, \ldots, A'_n$ будут равны 1. В этой конъюнкции распределение знаков отрицаний над переменными A_i будет совпадать с распределением нулей в наборе $\alpha_1, \alpha_2, \ldots, \alpha_n$. Оставляя в дизъюнкции только те конъюнкции, у которых постоянные множители равны единице (и удаляя их из конъюнкций по правилу $A \wedge 1 = A$), получим формулу, выражающую данную функцию в форме дизъюнкции конъюнкций вида $A'_1 \wedge A'_2 \wedge \ldots \wedge A'_n$. Эта форма называется дизъюнктивной совершенной нормальной формой (ДСНФ). Она обладает следующими свойствами: а) не имеет одинаковых слагаемых; б) каждое слагаемое ДСНФ содержит в качестве множителей либо основные переменные, либо их отрицания; в) ни в одном слагаемом ДСНФ нет двух одинаковых множителей и не содержится переменная вместе с ее отрицанием. Подобным же образом определяется конъюнктивная совершенная нормальная форма (КСНФ), представляющая собой конъюнкцию дизъюнкций, удовлетворяющих аналогичным условиям.

Совершенные нормальные формы могут использоваться при решении вопросов о равносильности сложных формул алгебры логики: две формулы алгебры логики являются равносильными, если они приводятся к одинаковым совершенным нормальным формам. Однако практическое применение этих форм затрудняется их громоздкостью, и поэтому часто используются так называемые минимальные нормальные формы.

Минимальная дизъюнктивная нормальная форма (МДНФ) представляет собой дизъюнкцию конъюнкций, в которой: а) нет повторяющихся множителей ни в одном слагаемом, б) нет таких пар слагаемых, в которых были

бы одинаковые множители, и в) для всяких двух слагаемых, в которых имеется одна общая переменная, входящая в одно слагаемое в прямом виде, а в другое — в виде отрицания, имеется третье слагаемое, представляющее собой конъюнкцию остальных множителей первых двух слагаемых.

Любая дизъюнктивная нормальная форма (не обязательно совершенная) может быть приведена к МДНФ путем следующих преобразований:

а) для каждой пары слагаемых вида $R_k \wedge A_i$ и $R_l \wedge \overline{A_i}$ (где R_k и R_l — произведения остальных множителей) добавляют дополнительные слагаемые вида $R_k \wedge R_l$ (при этом получается формула, равносильная исходной в силу соотношения $R_k \wedge A_i \vee R_l \wedge \overline{A_i} = R_k \wedge A_i \vee R_l \wedge \overline{A_i} \vee R_k \wedge R_l$);

б) применяя так называемые формулы поглощения и склеивания ($A \wedge B \vee A = A$; $A \wedge B \vee A \wedge \overline{B} = A$), соотношение $A \wedge 1 = A$, переместительный и сочетательный законы, устраняют повторяющиеся множители и слагаемые.

Пример преобразования к МДНФ:

$$A\wedge B\wedge C\vee A\wedge\overline{B}\wedge C\vee A\wedge\overline{B}\wedge\overline{C}\vee\overline{B}\wedge\overline{C}\vee\overline{A}\wedge C=$$
$$=A\wedge B\wedge C\vee A\wedge B\wedge C\vee A\wedge\overline{B}\wedge\overline{C}\vee\overline{B}\wedge\overline{C}\vee\overline{A}\wedge C\vee A\wedge C\vee$$
$$\vee\overline{A}\wedge\overline{B}=A\wedge B\wedge C\vee A\wedge C\vee A\wedge\overline{B}\wedge C\vee A\wedge\overline{B}\wedge\overline{C}\vee\overline{B}\wedge\overline{C}\vee$$
$$\vee\overline{A}\wedge\overline{B}\vee C=A\wedge C\vee A\wedge\overline{B}\wedge C\vee\overline{B}\wedge\overline{C}\vee\overline{A}\wedge\overline{B}\vee C=$$
$$=A\wedge C\vee C\vee\overline{B}\wedge\overline{C}\vee\overline{A}\wedge\overline{B}=C\vee\overline{B}\wedge\overline{C}\vee\overline{A}\wedge\overline{B}\vee\overline{B}=$$
$$=C\vee\overline{B}\wedge\overline{C}\vee\overline{B}=C\vee\overline{B}.$$

Существуют также и другие методы получения МДНФ. Практическое применение получил так называемый метод минимизирующих карт, позволяющий получать МДНФ для любой функции алгебры логики, заданной в виде совершенной дизъюнктивной нормальной формы. Минимизирующая карта для функций, зависящих от трех переменных, является таблицей следующего вида (табл. 7; точки означают знак конъюнкции).

Таблица 7

A	B	C	$A.B$	$A.C$	$C.B$	$A.B.C$
A	B	$\bar{C}$	$A.B$	$A.\bar{C}$	$\bar{C}.B$	$A.B.\bar{C}$
A	$\bar{B}$	C	$A.\bar{B}$	$A.C$	$C.\bar{B}$	$A.\bar{B}.C$
A	$\bar{B}$	$\bar{C}$	$A.\bar{B}$	$A.\bar{C}$	$\bar{C}.\bar{B}$	$A.\bar{B}.\bar{C}$
$\bar{A}$	B	C	$\bar{A}.B$	$\bar{A}.C$	$C.B$	$\bar{A}.B.C$
$\bar{A}$	B	$\bar{C}$	$\bar{A}.B$	$\bar{A}.\bar{C}$	$\bar{C}.B$	$\bar{A}.B.\bar{C}$
$\bar{A}$	$\bar{B}$	C	$\bar{A}.\bar{B}$	$\bar{A}.C$	$C.\bar{B}$	$\bar{A}.\bar{B}.C$
$\bar{A}$	$\bar{B}$	$\bar{C}$	$\bar{A}.\bar{B}$	$\bar{A}.\bar{C}$	$\bar{C}.\bar{B}$	$\bar{A}.\bar{B}.\bar{C}$

Табл. 7 содержит всевозможные конъюнкции заданных простых переменных и имеет всегда 2^n строк и 2^n—1 столбцов (сами простые переменные также могут рассматриваться как конъюнкции в силу соотношения $A \wedge 1 = A$). Порядок пользования минимизирующей картой следующий (табл. 8):

а) вычеркиваются все строки таблицы, соответствующие тем конъюнкциям правого столбца, которые отсутствуют в данной ДСНФ;

б) в оставшихся строках в каждом столбце зачеркиваются элементы, одинаковые с теми, которые уже зачеркнуты в этом столбце;

в) из каждой незачеркнутой строки выбирается по одной конъюнкции, содержащей минимальное число множителей, и эти конъюнкции соединяются знаками дизъюнкции. После исключения повторяющихся множителей и слагаемых получают МДНФ.

Таблица 8

~~A~~	~~B~~	C	~~$A.B$~~	$A.C$	$C.B$	$A.B.C$
~~A~~	~~B~~	~~$\bar{C}$~~	~~$A.B$~~	~~$A.\bar{C}$~~	~~$\bar{C}.B$~~	~~$A.B.\bar{C}$~~
~~A~~	$\bar{B}$	C	$A.\bar{B}$	$A.C$	$C.\bar{B}$	$A.\bar{B}.C$
~~A~~	$\bar{B}$	~~$\bar{C}$~~	$A.\bar{B}$	~~$A.\bar{C}$~~	$\bar{C}.\bar{B}$	$A.\bar{B}.\bar{C}$
~~$\bar{A}$~~	~~B~~	C	~~$\bar{A}.B$~~	$\bar{A}.C$	$C.B$	$\bar{A}.B.C$
~~$\bar{A}$~~	~~B~~	~~$\bar{C}$~~	~~$\bar{A}.B$~~	~~$\bar{A}.\bar{C}$~~	~~$\bar{C}.B$~~	~~$\bar{A}.B.\bar{C}$~~
~~$\bar{A}$~~	$\bar{B}$	C	$\bar{A}.\bar{B}$	$\bar{A}.C$	$C.\bar{B}$	$\bar{A}.\bar{B}.C$
~~$\bar{A}$~~	$\bar{B}$	~~$\bar{C}$~~	$\bar{A}.\bar{B}$	~~$\bar{A}.\bar{C}$~~	$\bar{C}.\bar{B}$	$\bar{A}.\bar{B}.\bar{C}$

Например, для функции

$$F(A,B,C) = A \wedge B \wedge C \vee A \wedge \bar{B} \wedge C \vee A \wedge \bar{B} \wedge \bar{C} \vee \bar{A} \wedge B \wedge C \vee \bar{A} \wedge \bar{B} \wedge C \vee \bar{A} \wedge \bar{B} \wedge \bar{C}$$

МДНФ согласно табл. 8 будет иметь вид $C \vee B$

Путем приведения к МДНФ можно установить равносильность различных сложных формул алгебры логики. МДНФ используются при решении задач синтеза параллельно-последовательных переключательных схем, обладающих минимальным числом элементов.

Использование аппарата алгебры логики в теории устройств дискретного действия (теории релейных схем, ЭЦМ и т. п.) основано на том, что элементы этих устройств являются двухпозиционными приборами, т. е. приборами, которые по условиям работы могут находиться лишь в одном из двух различных устойчивых состояний. Так, электрический контакт может быть замкнутым или разомкнутым, электронная лампа заперта или отперта и т. д. Одному из состояний двухпозиционного элемента можно поставить в соответствие 1, а другому — 0, и рассматривать 1 и 0 как значения истинности высказываний вида: «контакт α замкнут», «лампа Л отперта» и т. п. Конъюнкция такого рода высказываний будет тогда средством выражения последовательного соединения элементов, а дизъюнкция — их параллельного соединения. Это обеспечивает возможность применить средства алгебры логики к задачам анализа и синтеза переключательных схем.

Задача анализа схем состоит в следующем. Требуется, имея какую-либо готовую схему, описать ее работу логическим выражением, задающим некоторую функцию алгебры логики; затем путем алгебраических преобразований этого выражения исследовать вопрос об экономичности схемы, т. е. выяснить, нельзя ли получить более простую схему, содержащую меньшее количество элементов, которая тоже реализовала бы данную функцию алгебры логики.

Задача синтеза схемы состоит в том, чтобы, имея логическое выражение, описывающее некоторую логическую функцию, определить, из каких элементарных схем и каким образом должна быть построена сложная схема, реализующая данную функцию. Для этого необходимо логическое выражение рациональным образом преобразовать и расчленить так, чтобы каждый из членов мог быть представлен элементарной схемой, а общее число элементов было минимальным.

Помимо задач анализа и синтеза схем дискретного действия методы алгебры логики находят применение в теории ЭЦМ при составлении логических схем программы работы этих машин и для описания процессов переработки информации. С помощью формул алгебры логики описываются также различные математические действия, выполняемые этими машинами, что связано с использованием в машинах для представления величин различных двоично-кодированных систем счисления, применяющих только две цифры 0 и 1.

Важной областью применения алгебры логики является алгоритмизация процессов переработки информации и управления в технике, экономике и других областях. С помощью средств алгебры логики могут составляться и минимизироваться таблицы логических решений задач управления, в которых каждой возможной ситуации соответствует либо определенное решение, либо указание пути поиска такого решения (см, например, язык ТАБСОЛ [19]).

2. ОСНОВНЫЕ СВЕДЕНИЯ ОБ УСТРОЙСТВЕ ЭЛЕКТРОННЫХ ЦИФРОВЫХ МАШИН И ПРОГРАММИРОВАНИИ

§ 4 УСТРОЙСТВО ЭЛЕКТРОННЫХ ЦИФРОВЫХ МАШИН

Электронные цифровые машины (ЭЦМ) отличаются от других вычислительных средств (настольных счетно-клавишных, счетно-перфорационных машин и др.) двумя принципиальными особенностями: а) полной автоматизацией вычислительного процесса, обеспечиваемой применением программного управления, и б) наличием запоминающих устройств большой емкости.

Появление ЭЦМ относится к 1945 г., когда вступила в строй машина ЭНИАК (США), построенная специально для расчетов траекторий снарядов. За прошедшие 20 лет электронная цифровая вычислительная техника сделала огромный скачок и сейчас представляет собой одну из наиболее развитых отраслей промышленности.

ЭЦМ получили широкое применение в экономике, науке и технике и обеспечили успехи в таких областях, как ядерная физика и техника, ракетостроение и космонавтика, а также в автоматизации технологических процессов, в экономическом анализе и управлении.

Сейчас почти все ведущие фирмы за рубежом широко применяют ЭЦМ для экономических и технических расчетов, при этом резко сокращая ручной труд и повышая качество и скорость обработки информации. Широко применяются эти машины и в СССР,

Основные принципы устройства ЭЦМ

На рис. 10 показана общая структурная схема ЭЦМ. ЭЦМ состоит из следующих основных частей: устройства ввода информации в машину; устройства вывода информации из машины; внешнего запоминающего устройства (ВЗУ); оперативного запоминающего устройства (ОЗУ); арифметического устройства (АУ); устройства управления (УУ); пульта управления (ПУ).

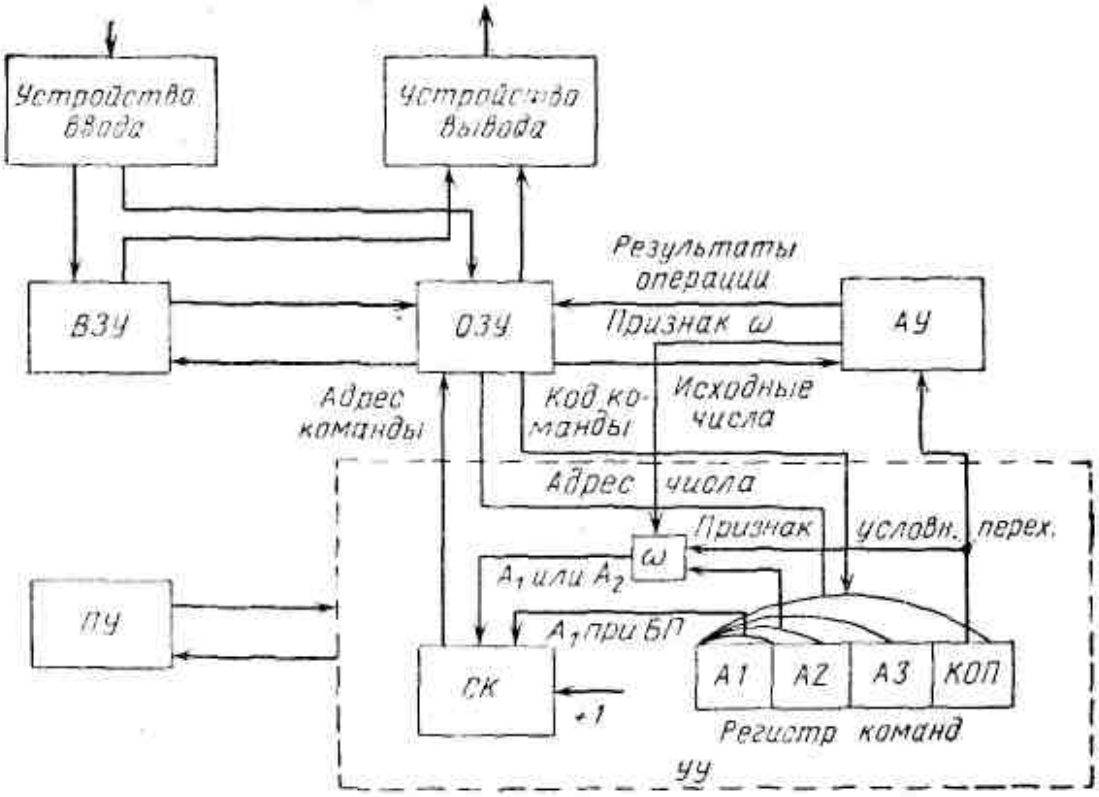


РИС. 10. Блок-схема цифровой вычислительной машины.

Указанные устройства соединены между собой магистралями, по которым передаются данные и сигналы, управляющие работой устройств.

С целью пояснения принципов работы машины на рис. 10 внутри УУ показаны его основные блоки, назначение которых будет описано в дальнейшем.

Перед решением задачи на ЭЦМ должны быть подготовлены программа и исходные данные. Эта информация пишется программистами на специальных бланках, с которых производится печатание ее человеком (перфораторщиком) на специальном приборе — перфораторе — в виде пробивок на стандартных листах картона (перфокартах) или бумажных лентах (перфолентах). Затем перфокарты или перфоленты вставляются в «читающее» устройство (прощупывающее с помощью фотоэлементов или механических контактов пробитые отверстия), которое превращает пробивки в электрические сигналы, поступающие в машину. Таким образом, к устройствам ввода относятся как внешние устройства, служащие для подготовки перфокарт или перфолент, так и собственно устройства ввода, осуществляющие «чтение» этих перфокарт или перфолент и ввод соответствующих сигналов в машину.

В последние годы появились различные новые устройства ввода: читающие оптическим способом обычный и цифровой текст стандартного шрифта и преобразующие его в сигналы, а также устройства, читающие буквенные и цифровые тексты, приготовленные специальным способом (в виде записей магнитными чернилами на чеках, в виде графитовых отметок на специальных ведомостях, нарядах и т. п.).

Устройства вывода информации осуществляют в большинстве машин перфорацию выводимых данных на перфокарты или перфоленты или печатают эти данные на бумажных лентах (узких и широких). Кроме того, есть устройства, выдающие данные в виде графиков или печатающие сразу готовые документы (ведомости, отчеты и т. п.).

Запоминающие устройства (ЗУ) ЭЦМ состоят из отдельных ячеек, в каждой из которых может храниться одно число. ЗУ делятся на два основных вида: внешние (ВЗУ) и оперативные (ОЗУ). ОЗУ имеют сравнительно небольшую емкость (4—64 тысячи ячеек), но обладают высоким быстродействием и, самое главное, позволяют обращаться к отдельным ячейкам в любой последовательности. Все ячейки ОЗУ перенумерованы подряд и каждой из них присвоен постоянный номер, называемый адресом ячейки. Для обращения к определенной ячейке необходимо указать ее адрес, после чего в эту ячейку может быть внесено новое число или «прочитано» (выбрано в виде кода, представленного электрическими сигналами) то число, которое там до этого хранилось. ОЗУ работают, как правило, по принципу замещений, при котором любое вновь записываемое в ячейку число автоматически стирает прежнее число, хранившееся в этой ячейке. ОЗУ служат для непосредственной совместной работы с арифметическим устройством (АУ). Из ОЗУ выбираются числа, которые по очереди поступают в АУ, там над ними выполняется требуемая операция и результат посылается в определенную (свободную) ячейку ОЗУ. После этого выбираются следующие числа из ОЗУ и т. д. Подобный процесс повторяется многократно согласно заданной программе, в которой для каждого такого этапа работы (так называемого такта работы ЭЦМ) имеется одна команда. Команда указывает адреса ячеек, из которых должны быть выбраны числа, указывает код операции, который нужно выполнить (сложение, вычитание, умножение и др.), а также адрес ячейки ОЗУ, в которую должен быть послан результат операции.

ОЗУ ЭЦМ строятся на самых различных технических принципах и физических элементах. Наибольшее распространение получили ОЗУ на магнитных ферритовых сердечниках с прямоугольной петлей гистерезиса.

Каждый сердечник может находиться в одном из двух устойчивых состояний, до тех пор, пока внешний сигнал не переведет его в противоположное состояние. Одному состоянию ставится в соответствие значение цифры 0, а другому — значение цифры 1, и таким образом каждый сердечник может быть использован для хранения одной так называемой двоичной цифры (см. ниже).

В последние годы начали применяться ОЗУ, построенные на тонких магнитных пленках, криотронах (запоминающих элементах, использующих эффект сверхпроводимости), туннельных диодах и других элементах.

ВЗУ обладают большой емкостью (сотни тысяч и миллионы ячеек), но не допускают возможности произвольного обращения к отдельным ячейкам. В ВЗУ в основном используются магнитные ленты (МЛ), т. е. ленты из пластического немагнитного материала, на который нанесено тонкое ферромагнитное покрытие. Данные на МЛ запоминаются в ячейках, объединенных в зоны, и отдельные адреса (номера) имеют зоны, а не ячейки, поэтому ВЗУ обеспечивают возможность только группового считывания или записи. ВЗУ непосредственно не связано с АУ и в вычислениях не участвует. ВЗУ служит резервом для ОЗУ, из которого переписываются по мере надобности исходные данные для обработки и в который выводятся результаты вычислений. Перепись данных из ОЗУ в ВЗУ или обратно осуществляется сразу целыми зонами.

Большое применение в ЭЦМ получили также ЗУ на магнитных барабанах (МБ) и магнитных дисках (МД), которые занимают по своим характеристикам промежуточное положение между ОЗУ и МЛ. На МБ и МД хранятся обычно большие программы, которые по частям переписываются в ОЗУ для выполнения, а также часто используемые таблицы данных, промежуточных результатов, которые потребуются в последующих вычислениях.

Принцип работы МБ и МД такой же, как и МЛ. Двоичные цифры (0 или 1) запоминаются в них в виде магнитных отметок определенной полярности, расположенных в виде строк или в виде дорожек вдоль (или поперек) образующей барабана или диска. Запись или считывание данных с МЛ, МБ, МД производится специальными магнитными головками в процессе перемещения под ними намагниченных участков.

Арифметическое устройство (АУ) служит для выполнения операций над числами. В АУ поступают из ОЗУ исходные числа (два или одно), а также указание (из УУ), какую операцию нужно выполнить. АУ выполняет эту операцию' и выдает число, являющееся результатом, которое записывается в ОЗУ по требуемому адресу. Любая ЭЦМ имеет строго фиксированный набор операций, выполняемых АУ. Каждой операции присвоен постоянный номер, называемый кодом операции. Этот код и поступает в АУ, определяя вид выполняемой операции. Существует много различных вариантов построения АУ. Тип АУ определяется системой счисления, в которой оно работает (двоичной или десятичной), и способом работы (параллельным или последовательным). АУ, работающее в двоичной системе счисления, проще по конструкции, но требует преобразования исходных данных из общепринятой десятичной системы в двоичную и обратного преобразования результатов вычислений. В двоичной системе основанием является число 2 и используется всего две различные цифры 0 и 1.

Подобно тому, как любое число в десятичной системе счисления представляется в виде суммы степеней основания 10 с соответствующими коэффициентами (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), в двоичной системе любое число может быть представлено в виде суммы степеней основания 2 с соответствующими коэффициентами (0, 1).

Например,

$$265=2\cdot10^2+6\cdot10^1+5\cdot10^0.$$

То же число в двоичной системе будет выглядеть так:

$$265=1\cdot2^8+0\cdot2^7+0\cdot2^6+0\cdot2^5+0\cdot2^4+1\cdot2^3+0\cdot2^2+0\cdot2^1+1\cdot2^0=100001001.$$

Двоичные эквиваленты первых 16 десятичных чисел представлены в следующей таблице:

Десятичное число	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Двоичное число	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111

В двоичной системе исключительно просто выполняются арифметические операции.

Ниже приведена таблица умножения и сложения для двоичных чисел:

$$\begin{array}{ll} 0\times0=0 & 0+0=0 \\ 0\times1=0 & 0+1=1 \\ 1\times0=0 & 1+0=1 \\ 1\times1=1 & 1+1=10 \end{array}$$

Действия над многоразрядными числами производятся так же, как и в десятичной системе, с учетом переносов между разрядами.

Ниже приведены примеры сложения и умножения пятиразрядных двоичных чисел.

$$\begin{array}{r} 11011 \\ +\ 10010 \\ \hline 101101 \end{array} \qquad \begin{array}{r} 11011 \\ \times\ 10010 \\ \hline 110110 \\ 1101100 \\ \hline 111100110 \end{array}$$

Обычно в ЭЦМ используются двоичные числа, соответствующие по точности 10 — 13-разрядным десятичным числам, что приводит к тому, что ячейки ЗУ рассчитывают приблизительно под 30 — 50-разрядные двоичные числа. Размер всех ячеек и, следовательно, разрядность чисел, с которыми оперирует машина, является фиксированным, однако это не означает, что в машине не могут быть представлены малые числа. Просто в этих случаях в старших разрядах ячеек записываются нули.

АУ, работающие в десятичной системе счисления, обычно применяются в ЭЦМ, предназначенных для экономических расчетов, в которых имеется большое количество исходных данных и результатов расчетов и для которых их перевод из одной системы счисления в другую потребовал бы слишком много времени работы машины.

Операции в десятичной системе выполняются в АУ в основном по тем же правилам, что и при ручном счете, с той только разницей, что отдельные цифры десятичных чисел представляются все же в двоичной системе.

Десятичные АУ помимо более сложного устройства обладают еще и меньшей скоростью работы, чем двоичные АУ. С технической стороны АУ представляют собой комплексы переключательных схем (счетчиков, регистров, клапанов, реле и т. д.), построенных на электронных лампах или полупроводниковых элементах (диодах, триодах).

Основой таких схем является схема триггера, т. е. схема с двумя устойчивыми состояниями, одно из которых соответствует запоминанию 1, а другое — 0. Кроме того, важнейшую роль играют схемы клапанов, т. е. такие схемы, которые выдают сигнал на выходе при наличии определенной комбинации (пли нескольких комбинаций) сигналов на входах.

АУ параллельного действия воспринимает из ОЗУ исходные числа и выполняет над ними действия сразу по всем разрядам, в то время как АУ последовательного действия воспринимает из ОЗУ числа и выполняет над ними действия последовательно разряд за разрядом, начиная с младших разрядов. Первые АУ более быстродействующие, но и более сложные по устройству, чем вторые.

Устройство управления (УУ) ЭЦМ служит для последовательной выборки команд программы и управления работой всех устройств ЭЦМ в процессе выполнения каждой команды.

На рис. 10 устройство управления показано более подробно: в него входят два основных блока: счетчик команд (СК) и регистр команд (РК). Счетчик команд осуществляет последовательный счет выполняемых команд и выдает каждый раз номер очередной команды, являющийся адресом ячейки ОЗУ, в которой хранится эта команда. Выбранный из этой ячейки код очередной команды поступает из ОЗУ на РК для исполнения. С РК выдаются для выполнения последовательно отдельные части команды. Сначала выдается в ОЗУ адрес одного числа ($A1$), участвующего в операции. После выборки из ОЗУ этого числа и передачи его в АУ, из РК выдается в ОЗУ адрес второго участвующего в операции числа ($A2$), которое также поступает из ОЗУ в АУ. В этот же момент в АУ с РК поступает код операции (КОП), которая должна быть выполнена. Затем в ОЗУ подается адрес ($A3$) ячейки, в которую записывается результат операции.

На этом такт выполнения очередной команды заканчивается; в СК прибавляется единица и увеличенный на единицу адрес поступает в ОЗУ. Производится выборка и выполнение таким же образом следующей команды программы.

Часто бывает необходимо изменить указанный последовательный порядок выполнения команд и перейти к какой-нибудь внеочередной команде. Такие переходы делаются по специальным командам и могут быть заранее полностью определенными (безусловные переходы) или альтернативными (условные переходы). Команда безусловного перехода передает один из своих адресов (например, $A1$) не в ОЗУ для выборки числа, а на СК, откуда он поступает в ОЗУ уже как адрес следующей команды. Команда условного перехода может передавать на СК один из двух адресов ($A1$ пли $A2$), причем передача их происходит не непосредственно на СК, а через специальный клапан ω, управляемый сигналом, поступающим от АУ. Характер этого сигнала зависит от результата выполненной операции и заранее неизвестен программисту; например, при вычитании двух чисел этот сигнал при положительном знаке результата будет пропускать с РК на СК первый адрес $A1$, а при отрицательном знаке результата — второй адрес $A2$.

Используя команду условного перехода, можно менять ход вычислений в зависимости от получающихся результатов, а также возвращаться к повторению одних и тех же участков программы для различных исходных данных.

Мы рассмотрели работу устройства управления для случая трехадресной команды, т. е. такой команды, которая содержит в себе помимо кода операции еще три адреса.

Кроме ЭЦМ с такими командами существуют ЭЦМ с двух- и одноадресными командами, а также четырехадресными командами, принцип работы которых аналогичен описанному.

Например, при одноадресной системе команд в каждой команде имеется только один адрес и с помощью такой команды явно может быть указана только одна величина. Поэтому для выполнения, например, операции сложения потребуется применить три одноадресных команды вместо одной трехадресной команды. Первой командой будет осуществлена посылка одного слагаемого в арифметическое устройство. Вторая команда обеспечит посылку второго слагаемого в это устройство и выполнение операции сложения. Третья команда необходима для посылки результата операции в нужную ячейку памяти. В четырехадресных командах четвертый адрес обычно используется для указания следующей команды; при этом команды программы могут располагаться в памяти машины не обязательно подряд, а в любом произвольном порядке.

Пульт управления (ПУ) ЭЦМ служит для пуска и остановки ЭЦМ и контроля за ее работой; кроме того, ПУ используется при проведении профилактических и ремонтных работ.

Все ЭЦМ по своему назначению и особенностям конструкции делятся на три основных типа:

а) для научно-технических вычислений;
б) для обработки экономической информации;
в) для управления технологическими процессами.

ЭЦМ для научно-технических вычислений характеризуются высокой точностью вычислений (представлением чисел в ячейках с большим количеством разрядов и большим диапазоном измерения чисел),

большим объемом ОЗУ, наличием небольшого числа (2 — 4) МБ и МЛ, сравнительно ограниченным составом устройств ввода и вывода данных (заметим, что для научно-технических вычислений характерен большой объем счета при сравнительно небольшом объеме вводимых и выводимых данных). Эти машины работают, как правило, в двоичной системе.

ЭЦМ для обработки экономической информации характеризуются большим количеством МБ и МЛ, разветвленной системой устройств ввода и вывода (сюда входят разнообразные устройства для печати готовых документов — ведомостей, сводок, отчетов и т. п. и устройства для восприятия информации с первичных документов — чеков, нарядов-заказов и т. п.). Эти машины работают, как правило, в десятичной системе счисления, что связано с большими объемами вводимых и выводимых данных, при которых работа по переводу этих данных из одной системы счисления в другую потребовала бы слишком много машинного времени.

ЭЦМ для управления технологическими процессами, так называемые управляющие машины, характеризуются, прежде всего, наличием устройств сопряжения машин с различными физическими датчиками первичной информации (датчики температуры, давления, перемещения, скорости, регистрации количества изделий и т. п.), а также с каналами связи, связывающими ЭЦМ с источниками информации и управляемыми органами. На основе поступающей информации о фактическом состоянии управляемых объектов ЭЦМ по имеющейся в ней программе периодически рассчитывает необходимые команды и выдает их для исполнения органам управления. Особенностью структуры управляющих ЭЦМ является наличие системы прерывания работы машины для вводов новой информации и выдачи команд управления, а также системы оперативной связи человека-оператора с машиной, позволяющей ему получать в наглядной форме информацию о ходе процесса управления и вносить корректуру в решения, вырабатываемые ЭЦМ.

Машины указанных типов в зависимости от производительности (быстродействие, емкость ЗУ, количество устройств ввода и вывода) делятся на три класса: большие машины (быстродействие от нескольких сот до миллионов операций в секунду, ОЗУ — 50—100 тыс. ячеек), средние машины (быстродействие от 10 до 100 тыс. операций в секунду, ОЗУ — 4 - 46 тыс. ячеек), малые машины (быстродействие до 10 тысяч операций в секунду, ОЗУ — 1 - 4 тыс. ячеек).

В последние годы стали выпускаться ЭЦМ, построенные по агрегатному принципу и допускающие наращивание производительности в зависимости от потребностей путем присоединения дополнительных блоков и устройств.

В качестве примера конкретного образца машины мы опишем полупроводниковую цифровую вычислительную машину «Минск-2», универсального назначения. Эта машина серийно выпускается нашей промышленностью для расчетов и обработки информации в различных областях народного хозяйства. Эта машина может быть отнесена к классу машин средней производительности. Она характеризуется следующими данными. Среднее быстродействие 5—6 тыс. операций в секунду. Емкость оперативного запоминающего устройства 4096 ячеек по 37 двоичных разрядов. Структура машины предусматривает возможность присоединения второго блока ОЗУ такой же емкости. Машина имеет внешний накопитель на магнитной ленте емкостью в 400 тыс. ячеек (по 37 разрядов). Предусмотрена также возможность увеличения емкости внешнего накопителя до 1,6 млн. ячеек. Машина имеет разветвленную систему команд двухадресного формата, обеспечивающую широкие логические возможности для построения различных процессов переработки информации. Особенностью машины является возможность ввода и вывода не только числовой, но и буквенной информации, что имеет большое значение для решения планово-экономических задач.

Ввод данных осуществляется с помощью бумажной перфоленты со скоростью 800 строк в секунду. Вывод данных может производиться либо на печатающую машину (БПМ-20) со скоростью 20 слов в секунду либо на перфоратор результатов со скоростью 20 строк в секунду. Машина «Минск-2» занимает площадь 40—50 *м*^{2}, потребляет электроэнергию мощностью около 4 *квт.*

Ниже приведена структура основного формата команд машины «Минск-2», осуществляющих арифметические и логические операции, а также передачи управления:

Зн 1...6	7, 8	9...12	13...24	24...36
Код операции 7 р.		(4 р.)	*А*1 (первый адрес) 12 р.	*А*2 (второй адрес) 12 р.

Знаковый разряд (Зн) вместе с первыми шестью разрядами (1— 6) образуют семиразрядный код операции.

Разряды 7 и 8 служат для расширения адресов *А*1 и *А*2 в том случае, когда машина «Минск-2» комплектуется двойным блоком памяти (общей емкостью в 8192 ячейки). Ясно, что в этом случае 12-разрядные адреса (2^{12}=4096) не охватывают всего объема ОЗУ.

Большой интерес представляет часть команды, помещенная в четырех разрядах с 9-го по 12-й и называемая адресом индексной ячейки. В машине «Минск-2» первые 15 ячеек ОЗУ являются так называемыми индексными ячейками и служат для автоматической модификации адресов команд *А*1 и *А*2.

При выполнении команды каждый раз проверяется значение ее индексного поля (т. е. разряды с 9-го по 12-й), и если оно отлично от нуля, то к адресам этой команды *А*1 и *А*2 прибавляется содержимое той индексной ячейки, адрес которой указан в индексном поле,

В индексных ячейках заранее запасается число, которое состоит из трех частей по 12 разрядов. Первая часть этого числа является счетчиком числа повторений цикла; она изменяется по специальной команде при каждом проходе циклического участка программы (см. ниже). Вторая часть содержимого индексной ячейки, расположенная в разрядах с 13-го по 24-й, является шагом переадресации для первого адреса команды *А*1, и третья часть содержимого индексной ячейки, расположенная в разрядах с 25-го по 36-й, является шагом переадресации для второго адреса команды *А*2. Наличие сравнительно большого количества индексных ячеек и

возможность использования в качестве индексных не специальных регистров, а ячеек основной оперативной памяти являются ценной особенностью машины «Минск-2».

Автоматическая модификация адресов команд с помощью индексных ячеек в сочетании с изменением счетчика циклов с выработкой сигнала перехода со по окончании заданного числа повторения цикла существенно повышает эффективность программирования сложных вычислительных и логических процессов переработки информации, позволяет получать достаточно компактные программы, являющиеся одновременно экономичными с точки зрения расхода машинного времени.

Наличие большого количества индексных ячеек позволяет также эффективно использовать метод составления программ в так называемых относительных адресах, когда программа и массивы исходных и промежуточных данных могут помещаться при разных вычислениях в разные места памяти. Настройка программы при этом может производиться путем задания так называемых базисных адресов, помещаемых в соответствующих индексных ячейках и определяющих начальные адреса массивов программы (для команд переходов) и данных (для команд, обращающихся к данным).

Весьма интересной и полезной особенностью структуры машины «Минск-2» является наличие программно-управляемой системы прерывания программ. Аналогичные системы имеются в ряде наиболее производительных зарубежных машин («Стретч» — США, «Атлас» — Англия и др.).

Система автоматического прерывания программы обеспечивает возможность оперативного реагирования вычислительной машины на различные внешние и внутренние сигналы, которые могут возникнуть в ходе вычислений. Наибольшее значение подобные системы имеют в вычислительных управляющих машинах, работающих в реальном масштабе времени и связанных каналами связи с различными внешними объектами, являющимися источниками данных для машины и потребителями информации, вырабатываемой машиной. Так как внешние сообщения поступают в такие машины в произвольные моменты времени, то необходима специальная система прерывания, которая бы приостанавливала выполнение основной программы на время переписи сообщения в оперативную память, а также на время выдачи очередной порции подготовленных результатов вычислений.

Помимо этих функций ввода и вывода информации система прерывания программы служит для реагирования на различные случайно возникающие события (сбои в работе машины, сигналы оператора, вводимые с пульта управления, получение ненормальных результатов и т. д.).

В системах прерывания программы почти во всех машинах, имеющих эти системы, для каждой возможной причины прерывания отведен отдельный разряд в специальном регистре, на который поступают сигналы прерывания.

Так как в различных условиях (в зависимости от важности и особенностей выполняемой в данный момент программы) может допускаться прерывание программы не от всех причин (а иногда требуется вообще запретить прерывание некоторых программ), то вводится еще так называемый защитный регистр, разряды которого соответствуют разрядам регистра прерывания. На этот защитный регистр по специальной команде могут устанавливаться различные коды, определяющие возможность прерывания программы разными причинами. Прерывание осуществляется в том случае, если есть сигнал прерывания на данном разряде регистра прерывания, а на соответствующем разряде защитного регистра стоит разрешение прерывания.

В машине «Минск-2» каждой причине прерывания соответствует специальная «ячейка прерывания», в которую автоматически передается управление при возникновении прерывания по данной причине. В этих ячейках прерывания заранее должны быть запасены команды перехода к программам, которые должны выполняться машиной при появлении соответствующих сигналов прерывания. Эти команды перехода своим первым адресом указывают адрес перехода к нужной программе, а вторым адресом указывают адрес ячейки, куда должна быть заслана команда возврата, обеспечивающая возвращение после прерывания к прерванному месту основной программы.

В машине «Минск-2» возможно только одноступенчатое прерывание, т. е. переход к некоторой прерывающей программе автоматически запрещает дальнейшее прерывание, до тех пор, пока оно не будет разрешено специальной командой.

§ 5 ПРОГРАММИРОВАНИЕ

Программирование — составление программ решения математических и информационных задач на электронных цифровых вычислительных машинах (ЭЦМ).

Под программированием понимают также область прикладной математики, занимающуюся разработкой методов программирования задач для ЭЦМ.

Различают два основных раздела программирования: непосредственное программирование и автоматическое программирование. При непосредственном программировании вся работа, начиная от разработки общей схемы программы и кончая кодированием и вводом программы в машину, выполняется человеком-программистом. При автоматическом программировании человек-программист составляет только схему программы и записывает ее в сокращенном символическом виде. Вся же техническая работа по составлению и кодированию программы выполняется самой ЭЦМ с помощью специальной программирующей программы.

Программное управление ЭЦМ. Сущность программирования состоит в представлении алгоритма решения задачи в виде последовательности элементарных операций, выполняемых ЭЦМ. Любая ЭЦМ может выполнять определенный состав арифметических и логических операций над числовыми кодами. К арифметическим операциям относятся сложение, вычитание, умножение и деление. Логическими операциями называют обычно операции, выполняемые с целью проверки наличия или отсутствия каких-либо признаков у чисел. Например, проверка равенства числа нулю, проверка знака числа, сравнение данного числа с другим, проверка равенства нулю некоторых разрядов числа и т. д. Подобные логические операции человек-вычислитель выполняет в уме, оценивая получающиеся промежуточные результаты для определения хода дальнейших вычислений.

Арифметические и логические операции над числами в электронных цифровых машинах выполняются чрезвычайно быстро — в миллионные доли секунды. Однако для достижения общей высокой скорости решения задач недостаточно, чтобы каждая отдельная операция выполнялась очень быстро. Необходимо еще, чтобы эти операции выполнялись автоматически в нужной последовательности и в большом количестве без вмешательства человека. Это обеспечивается в машинах применением программного управления, сущность которого сводится к следующему.

Каждую операцию машина выполняет по определенной команде. Команда — это условный код (число), заставляющий машину выполнять определенную операцию. Последовательность команд составляет программу работы машины. Программа решения задачи записывается на специальном бланке в условном числовом коде. Каждая команда после записи на бланк представляет собой по внешнему виду некоторое число, а вся программа — последовательность чисел.

Программа работы машины, представленная в виде последовательности чисел, вводится в запоминающее устройство машины таким же образом, как и обычные числа, и хранится в запоминающем устройстве машины.

В процессе решения задачи команды программы поочередно выбираются из запоминающего устройства устройством управления.

Каждая выбранная команда соответствующим образом расшифровывается устройством управления и выдается для выполнения в различные части машины. Каждая команда содержит две основные части — операционную и адресную. Операционная часть команды показывает, какую операцию должна выполнить машина по команде, адресная часть — откуда следует взять числа, над которыми должна быть выполнена указанная операция, и куда нужно записать результат операции.

Все основные арифметические и логические операции, выполняемые машиной, пронумерованы, и каждая операция имеет свой постоянный номер, называемый кодом операции. Этот код операции и указывается в операционной части команды при ее написании.

При написании команды указывают: в операционной части — код операции, а в адресной части — номера ячеек, содержащих исходные для операции числа, и номер ячейки, куда нужно записать результат операции. Например, может быть составлена команда следующего вида:

01	0026	0072	0136

Число 01 в левой клетке обозначает код операции сложения (обычно операция сложения имеет код 01). Числа 0026 и 0072 обозначают, что нужно взять первое слагаемое из ячейки с номером 0026, а второе слагаемое из ячейки с номером 0072. Число 0136, записанное в четвертой клетке команды, обозначает, что результат нужно записать в ячейку с номером 0136.

Это пример так называемой трехадресной команды, в которой указываются сразу три адреса: адреса двух исходных чисел и адрес результата операции. Существуют также машины с одно-, двух-, четырех- и пятиадресными командами. Соответствующим машинам присваивают названия одно-, двухадресных и т. д. машин.

В одноадресных машинах полная арифметическая операция выполняется, как правило, не по одной команде, как в трехадресных, а при помощи нескольких команд. Например, одна команда вызывает одно число, вторая команда вызывает второе число и выполняет операцию, третья команда отсылает результат по заданному адресу. Устройство управления машины обеспечивает последовательное выполнение команд программы и управляет работой машины в процессе выполнения каждой команды в течение такта работы машины.

Как упоминалось выше, после выполнения машиной очередной команды в счетчик команд автоматически прибавляется единица и производится выборка следующей команды из ячейки запоминающего устройства, имеющей номер на единицу больше, чем предыдущая команда.

Таким образом, выполняются команды арифметических и логических операций. В каждой из них, как правило, участвуют два исходных числа и получается одно число, являющееся результатом операции. Помимо этого, существуют так называемые команды управления и вспомогательные команды. Эти команды служат для пуска или остановки машины, переписи чисел из одного места запоминающего устройства в другое, для приема исходных данных и выдачи результатов операции, а также для изменения порядка выполнения команд программы.

Рассмотрим несколько подробнее назначение и порядок работы команд безусловного и условного переходов. Команда безусловного перехода, будучи поставлена в определенном месте программы, показывает, к какой команде программы перейти после выполнения данной команды.

Обычно команды программы выполняются подряд до тех пор, пока не встретится команда условного перехода или команда безусловного перехода. Команда условного перехода осуществляет переход к той или иной команде программы в зависимости от выполнения некоторого условия. Если рассматривать трехадресную команду условного перехода, то она может, например, иметь следующее строение и назначение.

В двух первых адресах указывают номера ячеек запоминающего устройства, где хранятся два некоторых числа, которые должны сравниваться между собой. По команде условного перехода производится выборка обоих чисел и сравнение их между собой. Если первое число больше второго, то происходит переход к команде, адрес которой указан в третьем адресе команды условного перехода. Если же второе число больше первого или равно ему, то после команды условного перехода выполняется следующая по порядку команда.

Команда безусловного перехода указывает адрес следующей команды, к которой должен быть совершен переход после выполнения данной команды независимо от каких-либо условий.

Команда условного перехода в трехадресной машине может иметь и такое строение. В первых двух адресах указываются адреса ячеек памяти, хранящих две команды программы, к которым может быть совершен переход после команды условного перехода, третий адрес не используется. Выбор той или иной команды из указанных двух команд осуществляется в зависимости от результата операции, выполнявшейся непосредственно перед командой условного перехода.

Большинство команд арифметических и логических операций в машинах построено таким образом, что помимо результата операции эти команды выдают и дополнительный признак, называемый сигналом ω (омега), характеризующий некоторые альтернативные свойства числа, являющегося результатом выполнения данной команды. Под альтернативными свойствами мы понимаем такие свойства, которые могут либо быть, либо не быть, т. е., условно говоря, признак ω может быть равен либо 0, либо 1. Например, операция сложения выполняется так, что если результат сложения отрицателен, то признак ω равен 1, а если положителен (или нуль), то ω равен 0.

Этот сигнал ω передается из арифметического устройства в устройство управления, которое использует его при выполнении команды условного перехода для определения адреса следующей команды, подлежащей выполнению. При этом команда условного перехода может работать, например, по правилу: если предыдущая команда дала признак ω, равный единице, то следующей выполняется команда, адрес которой указан во втором адресе команды условного перехода; если предыдущая команда выработала признак ω, равный нулю, то следующей выполняется команда программы, адрес которой указан в первом адресе команды условного перехода.

Таким образом команда условного перехода в обеих описанных модификациях позволяет произвести выбор того или иного направления в продолжении вычислений в зависимости от некоторого условия. Это весьма важное свойство программно-управляемых машин, обеспечивающее полную автоматичность их работы. Как известно, в процессе сложных вычислений часто приходится решать вопрос, как вести вычисления дальше, после того как получены те или иные промежуточные результаты. Выбор этих направлений можно возложить на машину благодаря наличию команды условного перехода, если соответствующим образом запрограммировать получение результатов с определенными признаками и поставить после этого команды условного перехода.

Непосредственное программирование. Непосредственное программирование рассмотрим на примере составления простой программы для трехадресной машины. Требуется составить программу для решения любой системы двух линейных алгебраических уравнений с двумя неизвестными:

$$Ax + By = C,$$
$$Dx + Ey = F$$

для случая, когда

$$AE — BD \neq 0.$$

Решая совместно эту систему, получим

$$x = \frac{CE - BF}{AE - BD}, \quad y = \frac{AF - CD}{AE - BD}.$$

Программу составляем для вычисления по этим формулам.

Порядок, в котором в данном случае должен осуществляться вычислительный процесс, очевиден.

Программу сначала составляют в «буквенно-числовых» обозначениях. Например, ячейки, отведенные для команд программы, обозначают знаками $a + 1$, $a + 2$, ячейки для исходных данных — знаками $b + 1$, $b + 2$, ... и т. д.

Будем использовать следующие коды операций: 03 — вычитание, 05 — умножение, 06 — деление, 07 — перевод числа из двоичной системы в десятичную, 10 — вывод на перфокарты, 11 —остановка.

Начальные данные задачи (числа *A*, *B, C, D, E, F)* разместим следующим образом:

Адрес ячейки	*b*+1	*b*+2	*b*+3	*b*+4	*b*+5	*b*+6
Содержимое ячейки	*A*	*B*	*C*	*D*	*E*	*F*

Программу поместим в ячейки с адресами *a*+1, *a+2,* ... Рабочими ячейками пусть будут ячейки *c*+1, *c*+2, . . . Для результатов отведем ячейки *d*+l*, d+2,* . ..

Составляя программу, будем слева от каждой команды записывать, пользуясь восьмеричной системой счисления, адрес ячейки памяти, предназначенной для ее хранения, и отделять этот адрес ячейки от команды закрытой скобкой.

В каждой команде двухразрядное число, стоящее слева, означает код соответствующей операции.

«Перемножить числа *A* и *E*. Произведение записать в рабочую ячейку *c*+1»:

a+1) 05, *b*+1, *b*+5, *c* +1

«Перемножить числа *B* и *D*. Произведение записать в ячейку *c* + 2»:

a+2) 05, *b*+2, *b*+ 4, *c* + 2

«Из числа *AE,* расположенного в ячейке c+1, вычесть число *BD,* стоящее в ячейке *c* + 2. Разность записать в ячейку *c*+1»:

a+3) 03, *c*+1 , *c*+2, *c* + 1

«Перемножить числа *C* и *E*. Произведение записать в ячейку *c* + 2»:

a + 4) 05, *b*+ 3, *b*+ 5, *c* + 2

«Перемножить числа *B* и *F*. Произведение записать в ячейку *c* + 3»:

a + 5) 05, *b*+2, *b*+ 6, *c* + 3

«Из числа *CE,* находящегося в ячейке *c*+2, вычесть число *BF,* стоящее в ячейке *c*+3. Разность записать в

ячейку *c*+2»:

a + 6) 03, *c*+2, *c*+3, *c*+2

«Разделить число *CE—BF,* помещенное в ячейку *c* + 2, на *AE—BD* из ячейки с+1. Частное записать в ячейку *d*+l»:

a + 7) 06, *c*+2, *c*+1, *d*+l

После выполнения этой команды будет получена искомая величина *x.*

«Перемножить числа *A* и *F.* Произведение записать в ячейку *c* + 2»:

A + 10)* 05, *b*+1, *b*+ 6, *c* + 2

«Перемножить числа С и *D.* Произведение записать в ячейку *c* + 3»:

A + 11) 05, *b*+ 3, *b*+ 4, *c* + 3

«Из числа AF, расположенного в ячейке *c* + 2, вычесть число *CD,* стоящее в ячейке *c* + 3. Разность записать в ячейку *c* + 2»:

A + 12) 03, *c* + 2, *c* + 3, *c* + 2

«Разделить число *AF—CD,* расположенное в ячейке *c* + 2, на число *AE—BD,* находящееся в ячейке *c* + 1. Разность записать в ячейку *d* + 2»:

A + 13) 06, *c* + 2, *c* + 1, *d* + 2

После выполнения этой команды будет получена вторая искомая величина *y.*

Выполняя составленные нами команды, машина решит заданную систему уравнения, но все вычисления она производит в двоичной системе счисления. Следовательно, и величины *x* и *y* будут получены в виде двоичных чисел. Необходимо преобразовать их в десятичные числа.

«Два числа *x и y,* расположенные в последовательных ячейках, начиная с ячейки *d*+1, перевести из двоичной системы счисления в десятичную. Результат снова записать в те же ячейки»:

A + 14)** 07, *d*+1,0002, *d*+1

Теперь надо предусмотреть вывод полученных результатов из машины на перфокарты.

«Два числа *x* и *y,* стоящие в последовательных ячейках *d+l* и *d+2,* перенести на перфокарты»:

A + 15) 10, *d*+l, 0002, 0000

Последняя команда «Останов»:

a +16) 11, 0000, 0000, 0000 Составление программы окончено.

Собрав вместе все команды, получим программу в буквенном виде (табл. 1). Заменив буквы конкретными числами, получим программу в ее окончательном виде.

Таблица 1

Номер ячейки	Команда, хранящаяся в ячейке			
a+1	05,	*b*+1,	*b*+5,	*c*+1
a+2	05,	*b*+2,	*b*+4,	*c*+2
a+3	03,	*c*+1,	*c*+2,	*c*+1
a+4	05,	*b*+3,	*b*+5,	*c*+2
a+5	05,	*b*+2,	*b*+6,	*c*+3
a+6	03,	*c*+2,	*c*+3,	*c*+2
a+7	06,	*c*+2,	*c*+1,	*d*+1
a+10	05,	*b*+1,	*b*+6,	*c*+2
a+11	05,	*b*+3,	*b*+4,	*c*+3
c+12	03,	*c*+2,	*c*+3	*c+2*
a+13	06,	*c*+2,	*c*+1	*d+2*
o+14	07,	*d*+1,	0002,	*d*+1

* После *a* + *7* следует *a* + 10, так как запись ведется в восьмеричной системе счисления. В этой системе счисления сочетание цифр 10 означает восемь

** В этой команде, так же как и в следующей, число, стоящее во втором адресе, указывает количество преобразуемых чисел, начиная с адреса *d*+l.

Номер ячейки	Команда, хранящаяся в ячейке			
a+15	10,	*d*+1	0002,	0000
a+16	11,	0000,	0000,	0000

Применение ЭЦМ целесообразно в том случае, когда машина выполняет много операций по программе, состоящей из небольшого количества команд. Этого можно добиться путем составления таких программ, по которым машина сначала осуществляла бы ряд операций над определенными данными, затем сама изменяла нужным образом адреса у части команд программы и снова выполняла уже видоизмененную программу над другими данными, и так до тех пор, пока решение задачи не будет доведено до конца, т. е. пока не будут обработаны все данные с помощью одной и той же программы.

Указанное изменение команд получило название переадресации. Переадресация команд осуществляется путем выполнения арифметических действий (сложения и вычитания) над числами, представляющими собой коды команд. Кроме изменения команд для повторения программы, нужно, во-первых, определить момент, когда следует и следует ли перейти к повторному выполнению команд, а во-вторых, нужно заставить машину «вернуться» обратно, т. е. нужно проверить выполнение некоторого логического условия, и, если это условие выполнено, перейти с конца некоторого участка программы к его началу.

Участки программы, выполняемые машиной многократно, называются циклами программы.

Участки программы, предназначенные для проверки какого-нибудь логического условия, в зависимости от выполнения или невыполнения которого машине надлежит перейти к одному из двух возможных вариантов дальнейших вычислений, называются разветвлениями программы.

Если нужно проверить истинность логического условия: «сумма двух чисел отрицательна», то достаточно произвести сложение этих чисел. Образование сигнала ω ($\omega=1$) означает выполнение этого условия. Отсутствие сигнала ω ($\omega = 0$) будет говорить о том, что условие не выполнено.

Вслед за командой, проверяющей выполнение логического условия, должна быть поставлена команда условного перехода. При наличии сигнала ω машина по команде условного перехода перейдет к выполнению команды, адрес которой стоит, например, во втором адресе команды условного перехода. При отсутствии сигнала будет произведен переход к команде, адрес которой стоит в первом адресе команды условного перехода.

В современных цифровых вычислительных машинах широкое применение получил способ автоматической модификации адресов команд с помощью так называемых индексных регистров. Этот способ по сравнению со способом модификации адресов с помощью основного арифметического устройства, применяемым, например, в машине «Стрела», обладает рядом важных преимуществ: сокращается число команд в программе и тактов работы машины.

Мы рассмотрим применение этого способа па примере машины «Минск-2», в которой первые 15 ячеек оперативной памяти играют роль индексных регистров, а в коде команды имеется специальное четырехразрядное индексное поле, содержащее адрес (номер) индексного регистра. Нулевой код в этом поле указывает на отсутствие модификации обоих адресов. Код, отличный от нуля, указывает адрес индексной ячейки, содержимое которой прибавляется соответствующими частями к первому и второму адресам команды. Подробнее этот процесс пояснялся при описании машины «Минск-2».

Рассмотрим пример простой программы для вычисления полинома *n*-й степени по схеме Горнера:

$$y = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_nx^n = (a_0 + x(a_1 + \cdots + x(a_{n-1} + xa_n)\cdots))$$

Все адреса и константы будем записывать, как это принято при программировании, в восьмеричной системе счисления, т. е. в системе, основанием которой является число восемь.

Для размещения аргумента *x* отведем ячейку 0077. Значение полинома *y* будем помещать в ячейку 0076. Коэффициенты полинома a_n, a_{n-1}, . . ., a_0 разместим в ячейках с адресами 0100, 0101, ..., 0100 +*n* Программу будем размещать в ячейках памяти, начиная с адреса 1000.

Приведем описание некоторых команд машины «Минск-2», которые нам понадобятся для составления указанной программы.

Заметим, что код операции в машине «Минск-2» пишется в виде двухразрядного восьмеричного числа со знаком. Адреса индексной ячейки и 1-й и 2-й адреса пишутся соответственно 2, 4 и 4-разрядными восьмеричными числами.

Некоторые команды машины «Минек-2»

Код операции		Индекс 2-го разряда	1-ый адрес 4-го разряда	2-ой адрес 4-го разряда	Пояснение
Знак	2-азрядный код				
–	10	*I*	*A*1	*A*2	В ячейку с адресом *A*2 и в сумматор АУ засылается число, находящееся в ячейке с адресом *A*1. Признак ω=1 вырабатывается при посылке 0.
+	35	*I*	*A*1	*A*2	Умножение числа, находящегося в ячейке с адресом *A*2, на число, находящееся в ячейке с адресом *A*1. Результат остается в АУ.
+	16	*I*	*A*1	*A*2	Сложение числа, находящегося в ячейке с адресом *A*1, с содержимым

Код операции		Индекс 2-го разряда	1-ый адрес 4-го разряда	2-ой адрес 4-го разряда	Пояснение
Знак	2-азрядный код				
					сумматора АУ, с посылкой результата в ячейку с адресом *A*2.
–	20	*I*	*A*1	*A*2	Проверка окончания цикла. К адресным частям А1 и А2 содержимого индексной ячейки с адресом I прибавляются значения соответствующих адресных частей содержимого ячейки с адресом А2, что обеспечивает увеличение шагов переадресации команд с помощью этой индексной ячейки. Из значения счетчика числа циклов, находящегося в 13 старших разрядах этой индексной ячейки, вычитается единица. Если результат будет ≥0, то происходит переход к команде с адресом А1; если результат <0, то происходит переход к следующей команде. Заметим, что начальное значение счетчика циклов должно быть на единицу меньше необходимого числа повторений циклов.

Ниже приводится программа, осуществляющая вычисление указанного полинома. Для модификации мы используем индексную ячейку 01, а в ячейке памяти 0075 хранится константа, представляющая начальное значение содержимого этой индексной ячейки. В качестве рабочей ячейки используем ячейку с адресом 0074.

Программа вычисления полинома

Адрес ячейки	Код команды				Пояснение
	Код операции	Индекс	*A*1	*A*2	
0001					Индексная ячейка
0074	....	....	....	....	Рабочая ячейка
0075	*n*—1		0001	000	
0076		Значение полинома *y*			
0077		Величина аргумента *x*			
0100		Коэффициент a_n			
0101		Коэффициент a_{n-1}			
. . . .	. . .	. . . . .	. .		. . . . .
0100+*n*		Коэффициент a_0			
. . . .	. . .	. . . . .	. . . .	. .	. . . . .
1000	—10	00	0075	0001	Посылка нач. знач. в индексн. ячейку 0001
1001	—10	00	0100	0074	Посылка коэф. a_n в раб. ячейку 0074
1002	+35	00	0074	0077	$a_n \times x$
1003	+16	01	0100	0074	$a_n \times x + a_{n-1}$
1004	—20	01	1002	0075	Проверка конца цикла
1005	—10	00	0074	0076	Посылка результата

В качестве другого примера, иллюстрирующего использование способа автоматической переадресации команд с помощью индексных ячеек, приведем программу для перемножения двух векторов — вычисления суммы парных произведений:

$$y = \sum_{i=1}^{n} a_i \times b_i .$$

Так же как в предыдущем примере, используем индексную ячейку 0001 и рабочую ячейку 0074; исходное значение содержимого индексной ячейки 0001 будем хранить в ячейке 0075; вычисленное значение величины *y* поместим в ячейку 0076. Величины a_1, a_2, .. ., $a_п$ поместим в ячейки 0101, 0102, . . ., 0100 + *n*, а величины b_1 b_2, ..., $b_п$ — в ячейки 0201, 0202,. . ., 200 + *n*. Считаем, что $n \leq 100$. Программу разместим в ячейках, начиная с 1000.

Основным приемом, облегчающим непосредственное программирование, является метод блок-схем. Перед написанием программы составляется ее блок-схема, представляющая собой графическое изображение последовательности выполняемых вычислений в виде набора прямоугольников — блоков, соединенных стрелками.

Программа перемножения двух векторов

Адрес ячейки	Код команды: Код операции	Индекс	A1	A2	Пояснение
0001					Индексная ячейка
. . .	. . .	. . .	. . .	. . .	. . . . .
0074					Рабочая ячейка
0075	$n-1$		0001	0001	Константа
0076	Значение y				
. . .	. . .	. . .	. . .	. . .	. . . . .
1000	—10	00	0075	0001	Посылка константы
1001	—10	00	0000	0074	Очистка рабочей ячейки
1002	+35	01	0100	0200	$a_1 \times b_1$
1003	+16	00	0074	0074	Сложение содержимого сумматора с содержимым рабочей ячейки 0074 с посылкой результата в рабочую ячейку 0074
1004	—20	01	1002	0075	Проверка окончания цикла
1005	—10	00	0074	0076	Посылка результата в ячейку 0076

Каждый блок соответствует определенному этапу вычислений; внутри блока записывается (словами или формулами) содержание этапа; стрелки, соединяющие блоки, показывают последовательность выполнения этапов.

В соответствии с намеченной блок-схемой составляются программы отдельных участков сначала в буквенных (условных) адресах. Затем производится объединение отдельных участков программы (подпрограмм) и присваивание действительных адресов.

Важным вопросом программирования является распределение памяти машины. Эта работа выполняется после составления общей блок-схемы программы. Наибольшие трудности вызывает распределение памяти в тех случаях, когда обрабатывается большой объем данных, которые приходится размещать на магнитных лентах и барабанах. При программировании в этом случае должны предусматриваться специальные команды для переписи исходных данных с лент и барабанов в оперативную память и вывода полученных результатов на ленты, барабаны или на печать.

Контроль вычислений. Естественно, что в процессе подготовки задачи к решению на машине, а также непосредственно в процессе решения необходимо осуществлять контроль правильности получаемых результатов.

Проверку правильности алгоритма и исходных данных задачи, а также проверку составленной программы производит сам специалист — математик, подготавливающий задачу.

Правильность переноса материала на перфокарты обеспечивается тем, что перенос этот осуществляется, как говорят, «в две руки» — двумя людьми независимо друг от друга, после чего два комплекта перфокарт сличаются между собой при помощи электрического устройства, называемого контрольником. Если эти перфокарты оказываются нетождественными, то они удаляются и вместо них пробиваются новые (опять «в две руки»). Полная тождественность двух комплектов перфокарт считается признаком того, что перенос информации с бланков на перфокарты выполнен без ошибок.

Правильность ввода программы в память машины контролируется автоматически, для чего в программу ввода включают специальные команды. При этом часто используется прием, называемый контрольным суммированием. Состоит этот прием в том, что после ввода программы и исходных данных материал, введенный в память, суммируют. Полученная сумма носит название контрольной суммы. Контрольную сумму по специальной команде машина сравнивает с заранее введенным в машину ее значением. Возможен случай двух последовательных вводов материала сначала с первого, а потом со второго комплекта перфокарт со сравнением контрольных сумм после первого и после второго ввода. Совпадение контрольных сумм считают признаком правильности ввода программы и исходных данных.

После того как программа правильно введена, можно пустить машину и решать задачу. Однако если программа применяется впервые, то перед тем, как приступить к решению задачи, производят отладку ее на машине.

Сущность отладки состоит в том, что с помощью программы производят ряд вычислений при упрощенных исходных данных. Результаты, выдаваемые при этом машиной, сличают с заранее заготовленными результатами, полученными путем ручного счета. При отладке программы на машине могут быть выявлены ошибки, допущенные в процессе составления программы. Только после устранения этих ошибок приступают к собственно решению задачи на машине.

Чтобы иметь уверенность в правильности решения задачи, необходимо также контролировать правильность вычислений, производимых машиной.

Наиболее простым способом контроля является двойной счет. В простейшем случае можно решить задачу дважды: полученные результаты напечатать и затем сравнить между собой. Тождественность результатов будет служить признаком того, что в процессе вычислений машина не допускала случайных сбоев.

Обычно контроль правильности вычислений предусматривают в программе. Тогда этот контроль осуществляется автоматически, самой машиной. Двойной счет может быть применен и в этом случае. Можно составить программу так, чтобы машина решила задачу один раз, просуммировала бы результаты, затем решила задачу второй раз, снова просуммировала бы результаты и, если обе контрольные суммы совпадут, выдала бы результаты на перфокарты. Такой контроль применяют довольно часто. Практика показывает его высокую надежность, но при этом время, расходуемое машиной па решение задачи, увеличивается вдвое.

Иногда для контроля правильности результатов предусматривают проверку выполнения какого-либо соотношения, связывающего вычисляемые величины, при условии, конечно, что это соотношение не используется для вычисления этих величин,

Операторное программирование. Значительным шагом вперед в развитии программирования явился операторный метод, предложенный в 1953 г. А. А. Ляпуновым.

При операторном методе программирования алгоритм, избранный для решения задачи, записывается на том или ином алгоритмическом языке, не зависящем от конкретной машины. Основу любого алгоритмического языка составляет набор операторов, представляющих собой автономные единицы действий, необходимых для преобразования информации в машине. В первоначальном виде операторный метод, предложенный А. А. Ляпуновым, выглядел следующим образом. Каждый оператор обозначается определенной буквой. По алгоритму задачи составляется логическая схема программы, в которой буквы, обозначающие операторы, располагаются в строку слева направо в порядке выполнения соответствующих операторов. В тех случаях, когда порядок записи операторов в схеме не соответствует порядку их выполнения, применяются специальные знаки переходов. Если оператор зависит от некоторого параметра, то букву, обозначающую этот оператор, снабжают сверху последовательностью индексов, обозначающих соответствующие параметры. Порядковые номера операторов в схеме обозначают нижними индексами.

При программировании задач чаще всего встречаются следующие типы операторов: арифметические (A^i_{m}), логические (P^i_m), операторы переадресации (F^i_m), операторы восстановления (O^i_m) и некоторые другие.

Обычно производится переадресация тех команд, адреса которых зависят от какого-либо параметра. Если переадресация связана с изменением значения, например, параметра j, то, чтобы это показать, пишут $F_m(j)$.

Переадресация, обусловленная изменением нескольких параметров, выполняется таким количеством операторов, каково число этих параметров.

Оператор восстановления O служит для восстановления команд и параметров, измененных в процессе работы. Если восстанавливающиеся команды зависят, например, от параметра i, то пишут $O_m(i)$ и т. д.

В логической схеме программы каждый оператор изображает некоторую совокупность команд будущей программы.

В логической схеме (и в программе) каждый нелогический оператор может передавать управление только в одном направлении, логический оператор может передавать управление в одном из нескольких направлений (удобно применять логические операторы, передающие управление в одном из двух направлений). Возможные переходы между операторами с пропуском других операторов, стоящих в логической схеме, принято обозначать стрелками. При этом переход от логического оператора в случае, когда проверяемое им условие выполнено, изображают стрелкой, проведенной над строкой операторов, а переход в случае, когда условие не выполнено, — стрелкой, расположенной под строкой операторов. Расположение стрелок, обозначающих переходы от нелогических операторов, безразлично (иногда вместо стрелок употребляют другие знаки перехода).

Составим, например, логическую (операторную) схему для следующей простой задачи. Пусть требуется вычислить значение определенного интеграла

$$\int_1^x t^n dt (x > 0). \tag{1}$$

Здесь исходными являются два числа x и n. При $n \neq -1$ значение интеграла вычисляется по формуле

$$\int_1^x t^n dt = \frac{x^{n+1} - 1}{n+1}. \tag{2}$$

При $n = -1$ применяется другая формула

$$\int_1^x t^n dt = \ln x. \tag{3}$$

Схема программы будет состоять из следующих операторов:

где P_1 — оператор проверки логического условия $n \neq -1$;
A_2 — арифметический оператор вычисления по формуле (2);
A_3 — арифметический оператор вычислений по формуле (3);
$Я_4$ — оператор окончания вычислений.

Операторный метод программирования имеет следующие достоинства.

1. Работа по программированию расчленяется на две стадии:

а) составление логической схемы программы, выполняемое работниками высокой квалификации (инженерами);

б) формальное расписывание команд программы, выполняемое техниками или автоматически самой ЭЦМ с помощью специальной программирующей программы.

2. Обеспечивается возможность независимого (а значит, и одновременного) программирования различных операторов несколькими лицами и возможность сравнительно простого объединения полученных участков

программы. Сюда же относится удобство возобновления работы по составлению программы после перерыва, что весьма затруднительно при непосредственном программировании.

В настоящее время создано большое количество различных алгоритмических языков, основанных на операторном подходе к записи алгоритмов. В дальнейшем мы подробно рассмотрим алгоритмический язык АЛГОЛ-60 и его расширение для программирования информационно-логических задач.

§ 6 СТРУКТУРА ЭЛЕКТРОННЫХ ЦИФРОВЫХ МАШИН

Структура электронной цифровой машины характеризуется составом запоминающих устройств, системой команд и схемой связи отдельных устройств. Основными характеристиками любой машины являются: быстродействие, разрядность и способы представления чисел, емкости ЗУ различных видов, скорости работы вводных и выводных устройств.

Система команд представляет собой машинный язык, с помощью которого задаются в машину алгоритмы решения задач. Система команд определяется форматом команды и набором элементарных операций, реализуемых в машине схемно.

Общая структурная схема машины определяет состав и систему взаимных связей основных устройств машины (АУ, УУ, ЗУ, ввод, вывод). Структурная схема характеризует направления передач информации и управляющих сигналов в машине и общую временную диаграмму ее работы. Рассмотрим общий подход к выбору структуры электронных цифровых машин.

Основными исходными данными для выбора структуры ЭЦМ являются, с одной стороны, состав задач и алгоритмов, для реализации которых предназначается проектируемая машина, и, с другой стороны, основные характеристики системы логических элементов, на базе которых строится данная машина. Последнее необходимо, так как для решения вопроса о способах обеспечения требуемого быстродействия машины, нужно знать, хотя бы ориентировочно, быстродействие отдельных элементов и устройств машины. На основе анализа указанных исходных данных определяются в первом приближении состав основных устройств, схема их соединения, технические параметры и система команд машины. После этого производится экспериментальное программирование типовых задач, анализ полученных программ и их моделирование на существующих ЭЦМ. По результатам экспериментального программирования и моделирования определяются времена решения задач, объемы программ, степень использования различных команд и их эффективность, объемы запоминающих устройств. На основе этих данных производится уточнение параметров, системы команд и структуры машины и по уточненным данным осуществляется вторичное 'программирование задач (при этом могут программироваться только наиболее характерные или сложные участки задач). Получающиеся на этом этапе результаты позволяют произвести окончательный выбор основных характеристик машины, отдельные параметры машин могут уточняться и на более поздних стадиях разработки. Требуемое быстродействие машины определяется путем анализа объемов вычислений, соответствующих типовым алгоритмам, и сопоставления их с требуемыми временами решения. Особенно жесткие требования предъявляются к быстродействию машин, работающих в реальном масштабе времени в системах автоматического управления быстропротекающими процессами, которые не допускают задержек или перерывов в вычислениях.

Емкость запоминающих устройств определяется на основании оценки связности алгоритмов, т. е. максимальных объемов данных, которые требуется одновременно хранить в процессе вычислений. Структура специализированных машин обычно в значительной степени зависит от характера задач и особенностей алгоритмов, в частности, от возможности их расчленения на независимые параллельные ветви, которые могут выполняться одновременно.

Отдельно исследуется вопрос о требуемой разрядности и способах представления данных. Разрядность АУ и ячеек памяти определяется на основе требований к точности решений с учетом дополнительных разрядов, необходимых для исключения ошибок округления. Способы представления данных в машине определяются в основном ее назначением.

В зависимости от характера решаемых задач в машинах используются следующие способы представления величин: с фиксированной запятой, с плавающей запятой, с переменной длиной слова, с разделением слова на слоги (как правило, по 6 двоичных разрядов), а также комбинации указанных способов. Фиксированная запятая применяется в большинстве малых и средних машин, предназначенных для научных и технических вычислений, а также в машинах, работающих в системах автоматического управления (в последнем случае решается, как правило, одна и та же задача и диапазон изменения величин заранее известен). Плавающая запятая, как правило, применяется в больших машинах, предназначенных для сложных вычислений с большой точностью; в этих же машинах применяются комбинированные способы (переменная длина слов, 6-разрядные слоги). Сочетание фиксированной запятой с 6-разрядными слогами применяется преимущественно в машинах, служащих для обработки учетной и плановой информации; в этих машинах наряду с расчетами выполняется операция сортировки буквенных и буквенно-числовых выражений (наименования объектов, позиций ведомостей, номенклатуры вооружения или оборудования и т. п.).

Определение основных параметров машины тесно связано с построением системы операций и команд машины. Машинной операцией называется элементарная неразрываемая совокупность действий машины, задаваемая постоянным кодом и обеспечивающая выполнение определенного шага в программе. Машинные операции образуют систему, включающую в себя, как правило, основные арифметические и логические операции, операции ввода и вывода информации и операции условных и безусловных переходов в программе. Для облегчения программирования часто в состав машинных операций вводятся сложные операции, выполняемые при помощи специальных схем или подпрограмм ($\sin x$, $\sqrt{x}$ и др.), а также модификации операций для выполнения действий с фиксированной или плавающей запятой, удвоенным числом разрядов, для действий с комплексными числами и т. д. В универсальных машинах небольшой и средней производительности общее

количество различных машинных операций изменяется в пределах от 32 до 64.

В мощных машинах («Стретч», «Атлас», «L-3060» и др.) количество различных операций порядка 100 и больше.

Формат команды определяет структуру и размер машинного слова команды (количество разрядов) и распределение функций между различными частями слова команды. Машины имеют, как правило, один основной формат и один или несколько дополнительных форматов.

В формате команды имеется в общем случае четыре основные части, называемые полями: операционная, индексная, признаковая и адресная. Операционное поле представляет собой группу двоичных разрядов (от 5 до 9 в разных машинах), служащую для представления кодов операций. Индексное поле состоит из одной или двух групп разрядов, представляющих собой короткие адреса (от 1 до 6 разрядов) и служащих для указания адресов индексных регистров, используемых для изменения адресов команд. Признаковая часть команды содержит ряд специальных признаков, модифицирующих действия команды.

К числу наиболее важных признаков относятся:

а) признак непрямой адресации, при котором адрес в команде является не адресом величины, а адресом ячейки, в которой хранится адрес величины;

б) признак непосредственного значения, при котором адресная часть команды используется не как адрес, а как число — величина;

в) контрольные признаки, в частности признак контроля «ода команды на четность, служащий для проверки правильности выборки и передачи команд.

Адресная часть включает в себя основной адрес (или несколько адресов) величины и в некоторых случаях дополнительные группы разрядов, определяющие положения величины внутри ячейки. Эти дополнительные группы разрядов в адресной части команды могут указывать, например, значение старшего разряда и количество разрядов, занимаемых величиной, а также величину и направление сдвига этих разрядов при передачах из ОЗУ в АУ или наоборот.

Проектирование структуры машины имеет целью определение такой организации машины, которая бы обеспечивала выполнение ею заданных функций при минимальном объеме оборудования, высокой надежности и удобстве эксплуатации.

В настоящее время основные тенденции в развитии структуры ЭЦМ состоят в максимальном повышении производительности оборудования за счет совмещения операций и исключения простоев отдельных устройств и в упрощении подготовки (программирования) задач за счет использования более сложных и *«выразительных» машинных языков,* приближающихся в отдельных случаях к обычному человеческому языку, а в других случаях к языку математических формул.

Одним из главных принципов проектирования структуры ЭЦМ является обеспечение взаимного согласования параметров отдельных устройств (их быстродействия, емкости памяти, разрядности и т. д.), с тем чтобы ни одно устройство не задерживало работу других устройств, само не простаивало из-за других устройств. Рассмотрим основные принципы построения структуры современных электронных цифровых машин.

Командно-программное управление. Процесс вычислений состоит в последовательном выполнении команд программы. Каждая команда определяет одну элементарную машинную операцию и реализуется специальными схемами как единая неразрываемая совокупность действий. Развитие командно-программного управления идет по пути усложнения структуры форматов команд и использования комбинированных приемов переадресации. Например, в машине «Стретч» (США) вместо простых индексных регистров, содержащих приращения адресов команд, используются сложные регистры, содержащие в себе составные индексы, называемые управляющими словами. Управляющее слово служит для заданий операций над массивами величин и включает в себя в общем случае следующие четыре части:

а) начальный адрес обрабатываемого массива данных;

б) счетчик, определяющий число повторений данной операции над элементами массива;

в) шаг изменения адресов элементов массива;

г) адрес связи, определяющий положение следующего управляющего слова в памяти машины.

При использовании способа управляющих слов достаточно просто осуществляются операции над группами массивов величин. Каждый массив задается своим управляющим словом, причем как сами массивы, так и соответствующие им управляющие слова могут размещаться в памяти произвольным образом (не обязательно подряд).

Обработка группы массивов данных начинается с того, что специальной командой выбирается управляющее слово первого массива и устанавливается на соответствующий регистр, который в дальнейшем управляет ходом групповой операции. После окончания обработки первого массива (что определяется по равенству нулю счетчика) производится выборка следующего управляющего слова по адресу связи, имеющемуся в первом управляющем слове, и обработка второго массива и т. д. Конец обработки группы массивов определяется по специальному признаку конца цепи управляющих слов, указываемому в последнем управляющем слове.

Микропрограммное управление. Эффективность решения различных задач на ЭЦМ в сильной степени зависит от того, насколько система команд машины приспособлена для реализации требуемых алгоритмов. В тех случаях, когда машина используется для массового решения однотипных задач, а сами типы задач могут время от времени изменяться, выгодно иметь возможность изменять систему команд, приспосабливая ее к классам решаемых задач.

Эта возможность обеспечивается при использовании принципа микропрограммного управления работой машины. В отличие от командно-программного управления, при котором структура и функции всех команд являются фиксированными, при микропрограммном управлении фиксируются не полные операции (сложение, умножение и т. д.), а выбор элементарных действий (актов), из которых строятся эти операции (сдвиг на один разряд, установка регистра в нуль, передача кода с регистра на регистр и т. д.). При этом отдельные команды машины представляются в виде последовательностей таких элементарных актов, называемых микропрограммами, которые хранятся в специальном ЗУ. Каждой машинной команде соответствует своя микропрограмма. Изменяя микропрограммы, можно сравнительно просто изменять в определенных пределах

систему команд машины. Достоинством микропрограммного управления является помимо упомянутой выше гибкости системы команд также существенное упрощение схем устройств управления, что особенно проявляется в случаях реализации сложных операций ($\sin x$, $\sqrt{x}$ и т.д.). Дальнейшим развитием микропрограммного управления является разработка машин с хранимой логикой, в которых указанный принцип применяется не только для образования команд, но и для выполнения ряда других действий.

Слоговое управление. При слоговом управлении вместо команд, хранимых в памяти и выбираемых для исполнения как единое целое, для построения программы используются отдельные, не связанные между собой схемно, части команд, так называемые слоги слова команды: коды операций, адреса, специальные управляющие коды и др. Этот способ использован в машине «В-5000» (США) и представляет собой принципиально новый путь организации структуры машины. Он сочетается с рядом других принципов: бесскобочной записью арифметических и логических выражений, способом динамического хранения операндов и способом непрямой адресации величин в ОЗУ машины, осуществляемой при помощи специальной отсылочной адресной таблицы. Слоговое управление обеспечивает высокую компактность программы, повышение быстродействия за счет сокращения пересылок данных между АУ и ОЗУ, а также широкие возможности автоматизации программирования с использованием алгоритмических языков типа АЛГОЛ.

Рассмотрим перечисленные принципы построения слогового управления. Суть принципа бесскобочной записи состоит в том, что знаки операций записываются не между операндами, а сбоку (обычно справа) и распространяют свое действие на ближайшие слева операнды (один или два в зависимости от вида операции — одно- или двухкомпонентная). После выполнения операции результат операции записывается вместо операндов и может быть использован в качестве операнда для следующей операции и т. д. После выполнения каждой операции знак ее стирается.

Например, выражение $(x + y) \times z$ в бесскобочной записи будет иметь вид

$$zxy + \times .$$

Принцип динамического хранения операндов состоит в циклическом перемещении операндов и результатов операций между регистрами АУ и специальным участком памяти ОЗУ, называемым магазином. В АУ для приема операндов имеются два регистра «А» и «В». Кроме того, предусматривается специальный счетчик адресов магазина ОЗУ «С», который хранит адрес последней посланной в ОЗУ из АУ величины. Вызываемая для операции из ОЗУ величина поступает сначала в регистр А; если этот регистр занят какой-либо величиной, то последняя предварительно пересылается в регистр В, а если и в этом регистре имеется величина, то эта величина пересылается в магазин ОЗУ по адресу, определяемому при помощи счетчика С (увеличением на единицу). Код очередной операции программы относится к текущим содержимым регистров А и В; результат операции всегда записывается в регистр В; при этом регистр А освобождается для следующего операнда. Если следующим слогом, взятым из программы, является код какой-либо операции, то производится обратная передача: содержимое В передается в А, а из ОЗУ по адресу, указываемому счетчиком С, извлекается последняя переданная туда величина; операция выполняется над числами, установленными в регистрах А и В.

Этот способ динамической очереди (магазина) хорошо согласуется со слоговым принципом построения программы и бесскобочной записью последовательности действий.

Суть способа непрямой адресации с использованием специальной отсылочной адресной таблицы состоит в том, что непосредственно в программе (в адресных слогах) указываются не фактические адреса операндов, а адреса ячеек специальной таблицы, в которых хранятся адреса операндов. Так как эта таблица имеет значительно меньший объем, чем весь объем ОЗУ, то ясно, что количество разрядов в слогах программы, являющихся адресами этой таблицы, может быть значительно меньше, чем это требуется для обращений к полному объему ОЗУ; за счет сокращения размеров слогов существенно уменьшается общий объем программы. Адреса ячеек отсылочной таблицы, указываемые в слогах, используются либо для вызовов из отсылочной таблицы начальных адресов массивов данных, либо для вызовов подпрограмм. Таким же способом могут вызываться и отдельные величины; в ячейках отсылочной таблицы могут храниться также и сами величины, либо их адреса. При работе с элементами массивов адреса, указываемые в адресных слогах программы, являются относительными адресами, так же как и адреса, указываемые в слогах команд условных и безусловных переходов, являются относительными адресами переходов внутри подпрограмм. Эти обстоятельства обеспечивают значительное сокращение объема отсылочной таблицы по сравнению с общим объемом ОЗУ.

Многопрограммное управление и система прерывания программ. Этот принцип является характерным для структуры машин, работающих в реальном масштабе времени в системах автоматического управления различными объектами. В устройстве центрального управления таких машин предусматриваются специальные схемы (схемы авторазрыва), обеспечивающие возможность прерывания текущей программы в любой момент времени по сигналам, поступающим либо от внешних устройств, либо от устройств самой машины. При авторазрыве автоматически запоминается текущее состояние машины (адрес очередной команды, содержимое АУ, регистров и т. п.) и происходит переход к выполнению одной из специальных подпрограмм, определяемых в зависимости от кода, сопровождающего сигнал, вызвавший авторазрыв. Такими подпрограммами могут быть подпрограммы для ввода и вывода данных, поиска неисправностей в машине и т. п. В зависимости от информации, поступающей при авторазрывах в процессе решения, машина может изменять ход выполнения основной программы или переходить от одной программы к другой.

Регулирование порядка выполнения различных подпрограмм осуществляется специальной программой — диспетчером. В достаточно мощных машинах («Стретч», «Атлас», «Гамма-60» и др.) предусматривается возможность одновременного выполнения нескольких программ для различных задач, чем обеспечивается наиболее полное использование оборудования машин. В то время как для одной задачи идет, например, перепись данных с магнитной ленты в оперативную память, для другой задачи могут выполняться вычисления, а третья задача будет заканчиваться выдачей на печать и т. д. Программа-диспетчер вызывает различные частные программы, осуществляет перепись необходимой для них информации, регулирует длительность их выполнения, распределяет объемы запоминающих устройств между различными программами, контролирует правильность работы машины.

В структуре многопрограммных машин предусматривается, как правило, большое количество индексных

регистров, специальные счетчики времени (абсолютный и относительный), схемы прерывания вычислений, специальные сверхбыстродействующие ЗУ для хранения программы-диспетчера, а также возможность параллельной и независимой работы основных устройств машины.

Совмещение операций. Основным структурным (логическим) способом повышения быстродействия машин является совмещение операций. Совмещение операций может производиться в машинах на различных уровнях. Высшим уровнем является одновременная работа различных устройств машины при решении нескольких различных задач по разным программам. Следующим уровнем совмещения является одновременное выполнение нескольких команд одной и той же программы. Типичным примером является трехкратное совмещение одноадресных команд: выборка следующей команды, выполнение операции для очередной команды и посылка результата для предшествующей команды. В трехадресных машинах часто совмещают выполнение операции и выборку следующей команды. В машине «Стретч» осуществлено семикратное совмещение команд. В специальной так называемой «виртуальной» памяти хранятся семь очередных команд, для которых одновременно выполняются необходимые действия (обращения к ЗУ, арифметические операции, модификации адресов или индексов и др.). Третьим уровнем совмещения является совмещение элементарных действий, выполняемых различными схемами машины при реализации каждой команды. Например, каждое отдельное обращение к ОЗУ расчленяется на ряд действий (посылка адреса, дешифрация адреса, выборка кода числа). Эти действия могут быть совмещены во времени для нескольких последовательных обращений к ОЗУ. Аналогичным образом выполнение операции в АУ состоит из ряда действий, допускающих совмещение. Для совмещения действий требуется во многих случаях в структуре соответствующих устройств предусматривать дополнительные регистры и схемы управления.

Одноуровневая организация системы оперативных и промежуточных запоминающих устройств машины с использованием непрямой адресации. С целью расширения объемов ЗУ, допускающих произвольное обращение к величинам, применяется единая система непрямой адресации для оперативной памяти (на сердечниках) и для промежуточной памяти (на магнитных барабанах или дисках). При этом способе вводится единая система индивидуальных адресов ячеек для всего объема указанных запоминающих устройств машины. Этот объем разделяется на блоки определенного постоянного размера (например, 256, 512 или 1024 ячейки). Старшие разряды единого кода адреса определяют номер блока, а младшие — адрес ячейки внутри блока. В соответствии с количеством блоков, которое может быть помещено в ОЗУ, вводится ряд адресных регистров, служащих для непрямой адресации ОЗУ. Каждый из этих регистров постоянно закреплен за определенной зоной ОЗУ, а его содержимое указывает номер того блока, который хранится в настоящий момент в данной зоне ОЗУ. Таким образом, содержимым указанных адресных регистров являются значения старших разрядов адреса. Для непрямой адресации остального объема памяти (кроме ОЗУ) используется специальная адресная оперативная память, объем которой равен общему количеству блоков. Каждая ячейка адресной оперативной памяти (ее адрес) соответствует постоянно определенному значению старших разрядов адреса, а содержимым ячейки является фактический адрес зоны магнитного барабана или диска, в которой находится в данный момент данный блок информации. При выборке или записи какой-либо величины в команде программы указывается просто ее полный адрес. По этому адресу специальными схемами сначала проверяется наличие требуемой величины (точнее ячейки с данным адресом) в ОЗУ. Это устанавливается путем сравнения старших разрядов заданного адреса со значениями кодов в адресных регистрах. При наличии требуемого блока в ОЗУ производится непосредственное обращение к соответствующей ячейке ОЗУ. При этом старшие разряды кода адреса задаются номером того адресного регистра, содержимое которого совпало со старшими разрядами заданного кода адреса. Если окажется, что блок, к которому относится искомая ячейка, не находится в данный момент в ОЗУ, то производится обращение к адресной оперативной памяти и по значению старших разрядов заданного полного адреса определяется ее ячейка. Содержимое этой ячейки указывает действительный адрес зоны МБ или МД, в которой находится требуемый блок. Этот блок переписывается на свободное место в ОЗУ (для этого в ОЗУ всегда одна зона остается свободной), в соответствующий адресный регистр записываются старшие разряды адреса данного блока. После 'переписи каждого нового блока в ОЗУ, если оно заполнено полностью, производится вывод из ОЗУ на МБ или МД одного из хранящихся там блоков. Таким образом, освобождается место для последующей записи. Выбор выводимого блока может быть сделан либо на основе непосредственного определения данных, которые не потребуются для ближайших вычислений, либо статистическим путем по принципу «самообучения» (на основе учета частостей обращений к отдельным блокам). Описанная одноуровневая организация адресации применена в английской машине «Атлас» для оперативной памяти на сердечниках емкостью в 16 384 слова и промежуточной памяти на МБ с максимальной емкостью в 1048 576 слов. Указанная система адресации обеспечивает большую гибкость в использовании ОЗУ и МБ, что особенно важно при многопрограммных режимах работы. Для оперативного регулирования заполнения ОЗУ данными в соответствии с решаемыми задачами заранее составляется таблица-справочник, показывающая для каждой программы необходимые блоки информации. По этой таблице программа-диспетчер перед включением в работу какой-либо программы переписывает в ОЗУ необходимые ей блоки информации.

Рассмотренные принципы построения структуры машин характеризуют некоторые пути и возможности современной вычислительной техники. Выбор того или иного принципа в каждом конкретном случае зависит в основном от назначения машины. Командно-программное управление применяется как основной вариант для построения большинства современных вычислительных Машин универсального назначения. Микропрограммное управление используется в основном в тех машинах, в которых требуется изменять систему команд в зависимости от классов решаемых задач. Слоговое управление выгодно применять в машинах, предназначенных для выполнения особо сложных и больших программ, насчитывающих десятки и сотни тысяч команд; оно упрощает автоматизацию программирования и резко сокращает объем программ.

Одноуровневая организация памяти, многопрограммность, прерывание вычислений и совмещение операций являются такими принципами построения машин, которые в той или иной степени используются при построении ЭЦМ различного назначения, в том числе ЭЦМ, предназначенных для решения информационно-логических задач.

Для решения информационно-логических задач, связанных с накоплением и логической обработкой больших объемов информации, поиском данных по признакам и т. п., эффективной оказывается также организация работы

электронных цифровых машин, основанная на схемной или программной реализации ассоциативных связей между данными, обладающими общими свойствами и признаками. Подробно эти вопросы будут рассмотрены в III части.

II. АЛГОРИТМИЧЕСКИЙ ЯЗЫК ДЛЯ ПРОГРАММИРОВАНИЯ ЭКОНОМИЧЕСКИХ И МАТЕМАТИЧЕСКИХ ЗАДАЧ

3. АЛГОРИТМИЧЕСКИЙ ЯЗЫК АЛГОЛ-60

§ 7 ОБЩИЕ СВЕДЕНИЯ ОБ АЛГОЛе

Для решения любой задачи на электронной цифровой машине должна быть составлена программа, представляющая собой в форме, воспринимаемой машиной, алгоритм решения задачи. Алгоритм — это система правил, четко и однозначно определяющих порядок всех действий, приводящих к решению задачи.

В первые годы развития электронных вычислительных цифровых машин применялось ручное непосредственное программирование задач, т. е. перевод алгоритмов на машинный язык осуществлялся непосредственно людьми-программистами. Программисты должны были хорошо знать систему команд машины и алгоритм решения задачи и полностью выписывать подряд все команды программы. Такая работа в случае сложных и громоздких программ была чрезвычайно трудоемкой, сопровождалась значительным количеством ошибок и в связи с этим составляемые вручную программы требовали, как правило, длительной доработки и проверки (отладки) на машине.

Первым этапом автоматизации процесса программирования явился переход к так называемому символическому программированию, когда адреса команд и коды операций пишутся программистом не в виде конкретных числовых значений, а в символической или буквенной форме. При этом адреса обозначаются, как правило, теми же буквами, что и величины, которые должны храниться в соответствующих ячейках, а коды операций — обычными математическими символами, например, +, —, × , : и т. п.

Переход от символических адресов и кодов операций к конкретным машинным кодам осуществляется самой машиной, которая снабжается для этого специальной переводящей программой. Хотя при символическом программировании полностью сохраняется соответствие между символическими и машинными командами, процесс программирования существенно облегчается и сокращается время на отладку программ. Следующим этапом автоматизации программирования явилось введение *метода автокодов,* при котором для некоторой конкретной машины составлялся набор так называемых макрокоманд, каждая из которых соответствует определенной совокупности машинных команд (машинным подпрограммам). В макрокомандах использовались символические обозначения для кодов этих команд и для адресов операндов. Например, для некоторых одноадресных машин автокодами служили системы команд трехадресных машин. Переход к укрупненным командам (макрокомандам) в сочетании с символическим заданием адресов обеспечил дальнейшее сокращение объема программ, составляемых вручную. Как символическое программирование, так и метод автокодов привязан к конкретным машинам и в этом отношении не обеспечивает необходимой гибкости и универсальности составляемых программ.

Дальнейшим этапом явилось применение *алгоритмических языков,* ориентированных не на определенные типы машин, а на определенные классы задач. Так широкое применение в международном масштабе для программирования научно-технических вычислений имеет язык АЛГОЛ-60; в США и Западной Европе широко распространен язык ФОРТРАН (II и IV), применяемый для научно-технических задач; для программирования экономических задач в США и Западной Европе применяется КОБОЛ; для описания алгоритмов машинного перевода — язык КОМИТ; для задач поиска информации — язык РЕКОЛ.

Особую группу составляют языки для неарифметической обработки информации: ИПЛ, ЛИСП. К числу алгорифмических языков следует отнести также адресный язык, в котором учитываются машинные особенности решения задач (принцип адресности).

Мы будем рассматривать вопросы программирования информационно-логических процессов и, в основном, построение алгоритмического языка для описания таких процессов. Следует заметить, что, как правило, информационно-логические процессы включают в себя и обычные вычислительные процессы (расчеты по формулам), поэтому язык для описания информационно-логических процессов должен позволять описывать обычные вычислительные задачи. В настоящее время определилась тенденция к созданию общего алгоритмического языка, который представлял бы собой единое семейство языков, приспособленных к различным классам задач. Это семейство языков должно иметь общую основную часть (ядро), служащую для описания вычислительных процессов, встречающихся во всех классах задач, и общий синтаксис, обеспечивающий единообразие построения семейства языков.

Создание единого алгоритмического языка (точнее, семейства языков) позволит сравнительно легко обучать практических работников различных областей пользованию этими языками в необходимом для них объеме, существенно облегчит переходы от одного языка к другому, позволит стандартизировать и упростить построение трансляторов на машинные языки и между алгоритмическими языками, позволит стандартизировать символику, применяемую в устройствах ввода и вывода данных в машинах. Особенно удобным будет применение такого семейства языков при программировании сложных процессов переработки информации, включающих в себя задачи различных классов.

В качестве основы для построения алгоритмического языка для программирования информационно-логических задач мы берем международный алгоритмический язык АЛГОЛ. Этот язык дополняется средствами,

необходимыми для описания процессов обработки экономической информации и для обработки списковой информации.

Для удобства изучения и использования описываемого здесь алгоритмического языка принят метод последовательного изложения отдельных его составных частей. Сначала описывается АЛГОЛ, затем излагаются дополнения к АЛГОЛу, необходимые для описания алгоритмов обработки экономической информации; эти дополнения вместе с АЛГОЛом составляют язык АЛГЭМ. После этого описываются дополнения, необходимые для обработки информации, организованной в виде списков и списковых структур. Этот раздел мы называем ассоциативным программированием.

Основным отличием описываемого здесь варианта АЛГОЛа от эталонного языка является замена основных английских словесных символов русскими.

Название АЛГОЛ (ALGOL) происходит от сокращения двух английских слов ALGorithmic Language, обозначающих алгоритмический язык. Число 60 указывает год принятия этого варианта языков.

Основным назначением АЛГОЛа является описание алгоритмов выполнения математических и научно-технических вычислений с помощью автоматических программно-управляемых машин. Он используется в качестве международного алгоритмического языка для обмена алгоритмами в международном масштабе. АЛГОЛ служит также эталонным языком для построения конкретных алгоритмических языков (входных языков).

Конкретные входные языки типа АЛГОЛ сохраняют все основные свойства эталонного АЛГОЛа и отличаются от эталонного составом используемых символов, что связано с различиями в конструкции клавиатур клавишных устройств, служащих для ввода данных в разные машины. В основу АЛГОЛа положен общепринятый язык математических формул, дополненный средствами, необходимыми для формального описания процессов автоматического выполнения алгоритмов. Программа, записанная на АЛГОЛе, перед выполнением должна быть переведена с помощью специальной (достаточно сложной) программы, называемой транслятором, в рабочую машинную программу для данной конкретной машины. Отличие АЛГОЛа от обычного языка математических формул заключается в линейности записи (в одну строку) всех символов (в том числе индексов, показателей степеней), а также в более полной формализации записи действия (явное указание операций умножения, четкое определение типов чисел при вычислениях и т. п.), в строгом разграничении способов использования всех символов (скобок, запятых, точек с запятой и т. д.).

Кроме эталонного языка АЛГОЛ используется еще как язык публикаций. В этом варианте языка допустимы отклонения от эталонного языка, связанные с удобством печатания и чтения людьми алгоритмов, записанных на АЛГОЛе (в частности, индексы и показатели степеней пишутся так, как обычно принято их писать в формулах).

В основу построения АЛГОЛа положен блочный принцип. Любая программа на АЛГОЛе представляет собой блок и сама может состоять из блоков, внутри которых могут быть другие блоки и т. д.

Блок в АЛГОЛе — это автономный участок программы, включающий в себя информацию двух видов: описания данных и других объектов, участвующих в вычислениях, и указания о непосредственно выполняемых в программе действиях. Эти указания принято называть операторами. Оператор охватывает законченную последовательность действий, например вычисление по формуле, переход к другому варианту вычислений и др.

Описания и операторы можно рассматривать как отдельные предложения языка. После каждого такого предложения ставится точка с запятой.

Блоки АЛГОЛ-программы строятся по общему плану. В начале блока идут все описания, затем пишутся подряд все операторы в порядке их выполнения. Описания имеют силу только внутри данного блока. Блок начинается словесной скобкой **начало** и заканчивается словесной скобкой **конец**. Эти же скобки используются и для образования так называемых составных операторов, представляющих собой группы операторов, разделяемых точками с запятой. Отличие составного оператора от блока заключается в отсутствии в составном операторе описаний. Заметим, что внутри любого блока или составного оператора в качестве операторов могут фигурировать снова составные операторы и блоки, так как они являются разновидностями операторов. Блочный принцип построения программ удобен при программировании сложных задач, так как при этом всю задачу можно разделить на ряд автономных участков (блоков), которые будут разрабатываться одновременно несколькими программистами. Каждый блок получает некоторые входные данные (от других блоков или в процессе ввода информации) и выдает выходные данные другим блокам или на выводные устройства машин (печать, перфорацию). Помимо входных и выходных данных в блоке используются и свои местные (локальные) величины, которые не используются вне данного блока. Эти величины должны быть описаны в данном блоке с помощью описаний.

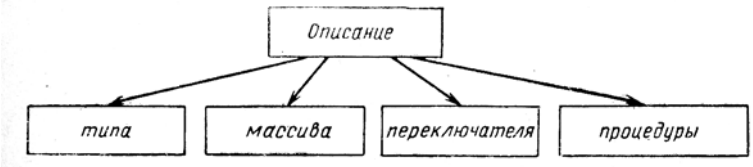


РИС. 11. Виды описаний в АЛГОЛе.

За символом **конец** ставится, как правило, точка с запятой либо другой символ **конец,** либо символ **иначе** (это будет пояснено ниже). Кроме того, за символом **конец** можно писать добавочный пояснительный текст, который не влияет на работу программ. Конец пояснительного текста указывается либо точкой с запятой, либо новым символом **конец,** либо символом **иначе,** а сам этот текст не должен содержать в себе указанных трех символов.

Блок обладает свойством локализации переходов внутри блока. Обращение к блоку возможно только в его начало; все операторы, используемые внутри блока, доступны только для операторов перехода, находящихся внутри блока. Переходы извне могут совершаться только в начало блока.

Перечислим все виды описаний и операторов, используемых в АЛГОЛе; приводимые названия будут

объяснены ниже. Описания делятся на четыре вида в соответствии со схемой, показанной на рис. 11.

Классификация операторов в АЛГОЛе представляется схемой, показанной на рис. 12. Всего имеется восемь различных видов операторов, причем некоторые из этих видов объединяются в более крупные группировки (основной оператор, безусловный оператор).

Для осуществления переходов между операторами некоторые из них снабжаются метками, которые ставятся перед операторами и отделяются от них двоеточиями. Перед одним оператором может стоять несколько меток, разделенных двоеточиями.

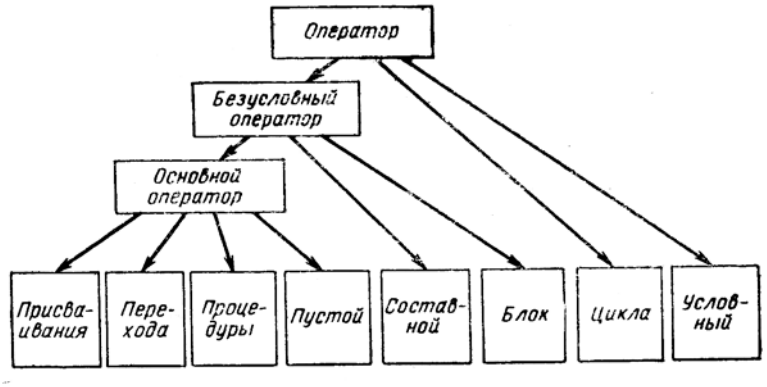


РИС. 12. Виды операторов в АЛГОЛе.

В АЛГОЛ-программе может стоять на некотором месте просто метка с пустым оператором. Понятие пустого оператора введено специально для возможности помещения одной метки, что бывает необходимо для отсылок к определенному месту программы, например, для выхода в конец цикла.

Рекурсивные определения понятий языка

В алгоритмических языках, как и вообще в языках, в основу построения их синтаксиса положен принцип иерархического определения более сложных понятий через более простые. Однако некоторые из сложных понятий языка не могут быть определены путем перечисления их компонентов. Например, точное формальное определение числа невозможно дать перечислением всех возможных значений чисел. В подобных случаях используются так называемые рекурсивные определения, в которых определяемое понятие само участвует в своем определении.

В качестве примера можно привести определение понятия блока. Можно считать, что блок состоит из двух половин, разделенных точкой с запятой. Первая половина называется «начало блока»», а вторая — «конец составного» (имеется в виду конец составного оператора).

Начало блока может содержать за символом **начало** одно или несколько описаний, разделенных точками с запятой, а конец составного может содержать перед символом **конец** один или несколько операторов, разделенных точками с запятой, в том числе могут быть блоки, являющиеся одним из видов операторов.

Таким образом, в определение понятия входит само понятие блока. Для представления подобных определений и вообще для описания различных конструкций языка удобна символика, известная под названием метаязыка Бэкуса.

В метаязыке используются металингвистические формулы, имеющие подобно обычным математическим формулам правую и левую части. Эти части соединяются специальным символом :: =, обозначающим «равно по определению». Левая часть является определяемой, а правая часть — определяющей. В металингвистических формулах участвуют основные символы языка, играющие роль констант и представляющие самих себя, и так называемые металингвистические переменные, представляющие более сложные понятия языка (операторы, описания, блоки и т. д.). Металингвистические переменные при постановке в металингвистические формулы заключаются в специальные угловые скобки (< — открывающая и > — закрывающая). В левых частях металингвистических формул ставятся только определяемые металингвистические 'переменные, заключенные в угловые скобки.

В правых частях металингвистических формул пишутся определяющие металингвистические выражения, которые строятся с использованием двух операций метаязыка:

— операции перечисления;

— операции построения определяющего выражения по составлению.

Операция определения путем перечисления имеет вид

$$< x >::= A \mid B \mid C \mid D \mid$$

где x — определяемое понятие, являющееся металингвистической переменной;

угловые скобки < > обозначают некоторое конкретное значение величины, заключенной в скобки;

A, B, C, D — некоторые элементы, являющиеся конкретными значениями определяемого понятия;

вертикальная черта — символ операции перечисления (операция ИЛИ).

В данном случае конкретными значениями некоторого понятия x будут: либо величина *A,* либо *B,* либо С, либо *D.*

Операция определения путем составления имеет вид

$$< x >::= ABC$$

где *ABC* есть конкретное выражение, образованное путем составления (последовательного написания рядом) трех выражений *A, B я C я* являющееся частным значением определяемого понятия «*x*». Возможно комбинированное использование указанных двух операций определения.

Например,

$$< x >::= AB \mid BC \mid CD .$$

При рекурсивном способе определения понятий определяемое понятие, указанное в левой части, снова выступает в правой части в качестве одного из элементов определяющего выражения.

Например, целое число без знака может определяться так:

<целое число без знака > : : = <цифра> | <целое число без знака> <цифра>

Основные символы АЛГОЛа

Основные символы — это наименьшие неделимые элементы языка, используемые для построения всех остальных конструкций языка. Пользуясь метаязыком, дадим следующее определение понятия символа АЛГОЛа:

<основной символ> : : = <буква> | <цифра> | <логическое значение> | <ограничитель>

<буква> :: = <латинская малая буква> | <латинская большая буква> |Ж|Ф|Щ|Э|Ы|Ю|Я|Ш|Н|Ч|И

В эталонном АЛГОЛе применяются только латинские малые и большие буквы. В конкретных вариантах АЛГОЛа. допускается изменение алфавита. Для удобства словесных обозначений мы вводим в состав алфавита АЛГОЛа еще буквы русского алфавита, несовпадающие с латинскими буквами.

<цифра> : : = 0|1|2|3|4|5|6|7|8|9

Состав цифр АЛГОЛа строго фиксирован; другие виды цифр не применяются. Цифры используются для представления чисел; буквы и цифры используются для представления различного рода наименований.

Логические значения будем обозначать с помощью словесных основных символов: **истинно** и **ложно**. Иногда для обозначения истинного значения применяют единицу (1), а ложного значения — ноль (0).

В эталонном АЛГОЛе для этого используются английские слова (true — истинно, false — ложно).

<логическое значение> : : = **истинно** | **ложно**

Логические значения являются логическими константами; они применяются для проверки логических условий, а также для образования логических выражений, т. е. выражений, которые могут иметь только два значения (истинно или ложно).

Ограничители — это служебные символы, входящие в состав языка и предназначенные для построения более сложных конструкций языка:

<ограничитель> : : = <знак операции> | <разделитель> | < скобка > | <описатель> | <спецификатор>

<знак операции> : : = <знак арифметической операции> | <знак операции отношения> | <знак логической операции> |<знак операции следования>

<знак арифметической операции> : : = + | — | × | / | ÷

Символ ÷ обозначает операцию целочисленного деления; результатом ее является наибольшее целое число, не превышающее фактического значения результата.

Заметим, что в АЛГОЛе операция умножения должна указываться явно. Например, $a \times b$ нельзя записать как *ab*; последняя запись будет воспринята как одно обозначение некоторой величины. Знаки операций отношения и логических операций имеют общепринятый смысл:

<знак операции отношения> : : = < | ≤ | = | ≥ | > | ≠

<знак логической операции> : : = ∨ | ∧ | ¬ |⊃|≡

Перечислим названия логических операций; подробно они рассматривались в § 3 гл. 1.

∨ — „или" (логическое сложение, дизъюнкция);

∧ — „и" (логическое умножение, конъюнкция);

¬ — „не" (отрицание), заметим, что в АЛГОЛе операция отрицания пишется не в виде черты над величиной, а в виде уголка, помещаемого перед величиной;

⊃ — „влечет" (импликация)

≡ — „эквивалентно" (равнозначность).

Логические операции служат для образования логических выражений.

<знак операций следования> : : = **перейти к | если | то | иначе | для | цикл**

К символам операций следования относятся такие символы, которые изменяют порядок выполнения операторов.

Символ **перейти к** используется для построения безусловного перехода; символы **если, то, иначе** используются при построении условных переходов и символы **для** и **цикл** — при построении циклов. Следует отметить, что словесные основные символы не имеют никакого отношения к буквам, из которых они составлены; они являются такими же основными символами, как и буквы. Вместо них можно было бы ввести какие-нибудь стрелки или другие значки, но выбраны слова, чтобы их было легче запоминать. Словесные основные символы при написании принято подчеркивать; при печатании они выделяются жирным шрифтом.

<разделитель> : : = , | . | $_{10}$ | : | ; |: = | |_| | **шаг | до | пока | примечание**

Перечисленные разделители имеют следующее назначение*:

запятая служит для разделения элементов списков, например список индексов [*i, j, k*] или список параметров

* В дальнейшем назначение каждого символа будет рассмотрено подробно.

процедуры *(a, b, c);*

точка служит для отделения целой части числа от дробной, точнее для обозначения дробной части числа (десятичной);

число 10, написанное с понижением, используется как символ десятичного порядка числа; следующее за этим символом число является порядком;

двоеточие служит для разделения граничных пар в описаниях массивов, а также для отделения меток от операторов;

точка с запятой служит для разделения отдельных участков программы (операторов и описаний);

двоеточие и равенство является символом присваивания и используется в операторах присваивания, в операторах циклов и в описаниях переключателей;

пробел |_| применяется при построении строк; слова **шаг, до, пока** используются при построении операторов цикла;

словесный символ **примечание** ставится после символа **начало** или точки с запятой и позволяет включать после него любой пояснительный текст, не содержащий точек с запятой, который транслятором не воспринимается, а служит только для удобства понимания программы людьми. Конец пояснительного текста указывается точкой с запятой.

В АЛГОЛе используются четыре вида скобок: <скобка> : : = (|) | **[** | **]** | **начало** | **конец** |‘ | ’

Круглые скобки используются при построении выражений для определения в них порядка выполнения действий и при построении так называемых процедур. Квадратные скобки называются индексными; они используются для заключения индексов величин, граничных пар в описании массивов величин, индексов в так называемых указателях переключателей (см. ниже). Словесные скобки **начало** и **конец** называются операторными и служат для объединения операторов в составные операторы и блоки. Кавычки используются для образования строк, обозначающих нечисловые величины (например, собственные имена, качественные характеристики и др.).

Описатели служат для построения описаний величин и других объектов, используемых в АЛГОЛ-программах. Описания, как уже упоминалось, ставятся в начале каждого блока; состав описаний в разных блоках может быть различным.

<описатель> : : = **целый | вещественный | логический | массив | собственный | переключатель | процедура**

Описатели **целый, вещественный** и **логический** служат для описания типов величин.

Описатель **массив** служит для описания массивов, т.е. групп величин (например, матриц), а описатели **переключатель** и **процедура** служат для описания типовых алгоритмических процессов, играющих роль подпрограмм при обычном машинном программировании.

<спецификатор> : : = **метка** | **значение** | **строка**

Спецификаторы — это такие описатели, которые используются только в одном разделе языка, а именно в разделе описаний процедур. Они служат для характеристики величин, используемых в процедурах. В качестве спецификаторов в этом разделе могут использоваться и другие перечисленные выше описатели, но роль их несколько меняется. Описатели, используемые в качестве спецификаторов, менее полно определяют соответствующие величины по сравнению с тем случаем, когда они используются в описаниях в блоках. Например, в спецификации **массив** *M* не указывается размерность массива *M.* Подробно эти вопросы будут рассмотрены в дальнейшем при описании процедур.

Необходимые дополнительные пояснения всех основных символов языка будут даваться по мере их появления в процессе изложения.

Числа

В АЛГ**О**Ле используются только десятичные числа. Простейшим числом является целое число без знака, представляющее собой последовательность цифр.

<целое без знака> : : = <цифра> | <целое без знака> < цифра>

<целое> : : = <целое без знака> | + <целое без знака> | — <целое без знака>

<десятичная дробь> : : = <целое без знака>

Заметим, что за рубежом, точкой принято отделять целую часть числа от дробной, поэтому в АЛГОЛе для указания десятичной дроби принята точка, а не запятая.

<десятичное число> : : = <целое без знака> | <десятичная дробь> | <целое без знака > <десятичная дробь>

<десятичный порядок> : : = $_{10}$ <целое>

<число без знака> : : = <десятичное число> | <десятичный порядок> | <десятичное число> <десятичный: порядок>

<число> : : = <число без знака> | + <число без знака> | — <число без знака>

В указанных определениях десятичная дробь, десятичное число и десятичный порядок определяются как числа без знака, при этом, например, число вида — 0,135 не будет подходить под определение «десятичная дробь», а будет подходить под определение «число».

Десятичный порядок это масштабный множитель, представляющий собой степень десяти. Показатель степени может быть только целым десятичным числом (положительным и отрицательным). Отрицательный показатель степени в скобки не заключается.

Идентификаторы

Идентификаторы — это наименования различных величин. В отличие от принятого в математике правила обозначать величины отдельными буквами (возможно с индексами) в алгоритмических языках величины могут обозначаться группами букв или букв и цифр, т. е. словами.

Идентификатором называется любая последовательность букв или цифр, начинающаяся с буквы:

<идентификатор> : : = <буква> | <идентификатор> <буква> | <идентификатор> <цифра>

Идентификаторы не имеют закрепленного за ними смысла, а используются для обозначения различных величин: простых переменных, массивов, меток, переключателей и процедур. Все идентификаторы, кроме постоянно закрепленных для стандартных функций (sin, cos и т. д., см. ниже), могут выбираться произвольно.

Чтобы облегчить человеку чтение алгоритмов, иногда удобно в качестве идентификаторов величин использовать названия этих величин (например, высота, скорость и т. п.). Каждый идентификатор должен быть описан в том блоке, внутри которого он используется. Исключение составляют идентификаторы ряда стандартных функций, за которыми в АЛГОЛе закреплены постоянные идентификаторы. Описание определяет вид величины, которая обозначается этим идентификатором (например, массив, процедура и т. д.). Различные величины, используемые в одном и том же блоке программы, должны обозначаться обязательно различными идентификаторами.

Строки

Строками называются группы основных символов, представляющие собой нечисловые константы. В отличие от идентификаторов, используемых для обозначения переменных величин, принимающих в процессе вычислений каждый раз некоторое конкретное числовое значение, строки представляют самих себя, т. е. являются конкретными нечисловыми значениями.

Например, ‘ИВАНОВ’, ‘ПЕТРОВ’, ‘СИДОРОВ’ и т. п.

Для отличия строк от идентификаторов используются кавычки, представляющие собой один из видов скобок.

Определение строки, данное с помощью метаязыка, имеет следующий вид:

<пусто> : : =

<чистая строка> : : = <любая последовательность основных символов, не содержащая символ ‘ или символ ’ > | <пусто>

«Пусто» обозначает строку, в которой нет ни одного символа.

<открытая строка> : : = <чистая строка> | ‘<открытая строка>’ | <открытая строка> <открытая строка>

Примеры открытых строк:

ПЛАТЕЖНАЯ |_| ВЕДОМОСТЬ

АНКЕТА |_| ‘ИВАНОВА’

‘ГОРОД |_| ТАШКЕНТ’

Открытая строка — это группа основных символов, которая может включать в себя кавычки, но не обязательно охватывается ими.

<строка> : : = ‘<открытая строка>’

Строка отличается от открытой строки тем, что она обязательно охватывается кавычками.

В эталонном языке АЛГОЛ предусматривается возможность использования строк только в качестве фактических параметров процедур. В рассматриваемом нами варианте языка (АЛГЭМ) предусматривается более широкое использование строк для построения выражений и операторов; при этом вводятся строчные величины, которые могут иметь своим значением как отдельные строки, так и группы строк (в случае строчных массивов).

Метки

Как уже говорилось, некоторые операторы могут иметь перед собой метки, отделяемые от операторов двоеточиями.

Метки это по существу те же строки, но они используются не для обозначения нечисловой информации, подлежащей обработке, а для различения операторов программы с целью осуществления переходов. Кроме, этого функционального отличия меток от строк, в эталонном языке АЛГОЛ дается ограничение на структуру меток: метками могут быть только числа (целые без знака) или идентификаторы, а не любые наборы символов основного языка, как это определено для строк.

Однако это отличие не является принципиальным: метки подобно строкам представляют самих себя и им не могут присваиваться какие-либо значения, как это делается для идентификаторов. Заметим, что подобную роль играют идентификаторы процедур и переключателей (см. ниже); эти идентификаторы также представляют самих себя и за ними не подразумевается конкретного числового значения. В указанных случаях идентификаторы пишутся без кавычек, так как сам способ их применения определяет их роль.

В отличие от идентификаторов процедур идентификаторы процедур-функций обозначают всегда некоторые конкретные значения, получаемые в результате вычисления функций, и поэтому они не могут рассматриваться как строки, а являются идентификаторами в полном смысле этого определения.

Переменные

Переменная — это наименование некоторой величины, которая может принимать определенные значения. Переменные в зависимости от типа значений, которые они могут принимать, делятся на числовые — целые и вещественные — и нечисловые — логические и строчные (последние не в АЛГОЛе).

Значение для числовых переменных (целых и вещественных)— это число или множество чисел (вектор, матрица и т. д.); для логических переменных — одно логическое значение **(истина —** 1 или **ложь** — 0) или множество логических значений; для строчных переменных — одна строка или множество строк. Значения переменных могут изменяться в процессе вычислений с помощью так называемого оператора присваивания (см. ниже).

Приведем формальное синтаксическое определение понятия переменной. Для приближения данного

изложения к эталонному АЛГОЛу мы сейчас не вводим в рассмотрение составные переменные и даем определение переменной, подразумевая под ней элементарную (несоставную) переменную.

< идентификатор переменной> : : = <идентификатор>

<простая переменная> : : = <идентификатор переменной>

<индексное выражение> : : = <арифметическое выражение>

<список индексов> : : = <индексное выражение> | <список индексов>, <индексное выражение>

<идентификатор массива> : : = <идентификатор>

<переменная с индексами> :: = <идентификатор массива> | <список индексов>

<переменная> : : = <простая переменная> | <переменная с индексами>

Для каждой переменной с индексом дается описание массива, элементом которого она является. Описание массива содержит все необходимые сведения об этом массиве (его идентификатор, размерность, размеры, тип значений). Переменная с индексами — это наименование одного элемента массива, заданного индексами. Примеры переменных с индексами:

обычные обозначения	обозначения по АЛГОЛу
a_1	a[1]
$y_{i,j}$	$y[i ,j]$
A_{x_i, y_j}	$A[x[i], y[j]]$

Список индексов состоит из одного или нескольких арифметических выражений, разделенных запятыми.

Арифметическое выражение, являющееся индексным выражением, по своему смыслу может быть только целым числом. Если в результате вычислений получится нецелое число, то в качестве значения индекса берется ближайшее целое. Это делается по формуле $E(A+0{,}5)$, где E означает ближайшее целое, не превосходящее значения выражения в скобках, а A — фактическое значение арифметического выражения.

Указатели функций

Переменные величины в АЛГОЛе могут быть представлены также в виде так называемых указателей функций. В АЛГОЛе имеется способ для однократного представления в программе часто используемых в разных местах программы автономных участков программы. Это делается с помощью так называемых процедур. Описание процедуры содержит полное описание (также на АЛГОЛе) данного участка программы и помещается в начале блока вместе с другими описаниями. Обращения к процедурам могут совершаться из любого места блока при помощи оператора процедуры или указателя функции. Обращения имеют стандартный вид: пишется идентификатор соответствующей процедуры, за которым в круглых скобках указываются ее аргументы, разделенные запятыми. Аргументы называются фактическими параметрами процедуры.

Процедуры в общем случае могут быть построены для выполнения разных действий и давать в результате одно или несколько значений или не давать вообще ни одного значения, а осуществлять, например, перегруппировку величин. Если же процедура используется для вычисления функции, то она обязательно в результате дает одно значение.

В тех местах программы, в которых нужно производить вычисления с помощью данной процедуры, ставятся обращения к этой процедуре, которые могут иметь, например, такой вид: sin (x) или ПЛОЩАДЬ (a, h) и т. д.

В последнем примере идентификатором процедуры является слово «площадь», а величины a и h представляют собой параметры этой процедуры и обозначают, например, основание и высоту треугольника.

Подобные процедуры, служащие для вычисления какой-либо одной величины (числовой, логической или строчной) называются процедурами-функциями.

Приведенные выше примеры обращения к процедурам-функциям называются указателями функций. В отличие от указателей функций обращения к процедурам, не являющимся функциями, называются операторами процедур. Операторы процедур по внешнему виду похожи на указатели функций, но между ними имеются следующие отличия:

а) обращение к описанию процедуры-функции осуществляется при помощи указателя функции, который в результате выполнения всех действий, заданных соответствующим описанием процедуры, принимает одно определенное значение, в то же время оператор процедуры может вычислять несколько значений или вообще не вычислять никаких значений, а выполнять некоторые другие действия в процессе вычислений (контроль вычислений, перепись данных и т. д.);

б) указатель функции стоит всегда внутри какого-нибудь выражения, в то время как оператор процедуры всегда фигурирует в виде отдельного оператора, т. е. после него ставится точка с запятой.

Примеры: a + sin(x) и ВВОД (a, b, c);

Здесь sin (x) является указателем функции, ВВОД (a, b, c) — оператором процедуры;

в) процедура-функция в своем описании должна обязательно содержать оператор присваивания, осуществляющий присваивание вычисленного значения идентификатору процедуры-функции (который совпадает всегда с идентификатором указателя функции).

Синтаксическое определение указателя функции в АЛГОЛе имеет следующий вид:

<идентификатор процедуры> : : = <идентификатор>

<фактический параметр> : : = <строка> | <выражение> | <идентификатор массива > | <идентификатор переключателя> | <идентификатор процедуры>

<строка букв> : : = <буква> | <строка букв> <буква>

<ограничитель параметра> : : = , |) <строка букв>:(

<список фактических параметров> : : == <фактический параметр> | <фактический параметр> <ограничитель параметра> <фактический параметр>

<совокупность фактических параметров> : : = <пусто> | (<список фактических параметров>)

<указатель функции> : : = <идентификатор процедуры> <совокупность фактических параметров> Ограничитель параметра в виде) <строка букв>:(позволяет вставлять пояснительный текст между параметрами. Например, функция *v,* зависящая от величин *t*, *D, H,* т. е. *v*(*t, D, H*), может быть записана так:

V (*t*) дальность: (*D*) высота: (*H*)

Здесь величина *t* не имеет пояснений, а величины *D* и *H* поясняются.

В качестве фактических параметров наиболее часто фигурируют числа и переменные, которые являются частными случаями выражений.

Из приведенного синтаксического описания процедуры-функции следует, что аргументы функции всегда должны заключаться в скобки. Например, нельзя писать, как принято в математике, sin *x,* а нужно писать sin (*x*), так как запись sin*x* будет воспринята согласно правилам АЛГОЛа просто как идентификатор некоторой переменной.

Далее, так как в описании процедуры-функции обязательно должен быть оператор присваивания, осуществляющий присваивание вычисленного значения идентификатору процедуры-функции, то здесь могут встретиться необычные для математических формул записи. Например, присваивание некоторого значения может производиться просто идентификатору sin без указания аргумента *x;* этот идентификатор sin будет иметь определенное числовое значение, равное значению sin (*x*).

В АЛГОЛе для ряда стандартных функций постоянно закреплены идентификаторы, которые запрещается использовать для других целей.

Стандартные функции не описываются в программах, а используются сразу в тех выражениях, где они необходимы.

К числу стандартных функций относятся следующие функции от выражения *E,* которое является аргументом (фактическим параметром):

abs (*E*) —для вычисления модуля (абсолютной величины) значения *E,*

sign (*E*) —для знака значения *E*(+ 1 для *E*>0, 0 для *E* = 0 и — 1 для *E*<0),

sqrt (*E*) —для квадратного корня из значения *E,*

sin (*E*) —для синуса значения *E,*

cos (*E*) —для косинуса значения *E,*

arctan (*E*)—для главного значения арктангенса значения *E,*

ln (*E*) —для натурального логарифма значения *E,*

exp (*E*) — для показательной функции значения *E* (e^E).

В АЛГОЛе принято, что все эти функции могут быть использованы как с аргументами, имеющими тип **вещественный**, так и с аргументами, имеющими тип **целый**. Все эти функции, кроме функции sign (*E*), дают значение, имеющее тип **вещественный**. Функция sign (*E*) дает значение целого типа.

К числу стандартных функций относится также функция entier (*E*), осуществляющая преобразование выражения *E,* имеющего тип **вещественный** в выражение типа **целый,** и присваивающая ему значение, являющееся наибольшим целым, не превышающим значение *E.*

Выражения

Выражения — это конструкции языка, имеющие определенное функциональное назначение; они показывают, какие действия и в каком порядке должны выполняться над величинами или как эти величины должны использоваться (например, в качестве операндов арифметических или логических операций, в качестве индексов или параметров и т. п.).

Составными частями выражений могут быть переменные, указатели функций, числа, логические значения, ограничители и некоторые символы операций.

Так как в синтаксические определения переменных и указателей функций входят выражения (например, индексное выражение, являющееся арифметическим выражением), то определение самих выражений, а также переменных и указателей функций может быть произведено только рекурсивным способом.

В АЛГОЛе используются три типа выражений: арифметические, логические и именующие:

<выражение> : : = <арифметическое выражение> | <логическое выражение> | <именующее выражение>

Выражения делятся на два вида: простые и условные.

Мы рассмотрим сначала простые арифметические и логические выражения, затем рассмотрим условные арифметические и логические выражения. Именующие выражения будут рассмотрены вместе с операторами перехода.

Простое арифметическое выражение

Это выражение представляет собой формулу, определяющую порядок вычислений некоторого числового значения.

Элементарными компонентами формул являются первичные арифметические выражения, которые будем называть просто первичными выражениями:

<знак операции типа сложения> : : = + | —

<знак операции типа умножения> : : = × | / | ÷

<первичное выражение> : : = <число без знака> | <переменная> | <указатель функции> | (<арифметическое выражение>)

Переменные и функции, являющиеся первичными арифметическими выражениями, могут быть только числовыми, т. е. типа целый или вещественный.

Из первичных выражений образуются множители, представляющие собой степени первичных выражений:

<множитель> : : = <первичное выражение> | <множитель> ↑<первичное выражение>

Показатель степени должен заключаться в скобки (круглые), за исключением случаев, когда он является числом без знака, переменной или указателем функции. Отрицательный показатель заключается в скобки.

Примеры записи множителей:

$$a\uparrow b\uparrow c = (a^b)^c; \quad a \uparrow (b \times c) = a^{bc};$$

Несколько операций возведения в степень выполняются последовательно слева направо. Суть правил, определяющих тип результата, сводится к следующему: при нулевом основании результат равен нулю или не определен (при показателе, равном или меньшем нуля). При целом положительном или равном нулю показателе тип результата совпадает с типом основания. При целом отрицательном показателе и основании, не равном нулю, тип результата вещественный.

При основании *a,* большем нуля, и вещественном показателе *r* (любом) результат для $a \uparrow r$ вычисляется по формуле exp ($r \times \ln(a)$) и имеет тип вещественный. При основании, меньшем нуля, и вещественном показателе результат, не определен.

<терм> : : = <множитель> | <терм> <знак операции типа умножения> <множитель>

В данном случае используется промежуточное понятие — «терм», представляющее собой одночлен без знака.

Пример терма:

$$a \times b / c$$

Тип результата определяется следующими правилами: если множители целые, то и результат целый, в противном случае — вещественный.

Если делитель равен нулю, то операция / не определена. Операция ÷ определена только для целых при условии, что делитель не равен нулю.

Результат операции всегда целый и определяется по формуле

$$m \div n = \text{sign}(m/n) \times \text{entier}(\text{abs}(m/n))$$

Здесь слово entier обозначает взятие целой части, т. е. наибольшего целого, не превышающего фактического значения аргумента.

<простое арифметическое выражение> : : = <терм> | <знак операции типа сложения> <терм> | <простое арифметическое выражение> <знак операции типа сложения> <терм>

Следует остановиться на порядке выполнения действий при вычислении арифметических выражений.

Сначала вычисляются выражения в скобках, затем выполняются операции в порядке следующего приоритета:

↑	первый приоритет;
×/÷	второй приоритет;
+ –	третий приоритет.

Операции одинакового старшинства выполняются в порядке записи слева направо.

Для выяснения возможности выполнения очередной операции каждый раз производится анализ следующей по порядку операции, и если она оказывается одинаковой или младшей по приоритету, то выполняется данная операция. В противном случае производится анализ следующей операции и т. д. до тех пор, пока не будет встречена младшая или равная по приоритету операция.

Пример

$$a + b \times c \uparrow (d + e)$$

Это выражение будет вычисляться в следующем порядке:

$$(d + e)$$
$$c\uparrow(d + e)$$
$$(c \uparrow (d + e)) \times b$$
$$((c\uparrow(d + e)) \times b) + a$$

Простые логические выражения

Логические выражения и логические переменные и функции могут иметь одно из двух значений: **истинно** (1) или **ложно** (0). Синтаксическое определение простого логического выражения имеет следующий вид:

<знак операции отношения> : : = < | ≤ | = | ≥ | > |≠

<отношение> : : = <простое арифметическое выражение> <знак операции отношения> <простое арифметическое выражение>

<первичное логическое выражение> : : = <логическое значение> | <переменная> | <указатель функции> | <отношение> | (<логическое выражение>)

<вторичное логическое выражение> : : = <первичное логическое выражение> | ¬ <первичное логическое выражение>

<логический одночлен> : : = <вторичное логическое выражение > | <логический одночлен> ∧ <вторичное логическое выражение>

<логический терм> : : = <логический одночлен> | <логический терм> ∨ <логический одночлен>

<импликация> : : = <логический терм> | импликация > ⊃ < логический терм>
<простое логическое выражение> : : = <импликация> | <простое логическое выражение> ≡ <импликация>

Переменные и функции, являющиеся первичными логическими выражениями, должны иметь тип **логический.**

Примеры простых логических выражений:

$$A \vee B \wedge C \supset (D \vee F)$$

$$x > y \wedge a \leq b \wedge \neg d$$

Таким образом, первичное логическое выражение — это величина, принимающая одно из двух возможных логических значений, а простое логическое выражение — это комбинация таких величин, соединенных знаками логических операций.

Определения пяти логических операций, включенных в состав АЛГОЛа, даны в следующей таблице (см. также гл. 1, § 3).

A	0	0	1	1
B	0	1	0	1
¬ *A*	1	1	0	0
A/*B*	0	0	0	1
A\/*B*	0	1	1	1
A ⊃ *B*	1	1	0	1
A≡*B*	1	0	0	1

В логических выражениях, как мы видели, могут участвовать арифметические операции, операции отношения и логические операции. Для всех этих операций имеет место следующий общий порядок приоритета (старшинства):

арифметические операции:

1) ↑
2) × /÷
3) + —

операции отношения:

4) < ≤ = ≥ > ≠

При рассмотрении операций отношения вопрос об их старшинстве по отношению друг к другу не возникает;

логические операции:

5) ¬ 8) ⊃
6) ∧ 9) ≡
7) ∨

Во всех случаях, когда нужно изменить порядок выполнения операций (всех трех видов), используются круглые скобки.

Условные арифметические выражения

Условные арифметические выражения включают в себя наряду с арифметическими операциями и проверку некоторых условий, определяющих выбор того или иного варианта вычисления данного арифметического выражения. Проверяемые условия формулируются всегда в виде логических выражений:

<условне> : : = **если** <логическое выражение> **то**

<условное арифметическое выражение> : : = <условие> <простое арифметическое выражение> **иначе** <арифметическое выражение>

<арифметическое выражение> : : = <простое арифметическое выражение> | <условное арифметическое выражение>

Следует обратить внимание на то, что после условия всегда следует простое арифметическое выражение, а после символа **иначе** — арифметическое выражение, в том числе условное арифметическое выражение и т. д.

Конструкция условных арифметических выражений схематически может быть представлена в виде

если *Л*1 **то** *А*1 **иначе** *А*

если *Л*1 **то** *А*1 **иначе если** *Л*2 **то** *А*2 **иначе если** *Л*3 **то** *А*3 **иначе** *А*

где *Л*1, *Л*2**,** *Л*3 — логические выражения;

*А*1, *А*2, *А*3 — простые арифметические выражения;

А — арифметическое выражение.

Вычисление подобных условных арифметических выражений производится следующим образом. Последовательно проверяются слева направо логические выражения (*Л*1, *Л*2, *Л*3), стоящие после символов **если.**

Первое истинное логическое выражение указывает, что следует взять в качестве значения данного условного арифметического выражения то простое арифметическое выражение, которое стоит за символом то, следующим за истинным логическим выражением. Если все логические выражения окажутся ложными, то в качестве значения условного выражения берется последнее арифметическое выражение, которое конечно будет простым.

Пример. Пусть величины *x, y, a, b* имеют тип вещественный. Тогда можно записать такое условное арифметическое выражение: КОЭФФИЦИЕНТ 2—*b*

если *x* = *a* /\ *y*<*b* **то** КОЭФФИЦИЕНТ 2 + *b* **иначе** КОЭФФИЦИЕНТ 2 – *b*

Заметим, что в случае истинности проверяемого условия значением этого условного арифметического выражения будет сумма двух величин: КОЭФФИЦИЕНТ 2 и *b,* а в случае ложности проверяемого условия значением этого выражения будет разность этих же величин.

Условные логические выражения

Условные логические выражения строятся подобно условным арифметическим выражениям, только вместо простых арифметических выражений используются простые логические выражения, а вместо арифметического выражения в конце условного выражения ставится логическое выражение.

<условное логическое выражение> : : = <условие> <простое логическое выражение> **иначе** <логическое выражение>

<логическое выражение> : : = <простое логическое выражение > | < условное логическое выражение>

Условие определяется точно так же, как и в случае условных арифметических выражений. Общий вид условного логического выражения может быть представлен следующей схемой:

если *Л*1 **то** *L*1 **иначе если** *Л*2 **то** *L*2 **иначе** *Л*3

Здесь *Л*1, *Л*2, *Л*3 — логические выражения;

*L*1, *L*2— простые логические выражения.

Особенностью вычисления подобных условных логических выражений является то, что стоящие после символов **если** логические выражения *(Л*1, *Л*2 и т. д.) сами могут быть условными логическими выражениями, и таким образом, могут встретиться несколько подряд стоящих символов **если** и за последним из них будет стоять простое логическое выражение.

Способ вычисления сводится к последовательному выделению условных логических выражений, начиная с самого внутреннего простого логического выражения, и замене этих условных логических выражений их значениями. При этом нужно придерживаться основного правила, что каждому символу **если** соответствует ближайший справа символ **иначе**, не считая уже выделенных выражений. Пример условного логического выражения показан на рис. 13,*а*.

Просматриваем выражение и после третьего **если** находим простое логическое выражение *L*1. Вычисляем это

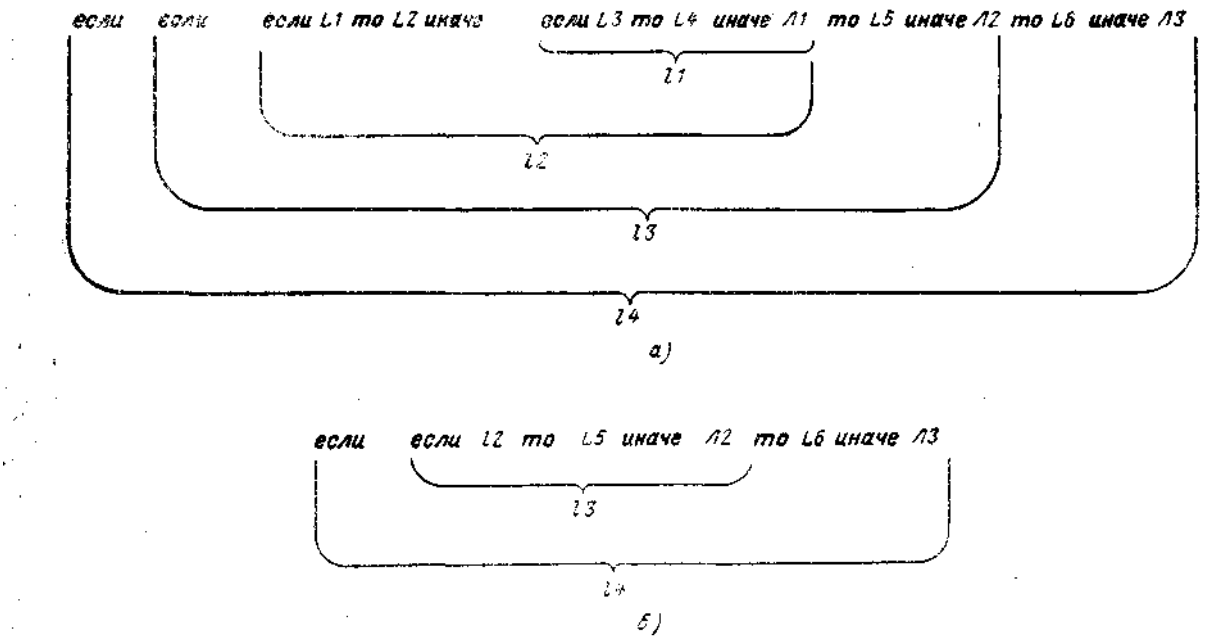


РИС. 13. Пример условного логического выражения:
а — исходное выражение; *б* — после первого этапа преобразования.

выражение, и если оно истинно, то берем *L*2, если ложно, то находим ближайший справа символ **иначе** и берем стоящее за ним логическое выражение (*l*1). В данном примере оно оказалось также условным логическим выражением; мы его вычисляем аналогичным образом, причем подобная цепочка действий может продолжаться, так как *Л*1 может также оказаться условным логическим выражением. Допустим, что мы нашли значение выражения *l*1 и при *L*1 ложном оно будет значением выражения *l*2. Теперь наше условное логическое выражение примет вид, показанный на рис. 13,*б*.

Подобным же образом вычисляем условное логическое выражение *l*3 и находим его значение *(L*5 или *Л*2*)*. Самое внешнее условное логическое выражение будет иметь вид

если *l*3 **то** *L*6 **иначе** *Л*3

Оно вычисляется описанным выше способом.

В условных выражениях весьма важно указание, что после символа **то** обязательно должно стоять простое (а не условное) выражение. Это обеспечит соответствие между символами **если, то** и **иначе,** относящимися к одному и тому же выражению. При отсутствии такого ограничения условных выражений могла бы возникнуть неопределенность в их понимании. Далее следует заметить, что в условных выражениях в отличие от условных

операторов (см. ниже) обязательно каждому символу **если** соответствует символ **иначе.**

Заметим, что любое условное логическое выражение может быть представлено эквивалентным ему простым логическим выражением. Использование условных логических выражений может в ряде случаев существенно сократить процесс вычислений по сравнению с представлением той же логической зависимости простым логическим выражением, так как при этом вычисляется одно из двух выражений: либо стоящее после **то**, либо стоящее после **иначе.**

Однако различия в порядке вычислений этих двух видов логических выражений могут привести иногда к побочным эффектам, если в этих логических выражениях имеются функции, которые при одном порядке вычислений должны вычисляться, а при другом — не должны вычисляться.

§ 8 ОПЕРАТОРЫ

Операторы — это единицы действий в алгоритмическом языке. Операторы могут объединяться в составные операторы и блоки, которые являются частными видами операторов, и поэтому определение понятия «оператор» по необходимости рекурсивное; при этом предполагается также, что синтаксические определения описаний уже даны и они могут использоваться при определении операторов.

Естественным порядком выполнения операторов называется тот порядок, когда они выполняются в порядке их написания в программе сверху вниз и слева направо. Этот естественный порядок может изменяться при помощи операторов перехода, которые явно определяют своего преемника, т. е. оператор, который должен выполняться следующим. При помощи условных операторов могут осуществляться пропуски некоторых операторов.

Для того чтобы можно было указывать те операторы, к которым должны совершаться переходы, эти операторы снабжаются метками.

Приведем общую классификацию операторов и синтаксические определения понятий, относящихся к операторам:

<непомеченный основной оператор> : : = <оператор присваивания> | <оператор перехода> | <пустой оператор> | < оператор процедуры>

<основной оператор> : : = <непомеченный основной оператор> | <метка> : <основной оператор>

<безусловный оператор> : : = <основной оператор> | <составной оператор> | <блок>

<оператор> : : = <безусловный оператор> | <условный оператор> | <оператор цикла>

Синтаксические определения понятий «составной оператор», «блок» и «программа» выглядят следующим образом:

<конец составного> : : = <оператор> **конец** |<оператор>; <конец составного>

<начало блока> : : = **начало** <описание> | <начало блока>; <описание>

<непомеченный составной> : : = **начало** <конец составного>

<непомеченный блок> : : = <начало блока>; <конец составного>

<составной оператор> : : = <непомеченный составной> | <метка>: <составной оператор>

<блок> : : = <непомеченный блок> | < метка>: <блок>

<программа> : : = <блок> | <составной оператор>

Оператор присваивания

Значения арифметических и логических выражений (а также строчных выражений в АЛГЭМе) могут присваиваться одной или нескольким переменным или функциям. (Последнее применяется только в описаниях процедур-функций.) Присваивание значений делается с помощью так называемого оператора присваивания, играющего основную роль в любом алгоритмическом языке. В машинной интерпретации оператор присваивания соответствует посылке кода в ячейку (или группу ячеек).

Синтаксическое определение оператора присваивания в АЛГОЛе:

<левая часть> : : = <переменная> : = | <идентификатор процедуры> : =

<список левой части> : : = <левая часть> | <список левой части> <левая часть>

<оператор присваивания> : : = <список левой части> <арифметическое выражение> | <список левой части> <логическое выражение>

Символ : = означает «принимает значение» или «становится равным». Примеры операторов присваивания:

ПРИЗНАК СОВПАДЕНИЯ := **если** $a = b$ **то** 5 **иначе** 10;

$z: = y: = x: = a + b;$

В первом примере в правой части оператора присваивания стоит условное арифметическое выражение.

Как было сказано раньше, в конце операторов ставятся точки с запятой. В отличие от обычных математических формул в списке левой части оператора присваивания может стоять переменная, используемая и в правой части, например: $n: = n+1$;

Идентификатору n в правой части соответствует значение этой переменной до выполнения оператора присваивания, а идентификатор n в левой части представляет значение этой переменной уже после выполнения оператора присваивания.

Типы переменных в левой и правой частях оператора присваивания не обязательно должны быть одинаковыми. Если переменная в левой части имеет тип целый, а в правой части результат вычисления арифметического выражения имеет тип вещественный, то при выполнении оператора присваивания производится округление результата до ближайшего целого по формуле entier $(A + 0{,}5)$, где A — фактическое значение результата.

Для обеспечения однозначности результатов в тех случаях, когда в операторах присваивания встречаются переменные с индексами, изменяемыми этим оператором присваивания, установлено следующее правило, определяющее порядок действий при выполнении таких операторов присваивания.

Сначала вычисляются значения всех индексных выражений, встречающихся в списке левой части оператора присваивания. Затем вычисляется значение правой части оператора присваивания. Полученное значение правой части присваивается всем переменным списка левой части, взятым со значениями ранее вычисленных индексных выражений. В АЛГОЛе индексы у переменных указываются в квадратных скобках.

Например:

$i := j := k := m + 1;$

$x[i, j, k] := i := j := k := m+n;$

Величина $m + n$ будет присвоена величине $x,,$ взятой со значениями индексов $i, j, k,$ равными $m+1$; а сами индексы после этого получат значение $m + n$.

Именующие выражения и оператор перехода

В алгоритмических языках, да и вообще в математике, действует правило, согласно которому порядок выполнения вычислений по нескольким формулам определяется порядком их записи: слева направо и сверху вниз. В таком порядке выполняются обычно и операторы в АЛГОЛ-программах.

При необходимости изменить этот естественный порядок выполнения операторов используется специальный оператор перехода, соответствующий по своему смыслу команде безусловного перехода в машинах.

Оператор перехода определяется синтаксически следующим образом:

<оператор перехода> : : = **перейти к** <именующее выражение>

перейти к — это основной символ, обозначающий одну из операций следования, а именно операцию перехода;

именующее выражение является правилом, определяющим метку того оператора, к которому должен быть совершен переход.

Как говорилось раньше, те операторы программы, к которым должны или могут совершаться переходы, снабжаются метками; метки пишутся перед этими операторами и отделяются от них двоеточиями.

Именующее выражение по своему назначению близко к строчному выражению, используемому в АЛГЭМе (см. ниже). Отличие заключается в том, что строчные выражения служат для представления обрабатываемой информации, а именующее выражение — для описания самого процесса обработки. Но и те и другие (выражения служат для представления нечисловых значений.

Приведем синтаксическое определение именующего выражения:

<метка> : : = <идентификатор> | <целое без знака>

<идентификатор переключателя> : : = <идентификатор >

<указатель переключателя> : : = <идентификатор переключателя> [<индексное выражение>]

<простое именующее выражение> : : = <метка> | <указатель переключателя> | (<именующее выражение>)

<именующее выражение> : : = <простое именующее выражение> | <условие> <простое именующее выражение> **иначе** <именующее выражение>

Как видно из этой записи, структура условного именующего выражения полностью совпадает со структурой условных выражений арифметического и логического типов.

Понятие метки было рассмотрено раньше; меткой может быть идентификатор или целое без знака, причем это целое, используемое в качестве метки, трактуется просто как строка символов. Часто ограничиваются использованием в качестве меток только идентификаторов, считая, что применение целого без знака в качестве метки может приводить к некоторым неудобствам.

Используя арифметические и логические выражения, операторы присваивания, метки и операторы перехода, можно уже строить разнообразные программы, в том числе программы, включающие в себя циклы.

Рассмотрим, например, задачу о нахождении суммы парных произведений двух групп чисел $p[i]$ и $q[i]$, где i меняется от 1 до n. Пусть S обозначает вычисляемую сумму.

Программа на АЛГОЛе (без описания информации) будет иметь вид:

$S := 0; \quad i := 1;$

$M : S := p[i] \times q[i] + S \qquad i := i + 1$

перейти к если $i \le n$ **то** M **иначе** ДАЛЬШЕ;

здесь идентификатор ДАЛЬШЕ условно обозначает метку оператора, к которому нужно перейти после окончания данного расчета. В программе использован оператор перехода с условным именующим выражением **если** $i \le n$ **то** M **иначе** ДАЛЬШЕ.

Рассмотрим назначение и строение переключателя.

Часто при программировании встречаются случаи, когда необходимо из одного и того же места программы перейти в разные места программы в зависимости от выполнения некоторых условий. Такие переходы можно запрограммировать, используя оператор перехода с условным именующим выражением, но более экономно переходы представляются с помощью специального средства, предусмотренного в АЛГОЛе и называемого переключателем.

Переключатель состоит из двух частей: указателя переключателя и описания переключателя.

Указатель переключателя представляет собой переменную с индексом: эта переменная в зависимости от значения индекса может принимать одно из значений, заданных описанием переключателя. Значением указателя переключателя, в конечном счете, должна быть метка.

Указатель переключателя ставится на место именующего выражения в операторе перехода, например:

перейти к $П[i]$;

здесь $П$ —идентификатор переключателя, а i — индексное выражение.

Описание переключателя имеет вид:

переключатель $П := И1, И2, И3, ..., Ин$;

здесь переключатель — основной символ, $П$ — идентификатор переключателя (в данном примере именно $П$), а

идентификаторы *И*1, *И2, И*3, ..., *Ип* образуют так называемый переключательный список, содержащий те именующие выражения, которые могут представлять значения данного переключателя. Они могут быть просто метками, а могут быть и снова указателями переключателей или условными именующими выражениями.

Описание переключателя помещается вместе с остальными описаниями, относящимися к данному блоку, т.е. в его начале. Действие переключателя происходит следующим образом.

Перед выполнением оператора перехода, содержащего указатель переключателя, должен быть выполнен оператор, определяющий значение индексного выражения в указателе переключателя. После этого выполнение оператора перехода сводится к тому, что по идентификатору переключателя находится описание соответствующего переключателя в начале блока, а по значению индекса указателя переключателя в переключательном списке выбирается именующее выражение, стоящее на том месте (считая слева направо), номер которого равен значению индекса. В общем случае найденное именующее выражение может оказаться снова указателем переключателя или даже условным именующим выражением, которое укажет другой указатель переключателя; в этом случае процесс повторяется аналогичным образом, т. е. производится обращение к другому описанию переключателя и т. д.

В эталонном АЛГОЛе допускается произвольная глубина подобных переходов. Практически же эти возможности, как правило, не используются. В связи с этим в сокращенном варианте АЛГОЛа сделаны следующие ограничения:

— в качестве именующих выражений в переключательном списке могут стоять только метки;

— исключаются условные именующие выражения.

Как будет видно из дальнейшего, условные переходы можно осуществлять не только с помощью условных именующих выражений, но и с помощью условных операторов, что, вообще говоря, практически применяется чаще.

Приведем синтаксическое определение описания переключателя:

<идентификатор переключателя> : : = <идентификатор>

<переключательный список> : : = <именующее выражение> | <переключательный список>, <именующее выражение>

<описание переключателя> : : = **переключатель**

<идентификатор переключателя> : : = <переключательный список>

Для пояснения назначения переключателя рассмотрим следующий пример. Пусть из некоторого места программы требуется осуществить переход к четырем различным операторам, имеющим метки *М*l, *М*2, *М*3, *М*4; причем переход должен происходить в зависимости от значения некоторой величины *К,* которая может принимать целые значения 1, 2, 3 или 4.

Используя условное именующее выражение, этот переход можно записать так:

перейти к если *К*=1 **то** *М*l **иначе если** *К* = 2 **то** *М*2 **иначе если** *К*= 3 **то** *М*3 **иначе** *М*4;

Этот же переход при помощи переключателя будет иметь вид:

переключатель *П*: = *М*1, *М*2, *М*3, *М*4;

…….

перейти к *П* [*К]*;

Ясно, что в обоих случаях предварительно должна быть вычислена величина *К.* Одно и то же описание переключателя может использоваться многими указателями переключателя, находящимися в разных местах программы.

В связи с рассматриваемым вопросом о реализации переходов в программе следует сделать несколько замечаний относительно пустого оператора. Это понятие, как мы уже говорили раньше, введено для того, чтобы можно было помещать в программах метки, за которыми не стоят операторы. Такие метки нужны, в частности, для указания перехода на конец блока или составного оператора.

Для иллюстрации применения переключателя приведем пример программы вычисления суммы парных произведений двух групп величин $p[i]$ и $q[i]$, где i изменяется от 1 до n (без описаний данных).

начало переключатель *М*: = *М*1, ВЫХОД;

$$i := 1;\ S := 0;$$

$$M1 : S := S + p[i] \times q[i];\ i := i+1;$$

перейти к *М* [**если** $i \le n$ **то** 1 **иначе** 2];

ВЫХОД: ПЕЧАТЬ (‘*S* =’, *S*)

конец

Этот же пример можно записать в следующем виде, имеющем чисто иллюстративную цель:

начало переключатель *М*: = **если** $i \le n$ **то** *М*1 **иначе** ВЫХОД;

$$i := 1;\ S := 0;$$

$$M1 : S := S + p[i] \times q[i];\quad i := i + 1;$$

перейти к $M[1]$;

ВЫХОД:ПЕЧАТЬ (‘*S*=’**,** *S*)

конец

Заметим, что после метки ВЫХОД стоит оператор специальной процедуры выдачи данных на печать. В этом операторе в круглых скобках указаны в качестве фактических параметров (т. е. аргументов) этой процедуры те величины, которые должны быть выданы на печать. Первая величина является строкой, состоящей из двух символов *S* и =, заключенных в кавычки. Вторая величина — просто величина *S*, обозначающая результат вычислений. Так как любая строка представляет самое себя и не принимает никаких значений, то на печать в качестве первой величины будут выданы два символа *S* = , за которыми будет напечатано то числовое значение, которое получит *S* в результате вычислений.

Условный оператор

Как мы уже видели, изменение порядка выполнения операторов в зависимости от некоторого условия может быть осуществлено с помощью оператора перехода, в котором после символа **перейти к** стоит условное именующее выражение. Однако более употребительным является другой способ, основанный на использовании условных операторов. Структура условного оператора напоминает структуру условных выражений, только вместо выбираемых выражений в нем стоят операторы. Синтаксическое определение условного оператора имеет вид

<условие> : : = **если** <логическое выражение> **то**
<безусловный оператор> : : = <основной оператор> | <составной оператор> | <блок>
<оператор «если»> : : = <условие> < безусловный оператор>
<условный оператор> : : = <оператор «если»> | <оператор «если»> **иначе** <оператор> | <условие> <оператор цикла> | <метка> : <условный оператор>

Заметим, что в операторе «если» разрешается использовать только безусловный оператор, а условный оператор с участием оператора цикла разрешается строить только без продолжения **иначе** <оператор>. Оба эти ограничения имеют целью предотвратить неопределенность в согласовании символов **если** и **иначе.** Эта неопределенность могла бы возникнуть в связи с тем, что условный оператор может быть двух видов: с продолжением **иначе** <оператор> и без такого продолжения. Поэтому если бы разрешить в условном операторе ставить после символа то снова условный оператор, то могла бы возникнуть, например, такая картина:

если *Л*1 **то если** *Л*2 **то** *S*1 **иначе** *S*2;

здесь не ясно, какому из двух **если** соответствует **иначе.**

Как будет видно из дальнейшего, в самом операторе цикла разрешается использовать условный оператор, что тоже могло бы приводить к подобной неопределенности, если бы не было указанных ограничений. Если все же нужно образовать условный оператор с оператором цикла, после которого должен идти символ **иначе** с последующим оператором, то это может быть достигнуто путем заключения оператора цикла в операторные скобки **начало** и **конец.** При этом оператор цикла превращается в составной оператор, т. е. относится уже к безусловным операторам. Заметим, что при наличии понятия пустого оператора перед символом **конец** можно ставить или не ставить точку с запятой. Перед символом **иначе** точку с запятой ставить нельзя ни в каких случаях. Из приведенного выше синтаксического определения видно, что условный оператор может быть двух видов:

а) односторонний условный оператор вида

если *Л* **то** *S*1; *S*;

где *Л* — логическое выражение, *S*1 — безусловный оператор или оператор цикла и *S* — какой-то оператор, который следует за данным условным оператором. Суть работы одностороннего условного оператора состоит в том, что в зависимости от того, истинно или ложно логическое выражение *Л,* оператор *S*1 выполняется или пропускается. В обоих случаях следующим выполняется оператор *S*;

б) двусторонний условный оператор вида

если *Л* **то** *S*1 **иначе** *S*2; *S*;

здесь *S*1 — безусловный оператор, а *S*2 и *S* — операторы. Если логическое выражение *Л* истинно, то выполняется оператор *S*1 и пропускается оператор *S*2, в противном случае пропускается оператор *S*1 и выполняется оператор *S*2. В обоих случаях следующим за условным оператором выполняется оператор *S* (если операторы *S*1, *S*2, а также *S*1 в предыдущем варианте не содержат внутри себя операторов перехода). Оператор, стоящий после символа **иначе**, сам может быть условным оператором и таким образом могут образовываться цепочки произвольной длины, включающие в себя несколько символов **если** с соответствующими логическими выражениями (*Л*1*, Л*2, *Л*3 и т. д.). Например,

если *Л*1 **то** *S*1 **иначе если** *Л*2 то *S*2 **иначе** *S*;

здесь *S*1 и *S*2 — безусловные операторы, а *S* — оператор.

Выполнение подобных условных операторов сводится к последовательной проверке слева направо логических выражений. Первое истинное логическое выражение указывает, что должен быть выполнен безусловный оператор, следующий за этим выражением (за символом **то**). Если все логические выражения окажутся ложными, то выполняется последний оператор, стоящий за символом **иначе**, а если его нет, то выполняется оператор, стоящий за данным условным оператором. Следует заметить, что возможны переходы извне к какому-нибудь оператору, входящему в состав условного оператора (и имеющему, конечно, свою метку). В этом случае, если оператор, к которому совершен переход извне, сам не определяет своего преемника, т.е. того оператора, который должен выполняться за ним, следующим будет выполняться преемник всего условного оператора таким же образом, как если бы выполнение условного оператора происходило с самого начала и выбор для выполнения данного оператора (т. е. оператора, к которому совершен переход) произошел в результате проверки предшествующих условий.

В качестве примера применения условных операторов приведем программу вычисления двух сумм положительных и отрицательных чисел, выбранных из общей последовательности чисел a_0 , a_1, ..., a_n (без описания данных):

начало *S*: = *t*: = *i*: = *0;*
ПРОВЕРКА: **если** $i \le n$ **то**
начало если $a[i] > 0$ **то** $S: = S + a[i]$
иначе $t: = t + a[i]$: $i: = i + 1$ *;*
перейти к ПРОВЕРКА;
конец
конец

Вторым примером будет служить запись с помощью условного оператора программы вычисления скалярного произведения двух *n*-мерных векторов *p* и *q,* которая раньше была записана с помощью переключателя и условного именующего выражения.

Условный оператор	Условное именующее выражение
начало i:=1; S:=0; $M1$:S:=S+$p[i] \times q[i]$; i:=i+1; **если** $i \le n$ **то перейти к** $M1$; ПЕЧАТЬ (‘S=’, S); **конец**	**начало** **переключатель** M:=$M1$, ВЫХОД; i:=1; S:=0; $M1$:S:=$S + p[i] \times q[i]$; i:=i+1; **перейти к** M [**если** $i \le n$ **то** 1 **иначе** 2]; ВЫХОД: ПЕЧАТЬ (‘S=’, S) **конец**

Пустой оператор

<пустой оператор> : : = <пусто>

Пустой оператор не выполняет никакого действия; он может служить для помещения метки.

Оператор цикла

В связи с важностью циклических вычислительных процессов для решения самых разнообразных задач с помощью цифровых программно-управляемых машин в АЛГОЛе предусматривается специальный оператор цикла. Этот оператор обеспечивает более наглядную и компактную запись таких процессов по сравнению с представлением их с помощью операторов перехода и условных именующих выражений или условных операторов. Синтаксическое определение оператора цикла:

<элемент списка цикла> : : = <арифметическое выражение> | <арифметическое выражение> **шаг** <арифметическое выражение> **до** <арифметическое выражение> | <арифметическое выражение> **пока** <логическое выражение>

<список цикла> : : = <элемент списка цикла> | <список цикла>, <элемент списка цикла>

<заголовок цикла>: : = **для** <переменная>: = <список цикла> **цикл**

<оператор цикла> : : = <заголовок цикла> <оператор> |<метка>:<оператор цикла>

Как видно из приведенного выше определения элемента списка цикла, эти элементы могут быть трех типов и соответственно им принято различать три типа циклов: цикл с перечислением, цикл с арифметической прогрессией и цикл с проверкой (последний тип цикла называют иногда циклом с пересчетом).

Цикл с перечислением имеет вид

для v:=$A1$, $A2$, $A3$**, ...,** An **цикл** S**;**

Здесь символ **для** является символом начала, а символ **цикл** — символом конца заголовка цикла. Идентификатор переменной v представляет собой так называемый параметр цикла, т. е. переменную величину, которая изменяется с каждым повторением цикла. В различных конкретных циклах этот параметр может быть обозначен любыми другими идентификаторами; здесь в общем случае может стоять и переменная с индексами, хотя часто делают ограничение, что параметром цикла может быть только простая переменная. Символ присваивания: = в данном случае условно показывает, что параметр цикла v должен последовательно принимать значения арифметических выражений $A1$, $A2$, $A3$, . ..,An, перечисленных за этим символом. Наконец, S условно обозначает некоторый оператор, который должен быть многократно повторен (выполнен в цикле) при различных значениях параметра цикла v.

Цикл с арифметической прогрессией имеет вид

для v: = $A1$ **шаг** $A2$ **до** $A3$ **цикл** S;

Здесь $A1$—арифметическое выражение, показывающее начальное значение параметра цикла v. Этот параметр при данном типе элемента списка цикла должен изменяться по закону арифметической прогрессии с шагом $A2$. Арифметическое выражение $A3$, представляющее собой ограничение значения параметра цикла, определяет момент окончания повторений цикла для данного элемента списка цикла. Все три выражения $A1$, $A2$, $A3$ могут быть как положительными, так и отрицательными; между ними должны соблюдаться естественные для арифметической прогрессии соотношения.

Порядок выполнения написанного выше оператора цикла может быть представлен следующим составным оператором:

начало v**: =** $A1$**;**

M: **если** $(v - A3) \times sign\ (A2) \le 0$ **то**

начало S; v**: =** $v + A2$; **перейти к** M **конец**

конец

Цикл с элементом списка типа пересчета имеет вид

для v: = $A1$ **пока** $Л$ **цикл** S**;**

Здесь $A1$—значение параметра цикла v, которое он принимает при каждом повторении цикла, а $Л$ — логическое выражение, которое проверяется перед каждым новым повторением цикла (т. е. перед выполнением оператора S). Пока это выражение является истинным, цикл повторяется; как только $Л$ становится ложным, повторение цикла прекращается.

Как видно из общего синтаксического определения списка цикла, в этом списке допускается участие элементов различных типов, расположенных в различном порядке. Например.

для v: — $A1$ **шаг** $A2$ **до** $A3$, $A4$, $A5$, $A6$ **пока** $Л$ **цикл** S**;**

Здесь сначала будет выполняться оператор цикла с элементом списка цикла типа арифметической прогрессии, затем два раза повторится цикл со значениями параметра цикла *v,* равными *A*4 и *A*6*,* а затем будет выполняться цикл с элементом списка цикла типа пересчета.

В отношении оператора цикла необходимо сделать следующие замечания.

Переходы извне к оператору *S,* который должен выполняться в цикле, или к операторам, входящим в его состав, правилами АЛГОЛа не допускаются.

Выходы из этого оператора могут быть двух видов:

а) естественный выход, происходящий при выполнении числа повторений цикла, заданного всем списком цикла; при таком выходе значение параметра цикла и считается неопределенным, т. е. эту величину нельзя использовать в операторах, которые будут выполняться вслед за оператором цикла;

б) «досрочный» выход, происходящий при наличии в составе оператора *S*, выполняемого в цикле, оператора перехода, ведущего к какому-нибудь оператору, находящемуся вне данного цикла. При таком («досрочном») выходе из цикла считается, что параметр цикла *v* сохраняет то значение, которое он имел непосредственно перед выходом из цикла, и эту величину можно таким образом использовать в последующих вычислениях (например, для того чтобы определить, сколько раз был повторен данный цикл).

В качестве примера программы с оператором цикла рассмотрим задачу о вычислении пяти значений полинома:

$$y_i = \sum_{i=0}^{n} a_i x_j^i \quad \text{для } j = 1, 2, 3, 4, 5.$$

Вычисления ведутся по схеме Горнера:

начало
для *j*: = 1, 2, 3, 4, 5 **цикл**
начало *y*[*j*]: = 0*;*
для *i*: = *n* **шаг** — 1 до 0 **цикл**

y[*j*]: = y[*j*] × x[*j*] + a[*i*]**;**

конец
конец

Пять значений $x_j (j = 1, 2, 3, 4, 5)$ считаются заданными. Они могут быть расположены в последовательных ячейках памяти машины, так же как и величины коэффициентов $a_n, a_{n-1}, \ldots, a_0$ и результаты вычислений y_j (j=1, 2, 3, 4, 5).

Другой вариант этой программы:

начало
для *j*: = 1, 2, 3, 4, 5 **цикл**
начало
для *i*: = n, *i* —1 **пока** $i \geq 0$ **цикл**

$$y[j]: = y[j] \times x[j] + a[i]$$

конец
конец

В АЛГОЛе допускается возможность изменения в процессе выполнения оператора *S*, входящего в цикл, величин, входящих в заголовок цикла (шага цикла, конечного значения параметра цикла, переменных, образующих логическое выражение). Это свойство оператора цикла АЛГОЛа называется динамической интерпретацией оператора цикла.

Например:

начало

$$h:=10;\ i:=1;$$

для *v*: = 0 **шаг** *h* **до** 100 **цикл**
начало

$$h: = h + 10;\ \ v: = v \times 2 + h;$$
$$y[i]: = h \times v;$$
$$i:= i + 1$$

конец
конец

В этом примере параметр цикла *v* меняется не только по правилу арифметической прогрессии, определяемому заголовком цикла, но и самим оператором, выполняемым в цикле так же, как и шаг *h.* В результате получим всего два значения величины *y:*

$$y[1] = 400,\ \ y[2] = 3300.$$

При третьем заходе (i = 3) параметр *v* уже примет значение 110+30=140, т. е. будет больше 100 и третий раз цикл выполняться не будет.

Рассмотрим примеры выхода из цикла.

Пусть требуется для заданной последовательности положительных чисел *a*1*, a.2, . .., an* определить номер первого числа *i,* находящегося в заданных пределах: $p \leq a_i \leq q$. Это может быть сделано с помощью следующего оператора цикла:

для *i:* == 1, *i* + 1 **пока** $i \leq n$ **цикл**

если $\neg(a[i] \leq q \wedge a[i] \geq p)$ **то иначе перейти к** *M;*

В этом примере возможны два случая выхода из цикла:

а) при исчерпании всех элементов ряда чисел, т. е. при $i > n$. После выхода из цикла следующим будет выполняться оператор, стоящий за этим оператором цикла, значение параметра цикла *i* будет неопределенным, а

искомый член ряда не будет найден (так как ни один член ряда не удовлетворяет заданному условию

При этом в цикле производится проверка условия $\neg(a[i] \le q \wedge a[i] \ge p)$ для каждого члена ряда $a[i]$; так как это условие выполняется, то каждый раз выполняется пустой оператор, стоящий за символом **то**;

б) второй случай выхода будет иметь место при обнаружении члена ряда $a[i]$, удовлетворяющего заданному требованию. При этом будет выполнен оператор перехода, стоящий за символом **иначе**. Метка *M* указывает некоторый оператор, который должен выполняться в случае обнаружения члена ряда, удовлетворяющего заданному требованию. Такой выход из цикла будет являться «досрочным» выходом, даже если он произошел после проверки последнего члена ряда ($i=n$), так как условие окончания цикла, требуемое заголовком цикла ($i>n$), еще не будет соблюдено. При этом параметр цикла *i* сохранит свое последнее значение, т. е. он укажет номер члена ряда, удовлетворяющего заданному условию.

Оператор процедуры

В разделе, посвященном величинам языка АЛГОЛ-60, мы рассматривали процедуры-функции: там было дано их синтаксическое определение и указаны их отличия от операторов процедур.

Основное отличие заключается в характере использования: оператор процедуры всегда фигурирует в программе в виде отдельного оператора (т. е. он может иметь метки, после него ставится точка с запятой), а процедура-функция — в виде так называемого указателя функции, который используется при построении выражений. Описания процедур-функций и операторов процедур и обращения к ним строятся по одним и тем же правилам, которые будут рассмотрены ниже.

Отличие сводится к тому, что в описании процедуры функции указывается тип величины, который должен иметь результат вычисления функции.

Синтаксическое определение оператора процедуры имеет следующий вид:

<фактический параметр> : : = <строка> | <выражение> | <идентификатор массива> | < идентификатор переключателя> | <идентификатор процедуры>

<строка букв> : : = <буква> | <строка букв> <буква>

<ограничитель параметра> : : =,|) <строка букв>:(

<список фактических параметров> : : = <фактический параметр> | <список фактических параметров> <ограничитель параметра> <фактический параметр>

<совокупность фактических параметров> : : = <пусто> |(<список фактических параметров>)

<оператор процедуры> : : = <идентификатор процедуры> <совокупность фактических параметров>

Обращение к процедуре либо с помощью указателя функции, либо с помощью оператора процедуры задает конкретные исходные данные в виде так называемых фактических параметров для выполнения стандартного участка алгоритма, определенного описанием процедуры. Эти фактические параметры представляются перечнем величин в круглых скобках после идентификатора процедуры. Один фактический параметр от другого отделяется ограничителем параметра либо в виде запятой, либо в виде последовательности символов, состоящей из круглой закрывающей скобки, буквенного пояснения с двоеточием и открывающей круглой скобки.

Фактическими параметрами могут быть числа и переменные, которые являются частными случаями выражений. Следует заметить, что идентификаторы массивов, процедур и переключателей, являющиеся фактическими параметрами, указываются в списке фактических параметров без сопровождающих их дополнительных выражений (индексов, именующих выражений или параметров). Фактические параметры должны быть описаны в том блоке, в котором находится оператор процедуры или указатель функции или во внешнем блоке (см. ниже). При их включении в состав фактических параметров оператора некоторой процедуры использование соответствующих величин внутри тела процедуры производится в соответствии с этими описаниями.

При этом, например, использование в качестве фактического параметра идентификатора некоторой процедуры приводит к тому, что внутри первой процедуры будет использоваться вторая процедура и, естественно, что для этого в первой процедуре должна быть предусмотрена подготовка всех фактических параметров для второй процедуры. Введение в состав языка такого средства как процедуры сильно усложняет язык, но в то же время придает ему большие возможности для представления различных процессов обработки информации и, главным образом, обеспечивает возможность использования предшествующего опыта при программировании новых задач. Если накоплен достаточно большой запас различных процедур, обеспечивающих выполнение типовых вычислительных процессов, то программирование новых задач будет в значительной степени сводиться к компоновке подходящих процедур в общую программу с добавлением недостающих участков. При этом сокращается время программирования и отладки программ.

Заметим, что более полное понимание процедур и особенностей их использования можно получить после ознакомления с правилами построения описаний процедур, которые изложены в следующем параграфе.

§ 9 ОПИСАНИЯ

Описания являются важной неотъемлемой частью каждой программы; они сообщают транслятору (переводящей программе) необходимые сведения для построения машинной программы: для распределения памяти, выполнения округлений и др. Описания следует отличать от примечаний, которые могут появляться в АЛГОЛ-программах. Примечания служат только для пояснения отдельных мест программы человеку, читающему эту программу. Описания же воспринимаются транслятором, и поэтому они должны быть написаны в строгом соответствии с синтаксисом АЛГОЛа. Описания ставятся в начале блока, к которому они относятся, и оформляются в виде предложений, заканчивающихся точкой с запятой. Описания имеют силу (т. е. область действия) только внутри этого блока. Расположение различных описаний в начале блока может быть произвольным.

Существует четыре вида описаний: типа, массивов, переключателей и процедур. Описания переключателей были приведены нами раньше при рассмотрении именующих выражений. Рассмотрим описания типа и массивов, а затем описания процедур.

Описания типа и массивов

Описания типа служат для указания типа простых переменных и переменных, являющихся элементами массивов. В описания простых переменных могут входить только описания типа, в описания массивов входят, кроме того, сведения о размерности массива и пределах изменения индексов переменных. В АЛГОЛе предусматривается три типа переменных: **целый, вещественный, логический**.

Каждый из этих типов переменных может быть локальным или собственным. Часто используют еще термин глобальные переменные как противоположность локальным, но этот термин определяет не тип переменных, а только степень их локальности. Переменные, являющиеся локальными в одном блоке, могут быть глобальными для другого блока. Подробнее этот вопрос будет рассмотрен несколько позже. Приведем синтаксическое определение описания типа:

<список типа> : : = <простая переменная> | <простая переменная>, <список типа>
<тип> : : = **целый** | **вещественный** | **логический**
<локальный или собственный тип> : : = <тип> | **собственный** <тип>
<описание типа> : := <локальный или собственный тип> <список типа>

Если переменная, имеющая по описанию тип **целый**, получает в процессе вычислений нецелое значение, то при выполнении присваивания будет произведено ее округление до целого значения по правилу

$$a := \text{entier}\ (A + 0{,}5);$$

где A — фактическое значение переменной, а функция entier — наибольшее целое, не превышающее значения аргумента, стоящего в круглых скобках.

Примеры описаний типа:

целый *i, j, k;* **вещественный** *x, y;*
логический ОМЕГА, ПРИЗНАК СРОЧНОСТИ;

Переменные, имеющие вещественный тип, могут принимать любые вещественные значения (в пределах разрядной сетки конкретной машины), а переменные логического типа могут быть представлены в машине одноразрядными двоичными кодами.

Описания массивов

Для описания массивов вводится вспомогательное понятие — сегмент массива. Сегмент массива — это группа однотипных массивов, отличающихся только своими идентификаторами; эти массивы должны иметь одинаковый локальный или собственный тип переменных, одинаковую размерность и одинаковые пределы изменения индексов.

Приведем синтаксическое определение описания массивов:

<нижняя граница> : : = <арифметическое выражение>
<верхняя граница> : : = <арифметическое выражение>
<граничная пара> : : = <нижняя граница> : <верхняя граница>
<список граничных пар> : : = <граничная пара> | <список граничных пар>, <граничная пара>
<сегмент массива> : : = <идентификатор массива> [<список граничных пар>] | <идентификатор массива >, <сегмент массива >
<список массивов> : : = <сегмент массива> | <список массивов>, <сегмент массива>
<описание массива> : : = **массив** <список массивов> | <локальный или собственный тип> **массив** <список массивов>

Размерность массива определяется числом граничных пар в списке.

Пример сегмента массива:

массив *A, B, C, D* [*a*1 : *b*1 , *a*2 : *b*2, *a*3 : *b*3]

A, B, C, D — идентификаторы трехмерных массивов; в списке граничных пар имеется три пары;
*a*1, *a*2, *a*3 — нижние границы;
*b*1, *b*2, *b*3 — верхние границы.

Таким образом, в одно описание массивов может входить несколько сегментов массивов, т. е. несколько групп массивов, каждая из которых имеет одинаковые списки граничных пар. Если в блоке используется несколько массивов различных типов, то должны писаться соответственно несколько описаний массивов. Описание массива может не содержать описателя типа; в этом случае считается, что массив имеет вещественный тип.

Например, **массив** *A, B, C* [1 : *a*], *x* [*b* : *c*];

Здесь указаны два сегмента массивов, в первом имеется три массива *A, B, C,* а во втором — один массив *x*.

Все массивы одномерные и имеют тип вещественный. Примеры описаний массивов с явным указанием типов:

собственный целый массив *z*, *x*[0 **:** 100, *a* **:** *b*], *y*[1 **:** 50**];**

вещественный массив *A* [0 **: если** $i \le n$ **то** *p* **иначе** $2 \times$ q];

Арифметические выражения, представляющие собой границы изменения индексов, по своей природе должны быть целыми.

Индексы переменных могут принимать все целые значения, начиная с нижней границы и кончая верхней. Если при вычислении границ получаются нецелые числа, то они должны округляться до ближайшего целого значения.

Все переменные или функции, входящие в арифметические выражения, представляющие границы индексов, должны быть описаны в одном из внешних блоков, в который входит блок, содержащий данное описание

массивов. При каждом входе в этот внутренний блок заново происходит определение границ массивов, и эти границы остаются неизменными в процессе выполнения данного внутреннего блока. При этом может быть случай, когда границы зависят от переменных, описанных во внешнем блоке, но являющихся глобальными для внутреннего блока (т. е. они используются и во внутреннем блоке). При работе внутреннего блока эти переменные, от которых зависят границы массивов, могут менять свое значение, но при этом границы массивов будут сохранять то значение, которое они имели при входе в этот внутренний блок.

Отсюда следует, что в самом внешнем блоке, представляющем собой всю программу, могут содержаться описания массивов, имеющих только постоянные (числовые) границы.

Нижняя граница всегда должна быть не больше соответствующей верхней границы, а значения индексов переменных не должны выходить за пределы границ. При невыполнении этих условий значение переменной с индексами считается неопределенным.

Области действий описаний типов и массивов в программах

Основой АЛГОЛа является блочный принцип построения программ, при котором отдельные ее участки, так называемые блоки, могут строиться одновременно и независимо разными лицами, что очень удобно при работе над сложными задачами. Это достигается строгим определением областей действий различных описаний или, как говорят, локализацией описаний. Все переменные, не имеющие в своем описании символа **собственный,** считаются локальными по отношению к тому блоку, в котором они описаны. Следует подчеркнуть, что все переменные, используемые в данном блоке, должны быть обязательно описаны в нем самом или в каком-нибудь внешнем блоке. Требуется также обязательно описывать и другие объекты, используемые в блоке (процедуры, переключатели), кроме стандартных функций (sin, cos, arctan, ln, exp, abs, sign, sqrt, entier). Смысл локализации переменных заключается в том, что такие переменные могут использоваться только в пределах того блока, в котором они описаны (за исключением, может быть, некоторых внутренних блоков, входящих в данный блок, о чем подробнее будет сказано ниже). При выходе из данного блока эти переменные теряют свое значение, и те же идентификаторы переменных (простых или с индексами) могут быть использованы для других целей (обозначения других переменных, меток, процедур, переключателей). Если же в своем описании некоторые переменные имеют символ **собственный,** то они также могут быть использованы только в пределах своего блока, но при выходе из данного блока сохраняют свое значение. При повторном обращении к этому блоку собственные переменные принимают то значение, которое они имели при выходе из этого блока. В отличие от этого локальные переменные при первоначальном или повторном входе в блок не имеют никаких значений и, перед тем как их использовать в этом блоке, им нужно присваивать определенные значения.

Отдельно нужно остановиться па том случае, когда в качестве собственных описаны массивы с переменными границами. Эти границы, как уже говорилось, могут зависеть только от переменных, описанных в некотором внешнем блоке по отношению к тому блоку, в котором описаны данные массивы. При каждом новом входе в блок границы собственных массивов могут изменяться операторами внешнего блока, поэтому значения элементов таких массивов сохраняются только в пределах совпадения границ массивов, которые они имели при последнем выходе из блока, с границами массивов, получающимися при данном входе в блок. Если массив расширит свои границы, то новые его области будут неопределенными; если массив уменьшит свои границы, то элементы отпавших областей будут утрачены.

Рассмотрим теперь понятие глобальности переменных. Если необходимо, чтобы некоторые переменные сохраняли свое значение в пределах некоторого блока, включая и все внутренние блоки, входящие в его состав, то необходимо такие переменные описать в самом внешнем блоке и не описывать их во внутренних блоках. Такие переменные будут называться глобальными по отношению к внутренним блокам, ими можно пользоваться одинаково в пределах всего внешнего блока, включая и те внутренние блоки, в которых эти переменные не описаны. Как уже говорилось, переменным, описанным в данном блоке и не являющимся собственными, перед их использованием в данном блоке для вычислений необходимо произвести присваивание определенных значений, так как при входе в этот блок их значения являются неопределенными.

Особенностью использования собственных переменных является то, что в том блоке, где они описаны, должны быть операторы, присваивающие им первоначальное значение при первом входе в данный блок. Эти операторы не должны выполняться при повторных входах в данный блок. При повторных входах собственные переменные будут сохранять то значение, которое они имели в момент предыдущего выхода из этого блока.

Как мы уже говорили, сохранить значения некоторых переменных, используемых во внутреннем блоке, при выходе из этого блока можно, описав эти переменные во внешнем блоке и не описывая во внутреннем. Однако между этим способом и способом, основанным на использовании описателя **собственный**, имеется следующее различие. При первом способе указанные переменные будут доступными для всех других блоков, входящих в состав внешнего блока, в которых эти переменные не описаны; при втором способе эти переменные будут недоступны никому, кроме операторов своего блока, даже если те же идентификаторы будут использованы в других блоках и описаны во внешнем блоке. Таким образом, можно обойтись без собственных переменных, но это потребует некоторого согласования использования общих переменных. Если же какой-то соседний блок (т. е. блок, входящий в состав того же внешнего блока) будет иметь описанными те же переменные, которые описаны во внешнем блоке, то внешние значения этих переменных будут недоступны операторам внутреннего блока, а внутренние переменные будут недоступны операторам внешнего блока.

Таким образом, повторное описание одних и тех же переменных во внутреннем и внешнем блоках устанавливает полное различие между внешним и внутренним использованием этих переменных так, как если бы это были совершенно разные переменные.

Локализация меток

Метки в АЛГОЛ-программах специально не описываются, а вводятся в действие при появлении. Областью действия метки является та совокупность операторов перехода, которые могут вести к оператору с данной меткой. Считается, что областью действия метки является тот внешний блок, в котором такая метка является

единственной. Если одна и та же метка используется и во внутреннем и во внешнем блоках, то операторы перехода, находящиеся во внутреннем блоке, будут вести к внутренней метке, а во внешнем блоке — к внешней; при этом невозможно будет осуществить переход к оператору во внешнем блоке, имеющем эту метку, из внутренних блоков, имеющих внутри себя оператор с такой же меткой. Как уже говорилось, в АЛГОЛе не допускаются переходы извне к отдельным операторам, входящим в состав блока. Переходы могут быть только в начало блока. Выходы же из внутренних блоков к другим операторам, входящим в состав одного и того же внешнего оператора, допустимы, если при этом выполняются указанные выше правила локализации меток.

Описания процедур

Структура описаний процедур. Каждая процедура, используемая в блоке, должна иметь в начале этого блока или какого-нибудь внешнего блока, включающего данный блок, описание, которое определяет порядок входа в процедуру и действия, выполняемые процедурой. Описания процедур являются единственным видом описаний, в которых используются операторы, образующие так называемое тело процедуры.

Синтаксис описания процедуры имеет вид:

<формальный параметр> : : = <идентификатор>

<список формальных параметров> : : = <формальный параметр> | <список формальных параметров> <ограничитель параметра> <формальный параметр>

<совокупность формальных параметров> : := <пусто> | (<список формальных параметров>)

<список идентификаторов> : : = <идентификатор> | <список идентификаторов>, <идентификатор>

<список значений> : : = **значение** <список идентификаторов>; | < пусто >

<спецификация> : : = **строка** | <тип> | **массив** | **<**тип**> массив** | **метка** | **переключатель j процедура** | <тип> **процедура**

<совокупность спецификаций> : : = <пусто> | <спецификация> <список идентификаторов>; | <совокупность спецификаций> <спецификация> <список идентификаторов>;

<заголовок процедуры> : : = <идентификатор процедуры> <совокупность формальных параметров>; <список зиачеиий> <совокупность спецификаций>

<тело процедуры> : : = <оператор> | <код>

<описание процедуры> : : = **процедура** <заголовок процедуры> <тело процедуры> | <тип> **процедура** <заголовок процедуры> <тело процедуры>

Операторы, входящие в состав тела процедуры, указывают те действия, которые должны выполняться процедурой.

В АЛГОЛе предусматриваются две возможности представления тела процедуры: запись необходимых операторов на языке АЛГОЛ или представление тела процедуры на каком-то другом языке, например, в виде машинной программы для конкретной машины. Последнее представление называется представлением процедуры в виде кода. Таким образом, понятие «код» является не определяемой на АЛГОЛе синтаксической единицей.

Типичными примерами процедур, задаваемых сразу в машинном коде, являются процедуры ввода и вывода данных, обмена между запоминающими устройствами. Определения таких процедур-кодов будут даны ниже при изложении АЛГЭМа.

Список значений и совокупность спецификаций не являются обязательными частями всех заголовков процедур. Для многих процедур они могут отсутствовать.

В качестве формальных параметров могут фигурировать только идентификаторы, а в качестве фактических параметров, указываемых в обращениях к процедурам, могут использоваться также выражения и строки.

Количество фактических параметров в обращении к процедуре должно быть равно числу формальных параметров в заголовке процедуры. Соответствие формальных и фактических параметров устанавливается путем их перечисления слева направо.

Формальные параметры, перечисленные в заголовке, являются обозначениями переменных или других объектов (переключателей или процедур), которые используются в операторах тела процедуры. При обращении к процедуре конкретный смысл каждого формального параметра задается соответствующим ему фактическим параметром. При этом возможны два способа конкретизации формальных параметров:

а) каждому формальному параметру присваивается численное значение соответствующего ему фактического параметра, которое вычислено к моменту обращения к процедуре. Этот способ называется конкретизацией формальных параметров значением (вызов по значению);

б) каждый формальный параметр заменяется в теле процедуры соответствующим ему фактическим параметром. При этом (так как фактические параметры могут быть сложными выражениями) там, где возможно и необходимо, фактические параметры заключаются в скобки. Такой способ называется конкретизацией формальных параметров наименованием (вызов по наименованию).

Для одной и той же процедуры возможно применение сразу обоих способов, при этом часть формальных параметров конкретизируется значением, а часть — наименованием. Для указания того, какие формальные параметры должны конкретизироваться значением, в заголовке процедуры пишется спецификатор **значение**, после которого перечисляются соответствующие формальные параметры. Этот перечень называется списком значений:

<список значений> : : = **значение** <список идентификаторов >; | <пусто>

<список идентификаторов> :: = <идентификатор> | <список идентификаторов>, <идентификатор>

Для всех формальных параметров, перечисленных в списке значений, обязательно должны быть заданы спецификации, т. е. указаны виды и типы объектов, которые представляются идентификаторами. Эти спецификации образуют четвертую составную часть заголовка процедуры, называемую совокупностью

спецификаций:

<совокупность спецификаций> : : = <пусто> | <спецификация> <список идентификаторов>; | <совокупность спецификаций> < спецификация> <список идентификаторов >;

<спецификация> : : = **строка** | **метка** | **переключатель** | **массив** | **процедура** | <тип> | <тип> **массив** | <тип> **процедура**

Последний вид спецификаций используется для процедур-функций. Кроме обязательного указания для формальных параметров, перечисленных в списке значений, спецификации могут указываться и для других формальных параметров, не перечисленных в списке значений. Это бывает необходимо для упрощения автоматического программирования процедуры транслятором. Ясно, что программирование процедур должно быть сделано независимо от вида конкретного обращения к ней. Вообще говоря, обращение к процедурам содержит дополнительную информацию о характере объектов, с которыми должна работать процедура. Такой информацией являются описания, относящиеся к фактическим параметрам, указанным в обращениях. Эти описания находятся либо в том блоке, из которого производится обращение к процедуре, либо в одном из внешних блоков. Так как процедуры рассчитываются па использование их в разных местах программы (в разных блоках), то в своем описании они должны содержать сведения, накладывающие необходимые ограничения на характер обращений к ним и обеспечивающие возможность программирования процедуры независимо от остальных частей программы. Для этой цели и служат спецификации, которые накладывают необходимые ограничения на фактические параметры и тем самым обеспечивают общность в построении и использовании данной процедуры.

Локализация величин в процедурах. Важной функцией списка значений является локализация перечисленных в нем величин в теле процедуры. При этом процедуру следует рассматривать как блок, хотя внешне она не оформляется в виде блока. Все формальные параметры, перечисленные в списке значений, являются локальными по отношению к телу данной процедуры. Это значит, что при входе в процедуру они получают определенные значения. В процессе выполнения процедуры эти величины могут изменяться операторами, входящими в тело процедуры, но соответствующие им величины, находящиеся вне процедуры, будут оставаться неизменными, т. е. будут сохранять те значения, которые они имели в момент входа **в** процедуру. При выходе из процедуры значения локальных объектов будут теряться, поэтому результаты работы процедуры нельзя перечислять в списке значений.

Формальные параметры, конкретизируемые наименованием, не являются локальными по отношению к данной процедуре. Если при выполнении процедуры соответствующие им фактические параметры изменяют свое значение, то изменения распространяются на весь блок, в котором описаны эти фактические параметры. При выходе из процедуры такие объекты сохраняют то значение, которое они получили в процедуре. Кроме объектов, перечисленных в списке формальных параметров, в теле процедуры могут использоваться различные объекты без указания их в списке формальных параметров и, естественно, без спецификации. Такими объектами могут быть простые переменные, массивы, переключатели, процедуры. Все эти объекты можно разделить на две группы: а) объекты, описанные в блоке, являющемся телом процедуры. Здесь мы уже имеем в виду не воображаемый, а явный блок, оформленный по всем правилам АЛГОЛа. Такие объекты, естественно, будут локальными для этого блока; б) объекты, не описанные в блоке, являющемся телом процедуры, и не являющиеся формальными параметрами. Такие глобальные по отношению к процедуре объекты должны быть обязательно описаны в каком-нибудь внешнем блоке, включающем в себя блок, содержащий описание данной процедуры, или в самом блоке, содержащем описание процедуры, т. е. область действия описаний таких объектов должна обязательно распространяться на блок, содержащий описание данной процедуры.

Отсюда вытекает, что все описания типа, массивов, переключателей и процедур, содержащиеся в начале данного блока, могут использоваться любой процедурой, описанной в этом блоке.

Заметим, что спецификатор **собственный** может использоваться в теле процедуры, если эта процедура оформлена в виде блока; при этом он действует так же, как если бы он использовался в обычном блоке программы.

Рассмотрим пример процедуры с локализацией переменных перечисленных в списке значений:

начало вещественные *m, x, y, z, p, q*;
вещественная процедура *П* (*a, z, Q*);
значение *a, z*; **вещественное** *a, z, Q*;
П: = $(a + y \uparrow m) \times Q / (z + y)$
.....
m: = 3; *p*: = 100; *q*: = 10;
x: = 5; *y*: = 15; *z*: = 20;
……
R: = *П* (*x, p, q*) + $z \uparrow m$;
……
конец

Заметим, что величина *R* должна быть описана в каком-нибудь внешнем блоке. Выполнение последнего оператора присваивания, использующего в правой части процедуру-функцию *П*, будет эквивалентно выполнению следующего составного оператора:

начало начало вещественное *a, z*;
a: = 5; *z*: = 100; *П*:= $(a + y \uparrow m) \times q/(z+y)$
конец; *R: = П* + $z \uparrow m$ **конец**

В теле процедуры *П* используются переменные *a, z, Q,* являющиеся формальными параметрами, и

переменные *m* и *y,* являющиеся глобальными переменными по отношению к этой процедуре. Кроме того, в блоке, содержащем эту процедуру, описана переменная *z,* которая совпадает по обозначению с одним из формальных параметров. Два формальных параметра *a* и *z* конкретизируются значением, а один *Q* конкретизируется наименованием.

При выходе из процедуры переменная *z* получит прежнее значение, а переменная *a* будет иметь неопределенное значение. Переменная *z,* используемая в теле процедуры, будет иметь значение фактического параметра *p* = 100, а переменная *z,* используемая в операторе присваивания *R*: = ..., будет иметь то значение, которое ей было присвоено перед выполнением данного оператора присваивания, т. е. z = 20.

В соответствии с правилами локализации действий описаний в блоках все описания, помещенные во внутренних блоках, не оказывают влияния на внешние блоки и поэтому нельзя обратиться из внешнего блока к процедуре, описанной во внутреннем блоке. Наоборот, из внутренних блоков в общем случае возможны обращения к процедурам, описанным во внешних блоках. При этом если в процедуре используются глобальные величины, не являющиеся формальными параметрами, но совпадающие по виду с величинами, описанными в том внутреннем блоке, из которого совершается обращение к процедуре, то при выполнении процедуры эти величины принимают те значения и смысл, которые они имели во внешнем блоке, содержащем описание процедуры. Следует подчеркнуть, что это правило относится только к нелокальным для процедуры величинам, не являющимся формальными параметрами. Что же касается формальных параметров, то во всех случаях обращения к процедурам, в том числе из внутренних блоков, конкретизация формальных параметров фактическими происходит с использованием описаний того блока, из которого произведено обращение, или тех описаний внешних блоков, действие которых распространяется на блок, содержащий обращение к процедуре. Таким образом, обращение к процедуре как бы несет с собой всю необходимую информацию о параметрах для работы процедуры, взятую из того блока, где помещается это обращение.

Рассмотрим пример некоторой процедуры, описанной во внешнем блоке и используемой во внутреннем блоке.

A:**начало вещестзенные** *b, x, y, z;*

вещественная процедура *p* (*a*);

значение *a*; **вещественное** *a*;

начало *x*: **=** $a \times x$**;** $p := x \mid 2 + y$ **конец**;

……

x: == 15; *b*: = 50; *y*: = 10;

……

B: **начало вещественные** *x, b*;

.....

x: = 100; *b:* = 10;

.....

$z = p\ (b) + y$;

……..

конец.....конец

Выполнение процедуры будет эквивалентно выполнению блока:

начало вещественное *a*;

a: = 10; *x*: = a × 15; $p := x \uparrow 2 + 1\,0$

конец

При выполнении операторов тела процедуры использовано значение величины *b* =. 10 (являющейся фактическим параметром), полученное во внутреннем блоке, имеющем метку *B.* Такая же величина *b,* описанная во внешнем блоке, имеет значение *b* = 50. В то же время глобальная переменная *x* в теле процедуры используется со значением *x* = 15, полученным во внешнем блоке, хотя такая же переменная *x,* описанная во внутреннем блоке, имеет значение 100.

Порядок выполнения процедур. При реализации в программе какого-нибудь обращения к процедуре, описанной на АЛГОЛе, с помощью оператора процедуры или указателя функции выполняются следующие действия:

а) формальные параметры, перечисленные в списке значений, конкретизируются значением, т. е. им присваиваются значения соответствующих фактических параметров;

б) остальные формальные параметры, не перечисленные в списке значений, конкретизируются наименованием, т. е. везде в теле процедуры вместо них подставляются выражения или идентификаторы соответствующих фактических параметров;

в) выполняется оператор, составляющий тело процедуры, так как если бы он находился *в* блоке, в котором находится описание данной процедуры, и к нему был бы совершен переход при помощи оператора перехода из того места, где находится оператор процедуры или указатель функции;

г) если в теле процедуры не содержится оператор перехода, выводящий из этой процедуры в какое-то другое место программы, то после окончания процедуры выполняются очередные действия программы, т. е. операторы, следующие за данным оператором процедуры или указателем функции.

Роль спецификаций и списка значений

Спецификации в описании процедуры накладывают определенные ограничения на фактические параметры.

Кроме того, сам характер использования различных переменных и других объектов языка в теле процедуры также предъявляет определенные требования к фактическим параметрам, которые могут задаваться для данной процедуры. Обе группы ограничений играют в общем одну и ту же роль. Можно сказать, что ограничения, связанные с характером использования объектов в теле процедуры и характером обращений, являются более полными, чем ограничения, задаваемые в явном виде спецификациями, так как спецификации не охватывают всех возможных случаев использования объектов. Поэтому всегда при составлении или использовании процедур нужно иметь в виду содержательную сторону дела и, используя общие принципы локализации объектов и конкретизации параметров, не допускать не предусмотренных вариантов работы процедур.

Однако применение спецификаций бывает полезно и необходимо с точки зрения упрощения работы транслятора, так как спецификации в явном виде указывают типы и виды объектов, допустимых в качестве фактических параметров в процедуре. К числу основных ограничений, задаваемых спецификациями, относятся:

1. Указание типа (вещественный, целый, логический) определяет, что для этого формального параметра фактический параметр должен быть выражением указанного типа. При конкретизации значением должно производиться (в случае различия в типах) преобразование типа к тому, который задан спецификациями.

2. Формальному параметру, конкретизируемому наименованием и используемому в теле процедуры в качестве левой части оператора присваивания, может соответствовать в качестве фактического параметра только переменная (а не более сложное выражение, так как операторы вида *x+y*: = *a +b;* в АЛГОЛе недопустимы).

3. Формальным параметрам, специфицированным как массивы, должны соответствовать фактические параметры — массивы того же типа, что и формальные параметры. При конкретизации значения также может происходить преобразование типов (целый в вещественный или наоборот). Если этот массив описан в блоке тела процедуры и имеет фиксированную размерность, то соответствующий фактический параметр-массив должен иметь ту же размерность. При конкретизации значением формальный параметр-массив принимает значения элементов и границ индексов соответствующего фактического параметра-массива.

4. Формальному параметру, имеющему спецификацию **метка** или используемому в качестве метки в теле процедуры, должно соответствовать в качестве фактического параметра именующее выражение.

5. Спецификация строка в АЛГОЛе указывает, что данный формальный параметр может иметь фактическим параметром строку; по правилам АЛГОЛа такие формальные параметры могут использоваться только в процедурах-кодах, т. е. написанных не на АЛГОЛе.

Рассмотрим роль списка значений. Указание списка значений позволяет использовать в процедуре в качестве исходных данных значения любых переменных или выражений, имеющихся в основной программе, сохраняя их в то же время в основной программе неизменными. Кроме того, используя конкретизацию значением, можно в качестве фактических параметров применять любые выражения, в том числе и для формальных параметров, фигурирующих в теле процедуры в качестве левых частей операторов (хотя последнее вряд ли имеет практический смысл). Ясно, что конкретизация значением неприменима к формальным 'параметрам, играющим роль идентификаторов процедур и переключателей. Формальные параметры, играющие роль именующих выражений, могут конкретизироваться как наименованием, так и значением, причем эти параметры могут использоваться для указания переходов из тела процедуры в основную программу, и в этом случае формальный параметр, конкретизированный значением, будет иметь смысл вне тела процедуры, т. е. в отличие от общего правила не будет локализован в теле процедуры.

Пример процедуры

Определение суммы элементов, находящихся на главной диагонали квадратной матрицы (вычисление следа матрицы).

Описание процедуры:

процедура СЛЕД (*a, n, S*); **значение** *n*;
массив *a*; **вещественное** *S*; **целое** *n;*
начало целый *k; S*: = 0;
для *k:* =1 **шаг** 1 **до** *n* **цикл** *S*: = *S*+a [*k*, *k*]
конец

Оператор процедуры для обращения к этой процедуре может иметь, например, вид, показанный в следующем фрагменте программы:

начало вещественное *z;* **массив** *A* [1 : 20, 1 : 20]; **целый** *b;*
b : = 20, СЛЕД (*A*, *b, z*)*;* ... **конец**

Выполнение оператора процедуры СЛЕД (*A*, *b, z*)*;* может быть представлено в виде блока:

начало целое *n*:
n : = *b*;
начало целое *k;*
z: = 0;
для *k* : = 1 **шаг** 1 **до** *n* **цикл**

$$z: = z+A[k, k]$$

конец
конец

Переменная *k* локализована описанием в теле блока процедуры. Переменная *n* локализована указанием в списке значений. Формальные параметры *a* и *S* конкретизируются наименованием, и результат процедуры сохраняется в виде значения переменной *z.*

В заключение настоящего раздела, посвященного описаниям процедур, следует сделать следующие

замечания.

Во-первых, при выполнении процедур, использующих нелокальные переменные, может происходить иногда присваивание новых значений таким переменным, даже если они не являются результатами процедуры. Это явление носит название побочного эффекта процедуры и его нужно учитывать при пользовании процедурами.

Во-вторых, правилами АЛГОЛа допускается таксе использование процедур, когда в процессе выполнения одной процедуры (до ее окончания) происходит вновь обращение к этой же процедуре либо непосредственно, либо через какую-нибудь другую процедуру. Такие многократно вызываемые процедуры называются рекурсивными. Их применение позволяет в ряде случаев весьма компактно записывать алгоритмы, носящие рекурсивный характер. Однако трансляция таких процедур связана с дополнительными трудностями и во многих конкретных вариантах языка АЛГОЛ от применения рекурсивных процедур отказываются.

4. АЛГОРИТМИЧЕСКИЙ ЯЗЫК АЛГЭМ ДЛЯ ПРОГРАММИРОВАНИЯ ЭКОНОМИЧЕСКИХ И МАТЕМАТИЧЕСКИХ ЗАДАЧ

В настоящей главе излагаются добавления к АЛГОЛу, составляющие вместе с АЛГОЛом содержание алгоритмического языка для программирования экономических и математических задач (АЛГЭМ). К ним относятся:

а) введение строчных переменных и выражений;
б) введение средств для описания вида переменных и указания разрядов;
в) введение составных переменных и составных массивов.

Эти добавления связаны со спецификой решения экономических задач, которые характеризуются достаточно сложной организацией данных (построение их в виде таблиц, ведомостей, отчетных форм и т. п.), а также большим удельным весом логических операций (проверка различных признаков, выделение знака числа и т. п.).

Заметим, что АЛГЭМ может быть применен не только при решении математических и экономических задач, но и при решении различных других информационно-логических задач, связанных с обработкой больших массивов информации, имеющей сложную, но полностью детерминированную структуру (обработка историй болезней в клиниках, поиск научной информации, обработка результатов экспериментов и т. п.).

§ 10 СТРОЧНЫЕ ВЫРАЖЕНИЯ. ВИД ПЕРЕМЕННЫХ

В разделе, посвященном строкам, дается определение строки в АЛГОЛе как любой последовательности символов, заключенной в кавычки. Строки в отличие от идентификаторов не являются наименованиями величин, а сами подобно числам представляют собой конкретные нечисловые значения.

Заметим, что в качестве одного из символов для образования строк используется пробел, который вне строки не имеет смысла, а в строке служит для ее разделения на отдельные слова.

Строками могут быть фамилии и имена людей, наименования каких-то конкретных объектов (изделий, городов и т. п.). В АЛГОЛе предусматривается возможность использования строк в процедурах-кодах в качестве фактических параметров, в частности, в процедурах вывода данных, когда вместе с численным значением переменной нужно выдать на печать и ее идентификатор. Однако эта возможность в АЛГОЛе детально не рассматривается, так же как и весь вопрос о процедурах-кодах, поскольку считается, что детализация этого вопроса должна производиться в конкретных представлениях языка.

В АЛГЭМе вводятся, во-первых, строчные переменные, т. е. переменные, значением которых могут быть строки (подобно тому, как значением числовых переменных являются числа, а логических переменных — логические значения). В связи с этим в АЛГЭМе используются четыре типа переменных. Описание типа имеет вид

<тип> : : = **целый** | **вещественный** | **логический** | **строчный**

Строчными могут быть как простые переменные, так и массивы переменных. Строчные переменные подобно переменным других типов должны быть описаны в начале того блока, в котором они используются. Все, что было сказано об областях действия описаний типов, полностью относится и к описателю **строчный**. Заметим, что в связи с появлением описателя **строчный** в АЛГЭМе исключается спецификатор **строка,** имеющийся в АЛГОЛе; вместо него в спецификациях можно пользоваться описателем **строчный,** так как сами строки имеют тип строчный.

Во-вторых, в АЛГЭМе вводятся строчные выражения и, таким образом, используется четыре типа выражений: арифметические, логические, именующие и строчные. Синтаксическое определение строчного выражения аналогично определениям других выражений:

<простое строчное выражеиие> : : = <строка> | <переменная> | <указатель функции> | (<строчное выражение)>

<строчное выражение>: : = <простое строчное выражение> | <условие> <простое строчное выражение> **иначе** <строчное выражение>

Ясно, что переменные и указатели функций, являющиеся строчными выражениями, должны иметь тип **строчный.**

Примеры строк и строчных выражений:

‘Иванов’
‘Ставка **[***i***]’**
если год [*n*] = 365 **то ‘**обычный’ **иначе** ‘високосный’

Описание вида переменных и указание разрядов

В АЛГЭМе строчные переменные обязательно должны иметь указание вида, показывающее строение этой переменной (сколько символов и какого вида входит в состав этой переменной). Числовые переменные (т.е. имеющие тип **целый** или **вещественный)** также могут иметь, когда это необходимо, указание вида, показывающее, сколько разрядов и каких (десятичных, восьмеричных или двоичных) входит в состав данной переменной, а также показывающее положение знака числа и десятичной запятой.

Логические переменные, очевидно, указания вида иметь не могут. Указание вида могут иметь только элементарные переменные в отличие от составных переменных, которые указания вида иметь не могут (составные переменные будут рассмотрены ниже). Указание вида переменной является дополнением описания типа переменкой. Дополненное описание типа будет выглядеть следующим образом:

<указатель символа> : : = 9 | 7 | 1 | *С* | *А* | *Л* | |_| | ·| *Д* | $_{10}$ | 0 | *Т* | *П* | + | - |
<повторитель> : : = (<целое без знака>)
<элемент указания вида> :: = <указатель символа> | <указатель символа> <повторитель>
< указание вида> ::= <элемент указания вида> | <указание вида> <элемент указания вида>
<элемент списка типа> : : = <идентификатор> | <идентификатор> **вид** <указание вида>
<список типа> :: = <элемент списка типа> | <список типа>, <элемент списка типа>
<тип> : : = целый | вещественный | логический | строчный
<локальный или собственный тип> : : = <тип> | **собственный** <тип>
<описание типа> : : = <локальный или собственный тип> <список типа>

В состав основных символов языка вводится, таким образом, новый описатель **вид**. Указание вида ставится за идентификатором соответствующей переменной и относится только к этой переменной.

Примеры:

целый *P, Q, S* **вид** 9 (5);
строчный дата **вид** 99.4 (10) 9 (4) *A;*
строчный деталь **вид** *С* (5) 99;
строчный фамилия **вид** *А* (12) |_| *А*;

Рассмотрим смысл отдельных указателей символов, перечисленных выше. Повторитель ставится за соответствующим указателем символа и показывает, сколько раз должен быть повторен подряд один и тот же символ. Повторители применяются для сокращения письма при указании вида. Указатель символа 9 показывает, что на данном разрядном месте может стоять только одна десятичная цифра (0, 1, 2, 3, 4, 5, 6, 7, 8 или 9). 7 означает, что в этом разряде может стоять только одна восьмеричная цифра (0, 1, 2, 3, 4, 5, 6, 7). 1 показывает, что в данном разряде может стоять только двоичная цифра (0 или 1).

Указатель *С* показывает, что в данном разряде строчной величины может стоять любой символ.
А означает, что в этом разряде может стоять буква русского алфавита или пробел.
Л означает разряд, занимаемый буквой латинского алфавита или пробелом.
|_| означает разряд, занятый пробелом,
· указывает помещение в данном разрядном месте в явном виде десятичной запятой.

Д означает, что на данном разрядном месте должна стоять подразумеваемая десятичная запятая, которая в явном виде не должна фигурировать и не должна учитываться при подсчете символов данного слова.

Указатель $_{10}$ определяет, что на данном разрядном месте должен стоять символ основания десятичной системы счисления, а правее этого символа будет идти порядок числа. Указатель 0 предписывает замену в данном разряде значащей цифры на нуль с округлением.

Указатель *П* предписывает замену пробелами ('|_|') всех нулей числа, стоящих левее первой значащей цифры (нуль, стоящий перед десятичной запятой десятичной дроби, считается значащей цифрой) и стоящих правее последней значащей цифры (вправо от десятичной запятой).

Указатель + определяет разряд знака числа или знака порядка (в последнем случае этот указатель стоит после указателя $_{10}$). Знак числа при этом указывается явно (+ для положительных чисел, — для отрицательных).

Указатель символа — имеет аналогичный смысл, только знак + у положительных чисел явно не указывается, а вместо него пишется пробел.

Указатель *Т* определяет замену в данном разряде значащей цифры на пробел без округления.

Указание вида может иметь в начале несколько подряд идущих знаков + или —. При этом положение знака числа не задается; знак числа должен быть помещен на место нуля, стоящего перед значащей цифрой, а перед знаком числа должны стоять пробелы. Это дает возможность сохранить в числе старшие значащие цифры, если они выходят за пределы, предусмотренные указателями символов 9. Аналогичную роль будут играть указатели + или —, когда они стоят в конце указания вида. Если же в указании вида отсутствуют несколько подряд идущих символов + или — (в начале и в конце) и фактически получающееся число имеет больше значащих цифр, чем для него предусмотрено в указании вида символами 9 (или 7 или 1), то происходит его преобразование к заданному виду. Для этого совмещаются положения десятичной запятой в числе с ее положением, заданным указанием вида, затем старшие цифры числа, выходящие за пределы, заданные указанием вида, отбрасываются; отбрасываются также и младшие значащие цифры, выходящие за заданные пределы, с той только разницей, что перед отбрасыванием они округляются. Подобные действия, как правило, могут происходить при выполнении операторов присваивания, причем присваивание всегда должно производиться с учетом указания вида, заданного для переменной левой части, независимо от того, имеется или нет указание вида в описании переменных, используемых в правой части. Если же вид правой части противоречит (кроме описанного выше случая) виду левой части, то действие оператора присваивания будет неопределенным. Общее число указателей символов (с учетом повторителей), содержащихся в данном указании вида, определяет количество разрядов в слове, представляющем величину. Все разряды слова независимо от их вида (двоичные, десятичные, буквенные и т. д.)

нумеруются подряд слева направо. Описанный способ указания вида переменных позволяет:

а) транслятору определять распределение памяти под соответствующие величины;

б) при программировании задач задавать точность и форму представления чисел;

в) при программировании задач осуществлять обращения к отдельным разрядам переменных.

Примеры действия указания вида. Пусть имеется число +167, 835. При указаниях вида, записанных в левой колонке, данное число будет иметь вид, показанный в правой колонке следующей таблицы:

+99.99	+ 67,84
++99.999	+ 167,835
– (5)9.9/7(3)	|_| |_| |_| 167,835|_|
– (5) *Д*9	|_| |_| 1678

В связи с введением в язык такого средства, как указание вида, необходимо предусмотреть возможность указания вида для процедур-функций. В соответствии с определением указания вида оно ставится после идентификатора соответствующей величины. В случае процедуры функции указание вида должно ставиться (если оно необходимо) после идентификатора процедуры-функции в заголовке процедуры. В связи с этим в АЛГЭМе дается следующее определение идентификатора процедуры:

<идентификатор процедуры> : : = <идентификатор> | <идентификатор> **вид** <указание вида>

Рассмотрим теперь способ указания разрядов, применяемый при программировании различных операций с отдельными разрядами величин или группами разрядов. Для тех переменных, которые в своем описании содержат указание вида, имеется возможность выделять отдельные разряды или группы разрядов по их номерам в слове в соответствии с указанием вида.

Для переменных, не содержащих в своем описании указания вида, также имеется возможность выделять разряды. В этом случае используется разрядная сетка конкретной вычислительной машины, и все двоичные разряды такой сетки считаются перенумерованными подряд слева направо от нуля до некоторого *n*. Величина, у которой выделяются разряды без описания ее вида, считается расположенной в ячейке стандартным для данной машины образом, и указание разрядов относится к номерам двоичных разрядов разрядной сетки данной конкретной машины.

Указание разряда, так же как и указание вида, может относиться только к элементарной (несоставной) переменной. Указание разряда представляет собой список номеров выделяемых разрядов, заключенный в круглые скобки и стоящий перед идентификатором соответствующей величины. Если требуется выделить несколько подряд идущих разрядов, то для сокращения письма могут указываться только номера старшего и младшего из выделяемых разрядов, разделенные двоеточием подобно граничным парам в описаниях массивов.

Синтаксическое определение указания разрядов:

<список разрядов> : : = <арифметическое выражение> | <граничная пара> | <список разрядов>, <арифметическое выражение> | <список разрядов> <граничная пара>

<указание разрядов> : : = <пусто> | (<список разрядов>)

В связи с введением указаний разрядов несколько расширяется определение переменной, которое будет выглядеть следующим образом:

<простая переменная> : : = <указание разрядов> <идентификатор>

<переменная с индексами> : : <указание разрядов> <идентификатор> [<список индексов>]

<элементарная переменная> : : <простая переменная> | <переменная с индексами>

В данном случае термин «элементарная переменная» используется как противопоставление понятию «составная переменная», которое будет рассмотрено в дальнейшем.

Примеры:

(1,5 : 10)*x*: — выделяются разряды 1, 5, 6, 7, 8, 9, 10 значения переменной *x*

(*i*: *i* + 5, *j*, *k* : *k* + *n*)*y* — выделяются разряды значения переменной *y* с *i* го по (*i* + 5)-й, затем *j*-й, затем с *k*-го по (*k* + *n*)-*й*.

§ 11 СОСТАВНЫЕ ПЕРЕМЕННЫЕ И МАССИВЫ

В АЛГЭМе предусматривается возможность использования простых составных переменных и массивов составных переменных.

Составными переменными называются такие, которые включают в себя другие переменные, причем этими переменными могут быть как элементарные, так и составные переменные. Примером составной переменной служит переменная «дата», в состав которой входят элементарные переменные: «день», «месяц», «год». Типичными примерами составных переменных являются так называемые записи — наборы данных, относящихся и одному какому-нибудь объекту или процессу. Например, набор данных об одном работнике, необходимых для расчета заработной платы, или набор данных, содержащих сведения об одной операции продажи какого-то товара и т. д. Как упоминалось во введении, сущность многих процессов решения экономических задач сводится к однотипной обработке массивов подобных записей. Составным массивом мы называем массив, образованный из составных переменных одинакового строения. Заметим, что в АЛГЭМе допускается, например, такая организация данных, когда элементами, входящими **в** состав простой составной переменной, являются массивы (как элементарные, так и составные). В общем случае никаких ограничений на количество уровней иерархии при построении составных переменных и массивов не накладывается.

Как и элементарные переменные и массивы, составные переменные и массивы должны быть описаны в том блоке, в котором они используются. Для описания составных переменных и массивов в число основных символов вводятся два новых символа: описатель **составной** и разделитель **уровень.** Таким образом, в АЛГЭМе по сравнению с АЛГОЛом используются четыре новых основных символа: **строчный, вид, составной, уровень.**

Синтаксическое определение описания составной величины (составной переменной и составного массива) имеет следующий вид:

<элемент описания> : : = <описание типа> | <описание массива> | <описание составной величины>

<идентификатор составной переменной> : : = <идентификатор>

< идентификатор составного массива> :: = <идентификатор>

<составная величина> : : = <идентификатор составной переменной> | **массив** <идентификатор составного массива> [<список граничных пар>]

<структура составной величины> : : = <элемент описания> | <элемент описания>; <структура составной величины>

<описание составной величины > : : = **составной** <составная величина>; <структура составной величины > **уровень**

Заметим, что составные величины не имеют указаний вида. Из приведенных синтаксических определений видно, что если в структуру составной величины входят на одном уровне несколько элементарных переменных одного типа, но разных видов, то они описываются обычным образом, т. е. за описателем типа перечисляются соответствующие элементы списка типа, разделенные запятыми. Обычным образом даются описания и элементарных массивов, являющихся компонентами составных величин. Эти описания входят в понятие элемент описания, которое включает в себя либо только одно описание типа (со списком типа), либо описание массива, либо описание составной величины. В АЛГОЛе принято после каждого описания (описания типа, массива и т. д.) ставить точку с запятой.

В АЛГЭМе также сохраняется это правило и после каждого отдельного описания ставится точка с запятой, в том числе и тогда, когда они входят в состав описания составной величины.

Описание любой составной величины (составной переменной или составного массива) начинается символом **составной,** после которого ставится либо идентификатор переменной, либо идентификатор массива со списком граничных пар в квадратных скобках. После этого в обоих случаях идет собственно описание структуры этой составной величины, представляющее собой перечисление описаний компонентов, входящих в состав составной величины. Описание составной величины заканчивается обязательно символом **уровень.**

Таким образом, каждому символу **составной** должен соответствовать один символ **уровень.** Между этими символами помещаются как идентификатор самой составной величины, так и описания компонентов, входящих в нее. Отдельные компоненты могут быть, в свою очередь, составными переменными или составными массивами, и их описания в этом случае также начинаются с символа **составной** и заканчиваются символом **уровень.**

Здесь имеется аналогия с использованием обычных (круглых) скобок при построении арифметических выражений или с использованием операторных скобок **начало** и **конец** при построении блоков или составных операторов. Более глубокая пара символов **составной** и **уровень** выделяет более высокий уровень иерархии компонентов составной величины. Для наглядности допускается при программировании записывать в явном виде номера уровней отдельных компонентов, однако синтаксически эти номера представляют собой избыточную информацию и транслятор может быть построен таким образом, чтобы производить обработку описаний составных величин без использования номеров уровней.

Рассмотрим в качестве примера описание составной переменной, имеющей следующую структуру:

173

ГРАФИК ДЕЖУРСТВ										
НАЧАЛЬНАЯ ДАТА			КОЛИЧЕСТВО ДНЕЙ	ФАМИЛИИ						
ДН	МЦ	ГД		Иванов	Петров	Сидоров	Ивановский	Петровский	Сидоровский	Палкин	Галкин
01	01	65	08								

В таблице даны некоторые конкретные значения.

составной ГРАФИК ДЕЖУРСТВ;
составной НАЧАЛЬНАЯ ДАТА;
целый ДН **вид** 99,
МЦ **вид** 99,
ГД **вид** 99 **уровень**;
целый КОЛИЧЕСТВО ДНЕЙ вид 99;
строчный массив ФАМИЛИИ **вид** *А* (15) [1 : 8] **уровень**

Рассмотрим второй пример. Составной массив ВЕДОМОСТЬ, состоящий из 500 элементов, имеет следующую структуру отдельного элемента:

ВЕДОМОСТЬ							
ТАБЕЛЬНЫЙ НОМЕР	ДАТА			ОКЛАД	КОЛИЧЕСТВО ПРОПУСКОВ	КОЛИЧЕСТВО ДЕТЕЙ	СУММА
	ДН	МЦ	ГД				
0125	06	01	05	150	03	01	140

Описание этого массива будет иметь вид:
составной массив ВЕДОМОСТЬ [1 : 500];
целый ТАБЕЛЬНЫЙ НОМЕР **вид** 9 (4);
составной ДАТА;
целый ДН **вид** 99,
МЦ **вид** 99,
ГД **вид** 99 **уровень**;
целый ОКЛАД **вид** 9 (3);
КОЛИЧЕСТВО ПРОПУСКОВ **вид** 99,
КОЛИЧЕСТВО ДЕТЕЙ **вид** 99,
СУММА **вид** 9(3) **уровень**

Это же описание составного массива с использованием в явном виде номеров уровней может быть представлено следующим образом:
составной массив ВЕДОМОСТЬ, [1 : 500];
1 **целый** ТАБЕЛЬНЫЙ НОМЕР **вид** 9(4);
1 **составной** ДАТА;
2 **целый** ДН **вид** 99,
МЦ **вид** 99,
ГД **вид** 99 **уровень**;
1 **целый** ОКЛАД **вид** 9 (3),
КОЛИЧЕСТВО ПРОПУСКОВ **вид** 99,
КОЛИЧЕСТВО ДЕТЕЙ **вид** 99,
СУММА **вид** 9(3) **уровень**

Обращение к компонентам составных величин

На основе описаний составных переменных и массивов можно при программировании выделять отдельные компоненты составных переменных и выполнять над ними различные действия. Кроме того, для элементарных переменных, как это было описано, имеется возможность выделения отдельных разрядов или групп разрядов. Для выделения какой-либо компоненты составной переменной применяется следующее правило. Выписывается идентификатор этой компоненты и за ним слева направо выписываются идентификаторы более крупных компонент, в состав которых входит выделяемая компонента, вплоть до самого внешнего идентификатора составной величины. Можно выписывать не все промежуточные идентификаторы структуры, если они не имеют индексов и если их отсутствие не приводит к неопределенности, связанной с тем, что одни и те же идентификаторы могут быть использованы для обозначения компонент разных составных величин. Выписываемые в порядке укрупнения структуры компоненты разделяются между собой точками.
Пример;

ДН. ДАТА. ВЕДОМОСТЬ [250]

Если составная величина ВЕДОМОСТЬ описана, как показано раньше, промежуточная структура ДАТА может быть опущена:

ДН. ВЕДОМОСТЬ [250]

С учетом сказанного полное определение переменной в АЛГЭМе будет выглядеть следующим образом:
<описок разрядов> : : = <граничная пара> | <арифметическое выражение> | < список разрядов>, <граничная пара> | <список разрядов>, <арифметическое выражение>

<указание разрядов> : : = (<список разрядов>) | <пусто>

<простая переменная> : : = <указание разрядов> < идентификатор переменной>

<переменная с индексами> : : = <указание разрядов> <идентификатор массива> [<список индексов >]

<элементарная переменная> : : = <простая переменная> | <переменная с индексами>

<уточнение> : : = <идентификатор составной переменной> | • < идентификатор составного массива> [<список индексов>]

<переменная> : : = <элементарная переменная > | <переменная> <уточнение>

Переменные с уточнениями, а также с указаниями разрядов могут стоять как в правых, так и в левых частях операторов присваивания. При этом если такие переменные стоят в левых частях операторов присваивания, то присваивание производится только указанным компонентам или определенным разрядам, а остальные части величины остаются без изменения. Если же в левой части оператора присваивания указана некоторая переменная

(элементарная или составная), имеющая большее количество разрядов или компонент, чем величина, представляющая собой правую часть оператора присваивания и у этой переменной, стоящей в левой части, не указаны принимающие разряды или принимающая компонента, то присваивание значения правой части производится всей переменной, стоящей в левой части, как единой переменной. При этом совмещаются десятичные запятые в правой и левой частях (если они указаны в описаниях видов) или младшие разряды (если положения запятых не указаны).

Заметим, что в понятие элементарной переменной входят как простая переменная, так и переменная **с** индексами, причем в обоих случаях указание разрядов может быть, а может отсутствовать (указание разрядов может относиться только к элементарной переменной).

Дополнение описания процедур в **АЛГЭМе по сравнению** с **АЛГОЛом**

Введение в состав языка описаний видов переменных, а также составных переменных и массивов и возможность их использования в качестве параметров в процедурах приводят к некоторому расширению описания процедур в АЛГЭМе по сравнению с АЛГОЛом. Это расширение обусловлено, прежде всего, тем обстоятельством, что для построения тела процедуры, использующего составные переменные или массивы, необходимо знать структуру составных величин, а поэтому все такие величины обязательно должны быть так специфицированы в заголовке процедуры, чтобы полностью определялась их структура. Указанные спецификации должны налагать ограничения на структуру составных переменных или массивов, являющихся соответствующими фактическими параметрами. При использовании в качестве формальных параметров составных величин в списке формальных параметров должны указываться только самые внешние идентификаторы составных переменных или массивов, а их полная структура должна быть описана в спецификациях.

Кроме того, в отдельных случаях может потребоваться задать в виде отдельных спецификаций спецификации строчных величин, являющихся формальными параметрами. Эти спецификации должны содержать указания вида строчных величин. Это необходимо делать для тех строчных величин, которые в теле процедуры используются с указанием разрядов.

В тех случаях, когда строчные величины используются без выделения специфичных разрядов, можно обходиться без указания их видов в спецификациях. При этом информация о видах величин, необходимая для распределения памяти под эти величины, будет получаться из указаний видов соответствующих фактических параметров.

В остальном описания процедур, способы конкретизации формальных (в том числе и составных) параметров наименованием и значением, правила локализации величин остаются такими же, как и в АЛГОЛе.

Итак, обязательно должны быть специфицированы формальные параметры, конкретизируемые значением, а также формальные параметры, которым могут соответствовать фактические параметры, являющиеся составными величинами.

Приведем синтаксическое определение понятия «совокупность спецификаций» в АЛГЭМе, которое является расширением соответствующего понятия АЛГОЛа: в него включаются дополнительно два вида спецификаций — спецификации строчных переменных и спецификации составных величин.

<отдельная строчная> : : = <идентификатор> | **массив <**идентификатор массива > | < идентификатор > **вид <**указание вида > | **массив** < идентификатор массива> **вид** <указание вида>

<список строчных> : : = <отдельная строчная> | <список строчных>, <отдельная строчная>

<спецификация строчных> : : = **строчный** <список строчных >

<отдельная спецификация> : : = <спецификация> <список идентификаторов> | <спецификация строчных> | <спецификация составной>

<структура спецификации>: : = <отдельная спецификация> | <структура спецификации>; <отдельная спецификация>

<спецификация составной> : : = **составной** <идентификатор составной переменной>; <структура спецификации> **уровень | составной массив** <идентификатор составного массива>; <структура спецификации> **уровень**

<совокупность спецификаций> : : = <пусто> | <отдельная спецификация>; | <совокупность спецификаций> <отдельная спецификация>;

Здесь понятие «спецификация» соответствует определению, данному в АЛГОЛе.

Таким образом, структура спецификации отличается от совокупности спецификаций тем, что она содержит ряд спецификаций, входящих в состав спецификаций составной величины, и в ней отдельные спецификации разделяются точками с запятыми,

Спецификация составной величины отличается от описания составной величины тем, что в ней не указываются списки граничных пар для массивов, а для элементарных переменных, в том числе и строчных, как правило, не указываются виды переменных.

Отдельные спецификации и отдельные описания в АЛГЭМе, так же как в АЛГОЛе, разделяются точками с запятыми. Отдельные спецификации, входящие в состав спецификации составной величины, и отдельные описания, входящие в состав описания составной величины, также разделяются точками с запятыми.

Приведем запись списка значений и совокупности спецификаций для некоторой процедуры:

значение фп, пп, рх, фонд;
составной массив фп;
составной годовой; **вещественный** н, о **уровень**;
составной массив кварт; **вещественный** н, о **уровень**
уровень;
составной массив пп; **целый** н, о **уровень**;
массив рх;

составной массив фонд;
составной годовой; **целый** н, о **уровень**;
составной массив кварт; **целый** н, о **уровень уровень**;

Приведем пример программы на АЛГЭМе, составленный А. П. Грушецкой и представляющий собой процедуру обсчета заявок предприятий для определения сводной потребности в материалах. Кроме переменных и массивов, являющихся формальными параметрами, конкретизируемыми значением (рост, деф, предпр, позиции, норма) и наименованием (заявки, потр, расход, остатки, ппотр), в теле процедуры используются переменные и массивы как локализованные, так и нелокализованные в теле процедуры.

процедура определение сводной потребности (потр) по заявкам: (заявки) на основе расхода: (расход) остатков: (остатки) плановой потребности: (ппотр) нормы переходящего запаса: (норма) среднего роста потребностей: (рост) признака дефицитности материала: (деф) по числу предприятий: (предпр) и числу позиций: (позиции);
значение рост, деф, предпр, позиции, норма;
вещественный рост; **логический массив** деф; **целый** предпр, позиции;
вещественный массив норма;
составной массив заявки;
вещественный произв, нир и окр, не обор, ремонт **уровень;**
вещественный массив ппотр, расход, остатки, потр;
начало вещественный ч, р, и, з, а; **целый** *i, j, k;* **массив** сумма [1 : позиции];
для / : = 1 **шаг** 1 **до** позиции **цикл**
сумма [*j*] : = 0;
для *i*: = 1 **шаг** 1 **до** предпр **цикл**
для *j*: == **1 шаг 1 до** позиции **цикл**
сумма [*j*]: = сумма [*j*]+ произв. заявки [*i, j*] +нир и окр. заявки [*i, j*]+нс обор, заявки [*i, j*] + ремонт, заявки[*I, j*];
примечание получены суммарные значения потребности, заявленной всеми предприятиями по всем позициям;
ч : = 1 + рост;
для *j*: = 1 **шаг** 1 **до** позиции **цикл**
если расход [*j*] ≠ 0 ∧ деф [*j*] **то**
потр [*j*]: **= если** сумма [*j*] > ппотр [*j*] **то** сумма **[*j*]** × ч **иначе** ппотр **[*j*]** × ч
иначе
если ¬ (расход **[*j*]** = 0**) то**
начало р : = расход [*j*] ч;
п : —ппотр [*j*] × ч;
з : =сумма **[*j*]** × ч**;**
если остатки [*j*] ≤ расход [*j*] × норма **[*j*] то**
начало а : **= если** р > п **то** р **иначе** п**;**
потр [*j*] : =если а > з **то** а **иначе** з **конец**
иначе а : **= если** р > п **то** п **иначе** р;
потр **[*j*]** : **= если** а < з **то** а **иначе** з **конец**
иначе потр [*j*]: **= если** ппотр [*j*] <сумма [*j*] **то**
ппотр [*j*] **иначе** сумма [*j*]
конец процедуры определения сводной потребности;

§ 12 ДОПОЛНИТЕЛЬНЫЕ СРЕДСТВА ДЛЯ ОПИСАНИЯ ПРОЦЕССОВ ВЫЧИСЛЕНИЙ

Оператор процедуры-кода в АЛГЭМе

В АЛГЭМе несколько конкретизируется по сравнению с АЛГОЛом понятие оператора процедуры-кода. Этот оператор позволяет использовать в программах, составляемых на алгоритмическом языке, отдельные стандартные подпрограммы, написанные на машинном языке.

Оператору процедуры-кода не дается описания процедуры на языке АЛГЭМ.

Синтаксическое определение оператора процедуры-кода имеет вид:

<оператор процедуры-кода> : = код (<список фактических параметров процедуры-кода>)

<список фактических параметров процедуры-кода> : = <спецификация процедуры-кода> | <список фактических параметров процедуры-кода> <фактический параметр>

<спецификация процедуры-кода> : = ‘ПЧ’ | ‘ПЧ2 — 10’ | ‘ПР’ | ‘ПР2 — 10’ | ‘ЧТ’ | ‘ЧТ10—2’ \ ‘запись’ 1 ‘чтение’

Таким образом, оператор процедуры-кода имеет постоянно закрепленный идентификатор «код», который запрещается использовать для других целей. За этим идентификатором в круглых скобках перечисляются фактические параметры процедуры-кода. Первым фактическим параметром является так называемая спецификация процедуры-кода, определяющая конкретный вид стандартной подпрограммы, которая должна быть использована. В качестве спецификаций процедур-кодов используются постоянно закрепленные строки.

Так, строка 'ПЧ' определяет подпрограмму выдачи данных на печать без перевода систем счисления, строка 'ПЧ2 — 10'—выдачу на печать с переводом из двоичной в десятичную систему счисления, строка 'ПР' — перфорацию без перевода, строка 'ПР2—10' — перфорацию с переводом из двоичной в десятичную систему

счисления, строка 'ЧТ' — чтение (ввод) данных без перевода систем счисления, строка 'ЧТ10—2' — чтение с переводом из десятичной в двоичную систему счисления. Строка 'запись' определяет подпрограмму переписи величины, заданной своим идентификатором, из оперативной памяти во внешнюю память на магнитной лейте, а строка 'чтение' определяет перепись величины из внешней памяти в оперативную ячейку (ячейки), закрепленную за соответствующим идентификатором.

Естественно, что список спецификаций процедур кодов может расширяться по мере накопления стандартных подпрограмм.

Следует сделать несколько пояснений относительно фактических параметров операторов процедур-кодов. Фактическими параметрами всех перечисленных операторов процедур-кодов, кроме 'записи' и 'чтения', могут быть выражения, идентификаторы массивов, составных переменных, составных массивов, компоненты составных переменных и массивов. Число фактических параметров в этих процедурах не ограничивается. При выполнении соответствующей процедуры производится перфорация, печать или ввод числовых значений величин, заданных своими идентификаторами.

Для процедур-кодов, имеющих спецификации 'запись' или 'чтение', число фактических параметров, не считая спецификации, должно быть четным, причем эти фактические параметры рассматриваются попарно.

Первые элементы пар должны быть идентификаторами массивов или переменных, определяющими некоторые группы ячеек (ячейку) в оперативной памяти, из которой производится выборка (при записи на МЛ) или в которую производится посылка (при чтении с МЛ).

Вторые элементы пар должны представлять собой некоторые условные идентификаторы, определяющие адреса зон и ячеек на магнитных лентах, а также номера блоков магнитных лент, из которых производится чтение или в которые производится запись данных.

Например:

код ('запись', *А, М* 105020, *В, М* 020501)

код ('чтение', С, *М* 070301, *Д, М* 110030)

Процедуры ввода и вывода в АЛГОЛе

Дополнительным сообщением, опубликованным рабочей группой по АЛГОЛу Международной федерации по обработке информации в 1964 г., вводится семь так называемых первичных процедур ввода и вывода данных.

В понятие ввода и вывода включается при этом и обмен данными между различными накопителями.

Первичными эти процедуры считаются потому, что их тела не могут быть выражены средствами АЛГОЛа, а должны быть представлены в виде кодов на языке конкретных машин. Заголовки же этих процедур являются стандартными, не зависящими от конкретных машин. Сочетая эти процедуры и средства АЛГОЛа, можно строить более сложные процедуры, осуществляющие ввод и вывод данных и обмен данными между различными накопителями.

Для унификации названий различных устройств ввода и вывода и накопителей вводится понятие канала, которым обозначается любой агрегат машины, участвующий в обмене информацией. Идентификатор «канал» используется как первый формальный параметр в указанных процедурах. Все такие агрегаты (оперативная память, магнитные ленты, барабаны, диски, перфораторы, печатающие устройства и т. д.) должны быть пронумерованы для каждой конкретной машины и за каждым из них должен быть закреплен постоянный номер. Этот номер будет первым фактическим параметром в операторе процедуры; он присваивается формальному параметру «канал» при обращении к соответствующей процедуре.

При использовании в качестве внешних агрегатов, участвующих в обмене информацией, магнитных барабанов, лент и дисков, соответствующий номер канала должен содержать в себе как признак конкретного агрегата (барабана, лентопротяжного механизма и т.д.), так и адрес начальной ячейки на барабане, магнитной ленте, диске. При этом последовательность чисел или символов вводится с агрегата или выводится на агрегат в группу ячеек или из группы ячеек с последовательно возрастающими адресами, начиная с указанной ячейки.

Упомянутые семь первичных процедур имеют следующие постоянно закрепленные идентификаторы,

insymbol	— ввод символа,
outsymbol	— вывод символа,
length	— длина строки,
inreal	— ввод вещественного числа,
outreal	— вывод вещественного числа,
inarray	— ввод массива,
outarray	— вывод массива.

Как известно, в процессе вычислений в машине вся исходная информация, а также промежуточные данные и окончательные результаты могут представляться либо в виде значений переменных, либо в виде строк (которые тоже являются значениями строчных переменных в АЛГЭМе). Собственно ввод и вывод информации может осуществляться только в виде последовательностей кодов отдельных символов. Допустимые наборы символов зависят от конструкций клавиатур вводных устройств и возможностей печатающих устройств, а также от конструкций входных и выходных перфораторов в 'конкретных машинах. Ничего кроме кодов отдельных символов, нельзя ввести в машину и получить из машины.

В состав основных символов АЛГОЛа входят символы, представляющие отдельные цифры, 'поэтому из линейной последовательности символов, подаваемых на вход машины, можно в машине формировать как числа, так и строки. Следовательно, в процессах обмена информацией внутри машины могут участвовать символы, числа и строки.

Процедура insymbol осуществляет ввод одного очередного символа из последовательности символов, подаваемых по некоторому каналу, и ставит этому символу в соответствие определенное целое число, которое будет теперь постоянным кодом этого символа в данной программе. Этот код символа присваивается указанной процедурой некоторой переменной, заданной в виде третьего фактического параметра в операторе процедуры.

Таким образом, код введенного символа, представленный теперь в виде значения заданной переменной, становится доступным для программы.

При этом внешний код данного символа, связанный с конструкцией конкретных устройств ввода и вывода, ставится в однозначное соответствие с внутренним кодом этого символа в программе, выработанным данной процедурой insymbol.

Описание процедуры insymbol может быть представлено следующим образом:

процедура insymbol (канал, строка, назначение): **значение** канал; **целые** канал, назначение; **строка** строка;
<тело процедуры>;

Оператор данной процедуры имеет вид:

insymbol (<арифметическое выражение> <ограничитель параметра> <строка> <ограничитель параметра> <переменная>)

Арифметическое выражение, являющееся первым фактическим параметром, представляет собой номер канала, по которому производится ввод символа.

Второй фактический параметр представляет собой строку символов, служащую для кодировки вводимых символов. Все символы этой строки, кроме внешних кавычек, считаются пронумерованными подряд слева направо. Введенный очередной символ сравнивается с символами этой строки, и номер совпавшего символа строки присваивается переменной, являющейся третьим фактическим параметром этого оператора процедуры. Ясно, что указанная строка не должна содержать повторяющихся символов, так как в противном случае может возникнуть неопределенность в их кодировании. Если введенный символ не совпал ни с одним из символов заданной строки, то переменной присваивается значение нуль.

Если введенный символ не является основным символом АЛГОЛа, то он должен кодироваться отрицательным целым числом.

Основным требованием к системе кодировки символов, как и вообще к процедурам ввода и вывода, является требование обратимости процессов ввода и вывода. Как видно из приведенного выше перечня процедур, шесть процедур из семи объединяются в три пары взаимно обратных процедур:

insymbol — outsymbol,
inreal — outreal,
inarray — outarray.

В каждой из этих пар действие процедуры ввода должно точно соответствовать действию процедуры вывода. Это значит, что если, например, процедурой insymbol введен какой-то символ и ему дан некоторый код, то процедура outsymbol, получив этот код, должна вывести в канал тот же символ.

Процедура outsymbol имеет следующее описание;

процедура outsymbol (канал, строка, источник);
значение канал, источник;
целые канал, источник; **строка** строка;
<тело процедуры>;

Оператор данной процедуры имеет вид:

outsymbol (<арифметическое выражение> <ограничитель параметра> <строка> <ограничитель параметра> <арифметическое выражение>).

Первый и второй фактические параметры в операторе процедуры outsymbol имеют такой же смысл, что и в операторе insymbol. Третий фактический параметр — арифметическое выражение, соответствующее формальному параметру «источник», служит для указания порядкового номера выводимого символа в строке символов; этот символ должен быть выведен в канал, номер которого задается арифметическим выражением, являющимся первым фактическим параметром.

Процедура-функция length имеет следующее описание:

целая процедура length (строка);
строка строка; <тело процедуры>

Для обращения к ней служит указатель функции length(*S*), в котором *S* обозначает произвольную строку символов. Эта функция принимает значение, равное числу основных символов, имеющихся в открытой строке (без внешних кавычек), соответствующей строке *S*.

Процедуры inreal и outreal служат соответственно для ввода и вывода действительного числа.

Описание процедуры inreal:

процедура inreal (канал, назначение);
значение канал; **целый** канал;
вещественный назначение; <тело процедуры>;

Оператор этой процедуры имеет вид:

inreal (<арифметическое выражение> <ограничитель параметра> < переменная >)

С помощью этой процедуры из канала, номер которого задан арифметическим выражением, вводится число, имеющее вещественный тип, и значение этого числа присваивается переменной, являющейся вторым фактическим параметром.

Процедура outreal выводит в канал с заданным номером значение переменной, являющейся вторым фактическим параметром. Описание этой процедуры имеет вид:

процедура outreal (канал, источник);

значение канал, источник; <тело процедуры>
Оператор указанной процедуры:
outreal (< арифметическое выражение> <ограничитель параметра> <арифметическое выражение>)

Для ввода и вывода массивов вещественных чисел (в частном случае — целых чисел, если они записаны в форме, принятой для вещественных чисел) служат следующие две процедуры. Описание процедуры ввода массива:

процедура inarray (канал, назначение);
значение канал; **целое** канал;
массив назначение; <тело процедуры>
Оператор этой процедуры:
inarray (< арифметическое выражение> <ограничитель параметра> <идентификатор массива>);
Описание процедуры вывода массива:
процедура outarray (канал, источник);
значение канал; **целое** канал;
массив источник; <тело процедуры>
Оператор этой процедуры имеет вид:
outarray (<арифметическое выражение> <ограничитель параметра> <идентификатор массива>);

В операторах процедур ввода и вывода массивов в качестве вторых фактических параметров указываются только идентификаторы массивов.

Необходимая информация о размерах и размерностях массивов берется из описаний соответствующих массивов.

Для ввода массива значения его элементов должны располагаться на внешнем носителе в виде линейной последовательности чисел в лексикографическом порядке.

Если, например, массив представляет собой матрицу, то элементы его должны идти по строкам сверху вниз и в строках слева направо. Вообще элемент $a\,[k_1, k_2, ..., k_n]$ предшествует элементу $a[j_1, j_2, \ldots, j_n]$, если $k_1 < j_1$ или $k_i = j_i$ (для $i = 1, 2, 3, ..., p$—1) и $k_p < j_p$,, причем $1 < p \leq n$. В таком же порядке будут выводиться элементы массива на внешний носитель (на МЛ, МБ и т. д.).

Используя указанные первичные процедуры, можно строить с помощью средств АЛГОЛа более сложные процедуры.

Рассмотрим некоторые примеры, приведенные в сообщении рабочей группы по АЛГОЛу.

Процедура для ввода целых чисел имеет следующее описание.

процедура ininteger (канал, целое); **значение** канал;
целые канал, целое;
примечание процедура осуществляет ввод целого числа, представленного на внешнем носителе в виде последовательности цифр, перед которой, возможно, стоит знак и за которой следует запятая, ограничивающая это число. Любой другой символ веред знаком не принимается во внимание;
начало целые *n, k;* **логическое** *b;*
целое : =0; *b* : = **истина;**
для *k* : = 1, *k*+1 **пока** *n* = 0 **цикл**
insymbol (канал, '0123456789 — + , ' , *n*);
если *n* = 11 **то** *b :* = **ложь; если** *n* > 10 **то** *n*: = 1;
для *k*: = 1, *k* + 1 **пока** *n* ≠ 13 **цикл**
начало целое : = 10 × целое + *n* — 1;
insymbol (канал '0123456789— + ', *n)*
конец
если ¬*b* **то** целое : = — целое
конец

С помощью первого цикла в данной процедуре производится ввод очередных символов из заданного канала и проверка их на наличие цифры или знака «—» или «+». При вводе знака «—» переменная *n* получит значение 11, при вводе знака «+» — значение 12, а при вводе какой-либо цифры — значение, на единицу большее значения этой цифры. Если первым опознанным символом будет знак «—», т. е. *n* будет равно 11, то логическая переменная *b* получит значение **ложь,** что в дальнейшем будет использовано для присвоения целому числу знака «—». После опознания знака числа переменной *n* присваивается значение 1.

Второй цикл в процедуре последовательно вводит цифры числа до тех пор, пока очередным символом не окажется запятая, ограничивающая число. Одновременно с вводом цифр производится вычисление абсолютного значения числа.

Рассмотрим процедуру, осуществляющую вывод заданной строки символов. Описание этой процедуры имеет вид:

процедура outstring (канал, строка);
значение канал; **целое** канал; **строка** строка;
начало целое *i;*
для *i*: = 1 **шаг** 1 **до** length (строка) **цикл** outsymbol (канал, строка, *i*)
конец

В теле данной процедуры используется оператор процедуры length (строка), который подсчитывает количество символов в выводимой строке и ограничивает тем самым число повторений оператора цикла. С каждым повторением цикла выводится один символ.

III. АССОЦИАТИВНОЕ ПРОГРАММИРОВАНИЕ

5. ОБЩИЕ СВЕДЕНИЯ ОБ АССОЦИАТИВНОМ ПРОГРАММИРОВАНИИ

§ 13 СУЩНОСТЬ АССОЦИАТИВНОГО ПРОГРАММИРОВАНИЯ И СПОСОБЫ ПОСТРОЕНИЯ СПИСКОВ

Сущность решения многих информационно-логических задач сводится к так называемой категоризации объектов, т. е. к разделению объектов на виды, типы, классы и т.д., в зависимости от их свойств и признаков. При этом запись новой информации об объектах в память машины или поиск информации в машине осуществляется по значениям этих признаков. Примерами подобных задач могут служить учет и планирование материально-технического снабжения, учет кадров, задачи библиографического поиска, обработки и накопления научной информации, машинной медицинской диагностики заболеваний и т. д.

Для эффективного решения подобных задач необходима специальная организация работы запоминающих устройств машин, которая позволяла бы осуществлять поиск информации не только по се адресам, но и по ее признакам.

Существует два основных пути организации машинной памяти, называемой ассоциативной и обеспечивающей быстрый поиск объектов, в зависимости от их признаков.

Первый путь — схемный; он состоит в построении специальной памяти, в которой запоминающие ячейки (частично или полностью) обладают свойством выполнять операции сравнения. Для поиска информации в такой памяти вместо адресов задаются наборы признаков. В ячейках памяти производится сравнение хранящихся в них признаков с заданными и на выход ЗУ выдаются адреса ячеек, в которых хранится информация, обладающая заданными признаками; в некоторых случаях выдается сразу искомая информация. Такой схемный способ организации памяти можно назвать *схемно-ассоциативным.* При этом необходимые наборы признаков хранятся во всех ячейках памяти и поиск информации, обладающей заданным набором признаков, производится одновременно и независимо по всему объему памяти. Связь между отдельными ячейками памяти, в которых хранятся сходные признаки, т. е. прямая ассоциативная связь, отсутствует. Свойство ассоциативности такой организации памяти проявляется в возможности поиска объектов по их признакам, т. е. в характере связи между памятью и внешним потребителем информации. Ассоциативность заключается в том, что одновременно извлекается из всех ячеек информация об объектах, обладающих общим набором признаков. Таким образом может выявляться наличие общих свойств у данных, хранящихся в разных ячейках, т. е. наличие ассоциативных связей между данными.

Второй путь организации ассоциативной памяти можно назвать *программно-ассоциативным.* Он основан на использовании обычных адресных систем памяти и заключается в том, что ассоциативные связи между различными данными, хранящимися в памяти, осуществляются либо путем специального упорядоченного расположения этих данных (или их наименований) в памяти машины, либо путем объединения их в последовательные цепочки при помощи специальных адресов связи, коды которых хранятся в тех же ячейках памяти совместно с данными. Группы последовательно расположенных данных или данных, связанных адресами связи, называются *списками.* При этом необходимые признаки информации могут храниться либо во всех объединяемых членах, либо только в отдельных ячейках, помещаемых в началах списков.

Некоторые члены списков, в общем случае, могут указывать ответвления к другим спискам, так называемым *подспискам и т. д.* Любой список со всеми отходящими от него подсписками называется *списковой структурой.* Программно-ассоциативный способ, по-видимому, является более реальным для практического осуществления при больших объемах переменной информации, так как не требует разработки специальной ассоциативной памяти большого объема. Он обладает гибкостью и обеспечивает возможность изменения в широких пределах алгоритмов поиска информации, состава и характера признаков, по которым производится поиск, в том числе реализацию многоступенчатых и рекурсивных поисковых процессов. *Ассоциативным программированием* мы называем совокупность способов решения информационно-логических задач, основанных на программной реализации ассоциативных связей между данными, хранящимися в памяти машины. За рубежом этот круг вопросов известен под несколькими названиями: списковая обработка, узловая обработка, цепная адресация, метод управляющих слов.

Ассоциативное программирование удобно применять при логической обработке информации, изменяющейся по своему составу и объему, когда процессы поиска и обработки имеют иерархический и рекурсивный характер.

В общем случае при ассоциативном программировании не требуется производить заранее жесткое распределение памяти. Распределение памяти осуществляется автоматически в ходе процесса обработки в соответствии с фактическим поступлением данных.

Ассоциативное программирование позволяет значительно (в сотни и тысячи раз, в зависимости от количества просматриваемых или обрабатываемых данных) ускорить поиск данных, их анализ и обработку.

Способы построения списков

Рассмотрим следующие способы построения списков в машинной памяти:

1) последовательный,
2) цепной,
3) гнездовой,
4) узловой.

Последовательные списки

При последовательном способе построения списков ассоциативные (списковые) слова, представляющие отдельные члены данного списка, размещаются в последовательно расположенных ячейках машинной памяти.

Достоинством последовательного способа построения списков является возможность быстрого поиска требуемых членов списков в тех случаях, когда эти члены располагаются в порядке монотонного изменения какого-либо признака (например, номера объекта). Тогда для поиска объекта с заданным номером может быть применен, например, способ последовательного деления списка пополам, обеспечивающий нахождение нужного члена списка приблизительно за $N=log_2 n$ операций сравнения, где n — общее число членов списка. Недостатками последовательных списков являются:

а) необходимость заранее знать количество членов в каждом списке и соответственно распределять память машины для построения каждого списка. При этом память машины используется нерационально, так как ожидаемые размеры списков часто не совпадают с фактическими. Некоторые зоны памяти оказываются недостаточными для размещения списков, в то время как другие зоны используются не полностью;

б) сложность процедур включения в список новых членов и исключения старых членов в тех случаях, когда списки должны быть упорядочены по каким-то признакам. При этом для вставки нового члена в середину списка или исключения старого члена из середины списка требуется сдвиг всех последующих членов. Если включение и исключение членов списка производится случайно по отношению к их расположению в списке, то в среднем на каждую операцию включения или исключения одного члена списка приходится сдвиг половины списка.

Цепные списки

При цепном способе построения списков отдельные члены списка располагаются произвольно в машинной памяти и связываются между собой при помощи так называемых адресов связи. Адрес связи помещается совместно с данным членом списка и указывает положение 192

следующего члена списка. Идея цепной адресации списков и подробная разработка методики построения и преобразования цепных списков была дана американскими учеными Невелом, Симоном и Шоу [15].

Обычно при обработке данных о некоторой совокупности объектов эти объекты распределяются между различными списками, причем один и тот же объект может находиться одновременно, в нескольких списках. Для того чтобы многократно не повторять во всех списках, куда включен объект, всю информацию об этом объекте, поступают следующим образом. Выделяют определенную область памяти (обычно на магнитных лентах), в которой последовательными участками, так называемыми записями, размещается вся информация об объектах, причем каждому объекту соответствует отдельная позиция (одна запись) со своим адресом. Эта область памяти называется областью записей объектов.

Списки объектов, формируемые по различным признакам, строятся обычно в оперативной памяти или в промежуточной памяти (на магнитных барабанах или дисках). Область памяти, в которой размещаются списки, будем называть списковой областью.

Возможны случаи, когда списки располагаются на магнитной ленте, а для обработки переписываются частями в оперативную память. При наличии отдельных записей, содержащих полную информацию об объектах, в самих списках объекты указываются только своими условными наименованиями. Обычно такими машинными наименованиями объектов являются адреса первых ячеек памяти, в которых хранятся записи для соответствующих объектов. Таким образом, адреса записей являются машинными наименованиями объектов; под этими наименованиями объекты фигурируют во всех списках, в которых они участвуют.

В табл. 1 показан пример области памяти, содержащей записи для некоторых объектов. В данном случае все записи одного типа и содержат некоторый внешний номер объекта *Ni* и координаты объекта *xi, yi, zi.* Адреса записей, показанные в левой колонке (1000, 1001, 1002 и т. д.) являются машинными наименованиями объектов; они указываются в цепных списках, приведенных в табл. 2.

Каждая ячейка в списковой области памяти делится на две части: в одной части хранится машинное наименование объекта, являющегося членом данного списка, а в другой части — адрес связи, указывающий положение следующего члена списка.

Следует заметить, что в дальнейшем изложении мы будем использовать название «списковый» и «ассоциативный» как синонимы. Например, списковый член и ассоциативный член, списковое слово и ассоциативное слово и т.д.

Таблица 1

Адреса записей	Записи объектов
1000	N1, x1, y1, z1
1001	N3, x3, y3, z3
1002	N9, x9, y9, z9
1003	N6, x6, y6, z6
1004	N2, x2, y2, z2
1005	N11, x11, y11, z11
1006	N8, x8, y8, z8

Для цепного способа построения списков характерной чертой является цепная организация свободных ячеек списковой области памяти. Все свободные ячейки этой области памяти (а в начальный момент, когда в памяти нет еще ни одного списка, то все ячейки списковой области памяти) завязаны в так называемый список свободных ячеек (СЯ). Одна ячейка в памяти (в нашем примере ячейка с адресом 100) постоянно закрепляется в качестве фиксатора (или указателя) свободных ячеек. В указателе свободных ячеек в первой половине ячейки хранится число имеющихся в наличии свободных ячеек, а во второй половине — адрес связи, показывающий положение первой ячейки из списка свободных ячеек. В первой ячейке имеется адрес связи, указывающий вторую ячейку, во второй — третью и т, д. В последней ячейке списка свободных ячеек (так же как и в последней ячейке любого цепного списка) вместо адреса связи стоит условный код (КС), показывающий конец списка. Список свободных ячеек служит резервом, из которого берутся ячейки для построения списков и в который возвращаются ячейки, освободившиеся из списков. Процессы взятия ячеек из СЯ и возвращения в СЯ не программируются программистами. Они должны выполняться автоматически либо стандартными подпрограммами, либо схемно.

Для построения каждого цепного списка программистом заранее выделяется одна ячейка, которая будет играть роль фиксатора или указателя этого списка. Адрес ячейки известен программисту и указывается во всех

командах, содержащих обращения к этому списку, в качестве адреса (машинного наименования) списка.

Таблица 2

	Адреса ячеек	Содержание списковых ячеек	
		Машинные наименования объектов	Адреса связи
Фиксатор свободных ячеек →	100	3	110
Фиксатор списка объектов →	101	6	108
	102	1003	105
	103	1004	107
	104	1005	КС
	105	1002	104
	106		109
	107	1001	102
	108	1000	103
	109		КС
	110		106

Таким фиксатором списка объектов в табл. 2 является ячейка с адресом 101. Ячейки для фиксаторов списков обычно берутся программистом не из СЯ; для них оставляется специальная группа ячеек, хотя, вообще говоря, эти ячейки можно брать также и из СЯ. Указатели списков могут строиться различными способами. Например, указатель списка, так же как и указатель СЯ, может содержать две части: первую, указывающую число членов в данном списке, и вторую, указывающую адрес первого члена списка. Все члены списка соединяются между собой в цепь при помощи адресов связи так же, как это было описано для списка свободных ячеек.

В отличие от списка свободных ячеек, в котором первые части ячеек не используются, в списках объектов первые части ячеек содержат машинные наименования (адреса их записей) объектов, являющихся членами данных списков.

В табл. 2 показан один цепной список, содержащий объекты (1, 2, 3, 6, 9, 11). Все остальные ячейки соединены в СЯ. Буквы КС означают условно код конца списка. Как видно из приведенного примера, ячейки с членами цепных списков могут размещаться в памяти в произвольном порядке; связи их между собой обеспечиваются адресами связи. При этом не требуется заранее выделять под каждый список определенное число ячеек. Эти ячейки берутся из общего резерва СЯ для всех списков по мере их нарастания. Таким образом, обеспечивается большая гибкость в использовании памяти.

Другим важным достоинством цепного способа организации списков является удобство включения новых членов списков и исключения ненужных членов списков. Как включение членов, так и исключение их может производиться из любого места списка без перемещения всех остальных членов.

Включение нового члена в цепной список обычно связано с появлением нового объекта, для которого составляется запись и заносится в одну из свободных зон области памяти, отведенной под записи. Заметим, что в этой области памяти также должен быть организован цепной список свободных зон или ячеек, из которого берутся зоны (группы ячеек) для записей новых объектов и в который включаются освободившиеся зоны после вычеркивания из всех списков ненужных объектов. При занесении новой записи из списка свободных зон берется очередная свободная зона, в которую заносится запись нового объекта. Адрес этой записи становится машинным наименованием нового объекта. Затем по значениям признаков, входящих в запись этого объекта, определяется, к каким спискам он должен быть отнесен, и производится последовательное включение данного объекта в соответствующие списки. Наиболее простым является включение нового члена в начало списка, но можно включать новый член и после любого другого члена без перестановки всех остальных членов списка.

Для включения нового члена в любой список (и в любое место списка) сначала должна быть взята свободная ячейка из СЯ. В левую половину этой ячейки записывается машинное наименование (адрес записи) нового объекта. Затем процесс включения может идти двумя путями. Если новый объект включается в начало списка, то на место адреса связи в новую ячейку записывается значение адреса связи, взятое из фиксатора данного списка, а в фиксатор данного списка на место адреса связи записывается адрес новой ячейки. Если новый член включается куда-то в середину списка, то сначала находится предшествующий член списка, после которого требуется включить новый член, а затем производится замена адресов связи: в предшествующем члене ставится адрес связи, указывающий новый член, а в новом члене ставится адрес связи, взятый из предшествующего члена. В обоих случаях производится увеличение счетчика числа членов в фиксаторе данного списка на единицу. В фиксаторе СЯ (так же как и в фиксаторе свободных зон) при взятии одной ячейки из СЯ (или зоны из списка свободных зон) производится уменьшение на единицу соответствующих счетчиков.

Таблица 3

Адрес ячеек	Содержимое ячеек	
	машинное наименование объектов	адрес связи
100	2	106
101	7	ПО
102	1003	105
103	1004	107
104	1005	КС
105	1002	104
106		109
107	1001	102
108	1000	103
109		КС
ПО	1006	108

В табл. 3 показаны те же списки (СЯ и список объектов), что и в табл. 2, только из СЯ взята одна ячейка (110), а в список объектов (в его начало) добавлен новый объект с машинным номером 1006, ранее не включенный в этот список.

Процесс исключения членов из цепных списков осуществляется также путем замены адресов связи. Находится член, предшествующий исключаемому (для первого члена это будет фиксатор списка) и в предшествующем члене адрес связи заменяется на адрес связи, взятый из исключаемого члена. Одновременно производится уменьшение числа членов в фиксаторе данного списка на единицу. Этот процесс был проиллюстрирован в предыдущем примере при исключении очередной ячейки из СЯ.

Исключение некоторого члена из данного списка может быть связано либо с переносом его в другой список объектов, либо с полной ликвидацией данного члена.

В последнем случае соответствующая списковая ячейка включается в СЯ. При полной ликвидации объекта соответствующая зона памяти с записью этого объекта включается в список свободных зон.

Таким образом, в списковых членах цепных списков в явном виде указываются два адреса: адрес следующего члена списка и адрес записи объекта,

являющегося данным членом списка. Этот способ построения списков является достаточно гибким, но приводит к некоторому увеличению объема памяти за счет хранения в явном виде указанных адресов. Несколько более экономичным в этом отношении является рассматриваемый ниже гнездовой способ.

Гнездовые списки

При гнездовом способе вместо указания явным образом в каждом члене списка адреса следующего члена этого списка, списковые слова, представляющие члены одного списка, располагаются подряд в последовательных ячейках памяти. При этом в списковых словах указываются только машинные наименования (адреса записей) объектов, являющихся членами данного списка, и некоторые дополнительные признаки. Так как состав различных списков переменный, то невозможно заранее точно знать длины списков и выделить им соответствующие места в памяти. Поэтому данный вариант реализуется не в виде сплошных последовательностей списковых ячеек, относящихся к одному списку, а в виде гнезд членов одного списка. Внутри гнезда члены размещаются подряд, а связь между гнездами осуществляется при помощи адресов связи. Отсюда происходит название «гнездовой» способ. Этот способ является сочетанием последовательного и цепного способов и обладает некоторым преимуществом перед указанным выше способом построения цепных списков в отношении расхода памяти и скорости просмотра списковых членов внутри гнезда. При гнездовом способе списковые слова содержат кроме машинных наименований объектов, являющихся членами списка, два признака: признак переходного слова (П) и признак конца списка (КС). Переходные слова служат для связи между гнездами одного списка. В табл. 4 и 5 показан пример гнездового списка. Так же как и при цепном способе, для каждого гнездового списка должен быть выделен фиксатор (указатель) списка, т.е. фиксированная ячейка, в которой хранится адрес, показывающий положение первой ячейки первого гнезда этого списка. Свободные ячейки при гнездовом способе объединяются не в цепи свободных ячеек, а в цепи свободных гнезд ячеек. Подробнее гнездовой способ будет описан в гл. 6 § 16.

Таблица 4

Адреса ячеек	Списковые слова			
	П	КС		
100			105	Фиксатор свободных гнезд
101			108	Фиксатор списка объектов
102	0	0	1004	2-е гнездо
103	0	0	1003	
104	1	0	114	
105		0	111	Свободное гнездо
106				
107				
108	0	0	1000	1-е гнездо
109	0	0	1001	
НО	1	0	102	
111		1		Свободное гнездо
112				
113				
11-1	0	1	1002	3-е гнездо (неполное)
115				
116				

Таблица 5

Адреса записей	Записи объектов
1000	*N*1, *x*1, *y*1, *z*1
1001	*N*3, *x*3, *y*3, *z*3
1002	*N*9, *x*9, *y*9, *z*9
1003	*N*5, *x*5, *y*5, *z*5
1004	N4, x4, y4. z4

Описок объектов в табл. 4 состоит из трех гнезд, причем в третьем гнезде занята всего одна ячейка. Список свободных гнезд состоит из двух гнезд. Свободные гнезда соединяются при помощи адресов связи, помещаемых в первых ячейках гнезд. Во втором свободном гнезде, в 1-й ячейке, адрес связи отсутствует, а признак КС равен 1, что указывает на конец списка свободных гнезд.

Узловые списки

Узловой способ построения списков является обобщением цепного способа и служит для образования многосписковых структур. В отличие от цепных списков, в которых от каждого члена списка может быть сделан переход только к одному следующему члену списка, в узловых списках от каждого члена списка могут быть сделаны переходы ко многим другим членам, т. е. каждый член является узлом пересечения многих списков. При узловом способе все списковые слова, представляющие один и тот же объект в разных списках, располагаются в памяти машины не независимо друг от друга, как это имеет место при построении простых цепных списков, а подряд. Обычно они располагаются совместно в одной ячейке памяти или в группе последовательно расположенных ячеек, образуя так называемый узел. Можно также сам узел представлять в виде цепного или гнездового списка. Описательная информация о данном объекте (запись объекта) может располагаться либо совместно с его узлом, либо в узле будет указываться машинное наименование объекта, т. е. адрес, определяющий положение записи объекта в памяти машины.

Совместное размещение узлов и записей объектов удобно при постоянном составе узлов и записей, раздельное их расположение — при переменном составе. В узловых списках (с объединенным или раздельным расположением узлов и записей) нет необходимости повторять в каждом списковом слове, входящем в состав узла, наименование данного объекта. В состав каждого узла входит одно слово, играющее роль заголовка узла, и нескольких списковых слов. В заголовке узла указывается наименование объекта (адрес записи) и некоторая дополнительная информация об объекте и самом узле (например, число списковых слов в узле). В списковых словах содержатся адреса связи, определяющие положение следующих членов соответствующих списков. Кроме того, могут указываться признаки членов списков, по которым производится их поиск, и некоторые дополнительные признаки (тип спискового слова, признак исключения объекта из данного списка и др.). Для каждого объекта образуется свой узел и своя запись, содержащая информацию о данном объекте. Общие признаки всех членов данного списка могут указываться не в каждом члене списка, а только в указателе (фиксаторе) списка (ячейке, хранящей адрес первого члена списка и некоторые другие данные о списке).

Используя узловой способ, можно строить списковые структуры, отражающие наличие сложных

ассоциативных связей между большим количеством различных объектов. Каждой отдельной цепочкой адресов связи объединяются в единые списки те объекты, у которых имеются одинаковые значения определенных признаков.

В табл. 6, 7, 8 показана схема построения узловой многосписковой структуры с раздельным расположением узлов и записей объектов. В данном примере в каждой ячейке размещается по одному списковому слову, состоящему из поискового признака (ПП) и адреса связи (АС). Каждый узел располагается в трех ячейках. Первые ячейки узлов отводятся под наименования объектов, представляющие собой адреса их записей. Поисковый признак служит для выбора нужного объекта при просмотре списков. Поисковый признак обычно состоит из двух частей: наименования признака и его значения. Наименование признака является общим для всех членов (узлов), входящих в данный список, а значения могут быть различными. Бывают случаи, когда в список объединяются только члены, имеющие одинаковое значение данного признака; тогда эти значения могут указываться только один раз в фиксаторе списка. В качестве кодов наименований могут использоваться адреса фиксаторов списков. Наименования признаков необходимо указывать в соответствующих списковых словах всех узлов в тех случаях, когда предусматривается оперативное исключение выбывающих объектов из всех списков, в которые он входит. При этом наименования признаков необходимы для того, чтобы, выполняя исключение некоторого объекта, иметь возможность по кодам наименований признаков, содержащимся в его узле (в соответствующих списковых словах), определить, к каким спискам относится этот объект. Если не требуется предусматривать возможность оперативного исключения объектов из списков, то коды наименований признаков можно не помещать в списковых словах, а указывать их только в фиксаторах соответствующих списков (или вообще не указывать, если этими кодами являются адреса фиксаторов).

В качестве адреса связи в списковом слове указывается адрес первой ячейки узла и номер спискового слова в узле, относящегося к данному цепному списку. Нумерация списковых слов идет без учета заголовка узла. Этот номер при записи на бумаге отделяется обычно от собственно адреса связи запятой. Если принято правило, что списковые слова, образующие один список, располагаются в узлах на одинаковых по порядку местах, то номер спискового слова в узле можно указывать только один раз в фиксаторе данного списка. Ассоциативные узлы могут быть переменного или постоянного состава. Постоянный состав узлов упрощает их обработку (перепись, включение и исключение), но приводит к излишнему расходу памяти, так как при этом, если какой-нибудь объект не входит в список, соответствующая ячейка спискового слова не используется.

В табл. 7 показана многосписковая структура с постоянным составом узлов (по три ячейки в узле) и постоянным закреплением списковых слов за списками. Первое списковое слово во всех узлах в нашем примере закреплено за списком *А,* а второе списковое слово — за списком *Б,* поэтому номера списковых слов при адресах связи не указываются. В общем случае списковые слова, занимающие одно и то же место в разных узлах (например, первые списковые слова), могут использоваться для построения нескольких списков, если эти списки строятся для взаимно исключающих значений признаков. Ясно, что при этом один и тот же объект (один и тот же узел) может одновременно входить только в один из таких списков.

В табл. 8 показан участок списковой области памяти с записями объектов, а в табл. 6 — участок памяти с фиксаторами списков *А и Б.*

В каждом фиксаторе списка указаны адрес первого члена списка, порядковый номер списковых слов в узлах, соответствующих данному списку, общее для всех членов списка значение признака объектов, объединенных в список, и число членов в списке. В качестве кодов наименований признаков в данном примере приняты адреса фиксаторов списков, которые указываются в соответствующих списковых словах. 202

Список *А* содержит все объекты, причем они следуют в списке в порядке возрастания их номеров. Список *Б* содержит два объекта *N8* и ЛИ. Концы списков обозначены условными кодами КС.

Таблица 6

	Адреса ячеек	Адрес первого члена списка	Номер спискозого слова	Значение признака	Число членов в списке
Фиксатор списка *А*	001	108	1	ПП1	4
Фиксатор списка *Б*	002	114	2	ПП2.	2

Таблица 7

	Адреса ячеек	Содержание ячеек ассоциативных узлов	
		ПП	АС
Узел для 5-го объекта	105	1026	
	106	001	114
	107	000	000
Узел для 1-го объекта	108	1024	
	109	001	111
	110	002	КС
Узел для 3-го объекта	111	1028	
	112	001	105
	113	000	000
Узел для 8-го объекта	114	1030	
	115	001	КС
	116	002	108
	117		

Таблица 8

Адреса ячеек записей	Записи объектов	
1024 1025	*N*1	1-ый объект
	*x*1	
	*y*1	
	*z*1	
1026 1027	*N*5	5-ый объект
	*x*5	
	*y*5	
	*z*5	
1028 1029	*N*3	3-й объект
	*x*3	
	*y*3	
	*z*3	
1030 1031	*N*8	8-ый объект
	*x*8	
	*y*8	
	*z*8	

Свободные списковые слова в узлах должны быть заполнены нулями. Это будет свидетельствовать о том, что данный объект не входит в соответствующий список. При практическом построении списковых структур рассмотренных типов (цепной, гнездовой, узловой) используются некоторые дополнительные приемы, в частности, применяются специальные слова (структурные слова), дающие возможность построения разветвлений в списках и отсылающие к фиксаторам подсписков. Строение фиксаторов может быть самым различным. С их помощью не только ведется учет числа членов в списках, но и осуществляются переходы к подспискам, обеспечивается возможность повторного использования ячеек гнездовых списков, освободившихся после исключения выбывших членов, просмотры и наращивания гнездовых списков и другие функции.

Способы адресации цепных списков

Можно указать четыре способа адресации цепных списков.

Полная прямая адресация. Адреса связи в списковых словах представляют собой полные прямые адреса ячеек ЗУ, используемого для размещения списков. При этом способе обеспечивается гибкость и простота программирования и упрощается построение схем адресации. Объекты имеют во всех списках одни и те же машинные наименования (свои полные прямые адреса), которые могут использоваться для их поиска. Недостатком такого способа является большая разрядность адресов связи, которая должна охватывать весь объем памяти, а отсюда и большой расход памяти под размещение списков.

Относительная плавающая адресация. При этом способе адрес связи в списковом слове указывает разность адресов (с соответствующим знаком) данного члена и следующего члена списка. Если разность адресов двух соседних членов списка превышает отведенную для адресов связи разрядную сетку, то используется непрямая адресация. При этом адрес связи указывает адрес ячейки, в которой находится полный прямой адрес следующего члена списка. Так как для хранения всех непрямых адресов может быть выделена определенная область памяти (например, начальная зона), то для указания этих ячеек не потребуется большая разрядность адресов. Относительная плавающая адресация в сочетании с непрямой адресацией может использоваться также и для указания адресов условных и безусловных переходов в командах. Относительные и непрямые адреса в списковых словах различаются соответствующим признаком.

Достоинства данного способа:

— экономия объема памяти, расходуемого пол адреса связи, так как относительные плавающие адреса могут иметь существенно меньшее количество разрядов (на 50% и больше);

— независимость адресов связи в списках и адресов условных и безусловных переходов в программах от расположения списков и программ в памяти машины.

Недостатки данного способа:

— наличие некоторых ограничений, налагаемых на расположение членов списков в памяти. Необходимо, чтобы члены одного и того же списка располагались в памяти по соседству (в одной и той же зоне), чтобы свести к минимуму число непрямых адресаций. В связи с этим возникает необходимость учитывать заполнение и освобождение отдельных зон памяти (нужны отдельные цепи свободных ячеек по зонам памяти);

— невозможность использования плавающих относительных адресов в качестве наименований объектов, так как адреса, обеспечивающие переходы к одному и тому же объекту из разных точек списка или разных списков, будут различными;

— усложнение схем адресации в связи с необходимостью сочетания непрямой и относительной адресации и преобразованием относительных адресов связи для выработки действительных (полных) адресов переходов.

Относительная индексная адресация. Адреса связи в списковых словах указывают относительные адреса следующих членов списка не по отношению к данному члену списка, а по отношению к некоторой постоянной (начальной) точке массива или зоны памяти. В этом случае для получения действительных адресов членов списков необходимо относительные адреса связи суммировать с некоторой постоянной для всего списка величиной, представляющей собой начальный адрес зоны (массива). Начальный адрес может устанавливаться на индексном регистре и автоматически складываться с относительными адресами.

При необходимости перехода к членам, находящимся за границами данной зоны, так же как и в предыдущем случае, должна применяться непрямая адресация.

Достоинства способа те же, что и в предыдущем случае, и, кроме того, в пределах зон (в одной зоне располагаются, как правило, списки объектов одного и того же типа) относительные индексные адреса связи являются одинаковыми во всех списках, в которых они используются, поэтому эти адреса могут использоваться в качестве наименований объектов для их поисков. Недостатки способа те же, что и в предыдущем случае, за исключением упомянутой выше возможности использования адресов в качестве наименований объектов.

Кроме того, использование при программировании относительных адресов, имеющих жестко закрепленное начало и являющихся только положительными числами, проще, чем плавающих относительных адресов, которые могут иметь положительные и отрицательные значения.

Сравнивая относительную плавающую и относительную индексную адресации, необходимо заметить, что плавающая адресация обеспечивает в принципе возможность произвольного расширения области памяти машины (на которую распространяются ассоциативные связи) путем последовательных переходов от одних точек памяти к другим. При этом каждый раз переход совершается на шаг, не превышающий предельного размера относительного адреса.

В то же время индексная адресация жестко ограничивает область действия ассоциативных связей предельным размером относительного адреса, и для выхода из этой области необходима непрямая адресация.

Непрямая адресация. Выделяется специальная зона памяти для непрямой адресации с числом ячеек, равным максимальному числу всех объектов различных типов, на обработку которых рассчитывается система. Содержимым этих ячеек будут полные прямые адреса первых ячеек записей (групп ячеек) объектов, а также

некоторые дополнительные признаки объектов (например, признак выбытия объекта). Непрямые адреса будут иметь существенно меньшую разрядность, чем полные прямые адреса записей объектов, за счет того, что общее число записей на один или два порядка меньше, чем общее число ячеек в памяти. Непрямые адреса могут служить непосредственно машинными наименованиями объектов и могут быть использованы в качестве адресов связей в цепных или гнездовых списках.

Указанный способ выгодно применять при совместном размещении ассоциативных узлов и записей объектов, так как при этом возрастает среднее число ячеек, занимаемых информацией об одном объекте.

Достоинства этого способа:

— уменьшение размеров адресов связей в списках;

— возможность записи в памяти машины информации об объектах различных типов в произвольном порядке, а не по зонам.

Недостатки данного оспособа:

— необходимость двойного обращения к памяти при адресации к записям объектов, что понижает скорость выборки и записи данных;

— необходимость рассчитывать разрядность адресов связи, исходя из общего числа объектов всех типов (а не одного типа, как при относительной адресации), что увеличивает размер адресов связи;

— необходимость иметь дополнительную память непрямой адресации, что уменьшает экономию в объеме памяти, получаемую за счет сокращения размера адресов связи.

§ 14. АССОЦИАТИВНЫЕ СПИСКОВЫЕ СТРУКТУРЫ

Основные типы ассоциативных списковых структур

Мы рассмотрели ряд способов построения списков: последовательный, цепной, гнездовой и узловой. Считалось, что членами этих списков являются отдельные объекты и каждому объекту соответствует одно списковое слово. Было также сказано, что членами списков могут быть другие списки, так называемые подсписки, которые вместе с основным списком образуют ассоциативную списковую структуру. В общем случае ассоциативной списковой структурой называется любая совокупность взаимосвязанных списков и подсписков.

Списковые структуры можно разделить на два основных типа: *объектные* и *признаковые.* В объектных структурах распределение объектов по спискам осуществляется относительно некоторых конкретных объектов, принимаемых за опорные точки системы деления или группирования объектов. Объектные структуры могут быть двух видов: включающие и исключающие.

Включающие объектные структуры используются для группирования однородных объектов, они строятся таким образом, что опорные объекты, от которых отходят подсписки, соответствующие определенным подразделениям группирования, сами входят в состав подразделений в качестве их первых членов. Причем от одного объекта может отходить несколько подсписков понижающихся уровней, и объект будет одновременно членом всех подсписков. Исключающие объектные структуры используются для группирования разнородных объектов, они строятся таким образом, что опорные объекты сами не являются членами отходящих от них подсписков.

Примером включающей объектной списковой структуры может служить структура, возникающая при решении задач селекции некоторых подвижных объектов методом последовательного деления всей совокупности этих объектов на группы, когда один и тот же объект может входить в состав нескольких групп объектов различной степени укрупнения в качестве их опорного объекта, т. е. объекта, относительно которого идет формирование группы.

Примером исключающей объектной структуры может служить система деления некоторой населенной территории: сначала вся территория делится на автономные части, представляющие собой страны. Страны образуют список объектов верхнего уровня. Каждой стране может соответствовать свой список областей, являющийся списком второго уровня. Каждой области может соответствовать свой список районов — список третьего уровня и т. д. В этом случае опорные объекты системы деления сами не являются членами отходящих от них списков.

Объектные ассоциативные структуры имеют древовидный или вообще разветвляющийся характер. Верхний уровень таких структур образует цепной список некоторых объектов, каждый из которых может быть точкой ответвления цепного подсписка (или нескольких подсписков) других объектов. Эти цепные подсписки будут представлять собой второй уровень данной списковой структуры; от них могут ответвляться цепные подсписки третьего уровня и т. д. Поиск объектов в объектных ассоциативных структурах осуществляется путем просмотра сначала цепного списка верхнего уровня и отбора в нем по заданным признакам объектов и ответвляющихся от них подсписков второго уровня. Затем идет просмотр отобранного подсписка второго уровня и отбор в нем уже по другим признакам объектов и ответвляющихся от них подсписков третьего уровня и т. д. до тех пор, пока не будут исчерпаны либо все уровни структуры (по просматриваемой ветви), либо заданный набор поисковых признаков.

При цепном способе построения списков объектные ассоциативные структуры образуются при помощи списковых слов двух видов: объектных и структурных. Эти слова различаются дополнительным указателем *S*, который для объектных списковых слов равен пулю, а для структурных списковых слов — единице.

Объектные и структурные списковые слова обычно имеют одинаковые форматы; в простейшем случае каждое слово состоит из трех частей (слогов): указателя вида слова *S,* наименования члена списка *H,* адреса связи АС.

В объектных списковых словах *H* представляет собой наименование объекта, являющегося членом списка. Таким машинным наименованием служит обычно адрес записи (формуляра объекта), содержащей собственную информацию об объекте. В структурных списковых словах *H* означает адрес вершины подсписка, ответвляющегося от того члена объектного списка, который стоит непосредственно перед данным структурным

словом.

Таким образом, включая после любого объектного спискового слова одно или несколько структурных слов, можно образовать один или несколько подсписков, ответвляющихся от данного объекта. На рис. 14 показан пример объектной ассоциативной структуры с двумя подсписками, ответвляющимися от основного списка. В основном списке находятся объекты *N*1, *N*2 и *N*4. От объекта *N*1 (после него) отходит подсписок, содержащий объекты *N*3 и *N*6. От объекта *N*4 отходит другой подсписок, в котором находится один объект *N*5.

Цифры, стоящие перед прямоугольниками, обозначают адреса соответствующих ячеек памяти. Символы КС на местах адресов связи обозначают, как обычно, концы списков. В структурных списковых словах адреса связи (АС) как бы заменяют собой адреса связи (АС) в предшествующих объектных списковых словах, от которых производится ответвление подсписков.

В признаковых структурах в качестве опорных точек системы классификации объектов используются не конкретные объекты, а определенные признаки или комбинации признаков, в общем, не связанные с конкретными объектами. Признаком объекта называется любое свойство, используемое для поиска и характеризующееся

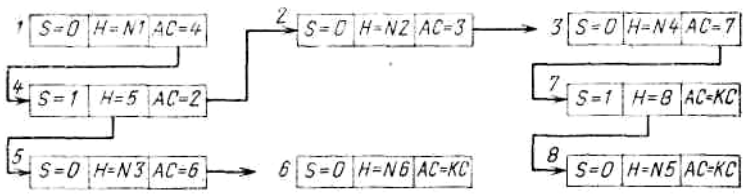


РИС. 114. Объектная ассоциативная структура.

наименованием и значением. Для каждого признака должно существовать одно наименование (код признака) и ряд значений, причем считается, что значения одного признака являются взаимно исключающими. Это означает, что один и тот же объект в одно и то же время не может характеризоваться несколькими значениями одного признака (т. е. он не может быть одновременно, например, белым и красным или большим и маленьким и т. д.). Значения признака могут носить качественный характер (цвет, вкус и т. д.), и тогда они представляются строчными величинами, а могут иметь количественный характер и представляться величинами типа целый или вещественный. Значения могут также представляться логическими величинами. В этом случае признаки называются альтернативными.

В общем случае ряд признаков может образовывать иерархическую систему и значения одного признака явятся точками разветвления для деления объектов по другому признаку. Например, сначала производится деление объектов по цвету, затем объекты одного цвета делятся по весу, затем объекты одного цвета и веса делятся по размеру и т. д. 210

Можно выделить два основных пути организации поиска объектов (или списков объектов) в признаковых структурах:

А. *Структурный путь,* реализуемый в виде различного рода поисковых деревьев (древовидных признаковых структур).

Б. *Вычислительный путь,* реализуемый в виде различного рода алгоритмов вычисления адресов объектов (адресов их записей или формуляров) по значениям признаков этих объектов.

Рассмотрим структурный путь. В общем случае древовидные признаковые структуры могут состоять из двух частей: собственно поискового дерева, определяющего систему классификации объектов по заданному набору признаков, и совокупности списков объектов, объединяющих объекты, обладающие определенными значениями признаков.

Способ поисковых деревьев имеет большое количество различных модификаций. Мы рассмотрим две модификации этого способа:

а) позиционное поисковое дерево, структура которого определяется заранее в процессе программирования. При этом способе каждый признак представляется определенным разрядом (позицией) в некотором много разрядном числе, заданном в позиционной системе счисления. Допустимый набор признаков и число значений каждого признака заранее фиксированы;

б) наращиваемое поисковое дерево, формируемое в процессе включения в систему новых объектов. При этом набор допустимых значений признаков заранее не фиксируется; он может произвольно изменяться в соответствии с фактическими значениями признаков, имеющимися у включаемых в списки объектов.

В качестве членов объектных списков в общем случае могут фигурировать также ассоциативные списковые структуры различных видов, в том числе древовидные признаковые структуры, объектные структуры, просто объекты и т. д.

Признаковые структуры с позиционными поисковыми деревьями

В данных признаковых структурах наборы значений признаков, которыми обладают те или иные объекты, представляются в виде многоразрядных чисел; при этом количество разрядов соответствует количеству признаков. За каждым признаком закрепляется определенный разряд, и количество значений этого разряда соответствует количеству значений соответствующего признака. В общем случае может получиться система счисления с различным количеством значений разных разрядов, т. е. с разными основаниями. Ясно, что наиболее просто оперировать с системой, имеющей одно основание, однако это может привести в некоторых случаях к неэкономной кодировке значений признаков, так как нужно будет выбирать основание системы счисления, исходя из признака, имеющего наибольшее число значений. При этом для других признаков эта возможность кодирования не будет использоваться в полной мере.

Позиционное поисковое дерево строится обычно следующим образом. В вершине дерева помещается один список, который будет списком верхнего (1-го) уровня, этот список осуществляет деление всех объектов по одному какому-нибудь признаку (например, цвету). От каждого члена этого списка отходят подсписки второго уровня, соответствующие делению объектов по другому признаку (например, весу). От членов этих подсписков отходят подсписки третьего уровня, соответствующие делению объектов по третьему признаку, и т. д. Членами подсписков самого нижнего уровня дерева будут объекты, обладающие теми значениями признаков, которые соответствуют всей цепочке членов подсписков всех уровней дерева, ведущей к данному члену нижнего подсписка. Вместо отдельных объектов в подсписках дерева нижнего уровня могут фигурировать списки объектов, обладающих заданным набором признаков. Такие списки объектов мы будем называть объектными списками. В общем случае членами подсписков нижнего уровня в поисковых деревьях могут быть не только объекты и объективные списки, «о и другие списковые структуры (объектные и признаковые).

Позиционные поисковые деревья удобно использовать для поиска объектов в тех случаях, когда задается полный набор признаков и порядок расположения этих признаков соответствует порядку построения поискового дерева, т. е. следованию признаков по уровням дерева. Тогда сначала просматривается список верхнего уровня и в нем отыскивается член, имеющий значение, соответствующее значению первого признака (старшего разряда) в заданном наборе. В подсписке, отходящем от найденного члена, находится член, имеющий значение, соответствующее второму разряду числа, и т. д.

Если же для поиска задается неполный набор признаков и они заданы в другом порядке, то процесс поиска усложняется. При этом необходимо просматривать для каждого незаданного признака все его значения, т. е. вести поиск сразу по нескольким параллельным ветвям дерева. Для облегчения поиска объектов по неполным наборам признаков могут быть использованы поисковые деревья с горизонтальными или поперечными связями. Как уже говорилось, поисковые признаковые деревья представляют собой иерархическую систему классификации объектов по заданному набору признаков. В этих деревьях в подсписках различных уровней кроме самого верхнего уровня могут повторяться однотипные подсписки, осуществляющие деление объектов по одному и тому же признаку. Например, если мы делим объекты по цвету и по весу и решили построить список верхнего уровня по цвету, то списки значений признака второго уровня, осуществляющие деление объектов по весу, могут частично или полностью повторяться. Аналогичным образом могут многократно повторяться и подсписки других уровней.

В поисковых деревьях с поперечными связями указанные однотипные подсписки или даже однотипные члены в этих подсписках объединяются в новые цепные списки при помощи дополнительных адресов связи; эти связи как бы пересекают древовидную структуру в горизонтальном направлении. Использование таких связей позволяет в некоторых случаях повысить эффективность поисков объектов по различным неполным комбинациям признаков по сравнению с обычными древовидными структурами. При этом каждый горизонтальный цепной список имеет свою вершину и кодом данного признака считается адрес ячейки, представляющей собой вершину этого горизонтального списка. Таким образом, любой признак будет иметь только один код, и, задавая комбинации признаков в виде последовательностей их кодов, можно однозначно находить соответствующие им списки объектов. Другим вариантом построения признакового поискового дерева является построение его без поперечных связей, но с указанием в каждом члене подсписков поискового дерева кода наименования признака того подсписка, который отходит от этого члена; значения же этого признака будут указываться специальным кодом в каждом члене этого ответвляющегося подсписка. Таким образом, в каждом члене любого подсписка помимо двух адресов связи будут находиться два кода признака: код значения данного признака и код наименования следующего признака.

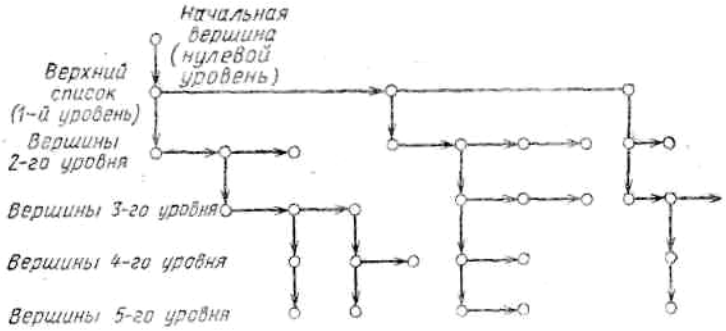


РИС. 15. Схема поискового дерева.

Поисковые деревья могут рассматриваться как ориентированные графы, имеющие одну начальную вершину и совокупность промежуточных и оконечных вершин. Вершины соединяются между собой ребрами. В каждую вершину может входить только одно ребро, а выходить может несколько. Все вершины дерева делятся на ряд уровней.

Вершины, из которых не выходит ни одно ребро, называются оконечными. Последовательность вершин и ребер, ведущая от начальной вершины к какой-либо оконечной вершине, называется путем. С точки зрения применения цепных списков для построения поисковых деревьев удобно представлять эти деревья графически следующим образом (рис. 15): будем считать, что все ребра дерева могут быть двух видов: вертикальные и горизонтальные. В списковых членах горизонтальным ребрам соответствуют одни адреса связи (например, первые), а вертикальным ребрам — другие адреса связи (например, вторые).

Из начальной вершины может выходить только одно вертикальное ребро. Переход от одной вершины к другой, осуществляемый с помощью вертикального ребра, будем считать переходом на другой уровень дерева

(переход к подсписку), а переход от одной вершины к другой, осуществляемый при помощи горизонтального ребра, означает перемещение вдоль одного уровня дерева (вдоль одного подсписка).

Таким образом, совокупность вершин, связанных горизонтальными ребрами, соответствует одному подсписку. Номер уровня любой вершины определяется количеством вертикальных ребер, ведущих от начальной вершины к данной вершине.

Поиск объектов в признаковых структурах, обычно, осуществляется в две стадии: сначала по заданным признакам отыскивается с помощью поискового дерева нужный объектный список, а затем производится поиск нужного объекта в данном списке (либо по другим признакам, либо по собственной информации об объекте). Поисковые деревья могут быть либо постоянными, либо переменными. Постоянные поисковые деревья строятся заранее в процессе программирования задачи и охватывают все возможные подразделения классификации объектов. Переменные поисковые деревья образуются в процессе обработки информации и включают только те подразделения классификации, которые относятся к объектам, фактически имеющимся в системе.

В качестве отдельных признаков могут использоваться и сочетания нескольких признаков. Тогда общее число значений такого комбинированного признака будет равно произведению количеств значений элементарных признаков (если совместимы все сочетания значений этих признаков). Количество различных объектных списков, отходящих от поискового дерева, равно общему количеству подразделений классификации нижних уровней (нижние уровни для различных путей не обязательно должны быть одинаковы).

Значения поисковых признаков, имеющих количественный смысл, обычно задаются в десятичной системе счисления. В этом случае можно строить двоично-десятичные поисковые деревья, в которых каждому разряду десятичного числа соответствует двухуровневое поисковое поддерево. На верхнем уровне поддерева строится один подсписок с тремя членами (00, 01, 10), а на нижнем уровне — три подсписка, из которых два первых имеют по четыре члена (00, 01. 10, 11), а третий имеет два члена (00, 01). Из подобных поддеревьев может строиться поисковое дерево для любых многоразрядных признаков, заданных в десятичной системе.

Пример позиционной поисковой структуры для поиска слов в словаре

Одна из типовых задач, встречающихся при обработке больших объемов информации, представленной на естественном языке (машинный перевод, поиск научно-технической информации и т. п.), заключается в нахождении слов в словаре. Если словарь составлен в алфавитном порядке и имеет неизменный характер, то здесь эффективно может быть применен вариант бинарного поиска (деление пополам). Этот метод требует проверки всего $\log_2 N$ членов словаря, где N — общее число членов в словаре. Однако такой словарь, представляющий собой, по существу, последовательный список, неудобен в тех случаях, когда требуется его часто изменять (исключать старые или добавлять новые члены). Как мы уже упоминали, если включаемые члены равномерно распределены по всему объему словаря, то на каждое включение будет приходиться в среднем $N/2$ сдвигаемых членов. Алфавитный словарь, организованный в виде простого цепного списка, удобен для изменений, но требует выполнения слишком многих операций при поиске.

Древовидная признаковая структура представляет собой такую форму организации данных, которая обеспечивает быстрый поиск (почти как при бинарном методе) и легкость изменения словаря.

Опишем вариант построения поискового дерева для такого словаря.

Это дерево строится по цепному способу из списковых слов, состоящих из трех частей (слогов):

1) наименования члена списка *H;* в данном случае наименование представляет собой код некоторой буквы;
2) первого адреса связи АС1, представляющего собой адрес ответвляющегося подсписка;
3) второго адреса связи АС2, представляющего собой адрес следующего члена данного подсписка.

Ниже показан формат ассоциативного слова. Окончание цепных списков обозначается, как обычно, символом КС.

H	AC1	AC2

Верхний уровень поискового дерева представляется цепным списком, в котором каждый член соответствует одной букве алфавита, причем в этот список входят только те буквы, с которых могут начинаться слова (например, буквы Ы, Й, Ъ, Ь не войдут в список верхнего уровня). От каждого члена списка верхнего уровня ответвляется цепной подсписок, включающий в себя буквы, которые могут находиться на втором месте в словах, начинающихся с букв, указанных в соответствующих членах списка верхнего уровня. От каждого члена списка второго уровня отходит подсписок третьего уровня, содержащий буквы, которые могут стоять на третьем месте в словах, имеющих две первые буквы, указанные на соответствующих местах в списках первого, второго уровней и т. д.

На рис. 16 представлена часть списковой структуры для некоторых слов, начинающихся с буквы Д.

Цифры, стоящие перед прямоугольниками, показывают адреса соответствующих ячеек памяти.

В данном примере показан фрагмент списковой структуры без указаний окончаний цепных списков (подсписков) одного уровня. Вместо вторых адресов связи в тех списковых словах, на которых оборваны подсписки, оставлены пустые места, которые обозначают, что этот подсписок можно продолжить.

В данном поисковом дереве различные ветви могут включать в себя общие участки (общие вертикальные звенья), причем ветви, соответствующие различным словам, имеющим одинаковые начальные группы букв, могут иметь различную длину. Например, ветви, соответствующие словам «дом» и «домна», имеют общую часть «дом», а все ветви, показанные на рис. 16, имеют общую вершину, соответствующую начальной букве Д.

Для указания окончания любой ветви используется так называемое списковое ассоциативное слово, в котором на месте наименования члена списка (кода буквы в нашем случае) ставится символ КС. В оконечных списковых словах АС1 используется для указания адреса записи в словаре, содержащей необходимую информацию о данном объекте (например, эквивалент данного слова на другом языке, грамматические сведения о слове и т. д.). Если это оконечное списковое слово используется для указания окончания слова, не являющегося самым длинным из данного семейства слов, то АС2 в этом списковом слове указывает адрес следующего члена в данном

подсписке, который будет указывать следующую букву более длинного слова, имеющего окончившееся слово своей начальной частью. В оконечных списковых словах наиболее длинных ветвей АС2 не используется.

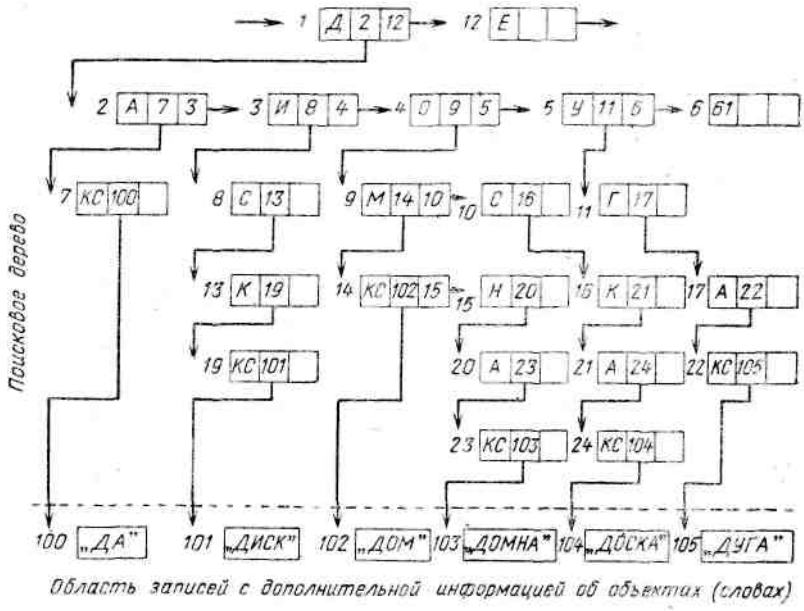


РИС. 16. Древовидный словарь.

Цепная адресация, примененная в данной списковой структуре, обеспечивает легкость изменения как горизонтальных (путем изменения вторых адресов связи АС2), так и вертикальных подписков (путем изменения первых адресов связи АС1).

Естественно, что все ячейки памяти, используемые для хранения членов подобных древовидных структур, могут располагаться в памяти произвольным образом. Скорость поиска объектов с помощью таких структур зависит от числа уровней в структуре и количества! членов в горизонтальных подписках.

К недостаткам подобных словарей следует отнести сравнительно большой расход памяти под поисковые деревья. Этот расход зависит от общего количества слов в словаре, среднего размера слов, а также от степени гнездования слов (от доли слов, имеющих общие начальные части).

Некоторые соотношения, дающие общую ориентировку при выборе размеров подписков и числа уровней в поисковых деревьях, будут приведены в следующем параграфе.

Для повышения скорости поиска слов в таком словаре целесообразно располагать буквы в подписках дерева в порядке убывания их частостей встречаемости на соответствующих местах в словах. Однако такое расположение букв требует большой работы по анализу текстов той области науки, для которой составляется словарь, с целью выявления частостей встречаемости букв на определенных местах в словах и после определенных сочетаний букв.

Можно пойти и по другому пути: не проводить подобных исследований заранее, построить описанный древовидный словарь сначала более или менее произвольным образом и обеспечить в процессе эксплуатации его «самообучение».

Для этого должен осуществляться учет частостей обращений к различным словам (буквам) в данном словаре и соответствующая его перестройка, т. е. изменение положений букв в подписках деревьев.

Очевидно, что при использовании цепного способа построения деревьев такое изменение подписков не вызовет затруднений.

Признаковая структура с наращиваемым поисковым деревом

Способ наращиваемых поисковых деревьев, предложенный В. И. Ландауером и Н. С. Прайсом [14], обеспечивает возможность изменения поисковых деревьев в процессе работы в соответствии с фактическими значениями признаков рассматриваемых объектов.

Наращиваемые поисковые деревья служат для нахождения по заданному набору значений признаков (дескрипторов) адресов объектов (или вершин списков объектов), обладающих этими признаками.

Наращиваемое поисковое дерево может использоваться в качестве составной части в признаковых структурах в сочетании с объектными списками, построенными по различным принципам: последовательному, гнездовому, цепному и узловому. Во всех этих случаях поиск объектов осуществляется в два этапа — сначала при помощи поискового дерева находится нужный объектный список, а затем находится нужная позиция данных (объект) внутри этого списка. Наращиваемое поисковое дерево представляет собой совокупность вершин и ребер, расположенных по определенным уровням табл. 1. Число ребер, отходящих от каждой вершины, определяется из условия минимума времени поиска (см. ниже). Для простоты в примерах рассматривается дерево, у которого от каждой вершины отходит не более трех ребер. Вершины дерева в памяти машины представляются отдельными гнездами ячеек (группами последовательных ячеек *N\, N2, N3* и т. д.). Каждое гнездо содержит списковые слова, размещаемые в отдельных ячейках. Таким образом, в каждом гнезде ячеек размещается последовательный подсписок поискового дерева.

Число членов (списковых слов) в подписке равно числу ребер, отходящих от данной вершины.

Адресом подписка является адрес первой ячейки (первого спискового слова) гнезда. Как показано в табл. 1, каждое списковое слово содержит два слога: признаковый слог и адресный слог.

Признаковый слог представляет собой некоторый код, с помощью которого различаются отдельные вершины дерева; это идентификаторы вершин дерева. Наращиваемое поисковое дерево строится для одного поискового признака, имеющего количественный смысл; наименование этого признака является общим для всего дерева, и поэтому оно в признаковых слогах не указывается. В них указываются только значения этого признака.

Особенностью рассматриваемого дерева является расположение списковых слов в гнездах (подсписках) в порядке возрастания значений поискового признака (например, в гнезде *N*1 *П*0<*П*1<*П*2; в гнезде *N*2 *П*5<*П*1<*П*8 и т.д.). Кроме того, в деревьях рассматриваемого типа значения признака в списковых словах всех подсписков (гнезд), относящихся к одному уровню, также возрастают при переходе от гнезда к гнезду слева направо (например, для всех подсписков первого уровня: *П*5<*П*1<*П*8<*П*9<*П*12<*П*10<*П*15 <*П*7<*П*6).

Адресный слог содержит в себе собственно адрес связи и два служебных разряда КП и КД. Адреса связи служат для указания переходов внутри поискового дерева от членов вышестоящих подсписков к нижестоящим ответвляющимся подспискам.

Служебный разряд КП (конец позиции) является признаком окончания данного гнезда (подсписка, соответствующего определенной вершине дерева). Этот разряд имеет смысл в тех случаях, когда подсписки поискового дерева имеют переменный состав. Служебный разряд КД является признаком окончания дерева; во всех подсписках, кроме подсписков нижнего уровня, этот разряд равен нулю. В списковых словах подсписков нижнего уровня этот разряд равен единице. Наличие единицы указывает на то, что адреса связи в подсписках нижнего уровня дерева являются уже не адресами нижестоящих подсписков дерева, а адресами вершин объектных списков. Служебный разряд КД имеет смысл в тех случаях, когда число уровней в различных ветвях дерева различно.

Для поиска объектов задается значение признака, общее для всех объектов, объединенных в одном объектном списке. Поиск производится путем прослеживания вершин дерева сверху вниз и сравнения заданного значения признака со значением признака в просматриваемых подсписках. Дерево строится таким образом, чтобы в каждом подсписке, ответвляющемся от некоторого спискового слова вышестоящего подсписка, наибольшее значение признака (т.е. значение признака в последнем члене этого подсписка) было равно значению признака в списковом слове вышестоящего уровня. Таким образом, при прослеживании подсписков дерева нужно переходить к нижестоящему подсписку, ответвляющемуся от того спискового слова данного подсписка, в котором значение признака оказалось равным или большим заданного значения признака (при условии, что в предшествующем списковом слове этого подсписка значение признака было меньше заданного).

Таблица 1

Номер вершины дерева	Адрес спискового слова	Списковое слово				
		Значение признака	КП	КД	Адрес связи	
N1	0100 010! 0102	П0 П1 П2	0 0 1	0 0 0	1022 0135 0511	адреса вершин дерева
N2	0135 0136 0137	115 111 118	0 0 1	1 1 1	0612 2031 5042	адреса вершин объектных списков
N3	0511 0512 0513	П8 П12 П10	0 0 1	1 1 1	0256 6011 1531	
N4	1022 1023 1024	1115 П7 П6	0 0 I	1 1 1	2041 2512 0172	

Таким образом, в наращиваемом поисковом дереве подсписок самого верхнего уровня делит весь диапазон изменения значения признака на некоторые достаточно крупные интервалы. Каждый подсписок следующего уровня, т. е. подсписок, ответвляющийся от каждого спискового слова верхнего подсписка, делит соответствующий интервал на более мелкие интервалы. Подсписки третьего уровня делят соответствующие подынтервалы на еще более мелкие подынтервалы и т. д.

Сбалансированным деревом называется такое дерево, у которого число уровней в разных ветвях отличается не больше чем на единицу. При этом образование нового уровня дерева начинается только после заполнения всех свободных списковых слов в подсписках предыдущего уровня. Сбалансированное дерево наращивается по мере добавления новых объектов и образования объектных списков с другими значениями признаков. При образовании нового подсписка дерева берется новое гнездо свободных ячеек памяти, т. е. группа ячеек, расположенных последовательно. Число ячеек в гнезде должно быть равно числу списковых слов, которые должны входить в новый подсписок. Это число должно быть фиксировано заранее и известно в момент образования нового подсписка. В последнем списковом слове нового подсписка записывается значение признака, взятое из вышестоящего исходного спискового слова, а' в предпоследнем списковом слове нового подсписка записывается новое значение признака, необходимость включения которого и обусловила образование нового подсписка. При этом несколько ячеек, находящихся в начальной части нового гнезда остаются временно свободными. В дальнейшем при поиске таких свободных ячеек этот факт устанавливается путем проверки только первых ячеек гнезд.

При записи каждого нового объекта производится проверка наличия в поисковом дереве того значения

поискового признака, которое характеризует записываемый объект. Если это значение признака уже имеется, то новый объект просто включается в соответствующий объектный список без каких-либо изменений в дереве. Если требуемого значения признака в дереве нет, то его включают в нижний уровень дерева в то место, которое соответствует условию монотонного возрастания значений признака. Если в соответствующем подсписке нет свободной ячейки, то производится поиск ближайшей свободной ячейки, в других подсписках нижнего уровня и затем сдвиг вправо или влево всех промежуточных членов в подсписках нижнего уровня с корректировкой значений признака в подсписках вышестоящих уровней.

На рис. 17 дан пример указанной признаковой структуры. Часть, расположенная выше пунктирной линии, представляет собой наращиваемое дерево. В дереве каждый прямоугольник представляет один подсписок, состоящий из трех членов (списковых слов). Для слов указаны только значения признака, а вместо адресов связи используются стрелки, ведущие от исходных слов к ответвляющимся подспискам. Под пунктирной чертой находится область объектных списков, содержащая данные об объектах с их значениями поискового признака, причем объекты с одинаковыми значениями признака объединены в общий список при помощи адресов связи. В данном случае допускается, что один объект может иметь несколько значений поискового признака (это может быть, например, если под одним объектом понимать группу предметов), т.е. различные значения признака, для которого построено данное поисковое дерево, не являются взаимно исключающими.

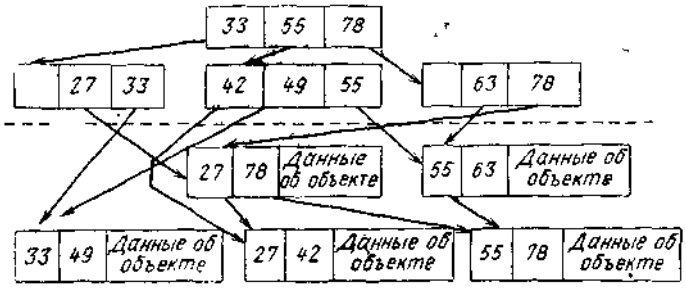


РИС. 17. Ассоциативная структура с наращиваемым деревом.

В вычислительной системе с описанной организацией памяти [И] фактически образуются две ассоциативные структуры: одна для команд программы и вторая для информации об объектах.

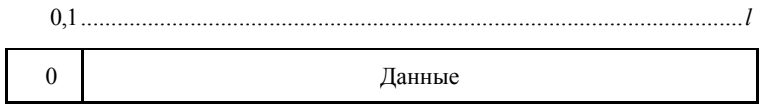
Исходное программирование для этой системы ведется на псевдоязыке, состоящем из набора списковых операторов, который может расширяться по мере надобности. Каждый оператор реализуется при помощи машинной подпрограммы, хранящейся в виде одного подсписка. Работа машины осуществляется по интерпретативному способу, при котором операторы исходной программы последовательно исполняются интерпретирующей программой.

Достоинствами наращиваемого поискового признакового дерева является возможность произвольного деления интервала изменения признака в зависимости от фактического состава поступающих объектов и гибкость использования памяти, не требующая ее предварительного распределения, а также эффективность поиска данных за счет оптимального выбора числа ветвей и уровней в дереве. Недостатком этого способа является необходимость сдвигов значений признака при наращивании дерева.

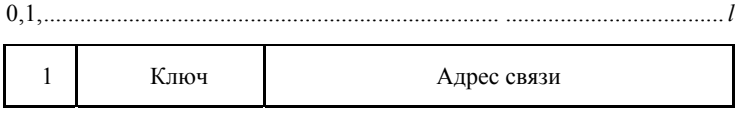
Многосписковые ассоциативные структуры

Н. С. Прайсом и Х. И. Греем предложена организация списковой структуры, аналогичная рассмотренной в предыдущем разделе. Она также основана на цепной адресации. Изложим кратко эту систему, описанную в [12]. Вся память машины разделена на две области: область поисковых деревьев и многосписковую область или область ассоциативных узлов.

Для построения поисковых деревьев и ассоциативных узлов используются слова (называемые катенами) двух видов: слова данных и списковые слова. Оба вида слов имеют одинаковые размеры. Формат слова данных имеет вид



Формат спискового слова имеет вид:



Из указанных слов формируются информационные позиции, т. е. отдельные записи, соответствующие определенным объектам. Эти позиции будут представлять собой в поисковом дереве вершины дерева, а в многосписковой области — ассоциативные узлы. Узлы могут иметь переменный состав, т.е. различное число слов. В каждой адресуемой ячейке памяти может размещаться фиксированное число слов; если требуемое число слов не помещается в *n-й* ячейке, то занимается следующая ячейка (n+1-я ячейка памяти); если же следующая

ячейка окажется занятой, то в последнем слове, находящемся в данной *n-й* ячейке, на месте ключа пишется адрес, указывающий продолжение данной позиции (продолжение ассоциативного узла). В каждом слове выделяется еще дополнительный разряд, указывающий окончание позиции. При помощи списковых слов различные позиции (узлы) объединяются в списки, обладающие одинаковыми ключами (признаками).

Каждый узел может входить во столько списков, сколько списковых слов имеется в узле.

Поиск объектов, обладающих заданным набором признаков, осуществляется в два этапа: сначала при помощи поискового дерева ключ, заданный в запросе, преобразуется в адрес начала списка объектов, обладающих данным ключом; затем просматривается этот список, и отбираются объекты, обладающие всеми заданными в запросе ключами. В [12] рассматривается пример информационной системы для учета кадров, рассчитанной на 1 млн. позиций. Каждое лицо характеризуется 15 признаками, причем каждый признак может иметь 10 значений; таким образом, информация о каждом лице представляется 15-разрядным десятичным числом.

В системе принят постоянный размер позиций, являющихся как вершинами поискового дерева, так и ассоциативными узлами. Каждая позиция содержит пять списковых слов. Для того чтобы представить в одной позиции (с помощью пяти слов) 15 признаков, эти признаки делятся на пять групп по три признака в группе.

Каждая группа из трех признаков образует один надпризнак, который может иметь 1000 различных значений, и, следовательно, одному надпризнаку может соответствовать 1000 объектных списков. Всем пяти надпризнакам соответствует 5000 различных объектных списков, которые могут использоваться в системе.

Поисковое дерево состоит из пяти и частично из шести уровней: на первом (верхнем) уровне имеется одна позиция — последовательный список, состоящий из 5 слов, от которого отходят пять ветвей к пяти подспискам второго уровня. Каждый подсписок второго уровня соответствует одному надпризнаку. От этих подсписков ответвляются подсписки третьего уровня, тоже состоящие из пяти членов. От подсписков третьего уровня отходят такие же подсписки четвертого уровня, а от подсписков четвертого уровня отходят подсписки пятого уровня.

В этих подсписках первые четыре члена указывают на вершины объектных списков, расположенных в многосписковой области, а последний член каждого подсписка пятого уровня указывает на подсписок шестого уровня, состоящий из четырех членов, каждый из которых содержит адрес вершины списка в многосписковой области.

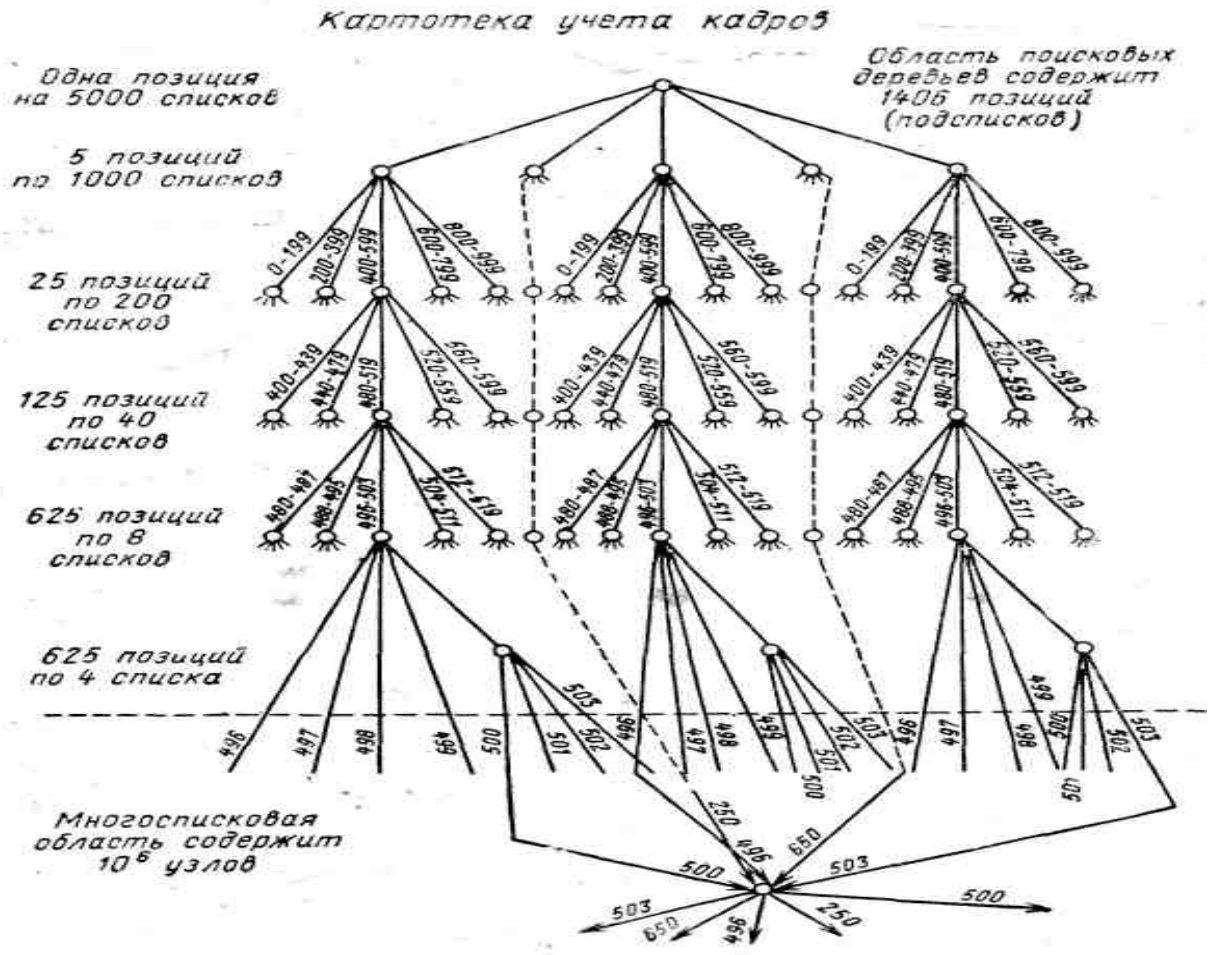


РИС. 18. Многосписковая структура.

На рис. 18 приведена схема описанной ассоциативной структуры. Пунктирными линиями показаны связи данного ассоциативного узла с поисковыми поддеревьями, соответствующими второму и четвертому надпризнакам. Чтобы не загромождать рисунок, связи показаны упрощенно. В действительности они имеют вид таких же ветвящихся деревьев, как и связи для остальных надпризнаков.

Ассоциативный узел будет состоять из пяти списковых слов и нескольких слов данных; для узла, показанного на рисунке, ключи в списковых словах, представляющие собой надпризнаки, будут иметь следующие значения:

500	Адрес связи	250	Адрес связи	496	Адрес связи	650	Адрес связи	503	Адрес связи 1

Для оценки эффективности подобных ассоциативных систем Прайсом и Греем предложен комбинированный критерий, представляющий собой произведение времени поиска заданного объекта на требуемую общую емкость памяти [12].

Более подробно эти вопросы будут рассмотрены в § 15.

Способ вычисляемых адресов

Способ вычисляемых адресов основан на преобразовании заданных наборов поисковых признаков либо в адреса записей с собственной информацией об объектах, либо в адреса ассоциативных узлов объектов или в адреса вершин списков объектов.

Естественными требованиями к таким вычислениям являются однозначность преобразования и соответствие получаемого диапазона изменения адресов размерам области памяти, отведенной для размещения искомых позиций. Различают вычисление прямых адресов, когда в результате получаются непосредственно адреса искомых позиций, и вычисление непрямых адресов, когда в результате получаются адреса ячеек, содержимым которых являются адреса искомых позиций. В общем случае значения признаков объектов могут изменяться произвольно, и многие объекты могут обладать одним и тем же значением данного поискового признака. Если считать, что в одной ячейке может быть записан только один член списка, то естественно все объекты с одинаковым значением поискового признака объединять в цепные списки, а при помощи вычислений определять адреса вершин таких списков.

Существует еще другой так называемый открытый способ размещения и поиска объектов, обладающих одинаковым значением поискового признака. При этом способе первый появившийся объект записывается в ячейку с адресом, полученным в результате преобразования значения признака в адрес, а каждый последующий объект записывается в ближайшую в сторону возрастания адресов свободную ячейку памяти. Поиск членов при этом происходит в два этапа. Сначала по заданному значению поискового признака определяется адрес первого объекта, а затем путем прямого просмотра в сторону возрастания адресов определяются остальные объекты, обладающие тем же значением этого признака. Как запись, так и поиск членов производятся циклично, т. е. после заполнения последней ячейки памяти снова используется первая ячейка и т.д.

Запись непрямых адресов может быть осуществлена двумя способами: для этого выделяются либо специальная группа ячеек памяти, либо определенная часть во всех ячейках памяти.

В последнем случае обеспечивается более широкий диапазон изменения адресов, но значительно возрастает расход памяти, так как во всех ячейках памяти необходимо выделять две адресные части: под вычисляемые (непрямые) адреса вершин списков и под адреса, используемые непосредственно для связи членов внутри цепных списков. Использование одной и той же адресной части в ячейке для обеих целей не представляется возможным, так как могут быть случаи, когда одна и та же ячейка должна указывать положение вершины некоторого цепного списка (т. е. содержать в себе непрямой адрес) и в то же время содержать член некоторого цепного объектного списка со своим адресом связи. Способ вычисляемых адресов удобно 'применять тогда, когда значения признака, по которому производится поиск объектов, изменяются монотонно, или когда их удается аппроксимировать монотонной функцией. Например, этот способ можно применять вместо постоянных поисковых деревьев с одинаковым числом членов во всех подсписках, как показано ниже.

Пусть, например, мы имеем систему классификации объектов, включающую в себя шесть признаков: *П*1, *П2, ..., П*6, каждый из которых может иметь три значения. Сравним для этого случая способы поискового дерева и вычисляемых адресов. Строим три поисковых дерева, объединив в каждом из них два признака в один надпризнак.

РИС. 19. Таблица значений признаков (двоичные числа).

На рис. 19 показана таблица значений признаков от *П1* до *П*6. Объединение в надпризнаки произведено следующим образом: признаки *П1* и *П*2 образуют первый надпризнак, которому соответствует поисковое дерево *I*; признаки *П*3 и *П4* образуют второй надпризнак, которому соответствует поисковое дерево *II*, и признаки *П*5 и *П*6 объединяются в третий надпризнак, которому соответствует поисковое дерево *III.*

На рис. 20 показаны три поисковых дерева *I*, *II, III.* . Подсписки расположены в последовательных ячейках памяти с адресами от 1 до 39. В каждой ячейке имеется две части: значение признака и адрес ответвляющегося подсписка. В подсписках нижнего уровня адресные части указывают положение вершин соответствующих объектных списков (с Al по A27). Над клетками указаны адреса соответствующих ячеек, занимаемых членами подсписков поисковых деревьев.

Используя данные поисковые деревья (*I*, *II* и *III),* можно осуществлять поиск объектов следующим образом:

а) из заданного набора значений шести признаков (*П*1—*П*6) выбираем одну пару признаков (*П*1 и *П*2 или *П*3 и *П*4, или *П*5 и *П*6), которая будет поисковым надпризнаком, и соответствующее этому надпризнаку поисковое дерево. Адрес вершины этого поискового дерева определяем при помощи подсписка верхнего уровня по коду поискового дерева (*I*-01, *II*-10, *III*-11);

б) в выбранном поисковом дереве просматриваем список верхнего уровня и находим в нем член, имеющий то же значение, что и первый признак в выбранном нами поисковом надпризнаке;

в) по адресу связи, имеющемуся в этом члене, находим подсписок второго уровня;

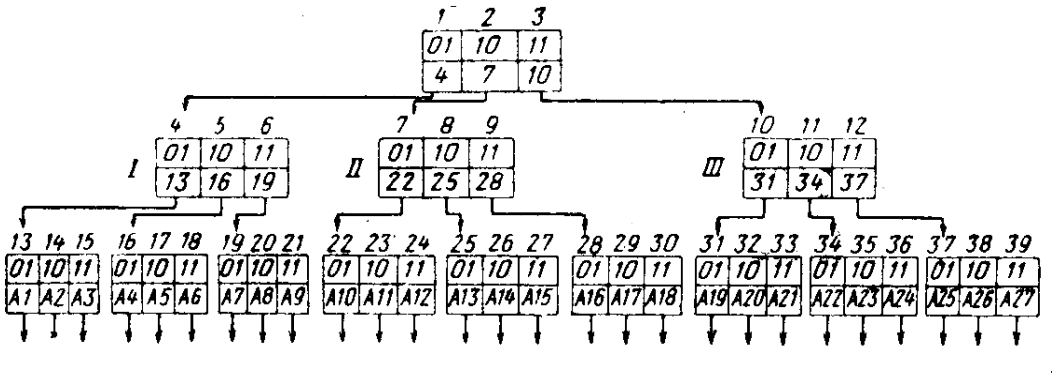


РИС. 20. Ассоциативная структура с постоянным поисковым деревом.

г) просматриваем подсписок второго уровня и находим в нем член, имеющий то же значение признака, что и второй признак, входящий в поисковый надпризнак;

д) по адресу связи, имеющемуся в этом члене, находим искомый объектный список;

е) просматриваем найденный объектный список и отбираем в нем объекты, удовлетворяющие заданным значениям остальных четырех признаков (не вошедших в состав поискового надпризнака).

При этом придется в среднем просматривать половину членов подписков первого и второго уровней и половину членов объектного списка (считаем, что все признаки равновероятны).

Поиск требуемого объектного списка (или отдельного объекта) при помощи поискового дерева может быть заменен вычислением адреса ячейки, в которой находится адрес вершины объектного списка. Для этого вместо трех поисковых деревьев *I, II, III* достаточно иметь группу из n последовательно расположенных ячеек памяти, в которых хранятся адреса вершин объектных списков ($A1$—An), где n — общее число объектных списков. Пусть $b0$ означает адрес ячейки, после которой располагаются ячейки указанной группы. При сделанных допущениях о системе кодирования признаков ($П1$—$П6$) и указанном расположении ячеек, содержащих адреса вершин объектных списков, формула для вычисления адресов вершин объектных списков по заданным значениям признаков n_i и n_{i+1} и номеру (коду) поискового дерева k (k=1, 2 или 3) будет иметь вид

$$A = b0 + 3^2(k\text{-}1) + 3(n_i — 1) + n_{i+1}.$$

Достаточно просто могут быть составлены формулы для вычисления адресов при других конфигурациях поисковых деревьев.

Вообще говоря, способ вычисляемых адресов во многих случаях позволяет получить экономию в количестве ячеек памяти и сокращение числа операций при поиске.

Число операций, необходимых для вычисления непрямого адреса, пропорционально числу признаков (т. е. числу уровней дерева) и не зависит от числа значений признаков (т. е. от числа членов в подсписках поискового дерева), в то время как число операций просмотров членов подписков при поиске по поисковому дереву пропорционально произведению числа признаков на число значений признаков (числа уровней дерева на число членов в подсписках). Следует заметить, что при переменных поисковых деревьях и постоянных деревьях, но имеющих различную длину признаковых подписков, необходимы еще ячейки для хранения сведений о числе членов в признаковых подсписках, а алгоритм вычисления непрямых адресов усложняется за счет учета размеров подписков.

Другим примером применения способа вычисляемых адресов может служить алгоритм, осуществляющий регистрацию некоторых объектов в зависимости от их положения на плоскости или в пространстве. Пусть положение объектов определяется прямоугольными координатами *(x, y, z)*. При помощи некоторого заданного числа старших разрядов координат объектов все пространство может быть разбито на области, для которых все находящиеся в каждой области объекты должны объединяться в один список. Из выделенных старших разрядов координат *x, y, z* может быть образовано одно двоичное число, являющееся адресом ячейки, в которой хранится адрес вершины соответствующего списка. Недостатком этого способа является, очевидно, необходимость резервирования ячеек для непрямой адресации всех возможных таких списков, хотя в действительности многие списки могут быть пустыми.

Одной из распространенных модификаций способа вычисляемых адресов является способ поиска слов в словаре при помощи сверток, получаемых по кодам букв, образующих слово.

Пусть, например, все буквы алфавита имеют свои постоянные коды, представляющие собой шестиразрядные двоичные числа. Тогда для любого слова может быть получена его свертка, т. е. двоичное число заданной разрядности (например, 10 или 15 разрядов). Один из простых способов свертывания состоит в том, что коды букв, образующих слово, складываются поразрядно со сдвигом на один разряд вправо или влево. Сдвиг производится в пределах заданной разрядности кода свертки, после чего коды остальных букв складываются без сдвигов. Пусть, например, имеем следующие коды: букв: *а* — 000001, *в* — 000010, е — 001000, *к* — 001010, *р* — 100100, *с*—100101, т — 100110. Тогда десятиразрядная двоичная свертка слова «свертка» может быть получена следующим образом:

Номера разрядов кода свертки

	1	2	3	4	5	6	7	8	9	10
с	1	0	0	1	0	1				
в		0	0	0	0	1	0			
е			0	0 1	1	0	0	0		
р					0	0	1	0	0	
т					1	0	0	1	1	0
к					0	0	1	0	1	0
а					0	0	0	0	0	1
	1	0	0	0	0	0	0	1	0	1

Коды отдельных букв

Код свертки, полученный путем поразрядного (без переносов) сложения кодов букв

Основным недостатком способа сверток является возможность появления неоднозначности сверток, т. е. случаев, когда различные слова после свертывания будут иметь одинаковые коды сверток. Для исключения неопределенности в этих случаях добавляют вспомогательные признаки, введение которых, естественно, усложняет процесс поиска. Кроме того, использование кодов сверток слов в качестве адресов ячеек памяти, в которых хранится информация о данных словах, приводит к неравномерности в заполнении памяти, так как получаемые адреса, вообще говоря, имеют случайный характер.

§ 15 НЕКОТОРЫЕ СООТНОШЕНИЯ ДЛЯ АССОЦИАТИВНЫХ СПИСКОВЫХ СТРУКТУР

Рассмотрим случай симметричного поискового дерева, т. е. такого дерева, в котором все ветви имеют одинаковое число уровней k и от каждого члена подсписков не нижнего уровня отходит подсписок с n членами. От каждого члена подсписков нижнего уровня могут отходить объектные списки с произвольным числом членов m. При указанном строении поискового дерева максимальное число объектных списков будет равно числу членов в подсписках нижнего уровня дерева:

$$N = n^k. \tag{1}$$

Для нахождения какого-нибудь объектного списка по заданной последовательности k признаков необходимо просмотреть k подсписков и выполнить в среднем в каждом подсписке μn проверок членов, где $\mu \leq 1$.

Может быть и такой случай поиска объектов, когда на каждом уровне дерева (в каждом подсписке не нижнего уровня) нужно проверять все n членов подсписка. Это может оказаться необходимым в случае объектных структур, в которых в одном подсписке находится несколько членов, удовлетворяющих условиям отбора, а по условиям поиска требуется отыскать все эти члены. В признаковых структурах в каждом подсписке может быть только один член с данным значением признакового кода, и поэтому просмотр каждого подсписка нужно вести только до встречи с членом, имеющим заданное значение проверяемого признака. Отсюда появляется множитель $\mu \leq 1$. Общее количество проверок членов поискового дерева, необходимых для нахождения требуемого объектного списка, будет равно:

$$L_{ПД} = \mu n k = \mu n \frac{\ln N}{\ln n}. \tag{2}$$

Величина $L_{ПД}$ при фиксированном N будет иметь минимум при $n=e$ (2,71). Таким образом, если выбирать параметры поискового дерева, исходя из требования максимальной скорости просмотра этого дерева, то целесообразно брать ближайшие к e=2,71 целые значения ($n = 2$ или 3).

Оценим ориентировочно количество ячеек памяти, занимаемых поисковым деревом. В подсписках нижнего уровня общее число членов будет равно общему числу объектных списков N. В подсписках предпоследнего уровня будет в n раз меньше членов, а в подсписках уровня, предшествующего предпоследнему, в n^2 раз меньше членов и т. д. Таким образом, общее число членов в поисковом дереве будет определяться геометрической прогрессией:

$$\Phi = N + \frac{N}{n} + \frac{N}{n^2} + \cdots + \frac{N}{n^{k-1}} = N \frac{n(n^k - 1)}{(n-1)n^k}.$$

Заменяя из (1) n на N, получим

$$\Phi = \frac{n(N-1)}{n-1} \approx \frac{nN}{n-1}. \tag{3}$$

Из формулы (3) видно, что, начиная с $n = 2$, для которого $\Phi = 2(N—1)$, величина Φ с ростом n уменьшается и стремится к N при $n = N$. Дальнейшее увеличение n не имеет физического смысла. Таким образом, видно, что с увеличением n, с одной стороны, уменьшается Φ, т. е. объем памяти расходуемой под поисковое дерево, а с

другой стороны, увеличивается число операций поиска (при n>e). Интересно получить совместную оценку, учитывающую расход памяти и количество операций поиска. В качестве такого критерия иногда принимают [12] произведение количества ячеек памяти, требуемых для построения поискового дерева, на число операций проверок членов поискового дерева, необходимых для нахождения нужного объектного списка:

$$K = L\Phi = \mu n \frac{\ln N}{\ln n} \frac{nN}{(n-1)}. \tag{4}$$

Величина K имеет минимум при $n = 4{,}2 \approx 4$.

Такое число членов в подсписках поискового дерева (при прочих равных условиях) будет наиболее выгодным, как с точки зрения расхода памяти, так и с точки зрения скорости поиска.

Число различных объектных списков, в которых может находиться одновременно некоторый объект, определяется количеством списковых слов в ассоциативном узле объекта. Будем считать, что в состав спискового слова входят адрес связи и код признака (дескриптор).

Система классификации объектов, по которой строится поисковое дерево, в общем виде может быть представлена следующим образом. Каждый объект может характеризоваться некоторым количеством (S) признаков (свойств): цвет, вес, длина, координаты и т.д., причем объекты могут обладать не обязательно всеми S признаками. Каждый из признаков может принимать определенное число значений, например для цвета: белый, красный и т. д. Обозначим число различных значений, которые может принимать i-й признак через b_i. Различные значения одного и того же признака, как уже упоминалось, несовместны (взаимно исключающие). Различные признаки являются совместными, хотя некоторые объекты могут и не иметь всех признаков. Для подобной системы классификации могут быть построены различные варианты поискового дерева. Рассмотрим два крайних случая.

1. Для каждого значения каждого признака образуется свой отдельный объектный список, в который будут включаться все объекты, обладающие данным значением этого признака. Общее число объектных списков будет равно

$$N = \sum_{i=1}^{s} b_i \tag{5}$$

Каждый объект может входить в S списков одновременно, и, следовательно, в ассоциативных узлах объектов должно быть по S списковых слов.

2. Для каждой комбинации значений всех S признаков составляется свой отдельный объектный список; такой «список» будет иметь либо один член, либо вообще не будет иметь членов, если считать, что рассматриваемые объекты обязательно различаются между собой значением хотя бы одного признака.

Ясно, что ассоциативные узлы объектов в этом случае будут отсутствовать.

Общее число членов в подсписках нижнего уровня, т. е. число оконечных вершин дерева, будет равно

$$N = \prod_{i=1}^{S} b_i. \tag{6}$$

Это число может значительно превышать фактическое число объектов, рассматриваемых в данной системе. Ясно, что в этом случае (при использовании взаимно-исключающих значений признаков) каждый объект может относиться только к одной оконечной вершине дерева. В первом случае получается минимальное число объектных списков, и поиск нужного объектного списка будет осуществляться наиболее просто, зато в каждом объектном списке будет большое количество членов, поиск которых усложняется. Также значительно возрастает расход памяти под списковые слова в ассоциативных узлах. Во втором случае вся ассоциативная структура вырождается фактически в одно поисковое дерево без объектных списков, которое может оказаться весьма разветвленным (при наличии большого числа признаков с большим числом значений). При этом весь процесс поиска объектов сводится к поиску по поисковому дереву нужной оконечной вершины. Весь расход памяти под ассоциативную информацию будет приходиться на поисковое дерево.

Промежуточные случаи между этими двумя крайними случаями могут быть получены путем разделения всех S признаков на несколько групп и представления каждой из таких групп признаков в виде одного так называемого надпризнака. Значением надпризнака будет комбинация значений признаков, входящих в его состав. Ясно, что общее число различных значений данного надпризнака будет равно произведению количеств значений входящих в его состав признаков. Так как различные значения каждого признака являются несовместными, то и различные значения одного надпризнака также будут несовместными.

Допустим, что мы разделили S признаков на l надпризнаков. Число признаков, объединяемых в i-м надпризнаке, обозначим через h_i. Очевидно:

$$\sum_{i=1}^{l} h_i = S. \tag{7}$$

Число различных значений i-го надпризнака будет равно

$$N_i = \prod_{j=q_i}^{r_i} b_i, \tag{8}$$

где $q_i = \sum_{k=1}^{i-1} h_k + 1,$

$$r_i = \sum_{k=1}^{i} h_k .$$

Мы считаем, что признаки, объединяемые в один надпризнак, располагаются подряд. Это можно всегда сделать, так как никаких ограничений на перестановку признаков не налагается. Теперь мы можем построить для каждого надпризнака свое поисковое дерево, причем общее число оконечных вершин в этом дереве будет равно числу различных значений этого надпризнака. Если все *S* признаков разделены на *l* надпризнаков, то мы можем иметь *l* поисковых деревьев. Для каждого значения надпризнака образуется отдельный объектный список, в который будут включаться все объекты, обладающие этим значением данного надпризнака. Общее число объектных списков, относящихся к некоторому *i*-му надпризнаку, будет определяться формулой (8), а общее число всех объектных списков для всех *l* надпризнаков будет равно их сумме;

$$N = \sum_{i=1}^{l} N_i . \qquad (9)$$

Каждый объект может входить только в один из объектных списков, относящихся к данному надпризнаку; но естественно, что он может входить в несколько объектных списков, относящихся к разным надпризнакам. Максимальное число объектных списков, в которые может входить один объект, равно числу надпризнаков *l*, и поэтому в ассоциативных узлах объектов должно быть предусмотрено *i* списковых слов. В связи с тем что в описанной системе классификации может быть построено не одно, а *l* поисковых деревьев (по числу надприззаков), данную ассоциативную структуру мы будем называть «многодеревной ассоциативной структурой».

В общем случае число уровней в поисковых деревьях, принадлежащих разным надпризнакам, может быть различным; оно будет определяться числом различных значений соответствующих надпризнаков N_i. В ассоциативном узле объекта для каждого надпризнака выделяется одно списковое слово. В этом слове размещается адрес связи, с помощью которого образуется объектный список членов, обладающих данным значением надпризнака; кроме адреса связи в списковое слово входит признаковый код данного объектного списка. Разрядность этого кода выбирается таким образом, чтобы можно было различать N_i объектных списков, соответствующих N_i значениям данного надпризнака. Заметим, что признаковый код в списковых словах, с помощью которых образуются подсписки поискового дерева, должен обеспечивать возможность различать члены только в пределах одного подсписка ($n = 2 \div 5$).

Поиск объектов в описанной многодеревной ассоциативной узловой структуре осуществляется следующим образом. Задается совокупность значений всех *S* признаков искомого объекта. Если какие-либо из признаков не заданы, то поиск в общем случае может оказаться неоднозначным, т. е. может быть найдено несколько объектов, отвечающих требуемым условиям. Заданные *S* значений признаков делятся на *l* надпризнаков в соответствии с принятой для построения поисковых деревьев схемой деления. Затем из *l* надпризнаков выбирается один надпризнак, который мы будем называть поисковым надпризнаком. С помощью поискового дерева, относящегося к выбранному поисковому надпризнаку, производится поиск объектного списка, соответствующего заданному значению поискового надпризнака. После нахождения требуемого объектного списка производится поиск нужного объекта в этом списке путем последовательного просмотра его членов (ассоциативных узлов) и сравнения значений признаковых кодов в остальных списковых словах каждого узла с заданными значениями остальных надпризнаков (кроме поискового).

Член, у которого все значения признаковых кодов в списковых словах узла равны заданным, выдается в качестве результата. Таким образом, весь процесс поиска объекта расчленяется на три стадии:

а) выбор поискового надпризнака;
б) поиск объектного списка, соответствующего заданному значению выбранного надпризнака;
в) поиск объекта в объектном списке.

В каждом узле объектного списка необходимо проверить максимум *l*—1 кодов надпризнаков. Если эти проверки производить последовательно, то в среднем придется проверять меньше, чем *l*—1 кодов, так как проверки нужно вести только до обнаружения первого несовпавшего кода. Для ускорения поиска все признаковые коды в каждом ассоциативном узле могут проверяться одновременно, т. е. за один такт работы машины.

Из изложенного видно, что при поиске объектов в многодеревной узловой структуре каждый раз в процедуре поиска участвует только одно из поисковых деревьев и один из объектных списков; в ассоциативных узлах объектов при этом используются: один адрес связи, входящий в списковое слово, соответствующее выбранному поисковому надпризнаку, и *l*—1 кодов надпризнаков, входящих в состав списковых слов, соответствующих остальным надпризнакам. Таким образом, объем памяти, фактически используемой при каждом конкретном поиске объекта, значительно меньше общего объема памяти, занимаемого данной многодеревной ассоциативной структурой (не считая памяти, занятой дополнительной информацией об объектах — формулярами объектов). Из сказанного вытекает, что для осуществления поисков объектов по заданной системе признаков можно построить структуру, состоящую только из одного поискового дерева, выбрав в качестве поискового надпризнака некоторую постоянно закрепленную группу *h* признаков ($h<S$). При этом в ассоциативных узлов объектов достаточно иметь одно списковое слово, включающее в себя один адрес связи и один сложный признаковый код, объединяющий в себе все остальные S—*h* признаков. Такие признаковые структуры мы будем называть однодеревными ассоциативными структурами.

Основное достоинство многодеревных структур состоит в возможности вести поиск объекта несколькими путями, выбирая в разных случаях в качестве поискового надпризнака любой из имеющихся *l* надпризнаков. При этом могут отбираться различные категории объектов путем задания неполных наборов их признаков. В качестве поискового надпризнака каждый раз может назначаться надпризнак, имеющий полный состав признаков, соответствующих отбираемой категории. Важным преимуществом многодеревных узловых структур является

возможность выбора для поиска объекта наиболее короткого пути, т. е. такого поискового надпризнака, которому соответствует наиболее короткий объектный список. Это может быть сделано на основе учета вероятностей или частостей появления объектов с различными значениями надпризнаков. Оценка целесообразности построения однодеревной или многодеревной ассоциативной структуры должна производиться, исходя из характера поиска объектов (поиск одиночных объектов с полным набором признаков или поиск категорий объектов с неполными наборами признаков), а также исходя из распределения вероятностей появления объектов с различными значениями признаков (надпризнаков). Для выработки подхода к разрешению этого вопроса рассмотрим некоторые зависимости, определяющие процесс поиска в однодеревной ассоциативной структуре.

Допустим, что в качестве поискового надпризнака мы выбираем h первых признаков (из S), Остальные S—h признаков объединим в один надпризнак, который будем называть объектным. Так как никаких ограничений на перестановку признаков не налагается, то под h признаками можно понимать любые (фиксированные) h из S признаков.

Согласно (8) число объектных списков, соответствующих данному поисковому надпризнаку, будет равно

$$N = \prod_{i=1}^{h} b_i \ . \tag{10}$$

Число операций проверок членов поискового дерева, необходимых для нахождения объектного списка, соответствующего заданному значению поискового надпризнака, будет согласно (2) равно

$$L_{ПД} = \mu n \frac{\ln N}{\ln n} \tag{2}$$

Эта величина имеет минимум при $n = e$, поэтому в подсписках поискового дерева целесообразно иметь по 2—3 члена. Коэффициент μ учитывает, что просматриваются не все члены подсписков поискового дерева, так как просмотр идет только до обнаружения требуемого значения проверяемого признака.

Будем считать, что число проверок членов выбранного k-го объектного списка, необходимых для нахождения одного заданного члена (объекта), пропорционально длине этого объектного списка (т. е. числу членов в нем)

$$L_{oc_k} = \eta Q_{oc_k} \ . \tag{11}$$

Длину каждого объектного списка естественно считать пропорциональной вероятности появления объектов с соответствующим значением поискового надпризнака. (Очевидно, что вероятность появления объекта с данным значением объектного надпризнака на процент проверяемых членов данного объектного списка влияния оказывать не будет.)

Обозначим вероятность появления объектов с некоторым k-м значением поискового надпризнака через p_k. Так как каждый из рассматриваемых объектов (общее число которых обозначим через P) должен находиться в одном и только в одном из объектных списков, относящихся к данному надпризнаку, то количества объектов, находящихся в объектных списках, соответствующих различным значениям поискового надпризнака, будут пропорциональны вероятностям появления этих значений надпризнаков:

$$Q_{oc_k} = P p_k \tag{12}$$

Считаем, что все S признаков взаимно независимы. При этом вероятность появления определенного значения надпризнака будет равна произведению вероятностей появления значений признаков, образующих данный надпризнак:

$$p_k = \prod_{i=1}^{h} p_{i,j_i} \ , \tag{13}$$

где p_{i,j_i} — вероятность появления j-го значения i-го признака.

Считаем, что для всех значений S признаков вероятности их появления заданы в виде таблицы, имеющей S колонок и в каждой i-й колонке по b_i строк (значений).

Таким образом, индекс j для каждого i может меняться от 1 до b_i. В формуле (13) участвует некоторый фиксированный набор значений j, соответствующий k-му значению поискового надпризнака. Заметим, что если некоторые признаки не являются независимыми, то они могут быть объединены в один признак (с большим числом значений — не надпризнак), который будет независим по отношению к остальным признакам, и таким образом случай зависимых признаков может быть сведен к рассматриваемому случаю независимых признаков. Частость обращений к данному объектному списку для поиска в нем объектов будет также пропорциональна вероятности появления данного значения поискового надпризнака p_k Таким образом, количество проверок, приходящееся на долю данного объектного списка, будет пропорционально длине этого списка и частости обращения к нему:

$$L_{oc_k} = \eta P \prod_{i=1}^{h} p_{i,j_i} M \prod_{i=1}^{h} p_{i,j_i} \ , \tag{14}$$

где индекс j имеет некоторый фиксированный набор значений, соответствующий k-му объектному списку, а коэффициент M характеризует собой общее число реализованных поисков объектов. При этом $M \prod_{i=1}^{h} p_{i,j_i}$ представляет собой количество поисков, приходящихся на k-й объектный список. Из (14) получим

$$L_{oc_k} = \eta MP \prod_{i=1}^{h} p_{i,j_i}^2 \ . \tag{15}$$

Общее количество проверок членов объектных списков, выполненных при всех M поисках объектов, будет равно сумме количеств проверок, приходящихся на все N объектных списков:

$$L_{oc} = \sum_{k=1}^{N} L_{oc_k} \ . \tag{16}$$

Эта сумма получится путем варьирования индекса j, который для каждого k принимает фиксированный набор значений. Сумма (16) получится следующим образом:

$$L_{oc} = \eta MP \sum_{j_h=1}^{b_h} \cdots \sum_{j_2=1}^{b_2} \sum_{j_1}^{b_1} \prod_{i=1}^{h} p_{i,j_i}^2 \ . \tag{17}$$

Выражение (17) можно записать более компактно, если заменить многократную сумму произведений произведением сумм:

$$L_{oc} = \eta MP \prod_{i=1}^{h} \sum_{j_i}^{b_i} p_{i,j_i}^2 \ . \tag{18}$$

Из (18) путем деления на M можно получить среднее количество проверок, приходящихся на один поиск объекта в объектном списке:

$$L_{oc}^{(cp)} = \eta P \prod_{i=1}^{h} \sum_{j_i=1}^{b_i} P_{i,j_i}^2 \ . \tag{19}$$

Объединяя формулы (2) и (19), получим общее среднее количество операций проверок в поисковом дереве и объектном списке, необходимых для нахождения одного объекта:

$$L^{(cp)} = L_{ПД} + L_{oc}^{(cp)} = \mu n \frac{\ln N}{\ln n} + \eta P \prod_{i=1}^{h} \sum_{j_i=1}^{b_i} p_{i,j_i}^2 \ . \tag{20}$$

Заменив N на $\prod_{i=1}^{h} b_i$, получим

$$L^{(cp)} = \frac{\mu n}{\ln n} \ln \prod_{i=1}^{h} b_i + \eta P \prod_{i=1}^{h} \sum_{j_i=1}^{b_i} p_{i,j_i}^2 \ . \tag{21}$$

Здесь μ и η— постоянные коэффициенты;

n — число членов в подсписках поискового дерева (которое является постоянным);

b_i — число различных значений i-го признака;

P — общее число рассматриваемых объектов;

P_{i,j_i} — вероятность появления j-го значения i-го признака.

Используя формулу (21), можно исследовать два вопроса:

1. Для заданной таблицы распределения вероятностей p_{i,j_i} появления объектов с различными значениями признаков выбрать из S признаков такие h признаков (и определить само число $h \le S$), чтобы среднее количество операций проверок, необходимых для поиска объектов, было минимальным. Эта задача построения оптимальной с точки зрения времени поиска однодеревной ассоциативной структуры.

Выбор признаков должен производиться с учетом числа различных значений каждого признака и распределения вероятностей появления отдельных значений. По-видимому, обобщенной вероятностной характеристикой признака может служить энтропия различных значений этого признака.

2. Для заданной таблицы распределения вероятностей p_{i,j_i} появления объектов с различными значениями S признаков произвести разбиение S признаков на l поисковых надпризнаков таким образом, чтобы при поиске объектов каждый раз при помощи наиболее выгодного поискового надпризнака (т. е. надпризнака, который имеет более короткий объектный список для заданного сочетания признаков) общее количество операций проверок было минимальным. Эта задача построения оптимальной с точки зрения времени поиска многодеревной ассоциативной структуры. Таким образом, имея таблицу распределения вероятностей появления объектов с различными признаками, можно оценить выгоду с точки зрения времени поиска от перехода к многодеревной узловой структуре и сравнить эту выгоду с дополнительным расходом памяти, связанным с этим переходом. Следует заметить, что приведенные формулы получены, исходя из предположения о том, что различные признаки являются взаимно независимыми. Как уже говорилось, в случае, если некоторые признаки будут зависимыми, то их можно объединить в один признак (не надпризнак), имеющий большее число значений. Этот признак будет независимым по отношению к остальным признакам, и таким образом этот случай будет сведен к рассмотренному случаю независимых признаков.

Исследование формулы (21) аналитическим путем не представляется возможным и поэтому упомянутые задачи можно решать численным путем (в частности, методом перебора с помощью электронной цифровой машины) применительно к конкретным значениям *P, S,* b_i и значениям таблицы p_{i,j_i}. Рассмотрим частный случай, когда все *S* признаков имеют одинаковое число значений ($b_1 = b_2 = \ldots = b_s = b$) и вероятности появления объектов с разными значениями признаков также одинаковы ($p_{i,j_i} = 1/b$). Тогда формула (21) примет вид

$$L = \mu n \log_n b^h + \eta P \frac{1}{b^h}. \tag{22}$$

Определим, при каком *h* величина *L* имеет минимум, считая *n* = e:

$$\frac{dL}{dh} = \mu e \ln b - \eta P \frac{\ln b}{b^h} = 0. \tag{23}$$

Отсюда

$$h = \log_b \frac{\eta P}{\mu e} = \frac{1}{\ln b} \ln \frac{\eta P}{\mu e}$$

Например (полагая η = μ = 0,5), при *P*=1024 и *b* = 2 *h* ≈ 8, а при $P=10^6$ и *b*=10 *h* ≈ 5. При этом среднее число проверок в объектном списке будет равно *ηe,* т. е. оно будет таким же, как число проверок в каждом из подсписков поискового дерева. Отношение числа проверок, приходящихся на поиск в поисковом дереве, к числу проверок в объектном списке, будет определяться числом уровней поискового дерева:

$$K = \ln \frac{\eta P}{\mu e}. \tag{24}$$

Полагая η = μ = 0,5, получим *K*=ln*P*—1. Например, при $P=10^3$ *K* ≈ 6, при P=10^6 *K*=13.

Таким образом, если исходить из условия минимизации общего числа проверок, то наиболее выгодной оказывается структура, состоящая как бы из одного поискового дерева. В этой структуре объектные списки в среднем должны иметь то же число членов, что и подсписки поискового дерева, но в отличие от них объектные списки будут иметь переменную длину. Основная часть проверок при этом приходится на долю поискового дерева, а на долю объектных списков остается незначительная часть.

Оценим для данного случая требуемый объем памяти под поисковое дерево и объектные списки. Если максимально допустимое число объектов *P,* а число всех признаков *S,* то расход памяти под списковые слова в объектных списках будет равен

$$Q_{ac} \approx P(\log_2 P + \log_2 b^{s-h}), \tag{25}$$

где первое слагаемое представляет адреса связи, а второе — признаковые коды.

Расход памяти под поисковое дерево будет равен

$$Q_{ПД} \approx \frac{nN}{n-1}\lambda,$$

где $N = b^h$— общее число объектных списков, а

λ— число разрядов в списковом слове дерева.*

$$\lambda \approx 2 + \log_2 \frac{nN}{n-1} + \log_2 n.$$

Отсюда

$$Q_{ПД} \approx \frac{nb^n}{n-1}(2 + \log_2 \frac{nb^h}{n-1} + \log_2 n). \tag{26}$$

Оценим соотношение в расходе памяти под списковые слова в объектных списках и поисковом дереве для некоторых конкретных случаев.

При $P = 10^3$, *b*=2, s = 16, *h* = 8, *n* = 3 получим $q = Q_{ac}/Q_{ПД} \approx 4$; при $p=10^6$, *b* = 10, *s*=16, *h* = 4, *n* = 3 получим *q*≈200. Вообще, расход памяти под объектные списки, в основном, зависит от числа объектов *P,* а расход памяти под поисковое дерево зависит от числа объектных списков $N = b^h$.

Так как число объектов *P* обычно значительно больше числа объектных списков *N,* то основной расход памяти приходится на объектные описки. Рассмотрим теперь частный случай построения ассоциативной структуры. Как и в предыдущем случае, будем считать, что все признаки имеют одинаковое число значений (b_1 =

---

* Считаем, что в списковом слове имеется два служебных разряда, адрес связи (относительный, он имеет примерно $\log_2 \frac{nN}{n-}$ разрядов; считается, что дерево занимает сплошную зону ячеек) и признаковый код, служащий для различения членов в пределах одного подсписка (он имеет примерно $\log_2 n$ разрядов).

$=b_2 = ...= = b_s = b)$ и равные вероятности появления объектов с разными значениями признаков ($p_{i,j_i} = \frac{1}{b}$ для i=l, 2, . .., s). S признаков делятся на l надпризнаков по h признаков в надпризнаке ($h = S/l$). В ассоциативном узле каждого объекта находится по l списковых слов, соответствующих отдельным надпризнакам. Будем считать, что все l признаковых кодов в узле проверяются одновременно.

Из формулы (21) получим (заменяя $p_{i,j_i} = \frac{1}{b}$ и $h=S/l$)

$$L^{(cp)} = \mu n \log_n b^{\frac{s}{l}} + \frac{\eta P}{b^{\frac{s}{l}}}. \qquad (27)$$

Дифференцируя по l и приравнивая нулю, получим значение l, при котором число проверок будет минимальным:

$$\frac{dL^{(cp)}}{dl} = -\frac{\mu n S}{l^2}\log_n b + \frac{\eta P S}{l^2}\frac{\ln b}{b^{\frac{s}{l}}}$$

Отсюда (полагая $\eta \approx \mu$)

$$l = \frac{\ln b^s}{\ln P + \ln\ln n - \ln n}$$

Если считать n = e, то получим:

$$l = \frac{\ln b^s}{\ln P - 1}. \qquad (28)$$

Из формулы (28) видно, что оптимальное число надпризнаков, на которое должны быть разделены S признаков, увеличивается с увеличением b^s, т. е. числа всех возможных комбинаций признаков, и уменьшается с увеличением числа рассматриваемых объектов P.

Например, для $P=10^3$, $b = 2$, S=16 получим $l \approx 2$; для $P=10^6$, $b = 10$, $S = 16$ получим $l \approx 3$.

Как уже говорилось выше, с увеличением числа надпризнаков растет число поисковых деревьев и ассоциативных слов в узлах, а следовательно, растет объем памяти, занимаемой ассоциативной информацией, причем основной расход памяти идет под списковые слова в узлах.

Рассмотренные частные случаи построения однодеревной ассоциативной структуры помимо непосредственного практического применения могут служить также для общей ориентировки при построении структур с переменным числом значений признаков и различными вероятностями их появления.

Для более точного анализа и выбора приемлемых параметров такого рода структур может быть непосредственно использована формула (21), которая должна исследоваться численным методом. Анализ показывает, что основная часть операций проверок, выполняемых при поисках объектов, приходится на просмотр поискового дерева, а основной расход памяти под ассоциативную информацию — на долю списковых слов в узлах объектов. Учитывая это, можно построить так называемую многодеревную ассоциативную структуру, которая будет наиболее эффективной, как с точки зрения быстроты поиска объектов, так и с точки зрения сокращения расхода памяти. При этом все S заданных признаков объектов разбиваются на l надпризнаков, и для каждого из надпризнаков строится свое поисковое дерево. Таким образом, всего в структуре будет l деревьев. Но в узлах объектов отводится место всего под одно списковое слово, которое будет включать в себя адрес связи и один объектный надпризнак. Следовательно, каждый объект может находиться только в одном из объектных списков, принадлежащих к одному из указанных поисковых деревьев. Выбор поискового дерева, т. е. поискового надпризнака, по которому должен производиться поиск объекта, осуществляется с учетом вероятностей появления объектов с различными значениями надпризнаков. Исходя из физического содержания задачи и имеющихся сведений о частостях появления объектов с разными значениями надпризнаков, при программировании заранее составляется таблица вероятностей появления объектов с различными значениями признаков ($p_{i\ i.}$). При поступлении очередного объекта по значениям его S признаков при помощи таблицы определяются вероятности появления каждого из l надпризнаков и выбирается в качестве поискового надпризнака тот надпризнак, который имеет минимальное значение вероятности. По этому надпризнаку будет производиться поиск. При указанном способе в каждом объектном списке будут находиться не все объекты, обладающие данным значением надпризнака, а только те, у которых это значение имеет наименьшую вероятность появления по сравнению со значениями его других надпризнаков. Этот способ обеспечивает значительное сокращение расхода памяти под ассоциативные узлы за счет того, что в каждом узле будет не l, а одно списковое слово. Этот способ приводит и к ускорению поиска за счет того, что в каждом объектном списке в среднем будет находиться в l раз меньше объектов и объекты будут распределяться по объектным спискам более или менее равномерно. При этом поиск будет идти каждый раз оптимальным образом, т. е. по тому объектному списку, который имеет наименьшую вероятность появления объектов с данным значением надпризнака. Дополнительные операции, связанные с определением по таблице вероятностей нужного надпризнака, могут быть оформлены в виде стандартной подпрограммы либо реализованы схемно в виде одной специальной команды.

6. ВАРИАНТЫ ОРГАНИЗАЦИИ МАШИННОЙ ПАМЯТИ ПРИ АССОЦИАТИВНОМ ПРОГРАММИРОВАНИИ

§ 16 ПОСТРОЕНИЕ АССОЦИАТИВНЫХ СТРУКТУР С ГНЕЗДОВЫМИ СПИСКАМИ

При построении ассоциативных структур на основе гнездовых списков используются списковые слова трех видов: объектные, структурные и переходные. Членами списков являются только объектные и структурные слова; переходные слова играют вспомогательную роль и не являются членами списков.

Объектные и структурные списковые слова содержат следующие компоненты:

1) указатель типа слова *Т*. Для объектных и структурных слов этот указатель равен нулю, для переходных слов *Т*=1;
2) указатель вида слова *S*. Для структурных слов *S*=1, для объектных *S* = 0;
3) поисковый признак *П;*
4) наименование члена списка *Н*.

Переходные слова помимо указателя *Т* имеют в своем составе указатель *В* (вида переходного слова) и адрес связи АС.

Признак *В* в переходных словах указывает назначение слова: *В*=0 — переходное слово является исключающим, т. е. оно заменяет исключенный член списка; *В*=1 — переходное слово является продолжающим, оно служит для связи между разными гнездами одного и того же списка.

Для построения ассоциативных структур с гнездовыми списками кроме указанных слов необходимы еще фиксаторы списков — специальные ячейки памяти, играющие роль заголовков списков.

Фиксатор имеет следующий состав:

— счетчика числа членов списка Сч;

— адрес вершины списка АВ;

— адрес очередной свободной ячейки гнезда АЯ

Для размещения списковых фиксаторов используется часть общей оперативной памяти. Адреса фиксаторов являются машинными кодами наименований списков и подсписков.

Идентификаторы фиксаторов или числовые значения их адресов указываются в качестве наименований членов (*Н*) в структурных списковых словах.

Счетчик числа членов списка (Сч) служит для указания числа объектных и структурных членов, находящихся в списке. При включении нового члена этот счетчик увеличивается на единицу, а при исключении члена уменьшается на единицу. Равенство счетчика нулю свидетельствует о полном исключении всех членов списка. Счетчик числа членов списка необходим:

а) для указания возможности исключения структурных членов. Эти члены могут исключаться при условии, что отходящие от них подсписки являются пустыми;

б) для указания количества объектов заданного типа, зарегистрированных в определенном списке;

в) для ограничения процессов просмотра списков.

При гнездовом способе организации ассоциативных списков весь объем оперативной памяти при программировании должен быть разделен ориентировочно на первоначальные гнезда в соответствии с ожидаемыми размерами списков. Все ячейки каждого первоначального гнезда соединяются в цепь путем помещения в каждую ячейку адреса следующей ячейки. Остальные разряды этих ячеек имеют значения, равные нулю. В последней ячейке цепи вместо адреса связи ставится специальный код конца списка КС.

Адреса первых ячеек списков помещаются в АВ, а адреса вершин (начальных ячеек) цепей свободных ячеек гнезд — в АЯ фиксаторов соответствующих списков.

Кроме указанных гнезд и фиксаторов списков, которые выделяются при программировании, оставляется общий резерв ячеек оперативной памяти и фиксаторов, который необходим для образования новых списков или наращивания имеющихся списков в процессе работы.

Все резервные ячейки оперативной памяти также связаны в единую цепь, адрес вершины этой цепи находится в постоянно закрепленной ячейке — фиксаторе цепи свободных ячеек СЯ.

Порядок работы с гнездовыми списками

Запись первого члена списка. Запись первого члена списка производится следующим образом. По значению Сч = 0, находящемуся в фиксаторе соответствующего списка, определяется факт записи первого члена списка, который будет представлять собой начало списка. Значение АЯ, указывающее адрес очередной (1-й) свободной ячейки первого гнезда этого списка, переписывается в АВ (теперь там будет уже не нуль). Это значение указывает адрес первой ячейки гнезда, куда должен быть записан первый член списка. Из этой ячейки в АЯ переписывается адрес следующей свободной ячейки, после чего в эту ячейку записывается первый член списка; значение Сч увеличивается на единицу. На этом запись первого члена списка заканчивается. При записи следующих членов списка адреса ячеек для них берутся аналогичным образом из АЯ. Члены списка должны записываться с признаком *Т* = 0 и признаком *S* = 0 или 1 в зависимости от вида члена.

Положение ячеек памяти после записи некоторого *n*-го члена списка при последовательной их записи представлено на рис. 21. Считаем, что для записи списка отведено гнездо из *m*+*n* ячеек. Идентификатор *Н* обозначает наименование соответствующего члена списка; оно включает в себя поисковый признак *П*, характеризующий данный член списка.

Стрелки вверху над клетками показывают естественную последовательную адресацию, используемую при просмотрах списков. Полукруглые стрелки снизу показывают цепную адресацию.

Исключение объектных членов списка. Члены списка, как правило, исключаются в произвольном порядке.

Для исключения некоторого *i*-го члена списка, заданного своим адресом *a+i,* необходимо:

а) на место исключаемого члена, т. е. в ячейку с адресом *a+i,* записать переходное слово со значениями признаков *Т*=1, *В* = 0 и адресом, равным значению АЯ;

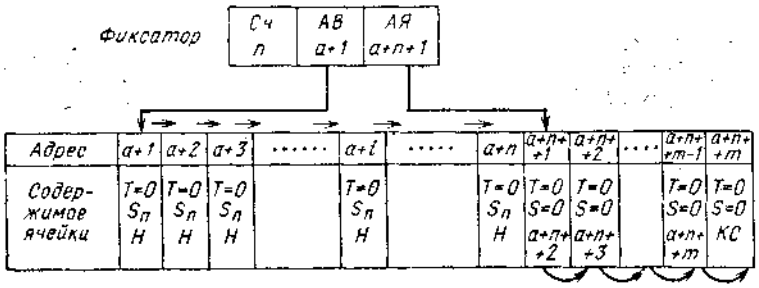


РИС. 21. Пример построения гнездового списка.

б) в АЯ записать адрес ячейки исключаемого члена *a+i;*

в) уменьшить в фиксаторе соответствующего списка значение счетчика Сч на единицу. При этом схема заполнения ячеек примет вид, показанный на рис. 22.

При исключении нескольких членов списка будут выполняться несколько раз такие же действия, как и при исключении одного члена описка. После исключения, например, трех членов (сначала *i-го,* 1-го, а затем *n*-го) схема заполнения ячеек будет иметь вид, показанный на рис. 23.

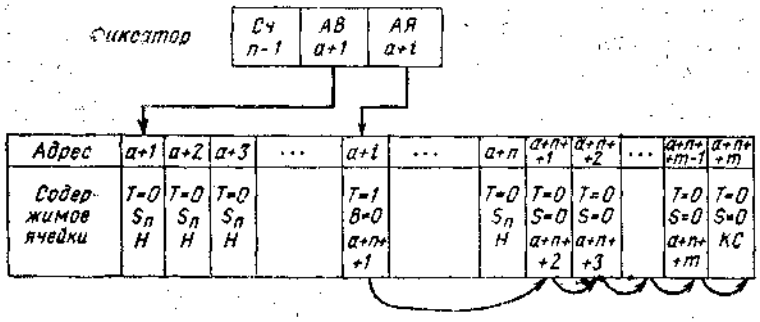


РИС. 22. Исключение члена из гнездового списка.

Последовательный просмотр списков и запись новых членов списка после многократных исключений членов. Просмотр списка всегда идет в порядке возрастания адресов слева направо. При просмотрах проверяются значения поискового признака *П* в объектных и структурных членах списков и отбираются члены, удовлетворяющие заданному критерию. Если при этом встречаются исключенные члены, то по признакам *Т*=1, *В* = 0 определяется факт, что данная ячейка содержит исключенный член списка и должна быть пропущена; при этом совершается переход к следующей ячейке.

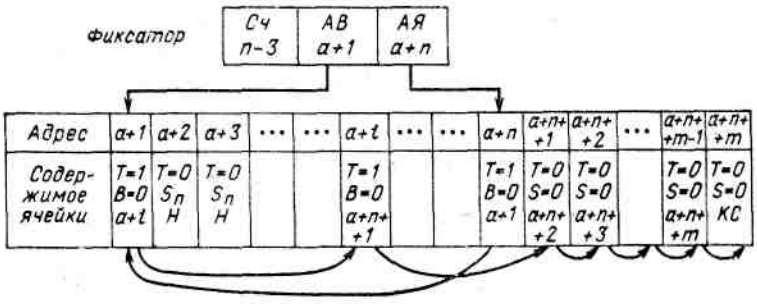


РИС. 23. Многократное исключение членов из гнездового списка.

Просмотры списка после многократных исключений производятся таким же образом, начиная с начальной ячейки, указываемой АВ. При этом ячейки *a*+l, *a+i* и *a* +*m* будут пропускаться (см. рис. 23).

При заполнении списка новыми членами сначала будет заполнена ячейка *a* + *n,* затем *a*+1 и затем *a* + *i,* после чего начнут заполняться новые ячейки *a* + *n*+1, *a* + *n* +2 и т. д.

Запись новых членов списка после исключения некоторых членов производится в ячейки, адреса которых указываются как и раньше АЯ. При указанной организации ассоциативной памяти всегда сначала заполняются внутренние ячейки гнезд, и только после того, как они все будут заполнены, происходит заполнение новых ячеек.

Продолжение списков при переполнении гнезд. При последовательном заполнении членами списка свободных ячеек гнезда может быть достигнут конец гнезда, отведенного первоначально под данный список, что будет обнаружено по значению специального кода конца списка КС, стоящего вместо адреса связи в последней свободной ячейке данного гнезда.

Для продолжения списка должны быть выполнены следующие действия (рис. 24):

а) к заполненному гнезду ячеек добавляется новое гнездо определенного размера, взятое из резерва свободных ячеек. Для этого из фиксатора цепи свободных ячеек СЯ берется адрес начальной ячейки нового

гнезда, а к содержимому фиксатора прибавляется величина приращения; в последнюю ячейку из группы ячеек нового гнезда записывается код конца списка КС;

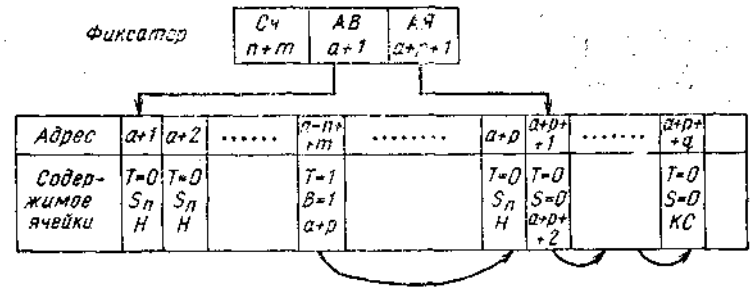


РИС. 24. Наращивание гнезда при переполнении.

б) в последнюю ячейку заполненного гнезда записывается вместо очередного члена списка переходное слово, содержащее признаки $T=1$, $B = 1$ и адрес связи АС начала нового гнезда. Это переходное слово необходимо для продолжения списка при просмотрах;

в) в АЯ записывается адрес, на единицу больший адреса первой ячейки нового гнезда;

г) в первую ячейку нового гнезда записывается новый член списка;

д) счетчик фиксатора увеличивается на единицу (а не на 2, так как переходное слово не является членом списка).

При следующей записи нового члена (если перед этим не будет производиться исключения членов) этот член запишется в ячейку $a+p+1$. В АЯ будет переписан адрес следующей свободной ячейки, взятый из заполняемой ячейки и равный $a+p+2$. При просмотрах списка сначала просматриваются члены в ячейках с $a+1$ до $a+n+m$, а затем по содержимому ячейки $a+n+m$ совершается переход к ячейке $a+p$, где находится следующий член списка, и далее обычным образом. Указанные выше наращивания гнезд могут производиться многократно по мере увеличения списков. Следует подчеркнуть, что всегда при записи новых членов в первую очередь заполняются внутренние ранее освободившиеся ячейки гнезд, чем обеспечивается плотность записи и экономное использование памяти.

Указанный процесс построения списков является необратимым в следующих отношениях:

а) при исключении некоторых членов списка остальные члены списка не сдвигаются, и при просмотрах списков будет затрачиваться время на пропуск переходных слов, заменяющих исключенные члены.

Таким образом, рост списка вправо внутри гнезда является необратимым, т. е. просматриваемая длина списка не уменьшается при исключении членов;

б) наращивание гнезд является также необратимым процессом, так как гнезда-приращения не возвращаются автоматически в резерв даже при полном исключении всех членов списка, входящих в состав данного гнезда-приращения. Для устранения этих недостатков при длительной непрерывной работе предусматриваются специальные процедуры уплотнения списков, которые должны выполняться периодически параллельно с основной программой. Эти процедуры осуществляют сдвиг всех членов списка с соответствующей корректировкой фиксаторов и возвращением освободившихся гнезд ячеек в цепь свободных ячеек.

Автоматический пропуск исключенных членов списков ($П=1$, $B = 0$) при просмотрах списков целесообразно решать схемным путем, используя для этой цели специальные регистры и операции. Схемным путем должны выполняться также действия, связанные с получением ячеек из цепей свободных ячеек гнезда (АЯ) и общих свободных ячеек (СЯ), а также с включением освобождающихся ячеек в эти цепи.

Повторное использование освободившихся гнезд ячеек

Для повторного использования освободившихся гнезд ячеек они должны соединяться в единую цепь, называемую списком освободившихся гнезд (ОГ). Адрес вершины этого списка хранится в специальной ячейке, называемой фиксатором ОГ.

Так как списки в общем случае состоят из нескольких гнезд (основного и приращений), связанных переходными словами, то возвращение освободившихся ячеек в список освободившихся гнезд производится также по гнездам. Для включения в ОГ освободившихся гнезд ячеек данного списка (который оказался уже ненужным для дальнейших вычислений) этот список просматривается, начиная с первой ячейки, адрес которой содержится в АВ фиксатора. При встрече первого переходного продолжающего слова ($T=1$, $B=1$) фиксируется его адрес, который указывает конец первого гнезда. Запоминается значение АС переходного слова, указывающее начало следующего гнезда, а в ячейку переходного слова заносится значение адреса, хранящееся в фиксаторе ОГ (т. е. значение адреса вершины списка свободных гнезд); при этом признаки $T=1$, $B=1$ в переходном слове сохраняются. В каждую из остальных ячеек первого гнезда записываются признаки $T = 0$, $B = 0$ и адреса связи, т.е. адреса следующих ячеек данного гнезда.

В фиксатор ОГ записывается адрес первой ячейки гнезда. На этом возвращение ячеек первого гнезда в список ОГ заканчивается. Возвращение остальных гнезд данного списка производится аналогично. Таким образом, все освободившиеся ячейки данного списка будут связаны в единую цепь по гнездам.

Внутри гнезд ячейки будут расположены в естественном порядке, в конце каждого гнезда стоит переходное структурное слово с признаком $T=1$, $B = 1$, указывающее продолжение цепи свободных ячеек. При этом гнезда в цепи ОГ следуют в порядке, обратном порядку их следования в стираемом списке. Очевидно, что это обстоятельство не имеет значения для их дальнейшего использования.

При программировании заранее должен устанавливаться постоянный минимальный размер гнезд ячеек (например, в 4 ячейки), и все списки и их приращения будут иметь размеры, кратные этой величине. Гнезда освободившихся ячеек, включаемые в список ОГ, также будут иметь размер, кратный четырем. Повторное использование гнезд ячеек должно производиться указанными порциями, что обеспечит совпадение переходных слов, находящихся на границах освободившихся гнезд, с переходными словами, которые будут ставиться на границах, включаемых в списки гнезд ячеек.

§ 17 ПОСТРОЕНИЕ ОБОБЩЕННЫХ АССОЦИАТИВНЫХ УЗЛОВЫХ СТРУКТУР

Ассоциативные структуры с узловыми списками могут использоваться для накопления и обработки больших объемов информации о различных объектах и выдачи этой информации по запросам.

Организация информации о различных объектах в виде узловых ассоциативных структур обеспечивает возможность автоматической категоризации и поиска объектов, обладающих определенными признаками. Это позволяет решать логические задачи, связанные с систематизацией и анализом экспериментальных данных, выработкой понятий, характеризующих классы объектов, выявлением некоторых закономерностей между различными свойствами и категориями объектов. Слово объект в данном случае следует понимать в широком смысле как любой предмет или явление, о которых имеется некоторая информация. Например, объектами могут быть населенные пункты, отдельные люди, документы, разделы науки, болезни т. д. Объектами могут быть также группы других объектов.

Как уже отмечалось, вся информация об объектах с точки зрения ее использования для поиска делится на два вида: ассоциативную и собственную.

В общем случае ассоциативная информация имеет двухчленное строение, т. е. для объекта имеется набор признаков, каждый из которых состоит из двух частей:

— наименования признака (например, цвета);

— значения признака (например, красного).

Наименования и значения признаков могут быть представлены условными числовыми кодами или задаваться в словесной форме с использованием определенного словаря и порядка построения фраз.

Ассоциативная информация об объектах в памяти машины в рассматриваемом варианте информационной системы может храниться в виде поисковых деревьев и в виде ассоциативных узлов. Каждый узел соответствует одному объекту и содержит кроме ассоциативной информации также отсылку к собственной информации об этом объекте. Возможны случаи, когда в узле совместно с ассоциативной информацией хранится и собственная информация. Собственная информация об объектах представляет собой некоторые дополнительные сведения об объектах, не используемые при поиске объектов. Удобно эту информацию хранить отдельно от ассоциативной в виде печатного материала или в виде записей на магнитных лентах. Поиск данных об объектах осуществляется на основе ассоциативной информации. Типовые запросы на поиск могут содержать в себе:

а) наименования искомых объектов;

б) наборы наименований и значений признаков объектов, которыми должны (не должны) обладать искомые объекты.

Построение ассоциативных узлов

Ассоциативные узлы представляют собой в общем случае группы ячеек памяти, в которых размещаются заголовок узла и несколько списковых слов. Таким образом, сам узел представляет собой список и может строиться, вообще говоря, по любому из трех способов построения списков: последовательному, гнездовому, цепному. В тех случаях, когда размеры узлов фиксированы или узлы, хотя и имеют различный состав, по, после того как они образованы, не меняются, а также в тех случаях, когда не производится оперативного исключения объектов (данные об объектах только накапливаются), удобно пользоваться последовательным способом. При этом для образования каждого нового узла из свободной зоны памяти выделяется необходимая группа последовательно расположенных ячеек, которая в дальнейшем не меняется. С другой стороны, если размеры узлов заранее неизвестны и могут меняться в ходе работы системы (могут исключаться одни списковые слова и включаться другие), то удобно пользоваться для построения узлов цепным способом построения списков.

При этом в каждом списковом слове кроме адресов связи, необходимых для включения этого узла в списки, соответствующие различным признакам объектов, выделяется еще один адрес связи, который служит для объединения в цепной список всех слов, образующих данный узел. Очевидно, что при этом обеспечивается простота изменения узлов, но увеличивается расход памяти и несколько увеличивается время обращения к отдельным словам внутри узла.

Промежуточным вариантом является гнездовой способ построения ассоциативных узлов, при котором узел образуется из одного или нескольких гнезд последовательно расположенных ячеек памяти.

При первоначальном образовании узла берется гнездо ячеек, соответствующее составу узла в данный момент. В дальнейшем, при необходимости увеличения узла, к первоначальному гнезду могут добавляться дополнительные гнезда. Этот способ обеспечивает определенную гибкость в построении узлов, а если большинство узлов будет состоять из одного гнезда, то сохраняются достоинства 'последовательного способа.

В этом параграфе рассмотрим гнездовой способ построения узлов. Для построения узлов будут использоваться гнезда ячеек одного заранее фиксированного размера, которые мы будем называть единичными гнездами. Вся память машины, предназначенная для размещения ассоциативной информации, заранее делится на единичные гнезда, которые соединяются в цепь свободных единичных гнезд (СЕГ).

Использование единичных гнезд постоянного размера упрощает процессы получения гнезд из СЕГ и возвращение в СЕГ гнезд, освободившихся после стирания или уменьшения узлов.

При первоначальном построении узлов из СЕГ может забираться сразу одно или несколько единичных гнезд, в зависимости от фактического размера образуемого узла.

При наращивании узлов приращения могут состоять также из одного или нескольких единичных гнезд, в зависимости от размеров приращений.

При гнездовом способе построения узлов в отличие от гнездового способа построения списков объектов нет необходимости предусматривать возможность оперативного включения и исключения отдельных членов внутри гнезд. В данном случае такими членами являются не объекты, а списковые слова, которые к тому же могут размещаться независимо от границ ячеек. В связи с этим в данном случае может быть использован упрощенный способ построения гнездовых списков.

Первая половина первой ячейки каждого гнезда отводится под так называемый фиксатор гнезда: здесь указывается либо номер очередного свободного отделения ячейки данного гнезда ячеек (см ниже), либо нулевой код, показывающий, что все ячейки (отделения) данного гнезда заполнены, либо полный прямой адрес следующего единичного гнезда (первой ячейки этого гнезда), являющегося продолжением данного ассоциативного узла. Различие между первым и третьим случаями состоит в том, что номер очередного свободного отделения гнезда должен быть всегда меньше общего числа отделений в единичном гнезде, а полный прямой адрес гнезда, являющегося продолжением данного узла, должен быть всегда больше общего числа отделений в единичном гнезде. Это всегда можно сделать соответствующим выбором размеров наращиваемых гнезд.

Проверки и изменения значений фиксаторов гнезд, необходимые для добавления в узел новых списковых слов или просмотров списков, могут осуществляться либо с помощью специальных подпрограмм, которые в этом случае могут быть оформлены в виде стандартных подпрограмм, либо при помощи специальных команд, реализуемых схемно.

В состав ассоциативного узла входят:

а) заголовок узла, содержащий наименование объекта и отсылку (адрес) к собственной информации об объекте. В частном случае адрес собственной информации об объекте может быть одновременно наименованием этого объекта;

б) списковые слова, представляющие ассоциативную информацию об объекте и служащие для включения объекта в соответствующие списки и для поиска объектов по признакам. Каждое списковое слово состоит из нескольких слогов: одного адресного слога и одного или нескольких слогов данных.

Адресный слог всегда начинается одним из символов адресного слога, после которого идет код адреса, и заканчивается любым служебным символом (см. ниже). Слоги данных, относящихся к данному адресному слогу, следуют непосредственно за этим слогом. Они могут начинаться любым служебным символом (кроме адресных) и заканчиваться либо одним из адресных символов (обозначающим начало следующего спискового слова), либо символом конца ассоциативного узла (для последнего в узле спискового слова).

В списковых словах для каждого признака указывается в явном виде наименование признака и его значение. Указание в явном виде наименований признаков несколько увеличивает объем хранимой информации, но обеспечивает гибкость в построении узлов и упрощает поиск объектов: при этом разные признаки объекта внутри узлов не обязательно должны располагаться в одном и том же порядке.

Для определенности будем ориентироваться на 48-разрядные ячейки памяти. Допустим, что максимальный объем ассоциативной информации будет ограничен размером памяти в 2^{24} ячейки. Полные прямые адреса всех этих ячеек будут представляться 24-разрядными двоичными кодами. При размещении узлов на магнитных лентах адреса связи могут включать в себя части, определяющие номер МЛ, адрес зоны на МЛ и адрес ячейки внутри зоны. Для простоты будем применять способ полной прямой адресации списковых членов. Ассоциативные узлы будем строить из 6-разрядных двоичных кодов, обозначающих отдельные символы. В узлах может быть только целое число гнезд, а, следовательно, и целое число ячеек, причем некоторые ячейки или части ячеек могут оставаться неиспользованными; при этом в них должны стоять нулевые коды.

Рассмотрим следующий способ использования памяти машины. 48 разрядов каждой ячейки делятся на восемь отделений по шесть разрядов в каждом отделении. Шестиразрядный двоичный код позволяет иметь 64 различных значения, т. е. 64 различных символа. В состав символов можно включить, например, все 32 буквы русского алфавита, десять цифр, символы логических операций «не» и «и» (логическая операция «или» может указываться просто написанием рядом соответствующих значений). Кроме того, в состав символов включается ряд служебных символов: начало узла, конец узла, начало дескриптора (этот же символ указывает конец адреса связи и начало наименования признака), начало значения признака, конец признака, символ объектного адреса связи, структурного адреса связи, признакового адреса связи, к ища гнезда (символ перехода), команды, градиента, семантический символ, символ обратного адреса связи (конца списка), символ фиксатора списка и др. Поясним значение некоторых служебных символов.

Символ объектного адреса связи указывает, что следующие за ним символы данных образуют адрес следующего объекта (узла), входящего в данный список. При следовании по объектным адресам связи мы переходим от узла к узлу при сохранении в этих узлах только значения признака (дескриптора), относящегося к данному объектному адресу связи. Остальные признаки объектов могут меняться; к объектным адресам связи могут относиться градиентные слоги (см. ниже), показывающие направление изменения некоторых других признаков.

Символ структурного адреса связи указывает, что следующие за ним символы данных образуют адрес узла, являющегося первым узлом подсписка, ответвляющегося от данного узла. Для всех членов подсписка сохраняются все значения признаков (дескрипторов) данного узла. Таким образом, ответвляющиеся подсписки должны представлять собой более мелкие подразделения классификации объектов некоторого типа, определяемого узлом, от которого ответвляется подсписок.

Символ признакового адреса указывает, что следующие за ним символы образуют адрес вершины некоторого поискового дерева, представляющего собой признаковую структуру, ответвляющуюся от данного объекта, причем все признаки данного объекта сохраняются для всех членов ответвляющейся структуры.

Символ команды указывает, что следующие символы образуют команду, которая должна быть

выполнена после проверки данного символа. Заметим, что при выполнении подобных команд, вклинивающихся в состав данных, естественно, должна быть обеспечена возможность многократного (рекурсивного) прерывания программы, так как включение отдельных команд или целых подпрограмм в состав обрабатываемой информации приведет к остановкам основной программы.

Символ градиента означает, что следующие за ним символы являются градиентами, показывающими направление изменения других признаков; при этом может быть применен такой, например, способ кодирования градиентов: в каждом шестиразрядном коде одного символа пять разрядов обозначают порядковый номер спискового слова в данном узле (не больше 32), а шестой разряд показывает направление изменения — вверх (1) или вниз (0).

Семантический символ означает, что следующие символы, взятые парами, являются указателями семантической связи данного спискового слова с другими списковыми словами данного узла: первый символ пары указывает номер другого спискового слова, а второй символ пары — семантическое отношение. Примерами семантических отношений могут служить такие отношения:

а) «быть объектом»; это значит, что данное списковое слово относится к списку, представляющему некоторую категорию объектов, которые являются объектами некоторого процесса, описываемого другим списковым словом;

б) «иметь принципом»; данное списковое слово представляет некоторую категорию объектов (понятие), принцип действия которых описывается другим списковым словом;

в) «иметь материалом»; категория объектов, представляемая данным списковым словом, характеризуется материалом, представляемым другим списковым словом;

г) «иметь частью»; данное списковое слово представляет собой категорию объектов, составной частью каждого из которых является объект, представляемый другим списковым словом.

Подобные семантические отношения (общим числом 10—30) используются при построении информационных языков для фактографических поисковых систем. Они позволяют образовывать сложные понятия, более полно и точно описывающие свойства объектов, чем простые наборы отдельных несвязанных между собой дескрипторов.

Адреса связи, как правило, имеют постоянный размер (в нашем примере по 24 разряда), но могут иметь и переменный размер. Остальные данные могут иметь переменный размер (разрядность).

Слоги данных, образуемые из нескольких символов, могут использоваться для обозначения наименований объектов или адресов записей (формуляров) с собственной информацией об объектах, а также для представления наименований и значений признаков. В случае обозначения наименований объектов или адресов собственной информации об объектах слоги данных располагаются в начале узла. Эти слоги данных будут составлять заголовок ассоциативного узла. Из сказанного ясно, что размеры заголовков узлов не являются фиксированными; конец заголовка определяется первым адресным символом, встретившимся в данном узле. За заголовком узла следуют списковые слова, которые также могут иметь различные размеры. Каждое списковое слово состоит из одного адреса связи и одного дескриптора (описателя). Адрес связи представляется адресным слогом (объектным, признаковым или структурным), а дескриптор представляется слогами данных.

Слоги данных, используемые для представления дескрипторов объектов (наименований и значений признаков), располагаются за соответствующими адресными слогами. К каждому адресному слогу относятся все слоги данных, стоящие за ним, вплоть до следующего адресного слога (или конца узла, включая и продолжение узла).

Простым признаком объекта является некоторая переменная, имеющая наименование. Эта переменная может принимать несколько значений (не обязательно два). Значения каждого признака должны иметь дискретный характер. Если признаки по своей природе представляются непрерывными величинами, их значения при программировании квантуются, исходя из физического смысла этих признаков и точности их представления.

Каждый простой признак содержит две компоненты: наименование и значение. Эти компоненты могут обозначаться при помощи одного или нескольких символов. Каждый дескриптор, стоящий за адресным слогом, может представлять собой либо простой признак, либо сложный, образованный в виде логической функции нескольких простых признаков.

Адресный слог вместе с соответствующим дескриптором образует одно списковое слово, соответствующее одному списку. Кроме заголовка и списковых слов в состав узла может входить переходное слово, служащее для продолжения данного узла в тех случаях, когда первоначально отведенное под данный узел количество ячеек, составляющих одно гнездо, окажется недостаточным в связи с поступлением в процессе обработки новой ассоциативной информации о данном объекте.

Переходное слово начинается символом начала переходного слова; за этим символом следуют символы, которые представляют собой полный прямой адрес первой ячейки нового гнезда, являющегося продолжением данного узла.

Объектные, структурные и признаковые адресные слоги (адреса связи) служат для построения списков и списковых структур различных видов (объектных и признаковых).

Объектные адреса связи служат для построения объектных списков, т. е. списков объектов определенного типа, связывающих, как правило, узлы одинакового формата. В этих слогах кроме символа типа адреса имеется адрес связи. Он указывает первую ячейку другого узла, находящегося, как правило, в той же зоне МЛ, но может указывать и узел, находящийся в другой зоне. После объектного адреса связи ставится один или несколько слогов данных (с соответствующими служебными символами), представляющих собой дескриптор списка, соответствующего данному адресу связи. Окончание дескриптора определяется следующим символом адресного слога или символом конца узла.

Фиксаторы списков — это отдельные ячейки, в которых могут храниться: адрес начала списка, число членов в списке, адрес последнего члена списка, общие признаки всех членов списка и другие данные.

Возможны списки без фиксаторов; в этих случаях роль фиксатора выполняет структурный или признаковый адрес связи, находящийся в узле списка вышестоящего уровня, отсылающий сразу к первому члену данного

подсписка.

В данном варианте ассоциативной списковой структуры можно использовать так называемые замкнутые цепные списки, в которых концы списков указываются при помощи адресов обратной связи, обозначающих адреса соответствующих фиксаторов списков. Замыкание цепных списков обеспечивает возможность повторных циклических просмотров списков, что может быть использовано для повышения устойчивости работы системы по отношению к случайным сбоям. Кроме того, при помощи циклических просмотров замкнутых цепных списков можно сравнительно просто находить для любого члена списка его предшественника, что бывает необходимо, в частности, при исключении членов списка.

При таком способе определения концов списков необходимы специальные схемы, которые будут автоматически запоминать адреса фиксаторов списков и сравнивать очередные адреса связи в просматриваемых членах списка с адресом фиксатора. Совпадение этих адресов должно означать окончание списка.

В объектных, структурных и признаковых адресах связи, как уже отмечалось, указываются адреса первых ячеек узлов, в которых находятся списковые слова прослеживающего списка. Можно было бы указывать сразу адреса ячеек соответствующих слов, что облегчило бы процесс прослеживания цепных списков, так как каждый адрес связи сразу же указывал бы ячейку, в которой находится следующий адрес связи того же списка, и т. д. Однако при прослеживании объектных списков, а также при переходах к подспискам с помощью структурных или признаковых адресных слогов все равно нужно проверять, как правило, все дескрипторы узлов, и поэтому определение соответствующего очередного адреса связи внутри узла не потребует дополнительного времени. Адресация же непосредственно по соответствующим словам внутри узлов неудобна тем, что при наличии приращений в узлах и в случае просмотров списков, у которых списковые слова располагаются в приращениях, для обеспечения возможности проверки начальных гнезд узлов необходимо иметь специальные адреса обратных переходов от последних гнезд к начальным гнездам узла. Кроме того, непосредственная адресация по списковым словам, а не по узлам потребует подобного размещения списковых слов внутри узлов.

При адресации же списковых переходов по начальным ячейкам узлов и определении очередных адресов связей по относящимся к ним дескрипторам (или при помощи специальных указателей номеров слов, ячеек или отделений ячеек) обеспечивается большая гибкость в построении узлов (см. пример 4 в § 21). Эти узлы даже для объектов одного типа, стоящих в одних и тех же списках, могут иметь различный состав и порядок рас положения списковых слов. Для проверки всех дескрипторов в узлах не требуется вводить адреса обратных переходов от последних гнезд к начальным гнездам, так как просмотр узлов будет всегда идти в одном направлении — от начала к концу узла. Кроме того, адреса первых ячеек узлов, указываемые в качестве адресов связей, будут одними и теми же для всех случаев отсылок к данному узлу и поэтому могут использоваться также в качестве закрепленных за данными объектами машинных наименований.

Построение поисковых деревьев

Поисковые деревья в рассматриваемой ассоциативной узловой структуре могут использоваться на различных уровнях этой структуры для образования признаковых структур. Отсылка к поисковым деревьям верхних уровней производится при помощи специальных фиксаторов; в фиксаторе указывается адрес первого члена подсписка верхнего уровня данного поискового дерева (при помощи признакового адреса связи) и дескриптор этого дерева.

Фиксаторы сами могут соединяться в цепные описки при помощи второго адреса связи. Отсылки к поисковым деревьям не верхнего уровня осуществляются при помощи признаковых адресов связи. За признаковым адресом связи ставится также дескриптор данного поискового дерева.

Сами поисковые деревья могут строиться, так же как и ассоциативные узлы, по гнездовому способу: для построения дерева выбирается единичное гнездо последовательных ячеек и все подсписки дерева, образуемые в данный момент, размещаются в этом гнезде. В дальнейшем дерево может наращиваться поддеревьями, которые могут размещаться либо в данном единичном гнезде (если в нем есть еще место), либо в других гнездах, являющихся приращениями данного гнезда.

Для указания переходов между членами подсписков и между списками и подсписками внутри поискового дерева может использоваться как прямая, так и относительная адресация. Так как все члены этих подсписков будут находиться, как правило, в одном единичном гнезде, то адреса связи могут иметь небольшое число разрядов (6 или 12). В то же время для членов подсписков поисковых деревьев не имеет смысла использование их адресов в качестве машинных наименований, что бывает нужным для узлов, представляющих отдельные объекты. Поэтому относительная адресация в данном случае не имеет недостатка, связанного с невозможностью использовать относительные адреса в качестве наименований объектов. В подсписках нижних уровней дерева для отсылок к фиксаторам объектных списков могут использоваться также относительные адреса, так как эти фиксаторы будут располагаться в тех же гнездах. В фиксаторах объектных списков для отсылок к первым членам объектных списков используются полные прямые адреса (объектные адреса связи). Каждый член подсписка дерева представляет собой одно списковое слово и состоит из двух адресных слогов и нескольких слогов данных. При построении поисковых деревьев используются адресные слоги трех типов. Каждое слово начинается структурным адресом связи, указывающим в данном случае адрес (относительный) следующего члена данного подсписка. За этим адресом идет слог (слоги) данных, показывающий значение признака, соответствующее данному члену подсписка. Затем следует признаковый адрес связи, указывающий адрес (тоже относительный) первого члена подсписка, ответвляющегося от данного члена. За признаковым адресом связи ставится слог данных, показывающий наименование признака, по которому строится ответвляющийся подсписок. В подсписках нижнего уровня вместо признаковых адресов связи ставятся объектные адреса связи, являющиеся адресами (тоже относительными) фиксаторов объектных списков. Таким образом, в поисковых деревьях в отличие от узлов две составные компоненты каждого дескриптора располагаются не вместе (рядом — в одном списковом слове), а разделены: наименование признака находится в члене подсписка вышестоящего уровня, а значения этого признака указаны в членах ответвляющегося подсписка. Концы подсписков фиксируются постановкой в слове на первом месте (т.е. вместо структурного адреса) адреса обратной связи.

Причем в качестве адресов обратной связи всегда указываются полные прямые адреса. При необходимости продолжения какого-либо подсписка поискового дерева и отсутствии места в данном гнезде вместо указанного адреса обратной связи ставится символ переходного адреса, а за ним — структурный адрес связи. Такое сочетание двух подряд идущих символов (переходного и структурного адресов связи) будет свидетельствовать о том, что следующие символы образуют относительный адрес следующего члена этого подсписка, который находится не в данном, а в другом гнезде. Полный прямой адрес этого члена определяется путем прибавления указанного относительного адреса к начальному адресу следующего гнезда, указанному в первой половине первой ячейки (фиксаторе) данного гнезда.

Рассмотренная система организации ассоциативной информации позволяет строить многоуровневые ассоциативные структуры, в которых от одного или нескольких поисковых деревьев отходят объектные списки, образованные узлами, от которых, в свою очередь, могут ответвляться новые структуры как объектные, так и признаковые и т. д.

Для ускорения поиска объектов с заданными признаками и придания поиску направленного характера можно строить ассоциативные узлы таким образом, чтобы все дескрипторы располагались в узлах подряд в порядке убывания их важности: в начальной части каждого узла будут располагаться все дескрипторы, а в конечной части — все адреса связи. Вместе с каждым адресом связи располагается дополнительный указатель дескрипторов, показывающий, к каким дескрипторам относится этот адрес связи. Адреса связи должны располагаться в порядке убывания этих указателей, рассматриваемых как двоичные числа.

В самих указателях дескрипторов отдельные разряды соответствуют определенным дескрипторам, причем самый первый разряд соответствует первому (наиболее важному) дескриптору, второй разряд — второму дескриптору и т. д. При включении нового объекта он вводится в такие места списков, чтобы быть как можно ближе к объектам, обладающим наибольшим числом ошибок с ним дескрипторов. Ответвляющиеся подсписки в этом случае должны отходить просто от узла, безотносительно к какому-либо адресу связи или дескриптору; поэтому все отсылки к подспискам могут располагаться подряд в конце узла.

Недостатком такого способа является дополнительный расход памяти под коды указателей дескрипторов, однако он позволяет осуществлять переходы по опискам с учетом многих дескрипторов.

Некоторые замечания по размещению ассоциативных списковых структур на МЛ

Как уже упоминалось, в данном варианте системы программирования используются адреса связи, которые являются полными прямыми адресами ячеек памяти в пределах всего адресуемого объема. Эти адреса обеспечивают возможность переходов между любыми двумя ячейками в пределах указанного объема памяти. При существующих запоминающих устройствах большие объемы информации могут располагаться только на МЛ, поэтому в процессе работы необходимо осуществлять периодическую перепись информации между МЛ и ОЗУ. Так как информация на МЛ располагается зонами и перепись производится зонами, то для сокращения количества переписей целесообразно списковые структуры также располагать по зонам. Для этого может быть предложен такой способ. Узлы, относящиеся к одной структуре (списку), размещаются в одной зоне; просмотр объектных списков данной структуры будет происходить при вызове в ОЗУ данной зоны. Построение каждой новой структуры целесообразно начинать в новой зоне, не дожидаясь заполнения ранее начатых зон (заполняемых другими структурами). Таким образом, в процессе работы заполнение МЛ будет идти параллельно по нескольким зонам. При таком зонном принципе расположения узлов для каждой зоны должна строиться своя цепь свободных гнезд.

При полном заполнении зоны и продолжении списковых структур в другую зону в первой полуячейке первой ячейки зоны должен ставиться условный код исчерпания данной зоны, а во второй полуячейке этой ячейки должен указываться номер следующей зоны, являющейся продолжением данной зоны.

§ 18 АССОЦИАТИВНОЕ ПРОГРАММИРОВАНИЕ ДЛЯ УПРАВЛЯЮЩИХ МАШИН

Основным отличием задач автоматического управления от задач накопления и поиска научно-технической информации является большая определенность в составе и характере обрабатываемой информации. При решении задач управления на ЭЦМ имеется возможность заранее определять ожидаемый состав списков и списковых структур, в которых должны фиксироваться объекты, состав и планировку ассоциативных узлов и формуляров (записей) объектов. Узлы и формуляры при этом имеют обычно небольшие размеры, и поэтому их целесообразно делать совмещенными подобно групповым ячейкам при обычном программировании. Ассоциативные узлы и формуляры для объектов разных типов могут быть различными, но для всех объектов одного типа они должны иметь одинаковый формат.

Так как состав формуляров объектов заранее жестко фиксирован, то для поиска объектов может использоваться любая информация, содержащаяся в формуляре. При этом как бы стирается разница между ассоциативной и собственной информацией, заключенной в формулярах объектов. Вся эта информация может, по существу, использоваться как ассоциативная; все зависит от конкретного алгоритма поиска объектов. В одних случаях одни и те же данные об объектах могут использоваться как их признаки при поиске, а в других случаях эти данные будут использоваться для вычислений, справок и т. д.

Заранее устанавливается состав типов объектов и максимальные ожидаемые количества объектов каждого типа. Указанная определенность в постановке задачи позволяет осуществлять предварительное распределение (планирование) оперативной памяти, что упрощает реализацию ассоциативных приемов программирования.

Система памяти ассоциативных управляющих машин в общем случае включает в себя три тина устройств:

1) память команд (ПК);
2) оперативная память (ОЗУ);
3) внешняя память на магнитных лентах (МЛ) или барабанах (МБ).

Внешняя память служит для хранения информации, не участвующей в данный момент в процессе обработки.

Можно было бы в принципе иметь общую оперативную память для команд программы и информации, но, учитывая, что программа в указанных ассоциативных управляющих машинах, как правило, является неизменной и что односторонние ферритовые ЗУ значительно экономичнее и надежнее оперативных ЗУ, целесообразно для хранения программы иметь специальные односторонние ЗУ. Кроме того, некоторые постоянно используемые константы, таблицы, списки и списковые структуры также целесообразно хранить в односторонних ЗУ. При этом часть ОЗУ будет заменяться этим односторонним ЗУ. Собственно оперативная память в рассматриваемых машинах функционально делится на три основные области:

а) рабочую область, содержащую рабочие ячейки и константы; эта область обычно располагается в начальной части оперативной памяти с тем, чтобы обращение к ячейкам этой области осуществлялось по прямым адресам, имеющим ту же разрядность, что и относительные адреса, которые будут использоваться для обращений в других областях памяти;

б) область ассоциативных узлов и формуляров объектов;

в) область поисковых деревьев и фиксаторов списков.

Постоянные поисковые деревья планируются заранее; для переменных поисковых деревьев заранее определяется число уровней и система кодирования признаков, состав же признаковых подсписков заранее не определяется. Эти подсписки образуются автоматически в ходе обработки данных.

В описываемой схеме организации памяти ассоциативной управляющей машины необходимо ее предварительное распределение подобно тому, как это делается для обычных вычислительных машин.

Применение ассоциативного программирования в этом случае имеет целью обеспечить более высокую скорость решения информационно-логических задач, упрощение логической структуры программы и процесса программирования. При рассмотрении организации памяти ассоциативных управляющих вычислительных машин можно исходить из того, что объединение объектов в списки будет производиться строго по принципу их однородности. Это значит, что в одном списке могут состоять только объекты одного типа, имеющие одинаковый состав ассоциативных узлов и формуляров с собственной информацией. В подсписках, отходящих от объектов данного списка, могут объединяться в общем случае и объекты других типов, но в каждом отдельном подсписке могут состоять только объекты одного типа.

Если считать, что все ассоциативные узлы и формуляры объектов одного типа будут размещаться в одной ассоциативной зоне памяти, то для адресации переходов между членами одного списка можно будет применить способ *относительной индексной адресации.* Выбор этого способа адресации обусловливается, во-первых, наличием заранее фиксированных размеров зон и, во-вторых, наличием заранее фиксированного состава ассоциативных узлов. Изменяя индексный (базисный) адрес зоны, можно при необходимости располагать зону в любом месте оперативной, памяти, не нарушая переходов внутри списков и списковых структур. Применение относительной адресации внутри зоны (по отношению к базисному адресу зоны) позволяет существенно (на 30—40%) сократить разрядность адресов связи в списках по сравнению с полной прямой адресацией.

Совмещенные формуляры и ассоциативные узлы мы будем в дальнейшем называть для краткости просто формулярами объектов (в их составе будет находиться определенное число списковых слов, а также собственная информация об объекте).

Внутри ассоциативной зоны формуляры объектов могут располагаться двумя способами:

а) *последовательно* по формулярам; при этом каждый формуляр занимает группу последовательных ячеек памяти и формуляры располагаются один за другим;

б) *параллельно* по словам, в виде подзон, объединяющих однотипные слова формуляров различных объектов. При этом ассоциативная зона делится на несколько подзон соответственно числу слов в формуляре. В первой подзоне располагаются все первые слова всех формуляров объектов; во второй подзоне располагаются в том же порядке все вторые слова формуляров объектов и т. д.

Так как все формуляры объектов одного типа одинаковы, т. е. списковые слова, относящиеся к одному списку, располагаются в них на одинаковых местах, то использование параллельного способа расположения информации дает некоторое упрощение схем адресации при просмотрах списков. При этом переходы между членами одного списка будут осуществляться в пределах только той подзоны, в которой находятся соответствующие списковые слова. В этом случае разрядность адресов связи будет определяться максимальным числом ячеек, имеющихся в подзоне, и для переходов между узлами в качестве адресов связи могут использоваться сокращенные относительные адреса.

Вообще говоря, можно использовать такую же сокращенную разрядность адресов связи и при расположении данных последовательно по формулярам объектов, но для этого необходимо при каждом обращении к очередному члену списка по относительному адресу связи, указанному в формуляре, вырабатывать действительный относительный адрес путем умножения указанного адреса связи на число ячеек в формуляре и прибавления относительного номера ячейки внутри формуляра. При данном рассмотрении мы делаем допущение о том, что слова внутри формуляра располагаются в отдельных ячейках. Однако способ расположения информации по формулярам, соответствующим отдельным объектам, обладает рядом существенных преимуществ с точки зрения гибкости использования объема памяти, отведенного под ассоциативную зону, и простоты обращения к различной информации, относящейся к одному и тому же объекту. При расположении по формулярам мы можем в широких пределах менять размер зоны даже в процессе вычислений, используя ее нижнюю часть под другую информацию, что трудно сделать при параллельном (пословном) заполнении зоны. При расположении всей информации об одном объекте в одной группе ячеек, обращение к любой величине внутри формуляра может осуществляться заданием номера старшего разряда этой величины по отношению к началу формуляра и указанием количества разрядов. Весь формуляр может рассматриваться как одна большая ячейка. При построении схем выработки действительных адресов для просмотров списков следует иметь в виду два обстоятельства. Во-первых, если размеры формуляров делать кратными двум, то умножение относительного адреса связи на степень двойки будет сводиться к простому сдвигу. Во-вторых, просмотры списков будут сопряжены, как правило, с какими-то преобразованиями или

проверками величин, относящихся к каждому просматриваемому члену списка. Поэтому процесс выработки очередного адреса для обращения к следующему члену списка может быть совмещен во времени с указанными проверками и преобразованиями информации об объектах и не замедлит общего процесса обработки информации.

Как упоминалось выше, в подсписках, отходящих от некоторых членов данного списка, могут объединяться объекты других типов, причем в каждом конкретном списке или подсписке будут объединяться объекты только одного типа.

Формуляры объектов разных типов будут иметь различную, структуру и размещаться в разных зонах. Для каждого типа объектов отводится своя зона. Таким образом, переходы к подспискам, ответвляющимся от членов данного списка, в общем случае будут связаны с переходами в другие зоны, и поэтому для задания таких переходов должны использоваться полные прямые адреса. Переходы к подспискам могут задаваться при помощи специальных ячеек, играющих роль фиксаторов этих подсписков. Для упрощения схемной реализации относительной адресации целесообразно начальные (базисные) адреса зон выбирать кратными степеням двойки. При этом старшие разряды полного адреса любой ячейки данной зоны будут автоматически устанавливаться на соответствующий регистр при первом обращении к списку, расположенному в этой зоне. При последующем просмотре членов списка будут изменяться только младшие разряды полного адреса, соответствующие относительным адресам связи. В качестве адресов связи в объектных списковых словах целесообразно использовать относительные индексные адреса начальных ячеек формуляров, которые будут являться машинными наименованиями объектов.

Во всех списках, в которые входит какой-либо объект, в качестве адресов связи будут фигурировать одинаковые машинные наименования объекта, что упростит поиск объектов. Однако при обращении к любому списку необходимо дополнительно указывать относительный номер ячейки внутри формуляра, в котором находится соответствующее списковое слово. Этот номер ячейки может указываться один раз в фиксаторе списка.

В целях конкретизации высказанных выше общих соображений об организации оперативной памяти для ассоциативных управляющих машин рассмотрим некоторые ориентировочные числовые характеристики. Допустим, что каждое списковое слово размещается в одной ячейке и занимает ее не всю, а только 12 младших разрядов. Оно включает в себя 10 разрядов адреса связи (что позволяет иметь в каждом списке до 1023 членов) один разряд признака структурного или объектного слова и один разряд признака конца списка. Адреса связи в последних членах списков используются в качестве адресов обратной связи (они указывают вершины списков) и служат для контроля. Остальные части каждой ячейки используются для размещения признаков объектов и других данных об объектах.

Наличие нулей на месте некоторого спискового слова будет указывать на то, что данный объект не входит в данный список.

Все ячейки памяти равноправны, и порядок заполнения их информацией и характер использования определяются программистом путем применения тех или иных команд обращения к ним. В частности, при ошибочном обращении при прослеживании списка к ячейке, содержащей только собственную информацию, 12 младших разрядов этой ячейки будут использоваться в качестве спискового слова, и, наоборот, при обращении для вычислений к ячейке, содержащей списковое слово, ее содержимое будет использоваться как обычное двоичное число.

Для определенности будем рассматривать оперативную память емкостью в 32 768 ячеек по 32 разряда. Считаем, что в каждой ячейке может располагаться не более одного спискового слова, размещаемого в 12 последних разрядах.

По существу в данном случае списковое слово представляет собой просто адрес связи, так как отдельных признаков объекта в составе спискового слова не предусматривается. Это обусловлено тем, что формуляры объектов имеют жесткий, заранее установленный формат и в качестве поисковых признаков могут использоваться любые данные, содержащиеся в формуляре. В этом случае 20 первых разрядов ячейки могут служить для размещения признаков объектов или собственной информации об объектах. Некоторые ячейки могут использоваться полностью для размещения собственной информации об объектах. Планировка заполнения ячеек в формулярах определяется программистом заранее и сохраняется жесткой на все время работы управляющей ассоциативной машины.

Как было сказано выше, для переходов между членами внутри одного списка применяется индексная относительная адресация, при которой в качестве адресов связи используются машинные наименования объектов, т. е. относительные адреса начальных ячеек формуляров, деленные на число ячеек в формуляре. При этом для выработки фактических адресов очередных членов списков необходимо выполнять следующие действия:

а) умножить адрес связи (машинное наименование объекта) на число ячеек в формуляре. Это число является постоянным для всех формуляров, расположенных в данной зоне, и может храниться в первой ячейке зоны. При обращении к некоторой зоне это число переносится на специальный регистр, на котором оно должно храниться в течение всего времени работы с данной зоной;

б) прибавить к полученному произведению постоянную величину, указывающую номер спискового слова (ячейки) внутри данного формуляра. Этот номер будет постоянным для всех списковых слов, относящихся к данному списку, и может быть указан в фиксаторе данного списка;

в) прибавить полученный результат к базисному адресу зоны, т. е. адресу, на единицу меньшему адреса первой ячейки этой зоны. При обращении к некоторой1 зоне этот адрес должен быть записан на специальный регистр, на котором он должен храниться в течение всего времени работы с данной зоной.

Для реализации ответвлений к подспискам используются структурные адреса связи (признак S=l), в которых собственно адрес имеет также 10 разрядов. При этом адрес связи указывает относительный адрес ячейки, являющейся фиксатором подсписка и располагающейся в определенной области оперативной памяти. Относительный адрес также индексный, т. е. он задается по отношению к началу этой области. Очевидно, что при

10-разрядном адресе в указанной области может быть не более 1023 фиксаторов; ясно, что при этом в ассоциативной управляющей машине может быть построено одновременно не более 1023 списков и подсписков. Будем считать, что каждый фиксатор занимает одну 32-разрядную ячейку, в которой первые 6 разрядов определяют номер спискового слова, относящегося к данному списку, внутри формуляров, следующие 15 разрядов представляют собой полный прямой адрес первого члена данного списка (являющийся подсписком по отношению к списку, который отсылает к данному фиксатору), а последние 11 разрядов — сочетание одного разряда признака типа адреса (*S*) и 10 разрядов адреса связи. При *S* = 0 этот адрес связи определяет продолжение основного списка, т. е. как бы полностью заменяет собой тот десятиразрядный адрес связи, который был использован в структурном слове для отсылки к данному фиксатору. При *S*=1 этот адрес указывает положение другого фиксатора, который определяет вершину другого подсписка, ответвляющегося от того же члена основного списка. В этом фиксаторе, в свою очередь, также может быть отсылка к следующему фиксатору и т. д. В последнем фиксаторе такой фиксаторной цепочки должен стоять признак *S*=0 и адрес связи указывает продолжение основного списка. Такой способ позволяет строить так называемые «грозди» ответвлений подсписков от одного члена основного списка, причем ответвления будут различных уровней. Это может быть использовано, например, для построения объектных включающих структур при селекции некоторых объектов, когда один и тот же объект может быть членом нескольких групп объектов различной степени укрупнения. При этом от одного члена списка верхнего уровня могут отходить несколько подсписков последовательно понижающихся уровней.

Фиксаторы списков могут быть адресуемыми и неадресуемыми. Адресуемые фиксаторы представляют собой отдельные ячейки, адреса которых известны программисту и которые служат для указания положения вершин основных списков или списковых структур. В таких фиксаторах первые 15 разрядов используются под полные прямые адреса вершин списков. Остальные 11 разрядов могут использоваться по усмотрению программиста для размещения некоторых признаков или характеристик списка. Неадресуемые фиксаторы служат для автоматического образования подсписков или неадресуемых списков в процессе обработки информации. Все ячейки области памяти, отведенной под размещение фиксаторов, за исключением ячеек, являющихся адресуемыми фиксаторами, завязаны в цепь свободных ячеек (СЯ). Из этой цепи они берутся автоматически при образовании подсписков и автоматически возвращаются в нее при стирании подсписков.

В ассоциативных зонах все свободные ячейки разделены на группы по числу ячеек в соответствующих формулярах и такие группы ячеек (гнезд) соединены в цепные списки свободных формуляров объектов (СФ). Из этих списков формуляры берутся при записи новых объектов; в эти же списки формуляры возвращаются при выбытии объектов. Завязка свободных формуляров в СФ производится при помощи полных прямых 15-разрядных адресов, помещаемых в старших разрядах первых ячеек свободных формуляров. В первой ячейке каждой ассоциативной зоны хранится (в первых 15 разрядах) адрес очередного свободного формуляра (его первой ячейки) и в следующих 6—8 разрядах — число ячеек в формулярах данной зоны. Другие разряды остаются резервными. При любом первом обращении к какому-нибудь описку или подписку в этой зоне по полному прямому адресу вершины списка, взятому из фиксатора, должно автоматически (схемно) производиться обращение к соответствующей части первой ячейки зоны для выборки из нее кода, определяющего число ячеек в формулярах объектов данного типа.

Этот код необходим для определения фактических адресов переходов при просматривании списков, расположенных в данной зоне. При записи новых объектов или исключении выбывших используется первая часть первой ячейки зоны, являющаяся указателем цепи соответствующих свободных формуляров. Следует подчеркнуть, что машинными наименованиями объектов будут относительные индексные адреса их формуляров, что может быть использовано для поиска объектов по их машинным наименованиям.

Изложенные выше общие соображения об организации машинной памяти в управляющих машинах, а также в информационно-логических машинах могут быть использованы при разработке систем программирования, основанных на ассоциативных приемах. Естественно, что в процессе конкретной реализации описанных приемов могут быть сделаны соответствующие изменения и уточнения организации памяти и способов кодирования.

7. МЕТОДИКА АССОЦИАТИВНОГО ПРОГРАММИРОВАНИЯ

Рассмотренный нами алгоритмический язык АЛГЭМ позволяет описывать различные алгоритмы переработки информации на ЭЦМ в тех случаях, когда объем и вид данных (количество и структура записей) заранее известны и могут быть точно описаны. Однако существуют задачи, в которых объем и состав информации заранее неизвестны. Более того, в ходе обработки структура информации меняется, что требует соответствующего перераспределения памяти. К числу таких задач относятся накопление и поиск научной и библиографической информации, машинный перевод, автоматическое программирование, опознавание образов, моделирование обучения и т. д.

Для эффективного описания подобных алгоритмических процессов необходим специальный алгоритмический язык. Работы в этом направлении ведутся уже в течение многих лет и привели к созданию ряда языков, предназначенных для так называемой неарифметической обработки данных (LISP, IPL-V, COMIT и др.).

В настоящее время определилась тенденция к унификации языков программирования и построению подобного языка на основе АЛГОЛа.

В настоящей главе рассмотрим некоторое расширение языка АЛГЭМ (представляющего, в свою очередь, расширение АЛГОЛа), предназначенное для более эффективного описания различных алгоритмов с гибкой системой организации данных. Это расширение сводится к введению двух адресных операций, представляемых двумя видами скобок (скобки адреса и скобки содержимого), описателя **список,** который служит для описания списковых величин и описателя **формат,** служащего для указания формата списковых величин (в случае, если форматов несколько).

Используя это дополнение и все остальные средства АЛГЭМа, в особенности метод процедур, можно записывать алгоритмы процессов переработки информации, включающей в себя как обычные (несписковые) величины, так и списковые величины.

Основными средствами ассоциативного программирования являются:

а) использование адресов связи для построения цепных списков различных видов, объединяющих объекты **с** общими признаками;

б) использование списковых структур, представляющих собой многоуровневые списки, т. е. списки с ответвляющимися от них подсписками, для предоставления иерархических систем организации данных;

в) использование продвигаемых списков (стэков или магазинов) для последовательного запоминания и восстановления данных при рекурсивных вычислениях;

г) организация свободной памяти в виде цепного списка ячеек, обеспечивающая гибкость и полноту использования всего объема памяти и исключающая необходимость в ее детальном предварительном распределении.

Существует много различных конкретных способов и форм машинной реализации ассоциативного программирования.

Указанные идеи усиленно разрабатываются многими учеными различных стран.

К числу первых и наиболее значительных работ в этой области относятся работы А. Ньювелла, Симона, Шоу, Тонге, Гелернтера. За рубежом этот круг вопросов известен под разными названиями: списковая обработка, узловая обработка, цепная адресация, метод управляющих слов. В некотором отношении сюда примыкает методика адресного программирования Е. Л. Ющенко. Работы в указанной области представляют собой, как правило, описания специальных языков программирования, связанных с определенной машинной реализацией (т.е. расположением данных в ячейках машинной памяти).

В настоящей главе рассматривается методика описания алгоритмов обработки списковой информации, основанная на использовании АЛГОЛа и его расширения — АЛГЭМа.

Язык АЛГЭМ предназначен для описания процессов обработки информации, имеющей жесткую, заранее установленную структуру. По сравнению с АЛГОЛом в АЛГЭМе имеются необходимые средства для описания составных величин, указания отдельных разрядов или символов, образующих величины, и, кроме того, введен тип строчных величин.

Приводимое ниже описание методики ассоциативного программирования следует рассматривать как дополнение АЛГЭМа, который является, в свою очередь, расширением АЛГОЛа.

Основной особенностью этой методики является возможность представления процессов обработки так называемой списковой информации, имеющей переменный состав и произвольное размещение в памяти машины.

§ 19 АДРЕСНЫЕ СООТНОШЕНИЯ

В АЛГОЛе различаются следующие классы величин: простые (или, как их можно называть, одиночные) переменные, массивы, метки, переключатели и процедуры, т. е. объекты, которые могут иметь свои идентификаторы. Правда метки могут быть представлены целыми числами без знака, и, таким образом, числа частично тоже попадают под определение величины.

Мы будем пользоваться более широким определением понятия «величина», включив в него кроме перечисленных объектов также числа, строки и списки. Можно дать следующее синтаксическое определение понятия «величина»:

<величина> : :== <число> | <метка> | <строка> | <простая переменная> | <переменная с индексами> | <массив> | <список> | <переключатель> | <процедура>

Таким образом, величина — это некоторая единица информации (не количества информации), представленная либо своим идентификатором (наименованием), либо непосредственно своим значением (последнее относится к числам, строкам, меткам).

В машинах все величины представляются цифровыми кодами (как правило, двоичными кодами). Этими кодами представляются как значения, так и наименования величин. Коды хранятся в ячейках памяти и называются содержимым. Каждой ячейке памяти ставится в однозначное соответствие некоторый постоянный код, который называется ее адресом. Наличие двух видов машинных кодов (адресов и содержимых) является принципиальной особенностью машинной переработки информации, которую необходимо учитывать при построении алгоритмов решения информационно-логических задач.

Обычно понятия адреса и содержимого привязаны жестко к конкретным машинам. Для того чтобы при программировании на алгоритмическом языке, не связанном с конкретными машинами, можно было учитывать указанную особенность машинного решения задач, необходимо ввести в язык общие понятия адреса и содержимого, не привязывая их к конкретной машине.

Дадим следующие формальные синтаксические определения понятий адреса и содержимого:

<адрес> : : = <арифметическое выражение> | |‾<величина>‾|

<содержимое> : : = |_<адрес>_|

Верхние уголковые скобки называются адресными: они показывают, что вместо величины, заключенной в эти скобки, должен быть взят ее адрес, т. е. некоторое целое число.

Нижние уголковые скобки называются скобками содержимого: они показывают, что вместо адреса, заключенного в эти скобки, должна быть взята величина, находящаяся в ячейке с данным адресом. В тех случаях, когда указанные символы (скобки адреса и скобки содержимого) используются в программах, написанных на алгоритмическом языке, без дополнительных описаний так называемых адресных соотношений (см. ниже), они имеют смысл буквально адресов и содержимых ячеек памяти той конкретной машины, на которой предполагается решение данной задачи. При этом предполагается, что каждая из величин, представляемых с использованием адресных скобок или скобок содержимого, размещается в отдельной ячейке памяти и располагается там стандартным для данной машины способом (с фиксированной запятой, с плавающей запятой или в виде набора алфавитно-цифровых символов). В тех же случаях, когда расположение величин в ячейках памяти не является стандартным (т. е. одна величина занимает несколько ячеек или в одной ячейке находится несколько величин), а также в тех случаях, когда в качестве содержимых, имеющих один адрес, желательно

иметь сложные величины (массивы, списки, составные переменные, процедуры, переключатели), необходимо в состав описаний данного блока программы включать дополнительные описания адресных соотношений.

Описание адресного соотношения строится следующим образом.

Адресное соотношение состоит из двух частей, правой и левой, соединенных знаком равенства. В левой части в адресных скобках указывается величина (или несколько величин), являющаяся содержимым по данному адресу (если указывается несколько величин, имеющих общий адрес, то они разделяются запятыми). В правой части указывается величина, являющаяся адресом. В качестве адресов могут указываться числа, переменные или арифметические выражения. Например, адресом может быть сумма двух переменных, из которых одна представляет собой базисный адрес, а вторая — относительный адрес. Каждое описание адресного соотношения, как и любое другое описание, заканчивается точкой с запятой. Принимается правило, что перечисление нескольких величин через запятые означает их расположение в одной ячейке или в нескольких подряд идущих ячейках памяти; величины, перечисленные через запятые, представляют собой последовательный список, положение которого полностью определяется адресом его первого члена. Подобным образом могут указываться адресные соотношения для величин, являющихся массивами или элементами массивов, для составных переменных, а также для величин, входящих в состав списков. Рассмотрим этот вопрос подробнее.

Если величиной, адрес которой требуется обозначить, является массив переменных, то в адресных скобках указывается только идентификатор этого массива. Тогда содержимым, которое будет извлекаться по данному адресу, будет весь массив полностью, причем размеры этого массива будут определяться его основным описанием, задаваемым обычным способом.

Если же содержимым, которое должно извлекаться по заданному адресу, должен быть не весь массив, а его один элемент, то при описании адресного соотношения за идентификатором массива в адресных скобках нужно указать индексные (квадратные) скобки (пустые).

Аналогичным образом при задании адресных соотношений для составных переменных, если требуется, чтобы содержимым по данному адресу считалась вся составная переменная, в адресных скобках должен ставиться только идентификатор этой составной переменной.

Если нужно, чтобы содержимым данного адреса являлась некоторая компонента составной переменной, то в адресных скобках указывается идентификатор этой компоненты с уточнением или без уточнения (если отсутствие уточнения не приводит к неопределенности).

Если в адресных скобках указан идентификатор списка, то содержимым данного адреса будет не весь список, а только его заголовок (см. ниже списковые описания). Если же нужно иметь в качестве содержимого по данному адресу какую-либо компоненту, входящую в состав заголовка списка, то это делается при помощи адресного соотношения, записываемого таким же образом, как для компоненты составной величины.

Адресные соотношения для величин, входящих в состав списковых членов, даются по правилам, аналогичным тем, которые указаны для компонент составных переменных. Если требуется устранить неопределенность, то идентификаторы этих величин указываются в адресных скобках совместно с уточнениями. В качестве уточнений могут выступать не только идентификаторы составных переменных, в которые входят данные компоненты, но и идентификаторы списков, в которые входят списковые члены, имеющие в своем составе данные компоненты.

Заметим, что в качестве адресов в правых частях адресных соотношений могут фигурировать не только простые переменные, но и элементы массивов, компоненты составных переменных, а также компоненты списковых членов. Во всех случаях, если не возникает неопределенности, эти величины указываются просто своими идентификаторами (например, адреса, являющиеся элементами массивов, указываются без индексных скобок, так как ясно, что адресом может быть только один элемент массива). В тех случаях, когда возникает неопределенность, эти величины указываются совместно с уточнениями, составленными по правилам АЛГЭМа.

Иногда адресом некоторой величины могут быть несколько переменных (или арифметических выражений); в этом случае все они перечисляются в адресном соотношении через знак равенства. Например, если адресом величины *A* будут величины *i*, *j*, *k,* то адресное соотношение имеет вид

$$|\overline{\ }A\overline{\ }| = i = j = k$$

При наличии описания адресного соотношения содержимым по данному адресу всегда является величина, указанная в адресных скобках в левой части описания.

В тех случаях, когда выдерживается стандартное расположение величин, согласно которому каждая величина помещается в одну ячейку, описания адресных соотношений приводить не нужно. Не нужно приводить описаний адресных соотношений и для величин, не использующихся при программировании совместно с адресными скобками или скобками содержимого, даже если они не имеют стандартного расположения.

Синтаксическое определение описания адресного соотношения имеет вид:

<описание адресного соотношения> : : |¯<величина>¯| = <арифметического выражения > |

<описание адресного соотношения> = <арифметическое выражение>

Примеры:

1. Пусть величины *p, q, r*, *s* должны помещаться в одну ячейку с адресом *a*. Описание адресного соотношения для этого случая будет иметь вид

$$|\overline{\ }p,\ q,\ \ r,\ s\overline{\ }| = a;$$

2. Пусть составная величина НАРЯД занимает несколько подряд идущих ячеек и адрес первой ячейки равен *b*. Адресное соотношение должно быть записано в виде

$$|\overline{\ }\text{НАРЯД}\overline{\ }| = b;$$

3. Пусть, например, адресами элементов массива «заголовок списка» являются соответственно элементы массива «дескриптор». Тогда адресное соотношение будет выглядеть следующим образом:

⌈‾заголовок списка [ ]‾⌉ == дескриптор;

4. Пусть имеется три величины *b, m, n,* которые образуют цепной список. Каждая из этих величин расположена в первой половине одной ячейки, во второй половине которой находится адрес связи, т. е. число, указывающее адрес ячейки, в которой расположена следующая величина данного списка. Во второй половине ячейки, в которой находится последний член данного цепного списка, помещается некоторый условный постоянный код, обозначаемый идентификатором КС и указывающий конец списка. Пусть адрес ячейки, в которой находится первый член этого списка, равен *a,* тогда адресное соотношение для указанного случая будет иметь вид

$$\lceil\!\overline{\ } b, \lceil\!\overline{\ } m, \lceil\!\overline{\ } n, \text{КС}\ \overline{\ }\rceil\ \overline{\ }\rceil\ \overline{\ }\rceil = a;$$

Понятия адреса и содержимого в общем случае можно интерпретировать как понятия наименования и значения некоторого объекта, не связывая их с машинной реализацией. Эти понятия носят относительный характер: одна и та же группа символов или один и тот же код может представлять собой наименование одной категории объектов и в то же время может быть значением одного объекта, являющегося членом категории более высокого уровня классификации.

В машинной интерпретации код адреса может рассматриваться как наименование той единицы информации, значение которой представлено кодом содержимого этой ячейки. Относительный характер понятий наименования категории объектов и значения объекта при машинной реализации проявляется в том, что содержимым некоторой ячейки может быть адрес другой ячейки и т. д.

Выделение компонент в содержимых

Адреса и содержимые могут участвовать в качестве операндов во всех операциях, предусмотренных в алгоритмическом языке, а также фигурировать в левых частях в операторах присваивания.

Адресные скобки и скобки содержимого могут применяться многократно и, таким образом, будут получаться адреса и содержимые более высоких рангов.

К адресам и содержимым полностью применимы описанные в АЛГЭМе способы выделения компонент составных переменных и разрядов. Если для содержимых не дано описания вида, то подразумевается, что указание разрядов относится к двоичным разрядам ячейки машинной памяти той машины, на которой предполагается решение задач. При этом считается, что все двоичные разряды ячейки перенумерованы подряд слева направо от 0 до некоторого *n* и указание разрядов определяет эти двоичные разряды ячейки.

Выделенные разряды переменной или выделенная компонента составной может использоваться в качестве адреса, т. е. заключаться в скобки содержимого. Заключение выделенной компоненты составной величины (или выделенных разрядов) в скобки содержимого означает, что эта компонента должна использоваться в качестве адреса независимо от ее исходного (т. е. до выделения) положения в составной величине. Таким образом, выделенная компонента как бы автоматически сдвигается в сторону младших разрядов адреса. Аналогичным образом используются выделяемые компоненты и при выполнении с ними арифметических операций, т. е. выделенная компонента всегда рассматривается как отдельная величина.

Согласно правилам АЛГЭМа имеют место следующие естественные соотношения между выделенными компонентами в правых и левых частях операторов присваивания. Если правой частью оператора присваивания является выделенная компонента, а для величины, стоящей в левой части, не указана компонента или разряды, то значение выделенной компоненты правой части присваивается всей величине, стоящей в левой части. Если же для величины, стоящей в левой части оператора присваивания, указаны разряды или компонента, то выделенная компонента правой части устанавливается так, чтобы совпали младшие разряды выделенных компонент в левой и правой частях оператора присваивания (это справедливо для целых чисел. В случае дробных или смешанных чисел выравнивание должно производиться по положению десятичной запятой).

В этом случае при выполнении оператора присваивания все остальные части переменной, указанной в левой части оператора присваивания, кроме выделенной компоненты (или выделенных разрядов), остаются без изменения.

Если выделенная компонента правой части имеет большее число разрядов, чем выделенная компонента левой части, то не совпавшие разряды (лишние) компоненты правой части пропадают. Например, запись вида *a.*|_*A*_|:=*C*; указывает, что значение *C* присваивается величине *a,* представляющей собой компоненту составной величины, имеющей адрес, равный значению величины *A*. В данном случае обозначение компоненты содержимого в виде *a.*|_*A*_| имеет смысл идентификатора величины и присваивание идет так, как если бы в левой части стоял просто идентификатор величины *a.* Такой же смысл идентификатора имеет задание части ячейки при помощи указания разрядов.

Запись вида (*i*:*j*) |_*A*_|*: = C;* определяет присваивание значения величины *C* группе разрядов ячейки, адрес которой равен значению величины *A*. Младший разряд принимающей группы имеет номер *i,* старший разряд — номер *j*. Если у величины, являющейся содержимым по адресу *A,* имеются описания адресного соотношения и вида, то левой частью оператора присваивания будет группа символов этой величины с номерами от *i* до *j*.

Если нужно указать, что выделенная компонента содержимого сама обозначает адрес ячейки, в которую посылается значение некоторой величины, то записывается следующее выражение:

$$|\_a.|\_A\_|\_|:=C; \quad |\_(i{:}j)|\_A\_|\_|:=C$$

Эти записи обозначают, что значение некоторой величины С присваивается содержимому ячейки, имеющей адрес, равный либо значению *a.* |_*A*_|, либо значению кода, находящегося в разрядах с *i* по *j* в ячейке с адресом, равным значению *A*.

§ 20 ОПИСАНИЯ СПИСКОВ

Как известно, все величины (за исключением некоторых стандартных), используемые в алгоритмических программах, должны быть описаны в начале тех блоков, в которых они используются (или во внешних блоках).

Это требование относится и к списковым величинам (спискам и списковым структурам), хотя назначение описаний списковых величин существенно отличается от назначения описаний несписковых величин. Описания обычных несписковых переменных (элементарных и составных) используются транслятором для:

а) распределения памяти под эти величины;

б) определения характера операций над величинами (с округлением или без округления и др.).

Списковые величины, как уже говорилось, не требуют для себя распределения памяти в обычном понимании этого термина. Размеры и структура списков могут меняться в процессе работы в широких пределах; они образуются автоматически в соответствии с фактическим составом обрабатываемой информации и располагаются на имеющихся свободных местах в памяти машины.

Однако, несмотря на указанную гибкость в использовании списковых величин, при программировании задач, в которых участвуют списковые величины, требуется соблюдение двух основных условий:

а) должна быть заранее определена структура (формат) списковых членов и расположение их относительно границ ячеек памяти машины;

б) должна быть произведена предварительная организация свободной области памяти, выделяемой подсписковые величины. Несписковая область распределяется обычным образом, а списковая область организуется
в соответствии с типом используемых списков (либо в виде общего цепного списка свободных ячеек, либо в виде цепного списка свободных гнезд ячеек фиксированного размера, либо в виде последовательного списка свободных зон на МЛ или в ОЗУ и т. д.).

Указанные ограничения вызваны тем обстоятельством, что процедуры обработки списковой информации привязаны к форматам списковых членов и способам организации свободной памяти машины.

Для описания списковых величин вводится описатель **список,** который вместе с символом **уровень** образует пару описательных скобок, подобную паре: **составной... уровень.** Описание списка состоит из двух частей: из описания заголовка списка (фиксатора списка) и из описания структуры спискового члена. Эти две части описания разделяются точкой с запятой. Описание списка начинается символом **список**, после которого идет описание заголовка списка, а после него дается структура спискового члена, которая заканчивается символом **уровень.** Как заголовок списка, так и списковый член в общем случае представляют собой составные переменные. В частном случае заголовок списка может вообще отсутствовать.

Так как в состав спискового члена может входить в качестве элемента другой список или даже несколько списков, то эту возможность необходимо предусмотреть соответствующим построением описания списка, а именно построением структуры спискового члена.

Структура спискового члена аналогична структуре составной переменной, за исключением того, что наряду с элементами описаний, принятыми в АЛГЭМе в структуру спискового члена могут входить и описания списков.

Приведем описание составной величины, принятое в АЛГЭМе:

<элемент описания> : : = <описание типа> | <описание массивов> | <описание составной величины>

<составная величина> : := <идентификатор переменной> | **массив** <идентификатор массива> [<список граничных пар>]

<структура составной величины> : : = <элемент описаниям> | <структура составной величины>; <элемент описания>

<описание составной величины> :: = **составной** <составная величина>; <структура составной величины> **уровень**

Используя это определение составной величины, можно построить следующее определение описания списка:

<идентификатор списка> : : = <идентификатор>

<описание заголовка списка> : : = <пусто> <идентификатор списка> | <описание составной величины>

<элемент структуры спискового члена> : : = <элемент описания> | <описание списка >

<структура спискового члена> : : = <элемент структуры спискового члена> | <структура спискового члена>; <элемент структуры спискового члена>

<описание списка> ::= **список** <описание заголовка списка>; <структура спискового члена> **уровень**

Следует сделать несколько пояснений относительно описания заголовка списка. Здесь возможны три варианта. Во-первых, описание заголовка списка может вообще отсутствовать. В этом случае сразу же за символом **список** будет стоять точка с запятой, за которой будет следовать описание структуры спискового члена. Во-вторых, описание заголовка списка может быть представлено просто идентификатором списка, за которым стоит точка с запятой. В этом случае идентификатор списка играет роль некоторого примечания, поясняющего название или назначение данного списка при чтении алгоритма людьми. Транслятором этот идентификатор списка не используется. В-третьих, описание заголовка списка может быть представлено в виде описания составной величины. В этом случае в качестве составной величины будет указываться идентификатор списка, который является фактическим идентификатором заголовка (фиксатора) данного списка. Структура этой составной величины будет представлять собой структуру заголовка списка. Здесь может указываться число членов в списке, адрес первого члена списка и другие данные, относящиеся к списку в целом. При наличии такого описания заголовка списка транслятор выделит соответствующую ячейку (ячейки) в памяти машины для размещения фиксатора списка. Отсюда ясно, что фактический заголовок списка может быть представлен в описании списка только в виде составной переменной. При этом за символом **список** должен сразу стоять символ **составной.** Таким способом могут описываться списки любого типа (цепные, узловые, гнездовые, последовательные).

Важно то, что для величины, заключенной между символами **список... уровень,** обычного распределения

памяти производить не требуется, а заданная структура списков учитывается при предварительной организации свободной памяти и в процессе составления операторов программы.

Сравнение адресного и индексного способов обращения к членам списков

Сравним возможности обращения к членам списков при помощи их адресов и при помощи индексов переменных, используемых в АЛГОЛе.

Операция получения содержимого по заданному адресу, вообще говоря, может быть представлена в виде операции обращения к элементу массива по заданному индексу. Для этого необходимо ввести в описание данных в соответствующем блоке программы условный одномерный массив, представляющий собой последовательность ячеек памяти. Индексы элементов массива будут совпадать с адресами ячеек. Иногда достаточно ввести условный массив, который будет охватывать не всю память, а только ту зону, в которой должна размещаться списковая информация. Естественно, что при этом каждый член списка должен занимать одну ячейку памяти.

Для иллюстрации двух указанных возможностей обращения к списковым величинам рассмотрим пример. Пусть требуется осуществить просмотр некоторого цепного списка, расположенного в оперативном запоминающем устройстве (ОЗУ), и отбор членов, у которых признак *П* равен заданному значению (ЗП). Введем условный массив ОЗУ [1 : *N*], охватывающий некоторую зону памяти из *N* ячеек, в которой будет располагаться список. Будем считать, что каждый член списка расположен в одной ячейке и состоит из трех компонентов: признака *П,* адреса связи АС и признака конца списка КС.

Адрес связи АС указывает положение следующего члена списка относительно начала данной зоны ОЗУ (т.е. это относительный, а не абсолютный адрес). Индекс элемента в условном массиве ОЗУ [1: *N*], представляющего собой первый член списка (вершину списка), обозначим через *Ф*. Он является заданным. Отобранные члены будем записывать в массив *T* [1 : *n*]; значение индекса этого массива будет указывать число отобранных членов.

Величины ЗП, *Ф, T*[*i*], *i, n* будем считать глобальными по отношению к рассматриваемому блоку программы. Адреса связи АС являются порядковыми номерами ячеек той зоны ОЗУ, которая выделена для размещения списка.

```
начало.....;
составной массив ОЗУ [1 : N];
целый   П вид 1 (10), АС вид 1 (20);
логический КС уровень
целый J, j;.....;
.....

i: =1, j: = Ф;
для  J : = j  пока ¯| КС. ОЗУ[J],  j цикл
начало если П. ОЗУ[J] = ЗП то
начало T [i] : = ОЗУ[J]; i : = i + 1 конец;
j: =АС. ОЗУ[J] конец
.....
конец
```

Этот же участок программы, записанный с использованием адресных символов и описателя **список**, будет иметь вид:

```
начало ..... ;
список цепь; составной элемент, целый П вид 1 (10), АС вид 1 (20); логический КС уровень
уровень;
целый J, j; ..... ;
|¯ элемент ¯| = АС;
.....
……..
i : = 1;  j : = Ф
для АС: = j пока ¯| КС. |_АС_|, j цикл
начало если П. |_АС_| = ЗП то
начало T [i]: = |_AC_|; i: = i+1; конец
j: = АС. |_АС_| конец
.....
конец
```

Здесь идентификатором всего списка является «цепь» (этот идентификатор фактически не используется), а идентификатором члена списка — «элемент».

Сравнение обоих вариантов записи показывает, что они примерно одинаковы; второй вариант содержит в описаниях блока дополнительное описание (а именно адресное соотношение). Но это описание в данном случае может быть опущено, так как согласно условию каждый член списка находится в отдельной ячейке, т. е. имеет место стандартное расположение списковых величин.

Принципиальным недостатком первого варианта является необходимость введения описания фиктивного массива ОЗУ [1 : *N*], в котором только некоторые ячейки будут заняты членами списка. При этом необходимо как-то отличать в описаниях такие фиктивные массивы от настоящих массивов, так как фиктивные массивы могут накладываться друг на друга, и, кроме того, они должны иметь по возможности большие размеры для того, чтобы можно было обеспечить гибкость в расположении списков в памяти машины.

Использование адресных символов для описания цепных списков является более удобным по следующим причинам:

а) они более естественны для машинного представления и наглядны для программирования, так как соответствуют существу машинных операций при обращении к списковым величинам;

б) применение скобок содержимого не связано с введением дополнительных описаний фиктивных массивов ОЗУ, которые транслятор должен отличать от настоящих массивов;

в) скобки содержимого могут нести дополнительные функции, выполняя роль обычных (круглых) скобок для выделения определенных выражений.

Описание многоформатных составных и списковых величин

Иногда бывает необходимо обрабатывать такие списки или составные величины, у которых списковый член или составная величина может иметь несколько различных вариантов структуры, т. е. несколько различных форматов, причем такие многоформатные величины органически образуют один список или массив и разделение их на несколько списков или массивов в зависимости от их форматов оказывается нецелесообразным или невозможным. Форматы одной и той же величины обычно различаются значениями определенных признаков-разрядов, и поэтому при описании таких величин необходимо указывать эти признаки и их значения.

Для этого можно использовать несколько модифицированную методику описания составных и списковых величин. После индентификатора признака, различающего форматы, будем писать, как обычно, указание вида, а после него — символ **формат**, за которым указываем значение этого признака для данного формата. За этим значением через запятую будем описывать обычным образом структуру составной или списковой величины для данного формата. За описанием структуры для данного варианта формата будем указывать снова символ **формат,** после которого писать другое значение признака и за ним (через запятую) — описание нового формата и т. д.

Таким образом, описание нескольких различных форматов одной и той же величины дается за идентификатором признака, различающего эти форматы; описания разных форматов разделяются символом **формат,** за которым стоит значение соответствующего признака. Ясно, что после последнего описания формата должен обязательно стоять символ **уровень**, так как любая многоформатная величина обязательно является составной величиной. Если это к тому же и списковая величина, то, естественно, в начале ее описания должен стоять описатель **список** с заголовком списка, а в конце описания — структура спискового члена и соответствующий символ **уровень**.

Описание списковых процедур

При обработке списковых величин может применяться метод процедур. В виде процедур можно оформлять различные типовые списковые процессы: включение нового члена в список, исключение члена из списка, объединение списков, разделение списка на части, поиск заданных членов в списке, копирование списка и т. п.

Эти процедуры оформляются по правилам АЛГОЛа с учетом добавлений, сделанных в АЛГЭМе, а также с использованием введенных в данной главе адресных соотношений и описательных скобок **список** ... **уровень,** а также разделителя **формат.**

Эти средства используются не только в теле процедуры, оформляемом в виде блока, но и в заголовке процедуры, в разделе спецификаций.

Подобно тому, как в АЛГЭМе предусматривается обязательная спецификация формальных параметров, являющихся составными величинами, при программировании списковых процедур еще обязательна спецификация формальных параметров, являющихся списковыми величинами. Спецификации списковых величин представляют собой описания списков и описания адресных соотношений, составляемые в полном соответствии с приведенными выше правилами построения таких описаний в блоках. На списковые процедуры полностью распространяются все правила локализации величин в блоках и в процедурах, действующие в АЛГОЛе. В частности, если какой-нибудь формальный параметр процедуры обозначен таким же идентификатором, как и некоторая величина, являющаяся глобальной по отношению к данной процедуре, то эта величина не может быть использована в данной процедуре. При построении спецификаций списковой процедуры (адресных соотношений) могут использоваться величины, являющиеся глобальными для данной процедуры и не являющиеся в ней формальными параметрами.

В описаниях списковых процедур могут использоваться все другие описания, входящие в состав описаний данного блока программы (или какого-нибудь внешнего блока), если это не нарушает общих правил локализации величин в блоках.

В адресных соотношениях, находящихся в составе описаний в блоке, являющемся телом процедуры, могут участвовать как величины, описанные в самом блоке, так и величины, описанные в разделе спецификаций этой процедуры, а также величины, глобальные по отношению к данной процедуре.

Полная спецификация списковых величин в процедурах необходима в связи с тем, что каждая конкретная процедура составляется для определенной структуры списков и поэтому использовать эти процедуры можно только при обработке списков, имеющих такую же структуру списков и соответствующую организацию свободной списковой памяти.

Перечислим общие дополнительные описания, используемые при ассоциативном программировании:

— описание списковых процедур, в том числе специальной процедуры организации свободной списковой памяти;

— описание адресных соотношений;

— описание списковых величин.

§ 21 СРАВНЕНИЕ АЛГОРИТМИЧЕСКИХ ЯЗЫКОВ

Как видно из предыдущего изложения, рассмотренный нами алгоритмический язык для программирования информационно-логических задач построен на основе использования ряда существующих алгоритмических языков и представляет собой, по существу, объединение нескольких языков в единый язык. Указанное объединение произведено не простым суммированием всех средств различных языков в один язык, а путем дополнения одного из языков (а именно АЛГОЛа), принятого за основу, теми средствами, взятыми из других языков, которые необходимы для программирования соответствующих классов задач.

К числу языков, использованных в той или иной степени при построении данного языка, относятся: АЛГОЛ-60, КОБОЛ, адресный язык Института кибернетики АН УССР, ЛИСП, разработанный Маккарти (США), ИПЛ-5, разработанный Ньювеллом, Симоном и Шоу (США). В отношении АЛГОЛ-60 было уже сказано, что этот язык с несущественными отклонениями, касающимися деталей обозначений или наименований нескольких понятий, просто положен в основу данного языка. Из языка КОБОЛ, предназначенного для программирования экономических задач, взята в основном методика описания составных переменных и массивов, а также методика описания видов величин и указания разрядов.

Из адресного языка института кибернетики АН УССР использовано понятие адресного алгоритма, введенное В. С. Королюком, определение ранга адреса и ряд общих положений, касающихся 'построения адресных формул и операций. Из языков списковой обработки данных (ЛИСП, ИПЛ-5, АЛП) взяты основные приемы работы с цепными списками и списковыми структурами (применение цепного списка свободных ячеек для организации свободной области памяти машины, применение продвигаемых списков — СТЭКОВ—для хранения промежуточных данных, применение двух видов списковых членов — структурных и объектных, построение ответвляющихся подсписков и др.).

Методика ассоциативного программирования (АП), изложенная в последней главе, должна рассматриваться в неразрывной связи с АЛГОЛом и АЛГЭМом, так как она содержит только специальные средства, используемые при работе со списковыми величинами. Но кроме этих специальных средств при обработке списковых величин должны обязательно использоваться и основные средства АЛГОЛа и АЛГЭМа.

Одним из основных понятий ассоциативного программирования является понятие содержимого. Оно составляет основу адресного языка программирования, разработанного Е. Л. Ющенко, и определяется в нем с помощью штрих-операций. В отношении использования понятия содержимого язык ассоциативного программирования имеет много общего с адресным языком. Но в адресном языке, который разрабатывался одновременно с АЛГОЛом и независимо от него, имеется своя символика и методика для представления всех остальных действий (операторы циклов, переходов, переключатели, описания и т. д.), в то время как в языке ассоциативного программирования все эти средства точно соответствуют АЛГОЛу. Кроме того, в языке ассоциативного программирования четко определено понятие, обратное понятию содержимого, а именно понятие адреса, для которого введено специальное обозначение (адресные скобки $|\bar{\ }\bar{\ }|$). Адресные скобки вместе с запятыми, принятыми в АП для разделения членов последовательных списков, соответствуют точечному обозначению (точки и круглые скобки), используемому в языке ЛИСП для представления внутренней структуры цепных списков. В языке ЛИСП используются списковые члены с постоянным форматом. Считается, что каждый член списка помещается в одну ячейку памяти и состоит из двух частей: символа и адреса связи. Символ помещается в первой половине ячейки, а адрес связи — во второй половине ячейки. Для обозначения концов списков используется символ КС, который ставится на место адреса связи. В ЛИСПе используются только цепные списки, т.е. каждый предшествующий член списка должен содержать адрес связи, указывающий положение последующего члена списка.

Суть точечного представления внутренней структуры списков, используемого в ЛИСПе, состоит в следующем.

Если две какие-либо величины *A* и *B* помещены в двух половинах одной ячейки, то адрес этой ячейки обозначается так:

$$(A.\ B)$$

При этом точка указывает на нахождение двух величин в одной ячейке, а круглые скобки обозначают адрес этой ячейки. Если во второй половине данной ячейки находится адрес другой ячейки, в которой записан следующий член списка *B,* и эта ячейка является последней в списке, то в ЛИСПе это запишется в виде

$$(A.\ (B.\ КС))$$

Графически члены списков в ЛИСПе обозначаются в виде прямоугольников с двумя отделениями, в первом отделении пишется символ, представляющий собой данный член списка, а второе отделение оставляется пустым, т. е. подразумевается, что в нем находится адрес связи, указывающий следующий член списка. Вместо записи адреса рисуется стрелка, отходящая от того отделения, в котором должен быть этот адрес связи, и подходящая к тому прямоугольнику, который должен быть указан адресом связи. Вместо символа КС в соответствующих отделениях ячеек чертится наклонная линия.

Первый пример. Цепной список, состоящий из трех членов *A*, *B*, *C*, в ЛИСПе запишется так:

$$(A.\ (B.\ (C.\ КС))).$$

Этот список изображен на рис. *25,а.*

С помощью введенной нами символики (АП) этот список запишется таким образом:

$$|\bar{\ }A,\ |\bar{\ }B\ |\bar{\ }C,\ КС\ \bar{\ }|\bar{\ }|\bar{\ }|$$

Второй пример. Список, имеющий графическое представление, показанное на рис. 25,*б*, в ЛИСПе будет записан так:

$$(A.\ ((B.\ (C.\ КС)).\ (M.\ ((K.\ (F.\ КС)).\ КС))))$$

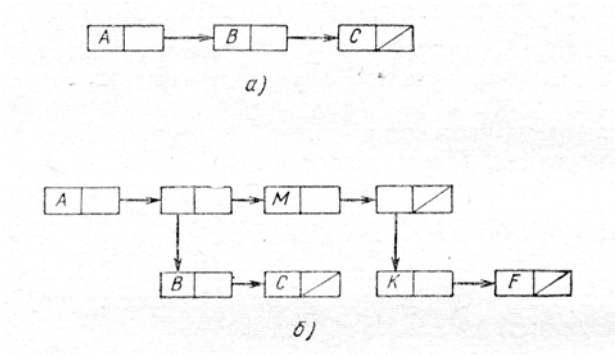
В АП этот список будет иметь аналогичную запись:

$$|\overline{\ }A,|\overline{\ }|\overline{\ }B,|\overline{\ }C,KC|\overline{\ }|,|\overline{\ }M,|\overline{\ }|\overline{\ }K,|\overline{\ }F,KC|\overline{\ }|,KC|\overline{\ }|\overline{\ }|\overline{\ }|$$

Таким образом, видно, что принятая нами методика использования адресных скобок и запятых полностью соответствует правилам построения точечного представления списков в ЛИСПе с помощью круглых скобок и точек.

Чем же вызвана необходимость замены точечной символики, принятой в ЛИСПе, на символику, принятую в языке ассоциативного программирования (АП)?

Во-первых, символика, принятая в АП, имеет несколько более широкие возможности. Запятая в АП служит не только для разделения двух величин, расположенных в двух половинах одной ячейки, как это делает точка в ЛИСПе, а вообще для разделения членов последовательных списков, т. е. величин, расположенных подряд в одной или нескольких ячейках памяти. Следовательно, принятым способом можно обозначать не только цепные списки, но и последовательные списки, а также комбинированные списки, включающие в себя как цепные, так и последовательные подсписки



.

РИС. 25. Изображение списка и списковой структуры:
а) описок (*A. (B.* (C. КС)); б) списковая структура (*A.* ((*B.* (*C. KC*)). (*M.* ((*K* (*F. KO*). *KC*)))).

Во-вторых, использование круглых скобок для обозначения адресов величин наряду с использованием их в АЛГОЛе для построения арифметических выражений может привести к неопределенности или потребует введения ограничения на обозначение величин, заключаемых в адресные скобки. Поэтому целесообразно для адресных скобок иметь отдельные символы.

В-третьих, в АЛГОЛе для разделения элементов различных последовательных списков принята запятая, а не точка (например, списки индексов у переменных с индексами, списки граничных пар, списки именующих выражений в переключательных списках, списки формальных и фактических параметров). Поэтому естественно и здесь для разделения элементов последовательных списков принять запятую. Кроме того, адресные скобки, имеющие обратный смысл по отношению к скобкам содержимого, логично обозначать символами, имеющими обратное начертание. Основным отличием описанного языка ассоциативного программирования от языков списковой обработки данных ЛИСП, ИПЛ-5, ФЛИПЛ, АЛП и других является непосредственная связь языка ассоциативного программирования с АЛГОЛом (и его расширением АЛГЭМом) и большая универсальность.

Все перечисленные выше списковые языки используют строго фиксированный формат списковых членов; более того, некоторые из них, например ФЛИПЛ, используют списковые члены, строго привязанные к разрядной сетке машины.

Язык АП обеспечивает возможность описывать алгоритмы обработки списков, имеющих различные структуры списковых членов. Это достигается возможностями:

а) описывать структуру списковых членов, используя описательные скобки **список … уровень;**
б) задавать с помощью адресных соотношений характер адресных связей между членами списков.

В качестве примера можно описать с помощью упомянутой методики структуру списковых членов, используемых в языке ИПЛ-5. В этом языке каждая ячейка памяти содержит одно списковое слово, состоящее из четырех частей: двух префиксов (P и Q), символа (symb) и адреса связи (link):

P	Q	symb	link

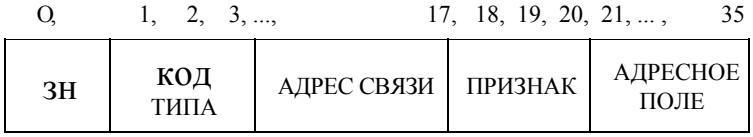
Пусть P занимает три двоичных разряда, Q — два разряда, symb — 20 разрядов и link — 20 разрядов. Тогда описание списка будет иметь вид (заголовок списка опущен):

список; составной ЧЛЕН СПИСКА;
целый P **вид** 1 (3), Q **вид** 1 (2);
вещественный symb **вид** 1 (20);
целый link **вид** 1 (20) **уровень уровень**

Адресное соотношение, определяющее связь между членами данного цепного списка, запишется следующим образом:

|¯ЧЛЕН СПИСКА¯| = link;

В языке ФЛПЛ используются списковые члены следующего формата:



Вверху указаны номера двоичных разрядов ячейки памяти, в которой размещается списковое слово

Описание списка будет иметь вид:

список; составной ЧЛЕН СПИСКА;
логический ЗН**; целый** КОД ТИПА **вид** 1(2),
АДРЕС СВЯЗИ **вид** 1(15), ПРИЗНАК **вид** 1(3),
АДРЕСНОЕ ПОЛЕ **вид** 1(15) **уровень уровень**

§ 22 ПРИМЕРЫ АССОЦИАТИВНОГО ПРОГРАММИРОВАНИЯ

Программирование для цепных списков и структур

Будем считать, что для построения цепных списков и списковых структур используются списковые слова, размещаемые по одному в каждой ячейке памяти. Каждое списковое слово состоит из трех частей: признака типа слова *S*, наименования И и адреса связи АС. Весь список (списковая структура) будет иметь идентификатор ЦЕПНОЙ СПИСОК, который в наших примерах не используется.

Конец списка (подсписка) обозначается АС, равным константе КС. При *S* = 0 *И* играет роль наименования объекта, являющегося членом данного списка, т. е. в этом случае списковое слово будет объектным.

При *S* = 1 часть *И* играет роль адреса, отсылающего к ответвляющемуся подсписку, т. е. в этом случае списковое слово является структурным.

Таким образом, списковое слово, обозначаемое сокращенно ЧС (член списка), имеет два формата, различающихся значением признака *S*, но оба эти формата совпадают. Распределение разрядов в описаниях видов элементарных величин, входящих в состав спискового слова, дано с таким расчетом, чтобы списковое слово полностью помещалось в ячейку машины «Минск-2», имеющую 37 двоичных разрядов.

Ниже приводятся описания типовых процедур, используемых при ассоциативном программировании. В качестве примера более сложной программы приводится программа копирования списковой структуры.

Аналогичная программа, но построенная иначе, была представлена на языке АЛП [13].

Некоторые списковые операторы, введенные в языке АЛП и описанные там словесно, здесь представлены на языке ассоциативного программирования в виде процедур.

Описания списковых процедур мы приводим не изолированно, а в виде перечня описаний процедур, входящих в состав описаний некоторого блока программы. В начале этого блока стоит символ **начало**, для которого отсутствует соответствующий ему символ **конец,** так как этот блок не является полным. Он содержит в себе описания нескольких величин (ЦЕПНОЙ СПИСОК, ЧС, КС, СЯ, *i*), глобальных по отношению к приводимым ниже описаниям процедур, а также описания процедур. На этом указанный блок программы как бы обрывается.

Идентификатор СЯ обозначает заголовок списка свободных ячеек, а идентификатор *i* — некоторую промежуточную (рабочую) переменную целого типа. Последние три примера программ следует рассматривать как блоки, входящие в состав данного блока:

начало

список ЦЕПНОЙ СПИСОК; **составной** ЧС**; логический** *S;* **целый** *И* **вид** 1 (18), АС **вид** 1 (18) **уровень уровень;**

целый КС **вид** 1(18), СЯ **вид** 1(18), *i* **вид 1(18);**

|¯ЧС¯| = АС = *i* = СЯ**;**

процедура ПОДГОТОВКА (АН, ЧЯ);

примечание производится соединение в цепной список свободных ячеек группы ячеек памяти, начиная с адреса АН, количеством ЧЯ. Адрес вершины этого списка присваивается величине СЯ. Величины СЯ, *i*, АС, КС являются глобальными по отношению к этой и последующим процедурам;

целый АН, ЧЯ; |¯ЧС¯| = АН + ЧЯ — 1**;**

начало

СЯ: = АН;
для *i*: = АН **шаг** 1 **до** АН + ЧЯ —2 **цикл**
АС. |_ *i* _|: = *i* + 1;
АС. |_АН + ЧЯ —1_|:=КС;

конец

процедура ЯЧЕЙКА (*К*);

примечание обеспечивается получение свободной ячейки из списка свободных ячеек с присваиванием адреса этой ячейки величине *К* и соответствующая корректировка списка свободных ячеек;

целый *К;*

если СЯ = КС **то** ОСТАНОВ **иначе**

начало *К* : = СЯ; СЯ : =АС.|_СЯ _| **конец**

процедура ВСТЭК (*К, В, Н*);

примечание в вершину СТЭКА, которым может быть любой список, заданную адресом *К,* записываются значения логического признака *В* и наименования *Н* на соответствующие им места в списковом слове. Прежние значения признака *S* и наименования *И* в этом слове переносятся на соответствующие места в свободную ячейку, взятую из списка свободных ячеек, которая включается в тот же список после вершины. При этом происходит как бы сдвиг всех прежних членов списка. В обращениях к этой процедуре обычно первый параметр является константой, которая фиксирует положение вершины СТЭКА, т.е. указывает ячейку, в которой находится первый член СТЭКА. Поэтому последующие члены СТЭКА вставляются за этой ячейкой;

целый *К;* |¯*ЧС*¯| = *К;*

начало

ячейка (*i*);

И. |_ *I* _|: = *И.* |_*K*_|; S. |_ *i* _|: = S. |_*K*_|;

И. |_*K*_| = *Н*; *S.* |_*K*_|: = *В*;

АС. |_ *i* _|: = АС. |_K_|;

АС. |_*K*_|: = *i*;

конец

процедура ИЗСТЭКА (*К*, *В, Н*);

примечание из СТЭКА, заданного адресом вершины *К,* исключается верхний член, значение признака *S,* наименования *И* которого присваиваются логической величине *В* и величине *Н*. Второй член СТЭКА пересылается на место первого члена, т. е. в ячейку *К,* остальные члены СТЭКА при этом как бы поднимаются на одну ступень вверх. Ячейка, в которой находился второй член СТЭКА, возвращается в список свободных ячеек;

целый *К;* **логический** *В;* **вещественный** *Н*; |¯ЧС¯| = *К*;

начало *В*: = *S.* |_*K*_|;

Н:=*И.* |_*К*_|

i: = АС. |_*K*_|;

|_*K*_|:= |_АС |_*К*_| _| ;

| _ *i* _ | := СЯ;

СЯ:= *i*;

конец

процедура ПОИСК (*К, Н*, *В, Д, М*);

примечание производится поиск в списке с адресом вершины *К* объекта, имеющего наименование *И* и логический признак *S* равными соответственно заданным параметрам *Н* и *В*. Адрес первого найденного объекта присваивается величине *Д,* и управление передается следующему оператору программы. Если не будет найден ни один объект, то величине *Д* присваивается адрес последнего члена списка и управление передается оператору с меткой *М;*

логический *В;* **вещественный** *Н;* **целый** *К*, *Д*;

метка *М*; |¯*ЧС*¯| = *К*;

N: **если** *И.* |_*K*_| = *Н**S.* |_*K*_| = *В* **то** *Д*: = *К* **иначе**

начало если АС. |_*K*_| = КС **то**

начало *Д*: = *К*; **перейти к** *М* **конец иначе**

начало *К* = АС**.** |_*K*_| **перейти к** *N* **конец конец**

процедура ВСТАВКА *(К, В,* Я);

примечание заданное значение логического признака *В* и наименования *Н* заносятся на соответствующие места в ячейку, и эта ячейка вставляется в список после ячейки с адресом *К;*

целый *К;* **вещественный** *Н*; **логический** *В;* |¯**ЧС**¯| — *К,*

начало

ЯЧЕЙКА (*i*);

И. |_*i*_|: = *Н; S.* |_*i*_|: = *В;*

АС. |_*i*_|:==АС. |_*K*_|;

АС. |_*K*_|:=*i;*

конец

процедура СВЯЗЬ (*К, Ч*);

примечание значение адреса связи в ячейке *К* устанавливается равным величине *Ч,* являющейся адресом некоторой ячейки. Если первая ячейка была последней в одном списке, а вторая — первой в другом списке, то при помощи этой процедуры оба списка будут объединены в один:

целый *К, Ч;* |¯ЧС¯| = *К*;

АС |_*К*_|:=*Ч;*

процедура СТИРАНИЕ (Л');

примечание все ячейки списка, начинающегося с ячейки с адресом *К,* вставляются в список свободных ячеек;

целый *К;* |¯ЧС¯| = *К*;

начало

ЯЧЕЙКА *(i);*

СЯ: = К;

***N*-.если** АС. |_*К*_|= КС **то перейти к** *М* **иначе**

К: = АС. |_*К*_|

перейти к *N;*

М:АС.|_*К*_|**:=** *i*

конец

Ниже приводятся три примера программ, использующих указанные процедуры.

Пример 1

Исключить из списка, адрес вершины которого равен *А,* все члены, у которых логический признак S и наименование *И,* равны соответственно логической величине *В* и вещественной величине *С.* Присвоить величине *J* число исключенных членов.

символ * (звездочка) обозначает любую неиспользуемую ячейку.

начало

целый *J, А*; **логический** *В;* **вещественный** *С***;**

|¯ЧС¯| = *А*;

J: = 0

М: ПОИСК *(А, С, В, А, N*);

ИЗСТЭКА *(А,* *, *);

J: =*J*+1;

перейти к М**;**

N: **конец**

Пример 2.

Произвести обмен наименованиями *И* между первым и последним членами списка, начинающегося с ячейки с адресом *А.*

начало вещественный *I***; целый** *А, J;* |¯ЧС¯| = *А* = *J***;**

*I***:=***И***.** |_*А*_|**;** *J***:** = *А***;**

*М:***если** АС**.** |_*А*_| ≠ КС **то начало**

А: = АС**.** |_*А*_|**, перейти к** *М* **конец;**

И.|_*J*_|:=*И.*|_*А*_|; *И.*|_А_|: = *I*;

конец

Пример 3.

Рассмотрим пример записи на языке ассоциативного программирования более сложной программы, осуществляющей копирование некоторой ассоциативной списковой структуры. Эта программа в качестве основной части имеет программу копирования цепного списка, но так как списковые структуры представляют собой в общем случае совокупности списков и подсписков различных уровней, то полная программа копирования структуры должна обеспечивать возможность многократного вызова собственно программы копирования цепного списка, т. е. она должна иметь рекурсивный характер. Такой вызов должен осуществляться в каждой точке разветвления прослеживаемого списка (подсписка). При этом необходимо каждый раз запоминать место возврата в данный список для того, чтобы после окончания копирования подсписка продолжить копирование ранее прослеживаемого списка. Такое запоминание точек возврата осуществляется с помощью СТЭКА *I*, в который засылаются адреса членов копируемой структуры (*i*), от которых происходит ответвление подсписков.

При ответвлении от данного списка некоторого подсписка в СТЭКЕ / запоминается адрес члена списка, от которого произошло ответвление; при ответвлении другого подсписка от данного подсписка в этот же СТЭК засылается адрес члена данного подсписка, от которого произошло ответвление, и т. д.

В программе используется второй СТЭК *J* для запоминания текущих адресов (*j*) членов структуры — копии, от которых ответвляются подсписки копии. Оба СТЭКА *I* и *J* работают синхронно и их, вообще говоря, можно было бы заменить одним СТЭКОМ. Использование двух раздельных СТЭКОВ *I* и *J* делает процесс более наглядным.

Кроме этих двух СТЭКОВ в данной рекурсивной программе используется еще и третий СТЭК *М.* Этот СТЭК служит для запоминания меток возврата при прерываниях хода программы и повторных обращениях к началу программы собственно копирования ассоциативной структуры. Такие прерывания осуществляются каждый раз

при встрече в основной (копируемой) структуре структурного члена. После окончания копирования очередного подсписка происходит восстановление СТЭКА *M*. В эти же моменты происходит и восстановление СТЭКОВ *I* и *J*.

Естественно, что при работе данной программы используется список свободных ячеек, из которого при помощи процедуры ЯЧЕЙКА (*j*) берутся свободные ячейки для построения ассоциативной структуры-копии.

Для начала работы программы копирования задается величина *A* — адрес вершины (первого члена, а не заголовка) списковой структуры-оригинала. *B* — это переменная, которой должен быть присвоен адрес вершины списковой структуры-копии; *i* — адрес текущего копируемого члена структуры-оригинала; *j* — адрес текущего члена в структуре-копии; *k* — промежуточная (рабочая) переменная.

Заметим, что, так как в СТЭКЕ *M* должны запасаться метки возврата, а не значения (числовые) некоторой переменной величины, то в качестве третьего фактического параметра в соответствующих обращениях к процедуре ВСТЭК указаны метки в виде строчных величин, т. е. метки, заключенные в кавычки. При обращении к СТЭКУ *M* при помощи оператора процедуры ИЗСТЭКА третий параметр этой процедуры *L* принимает значения соответствующих меток, т. е. величина *L* должна быть строчной величиной. В качестве второго параметра в операторах процедур ВСТЭК и ИЗСТЭКА фигурирует постоянно единица, так как этот параметр не используется, но указывать его нужно для общего счета числа параметров.

```
    начало
    целый i, j, K, I, J, M; строчный L вид A9;
   |¯ЧС¯| = i = j = K;
    начало
    i: = A;
    ЯЧЕЙКА (B);
   j: = B;
   ВСТЭК (M, 1, 'M3');
Ml :если ¯| S. |_i_|  то начало S.|_j_|:=0; И.|_j_|: = И.|_i_|;
   если AC.|_i_|: = КС то начало AC. |_j_|: = КС; перейти к M2 конец закончилось копирование последнего
члена подсписка; K: = j; ЯЧЕЙКА (j); AC.|_K_|:=j; i:=AC. |_i_|; перейти к Ml конец закончилось копирование
очередного объектного члена списка;
    S.|_j_|:=1; K:=j; ВСТЭК (J, 1, j);
    ЯЧЕЙКА (j); И.|_K_|:=j
    ВСТЭК (I, 1, i); i= И.|_i_|;  ВСТЭК (M, 1, 'M4');
перейти к Ml;
    M4:ИЗСТЭКА (I, 1, i); i:=AC. |_i_|
    ИЗСТЭКА (J, 1, K); ЯЧЕЙКА (j); AC.|_KA_|=j
   перейти к Ml;
M2: ИЗСТЭКА (M, 1, L);
   перейти к L;
M3: ОСТАНОВ;
    конец
    конец
```

Поиск документов в ассоциативно- адресной дескрипторной поисковой системе

Общий принцип действия дескрипторных поисковых систем был рассмотрен во введении. Сейчас рассмотрим один из способов машинной реализации такой системы и алгоритм поиска документов в ней, который запишем на алгоритмическом языке

Для простоты будем считать, что вся информация помещается в оперативной памяти машины и поэтому вопросы обмена данными между ОЗУ и МЛ или МБ рассматривать не будем.

Для построения дескрипторной поисковой системы необходимо иметь словарь дескрипторов, т. е. фиксированный набор терминов-определений и массив поисковых образов документов. Поисковый образ документа это группа дескрипторов, характеризующих содержание документа; у разных документов эти поисковые образы могут иметь различное число дескрипторов (в среднем по 5—10 дескрипторов). Будем использовать прямой способ организации массива поисковых образов документов. Все документы, учитываемые в системе, так же как и все дескрипторы, имеют свои постоянные коды, чаще всего такими кодами являются порядковые номера.

При прямом способе организации массива поисковых образов документов за каждым кодом документа перечисляются все коды дескрипторов, которыми обладает этот документ.

Каждый документ в памяти машины будем представлять в виде одного ассоциативного узла (см. гл. 5), в котором число списковых слов соответствует числу дескрипторов, имеющихся у документа. Таким образом, каждому дескриптору будет соответствовать одно определенное списковое слово в узле. Ясно, что одни и те же дескрипторы будут фигурировать в поисковых образах многих различных документов.

Представление поисковых образов документов в виде ассоциативных узлов позволяет с помощью списковых слов связывать в единые цепные списки все документы, имеющие одни и те же дескрипторы. Очевидно, что один и тот же документ будет входить во столько различных цепных списков, сколько дескрипторов он имеет в своем поисковом образе.

В памяти машины выделяется специальная зона (группа последовательных ячеек) под так называемый массив заголовков списков. Каждому дескриптору соответствует один цепной список и один заголовок списка; каждый заголовок списка размещается в одной ячейке. Адрес этой ячейки можно считать кодом данного дескриптора;

зная код дескриптора, можно обратиться к нужному заголовку списка. Вопрос о том, как от словесного представления дескриптора перейти к его коду, мы рассматривать не будем. Это может быть сделано, например, методом сверток, аналогичным методу, описанному в гл. 5, § 14.

Заголовок списка включает в себя пять элементов, т. е. он является составной величиной. Первым элементом, входящим в состав заголовка, будет адрес связи начала цепного списка (АС) документов, имеющих в своем поисковом образе данный дескриптор. Этот адрес указывает первый ассоциативный узел, а точнее, первую ячейку узла, представляющего первый документ в этом списке документов. Так как каждый документ характеризуется несколькими дескрипторами, каждому из которых соответствует свое списковое слово в его ассоциативном узле, то для следования по цепному списку необходимо указывать не только адрес очередного узла, но и номер того спискового слова в узле, которое соответствует данному списку (т. е. данному дескриптору) Для этой цели служит второй элемент в заголовке списка, называемый номером слова начального (НС). Оба элемента АС и НС образуют слово ССН — списковое слово начальное.

Третий элемент заголовка представляет собой адрес связи, указывающий конечный (последний) ассоциативный узел в данном цепном списке, соответствующий последнему документу в данном списке (АС — адрес связи конца).

Четвертый элемент указывает номер спискового слова в ассоциативном узле, соответствующем последнему документу (НС — номер слова конечного). Эти АС и НС образуют ССК — списковое слово конечное и используются при включении в систему новых документов. Каждый новый документ, вводимый в систему, должен включаться в те цепные списки, которые соответствуют имеющимся в его поисковом образе дескрипторам. Новые документы включаются в концы списков, а просмотр списков при поиске производится с начала.

При таком порядке включения новых документов документы в цепных списках будут располагаться в хронологической последовательности, соответствующей порядку их поступления в поисковую систему. Можно было бы не иметь в заголовках списков элементов для концов списков, но тогда для нахождения последних ассоциативных узлов в цепных списках (т. е. концов этих списков) пришлось бы каждый раз при включении нового документа просматривать полностью соответствующие цепные списки, что привело бы к излишней затрате машинного времени. Кроме того, хранение в заголовках списков адресов двух граничных точек списков (начала и конца) полезно с точки зрения контроля работы системы.

Возможен также такой вариант, когда используется адрес только одного (последнего) узла и запись узлов идет в обратном порядке, т. е. от конца МЛ к ее началу.

Пятый элемент заголовка списка представляет собой число членов в данном списке, т. е. число документов (ЧД), введенных в систему и обладающих данным дескриптором. Этот элемент необходим для периодической проверки размеров списков, соответствующих разным дескрипторам. Знание размеров списков необходимо для корректировки словаря дескрипторов. Кроме того, знание размеров списков может быть использовано для построения более рациональных запросов и организации оптимального процесса поиска документов. Ниже показан формат заголовка списка

ССН		ССК		
АС	НС	АС	НС	ЧД

Рассмотрим теперь строение ассоциативного узла. В первой ячейке узла размещается составная величина, которую мы назовем заголовком узла. В состав заголовка узла входят три элемента. Первым элементом будет указатель документа (УД), представляющий собой некоторый код документа, например его порядковый номер, с помощью которого этот документ может быть найден в хранилище.

Следующим элементом заголовка узла является так называемая дополнительная информация о документе (ДИ), которая содержит дополнительные сведения о документе и сама может являться составной величиной. Конкретное строение ДИ зависит от конкретных условий применения системы, и мы его рассматривать не будем. Сюда могут входить, например, признаки вида документа (книга, журнал, отчет и т. д.), год и место издания, язык, на котором написан документ, и т. д.

Третьим элементом заголовка узла является число списковых слов в узле (ЧС). Оно необходимо для указания общего размера узла (при вводе документов в систему и построении узлов, а также при переносе узлов с места на место). Этим элементом заканчивается заголовок узла.

Во второй и последующих ячейках узла располагаются списковые слова, по одному слову в ячейке памяти. Каждое списковое слово состоит из двух частей: адреса связи (АС), указывающего первую ячейку другого узла, представляющего следующий документ в данном цепном списке, и номера слова (НС), указывающего, какое списковое слово в следующем узле соответствует данному дескриптору. Окончания цепных списков обозначаются специальным кодом КС, который ставится на месте адресов связи.

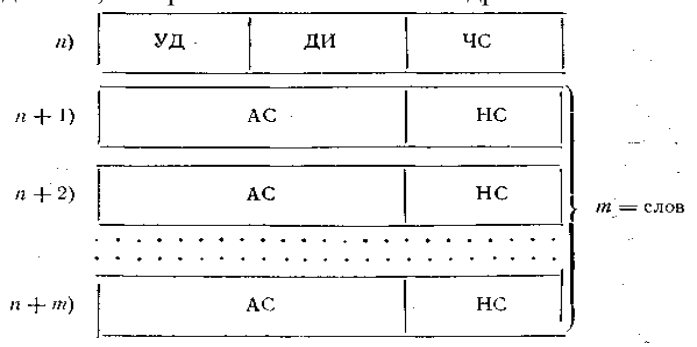


РИС. 26. Ассоциативный узел.

Заметим, что код КС в соответствии с алгоритмом поиска, который описывается ниже, должен быть больше по своей величине любого из адресов связи АС. Таким кодом может быть код, состоящий из единиц во всех разрядах адреса связи.

На рис. 26 приведен формат ассоциативного узла.

Для пояснения общего принципа работы системы на рис. 27 приведен фрагмент ассоциативно-адресной структуры, включающий в себя словарь из пяти дескрипторов (пять заголовков списков, размещенных в ячейках с адресами от 107 до 111) и пять ассоциативных узлов, соответствующих пяти документам. В рассматриваемой ассоциативно-адресной системе все цепные списки имеют направленный характер, т. е. адреса связи в них при движении вдоль по спискам монотонно возрастают. Это связано с тем, что расположение ассоциативных узлов на магнитной ленте соответствует порядку ввода документов в систему.

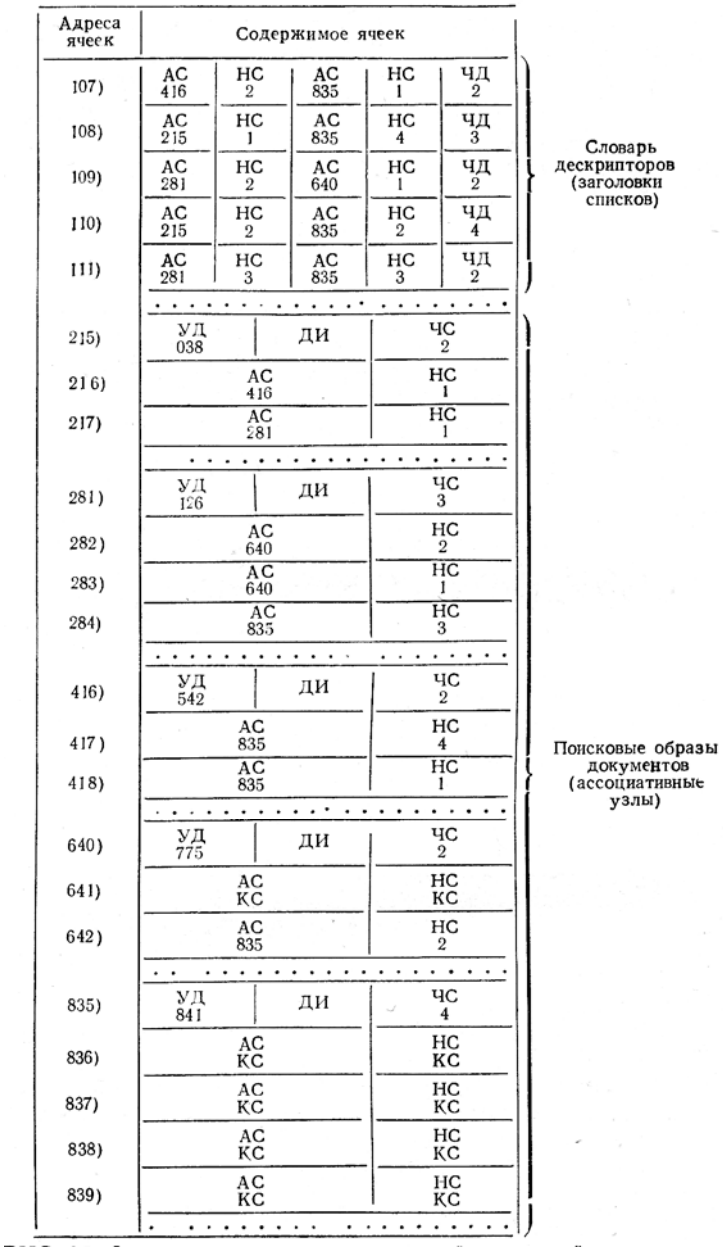


РИС. 27. Фрагмент ассоциативно-адресной поисковой системы.

В общем случае в цепных или узловых списках члены списков будут располагаться в памяти машины произвольным образом, и поэтому адреса связи у них могут меняться при переходе от одного члена списка к следующему как в сторону возрастания, так и в сторону убывания. В приведенном фрагменте ассоциативно-адресной системы выдерживается свойство монотонного возрастания адресов связи при перемещении вдоль списков.

Монотонное изменение адресов связи является существенным достоинством ассоциативно-адресного способа построения дескрипторной поисковой системы, так как позволяет производить просмотр цепных списков, расположенных на магнитной ленте за один прогон ленты, без реверсов и повторных протонов.

При реализации подобной ассоциативно-адресной поисковой системы на магнитных лентах возникают некоторые особенности в программировании и распределении памяти, которые мы сейчас кратко рассмотрим.

Оперативная память машины под все несписковые величины распределяется обычным образом (т. е. выделяются зоны констант, рабочих ячеек, зона для размещения программы и т. д.). Из остающегося свободного объема ОЗУ выделяется зона под обрабатываемый списковый блок (например, размером в 2048 ячеек). В

соответствии с размером этой рабочей списковой зоны, выделенной в ОЗУ, производится запись спискового массива на магнитной ленте в виде блоков данных по зонам, размер которых равен размеру рабочей списковой зоны в ОЗУ. В процессе работы системы списковый массив будет переписываться для обработки по блокам в рабочую списковую зону ОЗУ. Так как размеры ассоциативных узлов для разных документов могут быть различными, а размеры всех зон на МЛ устанавливаются одинаковыми, то ясно, что в разных зонах может быть различное число ассоциативных узлов, а в конце зон могут оставаться свободные ячейки. Для контроля заполнения МЛ списковым массивом в ОЗУ должны быть выделены постоянно две ячейки: в одной, называемой фиксатором зоны, должен храниться адрес очередной заполняемой зоны МЛ, а во второй — фиксаторе ячейки — адрес очередной свободной ячейки в этой зоне, (практически это может быть один код). Эти фиксаторы используются только при включении новых документов в систему. При необходимости включить новый документ в систему сначала по значению фиксатора зоны определяется нужная зона, а затем по значению фиксатора ячейки определяется адрес ячейки внутри зоны. Производится проверка, достаточно ли свободных ячеек в этой зоне для записи НОЕОГО ассоциативного узла, и если достаточно, то эта зона переписывается с МЛ в ОЗУ и в нее вносится новый ассоциативный узел с включением его во все необходимые списки. После этого величина фиксатора ячеек уменьшается на соответствующее количество ячеек.

Если количество свободных ячеек в данной зоне окажется меньше требуемого для записи нового узла, то запись нового узла производится в следующую зону; при этом значение адреса текущей зоны в фиксаторе зоны увеличивается на единицу, а значение фиксатора ячеек сбрасывается в нуль. При поиске нужных документов весь списковый массив поочередно по блокам переписывается с МЛ в рабочую списковую зону ОЗУ. Ясно, что при записи в ОЗУ ассоциативные узлы каждого блока попадают на определенные места (в ячейки) внутри рабочей списковой зоны ОЗУ и поэтому адреса связи, используемые в узлах, должны соответствовать адресам, которые будут иметь требуемые узлы в рабочей списковой зоне.

Ниже приводится запись на алгоритмическом языке алгоритма поиска документов в ассоциативно-адресной дескрипторной поисковой системе. Для простоты предполагается, что вся информация помещается в оперативной памяти машины и поэтому описание распределения памяти не приводится. Алгоритм оформлен в виде процедуры. В качестве формальных параметров фигурируют: массив заголовков списков (СЛОВАРЬ), массив дескрипторов запроса (ЗАПРОС) и массив отобранных документов (ОТВЕТ). Все эти формальные параметры конкретизируются наименованием, т. е. вместо их идентификаторов при обращении к процедуре будут подставляться идентификаторы соответствующих фактических массивов. Основной списковый массив документов, в котором производится поиск, не фигурирует в качестве формального параметра, так как заданием массива заголовков списков (словаря дескрипторов) однозначно определяется и этот массив документов. Ясно, что в спецификациях составного массива СЛОВАРЬ и целых массивов ЗАПРОС и ОТВЕТ границы изменения индексов не указываются. Они будут заданы при конкретизации этих массивов фактическими параметрами.

К формальным параметрам относится, кроме того, простая переменная, представляющая собой число дескрипторов в запросе. Эта переменная, определяющая размер массива ЗАПРОС, должна быть введена в качестве отдельного параметра, так как она используется в теле процедуры независимо от массива ЗАПРОС. Для составного массива СЛОВАРЬ, являющегося формальным параметром, в соответствии с правилами АЛГЭМа дается подробная спецификация, определяющая структуру этой составной величины. В теле процедуры, оформленном в виде блока, описан ряд величин, являющихся локальными по отношению к данному блоку. Сюда относятся целые 'простые переменные *i, j*, играющие роль параметров циклов, величина МАКС АС, используемая в качестве промежуточной (рабочей) величины (для запоминания максимального значения адреса связи). Здесь же описана структура основного спискового массива документов, в котором производится поиск. Последнее описание необходимо для построения операторов тела процедуры. После перечисленных описаний величин идут описания адресных соотношений, используемые при построении операторов тела процедуры. В данной процедуре отсутствует описание распределения памяти, так как предполагается, что вся информация будет располагаться в оперативной памяти машины.

После описания адресных соотношений следуют операторы, образующие тело процедуры.

Сначала идет оператор цикла, осуществляющий выборку необходимых заголовков списков из словаря дескрипторов. Используя элементы массива ЗАПРОС в качестве адресов элементов массива СЛОВАРЬ, выбираем *n* значений величины ССН; эти значения присваиваются элементам рабочего массива ССР (списковое слово рабочее). Затем идет оператор цикла, осуществляющий проверку на совпадение адресов связи (АС) в выбранных словах ССН.

Совпадение адресов связи будет свидетельствовать о том, что все адреса связи указывают на один и тот же ассоциативный узел, т. е. на один и тот же документ, который должен быть выдан в составе ответа. Здесь сначала проверяется наличие среди адресов связи символа КС (конца списка). Если среди проверяемых адресов связи имеется этот символ, то дальнейшие поиски бесполезны, так как среди дескрипторов запроса имеется дескриптор, у которого в цепном списке вообще нет документов. Если окажется, что среди АС нет символа КС и, кроме того, не все АС совпадают между собой, то необходимо осуществить переход вдоль цепных списков, соответствующих заданным дескрипторам запроса. Для этого сначала выбирается из всех адресов связи (АС), имеющихся в составе элементов массива ССР, максимальный адрес связи (МАКС АС); затем производится движение вдоль по всем цепным спискам, соответствующим заданным дескрипторам, до тех пор, пока текущие адреса связи в них не станут равными или большими этого МАКС АС. Очевидно, что в силу монотонности изменения адресов связи при движении вдоль списков, совпадения адресов связи не произойдет ни для одного значения адреса связи, меньшего, чем МАКС АС. При наличии на каком-то этапе прослеживания списков, совпадения всех текущих адресов связи процесс движения по спискам приостанавливается и производится присваивание заголовка данного узла очередному элементу массива ОТВЕТ (фиксируется найденный документ).

После этого делается один шаг по всем прослеживаемым цепным спискам, затем осуществляется переход к оператору с меткой ПРОВЕРКА" НА СОВПАДЕНИЕ.

Шаг по списку, т. е. переход от данного члена цепного списка к следующему, указываемому адресом связи, содержащимся в данном члене списка, производится путем замены соответствующего элемента массива ССР

списковым словом с адресом АС + НС, где АС — адрес ассоциативного узла, а НС — номер спискового слова в узле.

Описанный процесс движения по цепным спискам и проверки адресов связи на совпадение продолжается до тех пор, пока не окончится какой-нибудь из прослеживаемых цепных списков.

После этого производится выдача на печать массива ОТВЕТ, содержащего заголовки узлов отобранных документов. Дополнительные детали алгоритма поиска указываются в виде примечаний в самом алгоритме, "представленном на алгоритмическом языке.

Естественно, что в этом алгоритме, описывающем только процесс поиска документов по заданному набору дескрипторов (запросу), отсутствуют какие-либо сведения, касающиеся образования как основного спискового массива документов, в котором производится поиск, так и массивов словаря дескрипторов и дескрипторов запроса, используемых при поиске.

Основной списковый массив документов образуется автоматически путем последовательного включения в этот массив новых документов. Это осуществляется при помощи специальной процедуры включения документа. Для каждого включаемого документа задается заголовок узла (УД, ДИ, ЧС) и набор его дескрипторов. Сначала производится запись нового ассоциативного узла на очередное свободное место в памяти машины (при этом используются упомянутые выше фиксатор зоны и фиксатор ячейки); в этом новом узле на местах адресов связи (АС) во всех списковых словах ставится сразу символ КС (конец списка), так как этот узел, естественно, будет последним во всех списках, в которые он будет включен. Заметим, что для правильной работы рассмотренного нами алгоритма поиска необходимо, чтобы этот символ КС был больше по своему значению любого возможного значения адреса связи (АС). После записи нового ассоциативного узла производится его включение во все списки, соответствующие имеющимся у него дескрипторам. Включение производится поочередно для каждого заданного дескриптора.

По коду дескриптора находится в словаре дескрипторов соответствующий заголовок списка; в этом заголовке выбирается вторая часть, а именно ССК, и по АС. ССК находится последний узел в данном списке, а по НС.ССК — номер спискового слова в этом узле. Ясно, что в качестве АС в этом списковом слове должен стоять КС.

Затем на место АС в это слово записывается адрес нового ассоциативного узла, а на место НС в это слово записывается порядковый номер дескриптора в наборе дескрипторов нового документа.

Эти же значения АС и НС записываются в ССК данного заголовка списка, так как теперь уже последним членом в списке будет ассоциативный узел нового документа. На этом включение нового узла в один список заканчивается. Таким же образом производится включение нового узла во все остальные списки, соответствующие остальным дескрипторам нового документа. Словарь дескрипторов вводится в систему отдельной процедурой один раз в начале работы поисковой системы. При этом в начальный момент, когда в систему не включен еще ни один документ, на местах всех АС в ССН и ССК заголовков списков должны стоять символы КС. Это необходимо для правильного выполнения описанного выше алгоритма поиска. Затем по мере образования списков документов символы КС будут заменяться соответствующими АС. Заметим, что можно для повышения контроля работы системы ввести еще обратные адреса связи, т. е. в последних членах списков (ассоциативных узлах) на местах АС ставить не КС, а адреса соответствующих заголовков списков в словаре дескрипторов; для указания же концов списков ставить символ КС (все единицы) на месте НС в этом же списковом слове. Кроме того, факт окончания списка будет устанавливаться и по равенству текущего адреса связи в прослеживаемом цепном списке адресу заголовка этого списка. Следует заметить также, что в описанной процедуре поиска остается открытым вопрос о вводе в систему дескрипторов запроса.

Это должно делаться специальной процедурой (или оператором) ввода. При обращении же к процедуре поиска считается, что этот массив дескрипторов конкретного запроса уже введен в машину и с его помощью производится конкретизация наименованием формального параметра ЗАПРОС в данной процедуре. Также считается заданной и величина *n* — число дескрипторов в запросе, которая должна вводиться при вводе запроса.

```
процедура ПОИСК (СЛОВАРЬ, ЗАПРОС, n, ОТВЕТ);
значение n; целый n;
составной массив СЛОВАРЬ; составной ССН;
целый АС вид 1(20), НС вид 1(5) уровень;
составной ССК; целый АС вид 1 (20), НС вид 1 (5)
уровень;
целый ЧД вид 1(12) уровень;
целый массив ЗАПРОС, ОТВЕТ;
начало
целый i, j, K, МАКС АС вид 1 (20);
составной массив ССР [1 : n]; целый АС вид 1 (20),
НС вид 1 (5) уровень;
список МАССИВ ДОКУМЕНТОВ;
составной ЗАГОЛОВОК УЗЛА; целый УД вид 9 (5),
ДИ вид 9 (6), ЧС вид 1 (5) уровень;
список ГРУППА АССОЦИАТИВНЫХ СЛОВ;
составной АССОЦИАТИВНОЕ СЛОВО; целый АС
вид 1 (20), НС вид 1 (5) уровень уровень уровень;

|¯СЛОВАРЬ [ ]¯| = ЗАПРОС;
|¯ЗАГОЛОВОК УЗЛА¯| = АС;,
|¯АССОЦИАТИВНОЕ СЛОВО¯| = АС + НС;

ВЫБОРКА ЗАГОЛОВКОВ СПИСКОВ:
для i:=1 шаг 1 до n цикл
ССР [i]:= ССН. |_ ЗАПРОС [i]_|; K: = 0;
```

ПРОВЕРКА НА СОВПАДЕНИЕ:
для *i:* = 1 **шаг** 1 **до** *n*—1 **цикл**
начало если АС. ССР[*i*]=КС **то перейти к**
ОКОНЧАНИЕ;

примечание случай АС. ССР [*n*] = КС будет обнаружен при выполнении оператора с меткой ВЫБОР МАКСАС;

если АС. ССР [*i*] ≠ АС. ССР [*i*+1] **то**
перейти к ВЫБОР МАКС АС **конец**;
K:=K+1*;*
ОТВЕТ [*K*]*:* = |_*AC.* ССР [1]_|;

примечание этот оператор осуществляет фиксацию найденного документа, а следующий оператор цикла осуществляет один шаг по всем прослеживаемым цепным спискам;

для *i*: = 1 **шаг** 1 **до** *n* **цикл**
ССР [*i*]:=|_АС. ССР [*i*] + НС. ССР [*i*]_|;
перейти к ПРОВЕРКА НА СОВПАДЕНИЕ;
ВЫБОР МАКС АС:
МАКС АС : =АС. ССР [*i*];

примечание *i* сохранило полученное значение, а все предшествующие АС равны *i* — тому АС;

для *j*: = *i* + 1 **шаг** 1 **до** *n* **цикл**
начало если АС. ССР [*j*] = КС **то перейти к**
ОКОНЧАНИЕ;

примечание все предшествующие АС уже проверены на равенство КС;
если МАКС АС <АС. ССР [*j*] **то**
МАКС АС: = АС. ССР **[*j*] конец;**

примечание при этом МАКС АС не может оказаться равным КС. Следующая часть программы осуществляет движение по всем прослеживаемым цепным спискам. Исходное положение характеризуется тем, что все текущие АС не равны КС и среди них выбран МАКС АС. Проверка на КС при движении по спискам будет производиться автоматически за счет того, что нами принято условие, что значение КС больше любого АС;

i: = 1
ШАГ ПО СПИСКУ:
если АС. ССР [*i*] < МАКС АС **то**
начало ССР [*i*]:=|_АС. ССР [*i*] + НС. ССР [*i*]_|;
перейти к ШАГ ПО СПИСКУ **конец иначе**
начало *i:* = *i* + l**; если *i* > *n* то**
перейти к ПРОВЕРКА НА СОВПАДЕНИЕ **иначе**
перейти к ШАГ ПО СПИСКУ **конец**
ОКОНЧАНИЕ:
конец

ЛИТЕРАТУРА

1. П. С. Новиков. Элементы математической логики. Физматгиз, 1959.
2. А. А. Ляпунов. О логических схемах программ. «Проблемы кибернетики», 1958, вып. 1, стр. 46—74.
3. Дж. Бекус и др. Сообщение об алгоритмическом языке АЛГОЛ-60. «Журнал вычислительной математики и математической физики», 1961, т. 1, № 2, стр. 308—342.
4. Р. Ледли. Программирование и использование цифровых вычислительных машин. Пер. с англ. Изд-во «Мир», 1966.
5. Е. Л. Ющенко. Адресное программирование. Госиздат техн. литературы УССР, Киев, 1963.
6. С. С. Лавров. Универсальный язык программирования. Изд-во «Наука», 1964.
7. М. И. Агеев. Основы алгоритмического языка АЛГОЛ-60, 1963.
8. А. И. Китов и П. А. Криницкий. Цифровые вычислительные машины и программирование. Физматгиз, 1961.
9. А. П. Ершов, Г. И. Кожухин, Ю. М. Волошин. Входной язык системы автоматического программирования (предварительное сообщение). ВЦ АН СССР, 1964.
10. М. Р. Шура-Бура, Э. З. Любимский. Транслятор АЛГОЛ-60. «Журнал вычислительной математики и математической физики, 1964, т. 4, № 1, стр. 96—112.
11. D. G. Bobrow and B. Raphael. A comparison of list-processing computer languages. Communications of the A. C. M., 1964, v. 7, № 4,
12. N. S. Prywes and H. J. Gray. The organisation of a multilist-type associative memory. IEEE Trans., 1963, Sept.
13. D. C. Cooper and H. Whi field. Computer Journ., 1962, v. 5, № 1. ALP: An autocode list-processing language.
14. W. I. Landauer and N. S. Prywes. A growing tree for descriptor language translation. Symbol languages Data Process, New-York–London, Cordon and Breach, 1962, p. 153—172.
15. A. Newell, F. M. Tonge. An introduction to the IPL-V. Communications A. C. M., 1960, v. 3, № 4.
16. J. Mc Carthy. Recursive functions of symbolic expressions and their computation by machine, p. 1. Communications A.C.M., 1960, v. 3, № 4.
17. H. Gelernter and ot. A fortran-compiled list-processing language. Journ. Association for Computing Machinery, 1960, v. 7, № 2.
18. М. А. Королев. Сообщение об алгоритмическом языке для экономических расчетов — АЛГЭК.
19. М. Н. Ефимова. Алгоритмические языки (обзор). Изд-во «Советское радио», 1964.

Редактор Т. М. Любимова. Художественный редактор В. Т. Сидоренко. Технический редактор Г. З. Шалимова.

Сдано в набор 21ЛХ. 66 г. Подписано в печать 7/II 1967 г. Т-00362

Формат 84ХЮ8'/м Бумага типографская № 2 Объем 17,22 усл. п. л.
Уч.-изд. л. 17,679 Тираж 15 800 экз.

Изд-во „Советское радио", Москва, Главпочтамт п/я 693,
Зак. 2634 Московская типография № 10 Главполиграфпрома Комитета по печати при Совете Министров СССР Шлюзовая наб., 10. Цена 1 р, 25 к.