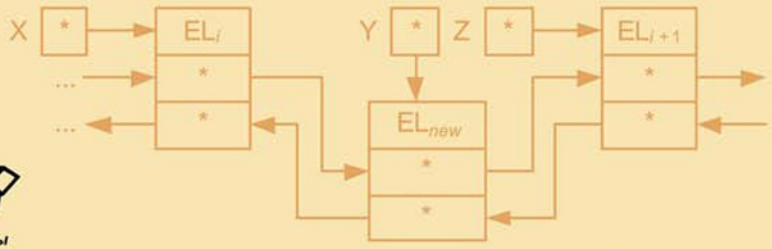
В. А. Касторнова

Структуры данных и алгоритмы их обработки на языке программирования Паскаль

- *Теоретический материал*
- *Вопросы и задания для лабораторных, практических и самостоятельных работ*
- *Для студентов высших и средних специальных учебных заведений*
- *Для всех форм обучения*



Материалы на www.bhv.ru

УЧЕБНОЕ ПОСОБИЕ

bhv®

УДК 004.438 Pascal
ББК 32.973.26-018.1
К28

Касторнова В. А.

К28 Структуры данных и алгоритмы их обработки на языке программирования Паскаль: учеб. пособие. — СПб.: БХВ-Петербург, 2016. — 304 с.: ил.

ISBN 978-5-9775-3622-6

Учебное пособие предназначено для изучения теоретического материала и выполнения лабораторных работ при изучении дисциплин «Алгоритмы и алгоритмические языки», «Структуры и алгоритмы обработки данных». Материал книги условно разбит на две части: язык программирования Паскаль и структуры данных. Описание языка включает: алфавит, структуру программ, типизацию данных, операторы, ввод/вывод данных, алгоритмические структуры, подпрограммы, процедуры и функции, типы данных, массивы, работу с файлами. При рассмотрении структур данных описываются структуры прямого и последовательного доступа, сортировка массивов, линейные списки и двоичные деревья, включая операции с ними. Рассмотрение типов данных и управляющих структур сопровождается графическими схемами и диаграммами, способствующими лучшему пониманию изучаемого материала. Вопросы и задания лабораторных работ также могут использоваться для организации практических и самостоятельных работ. Материалы книги будут полезны учителям информатики средних и специальных учебных заведений, готовящих специалистов в области информатики и вычислительной техники.

На сайте издательства находится электронный архив с исходными кодами примеров программ.

Для студентов и преподавателей профильных вузов

УДК 004.438 Pascal
ББК 32.973.26-018.1

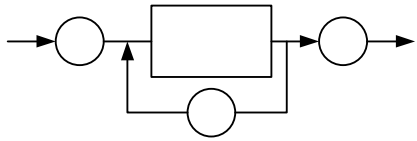
РЕЦЕНЗЕНТЫ:
О. А. Козлов, д-р пед., наук, канд. техн. наук, проф., завлабораторией ФГБНУ «ИУО РАО»
С. Н. Григорьев, д-р техн. наук, проф., ректор ФГБОУ ВПО МГТУ «СТАНКИН»

Подписано в печать 31.07.15.
Формат $60\times90^{1}/_{16}$. Печать офсетная. Усл. печ. л. 19.
Тираж 600 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

ISBN 978-5-9775-3622-6

Оглавление



ГЛАВА 1

Общая характеристика языков программирования

На этапе становления программирования программы для ЭВМ составлялись в машинных кодах, что создавало трудности при использовании этих программ для разных типов машин. Поэтому были разработаны языки программирования, позволяющие составлять программы на уровне манипуляции структурами данных, независимо от машинных кодов конкретной ЭВМ. Кроме того, это позволяло составлять программы людям, далеким от программирования в машинных кодах, на языке, близком к математике.

1.1. Языки программирования

Языки программирования — это формальные языки, применяемые для описания данных и алгоритма их обработки на ЭВМ. Они подразделяются на языки низкого и высокого уровней.

Язык низкого уровня представляет собой систему команд в машинных кодах. Программист при этом общается с машиной на "ее языке", который понимается ЭВМ без преобразований введенной программы. Примером такого языка является ассемблер. Этот язык используется, в основном, программистами-профессионалами, однако, обладает одним существенным недостатком — машинной зависимостью, т. е. невозможностью непосредственного переноса программы на другой тип машин (на другой тип процессора).

Работа с *языками высокого уровня* в ЭВМ происходит более сложным образом. Вначале команды языка преобразуются в шестнадцатеричные коды, затем транслируются (каждому коду ставится в соответствие одна или несколько машинных команд), и только после этого происходит процесс исполнения. Примерами языков высокого уровня являются Паскаль, Бейсик, Си и др. В отличие от языков низкого уровня, на языках высокого уровня удобнее программировать, т. е. общаться с машиной. Однако часто с простотой общения теряются некоторые преимущества системы команд процессора, поэтому практически в каждом языке высокого уровня есть возможность писать команды непосредственно на машинном языке (программировать в "кодах", а точнее — делать ассемблерные вставки).

1.2. Трансляторы

Процесс перевода программы с языка программирования высокого уровня на машинный язык называется *трансляцией*. Трансляция осуществляется с помощью специальной программы, называемой *транслятором*. Различают два вида трансляции: интерпретация и компиляция, а соответствующие им программы — интерпретаторы и компиляторы. Трансляторы с языков являются составной частью системы программирования, поэтому запуск программы предполагает сначала трансляцию, а потом собственно ее выполнение.

Рассмотрим процесс выполнения программы в режимах интерпретации и компиляции.

Интерпретация.

1. Машина считывает очередной оператор программы.
2. При обнаружении ошибки интерпретация прерывается, и машина указывает на это.
3. Переводит оператор в свои, ей понятные команды.
4. Выполняет переведенные команды.
5. Освобождает место в памяти после выполнения этих команд.

6. Переходит к п. 1, пока не выполнятся все операторы, т. е. пока не дойдет до указателя конца программы.

Процесс интерпретации можно проиллюстрировать в виде схемы, представленной на рис. 1.1.

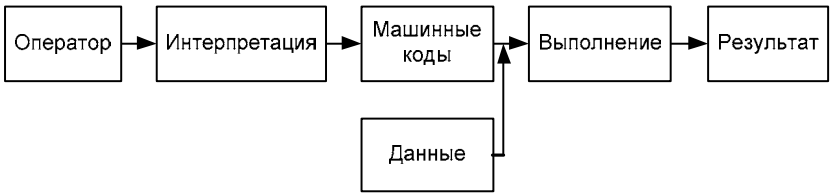


Рис. 1.1. Схема работы в режиме интерпретации

Компиляция.

1. Машина считывает очередной оператор программы.
2. При попадании на ошибку процесс компиляции прерывается, и машина указывает на это.
3. Переводит оператор в свои, ей понятные команды.
4. Переходит к п. 1, пока не дойдет до указателя конца программы.
5. Выполняет переведенную программу целиком.

Схема работы в режиме компиляции показана на рис. 1.2.

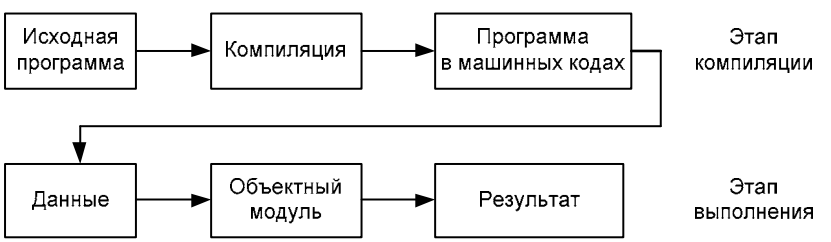


Рис. 1.2. Схема работы в режиме компиляции

Из описания указанного выше процесса выполнения программы следует, что интерпретатор работает медленнее, при запуске не обнаруживает сразу всех ошибок (а лишь при попадании на них

во время выполнения очередного оператора). Достоинством интерпретации является то, что она позволяет работать с программой в диалоговом режиме. Ее недостаток заключается в том, что при повторном запуске программы процесс ее трансляции повторяется заново.

Компилятор транслирует программу целиком и по ее завершении выдает машинный код, который может быть сохранен в виде отдельного файла и использоваться при работе с этой программой. Однако это требует выделения дополнительного объема памяти, т. к. в конце трансляции в ней находятся и исходный текст программы на языке высокого уровня, и ее машинный перевод.

1.3. История создания языков

Одним из первых языков программирования, созданных специально для учебных целей, был Бейсик, разработанный в 1964 году в Дартмутском колледже (США). Его создание преследовало цель предоставить возможность студентам пользоваться ЭВМ без длительной предварительной подготовки. Предполагалось также, что Бейсик будет использоваться в качестве универсального языка программирования людьми, не имеющими опыта работы на ЭВМ — рядовыми пользователями. Одним из достоинств языка является его удобство для работы в интерактивном режиме, что послужило использованию Бейсика при разработке диалоговых обучающих программ.

К концу 1960-х годов сложилась ситуация, когда для профессиональных целей использовались языки Фортран, Кобол, ПЛ/1 и пр., а большинство представителей учебного мира предпочитало Бейсик. Естественно, что многие считали такую ситуацию неудовлетворительной. По этой причине две группы исследователей приступили к созданию универсальных языков программирования, отвечающих требованиям того времени. Эти языки должны были включать в себя все достоинства существующих языков, иметь логически обоснованные структуры и быть легкими для восприятия. Такие языки были созданы. Одним из них стал Алгол-68, другой был разработан в Институте информатики Цюриха

(Швейцария) Никлаусом Виртом в 1971 году. Этот язык получил название Паскаль в честь великого французского ученого XVII века, сумевшего первым в мире изобрести автоматическое устройство для проведения вычислений. Транслятор с этого языка был разработан в 1973 году.

Хотя Паскаль почти так же прост, как и Бейсик, он имеет перед ним ряд преимуществ. Так, Паскаль способствует внедрению технологии программирования, основанной на поэтапном построении программы (принцип "сверху вниз"), состоящей из небольших, четко определенных процедур (структурный подход). Таким образом, преодолеваются главные недостатки, присущие Бейсику, — неэффективная организация подпрограмм и использование безусловного перехода. Разработанный Н. Виртом вариант языка является его стандартом. Помимо стандарта языка, в связи с разработкой различных компиляторов, появились версии Паскаля, среди которых наиболее популярными являются системы Turbo Pascal, Borland Pascal (для ОС MS-DOS), Delphi для ОС Windows.

1.4. Базовые структуры языков программирования

Понятие "структурное программирование" появилось в 1968 году, когда была опубликована статья одного из видных программистов того времени — нидерландского ученого Э. Дейкстры, в которой обосновывалась нежелательность применения оператора безусловного перехода (оператора, позволяющего сделать переход от одного оператора к другому, находящемуся в любом месте программы). Он четко обосновал, что использование основных базовых конструкций способствует обеспечению ясности и читабельности при составлении и дальнейшей доработке программ.

Таким образом, *структурным языком* назовем тот, у которого блок-схема любой программы состоит только из базовых структур и каждая структура имеет один вход и один выход. Имеются четыре типа управляющих структур: следование, выбор (ветвление, развилка), повторение (цикл) и группирование (вложение).

Для реализации *следования* существует правило: все команды выполняются в порядке их расположения в программе. Для выбора и повторения есть свои специальные инструкции (операторы, команды). *Выбор* предусматривает проверку условия с последующим выполнением одной или нескольких команд, в зависимости от истинности или ложности условия. Выбор (развилка) бывает полным или неполным (рис. 1.3, 1.4), в зависимости от числа выполняемых им операторов.

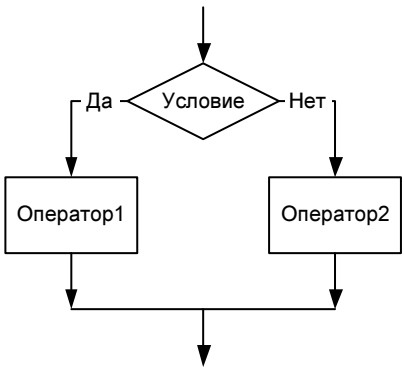


Рис. 1.3. Полная развилка

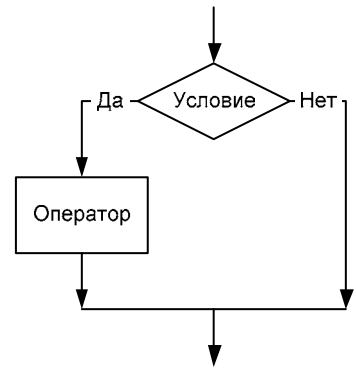


Рис. 1.4. Неполная развилка

Конструкция полной развилки работает следующим образом: при истинности "Условия" выполняется первая серия команд "Оператор1", а при ложности — вторая серия команд "Оператор2". В неполной развилке также осуществляется ветвление, но только в ней по истинности условия выполняется серия команд "Оператор", а при ложности — не выполняется никаких действий.

Повторение (цикл) представляет собой конструкцию, которая состоит, как и выбор, из проверки условия и серии команд. Однако, в отличие от выбора, данная серия команд может выполняться неоднократно, в зависимости от результата проверки условия. Повторения подразделяются на циклы с предусловием (циклы "пока") (рис. 1.5) и циклы с постусловием (циклы "до") (рис. 1.6). Серия команд "Тело цикла" повторяется до тех пор, пока "Условие" истинно в первом случае и ложно — во втором.

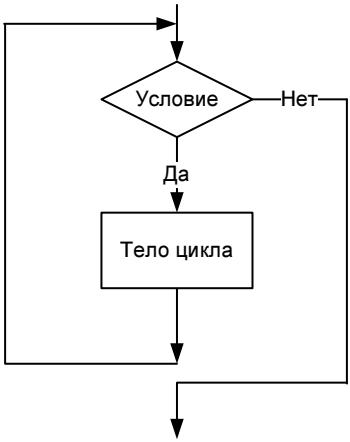


Рис. 1.5. Цикл с предусловием

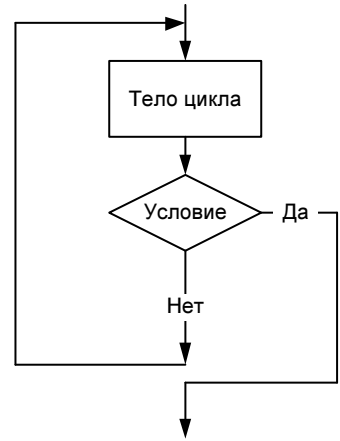


Рис. 1.6. Цикл с постусловием

Группирование означает объединение одной или нескольких конструкций внутри новой конструкции. Во всех языках имеются средства для формирования единого блока из группы конструкций (подпрограммы в Бейсике, составные конструкции и процедуры в Паскале). Примером группирования может являться также выполнение в конструкциях циклов следования или выбора и т. д.

1.5. Синтаксические диаграммы

Программирование на любом языке предполагает знание определенных форм записи, которые необходимо соблюдать при составлении программ. Порядок записи объектов программы и их структуры часто бывают трудными для понимания, поэтому прибегают к различным способам их отображения. Один из формальных методов, наглядно представляющий синтаксические конструкции языка в графическом виде, использует *синтаксические диаграммы*. Популяризировал эти диаграммы создатель языка Паскаль Н. Вирт, поэтому их часто называют синтаксическими диаграммами Вирта.

Синтаксическая диаграмма — это графическое представление отдельных объектов языка и строения самой программы. В син-

таксической диаграмме, как и в блок-схеме, используются плоские геометрические фигуры, соединенные стрелками. Однако, в отличие от блок-схемы, в которой отображается порядок выполнения действий определенной алгоритмической структуры, здесь указывается порядок синтаксического написания той или иной конструкции языка программирования.

На синтаксических диаграммах используются два вида четырехугольников — с прямыми и скругленными углами (иногда их заменяют кружками или овалами). В прямоугольники заключаются элементы языка, значения которых должны быть определены. В четырехугольниках со скругленными углами размещаются символы или иероглифы языка, значения которых в определении не нуждаются. Направление движения по диаграмме при раскрытии структуры понятия, записанного при входе в диаграмму, указывают стрелки (рис. 1.7).

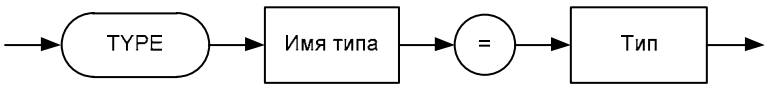


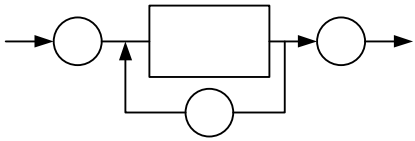
Рис. 1.7. Пример синтаксической диаграммы

На рис. 1.7 элементы "`Type`" и "`=`" являются служебными словами, а "Имя типа" и "Тип" требуют дальнейшей конкретизации и определения.

Чтобы получить правильные грамматические конструкции языка, используя синтаксические диаграммы, нужно идти по пути, указанному стрелками, от одного четырехугольника к другому до тех пор, пока не встретится выход. Там, где предусмотрено более одного направления движения, можно выбирать любое. Если по пути встречается ссылка на другую синтаксическую диаграмму, то следует войти в эту новую диаграмму, пройти по ней, выйти из нее и возвратиться на исходное место в первоначальной диаграмме.

1.6. Вопросы для повторения

1. Для чего служит язык программирования?
2. Какие уровни языков программирования выделяют? Дайте краткую характеристику каждого уровня.
3. Как называется программа, осуществляющая перевод с языка высокого уровня на язык машинных кодов?
4. Назовите достоинства и недостатки компилятора и интерпретатора.
5. В чем важность структурного подхода при составлении программ на языках высокого уровня?
6. Перечислите управляющие структуры языков программирования.
7. Какие бывают типы развилок и в чем их отличие?
8. Какие две части включает в себя структура "цикл"?
9. В чем состоит основное отличие структур "цикл „пока“" и "цикл „до“"?
10. Можно ли вкладывать одну управляющую структуру в другую или нужно располагать их только последовательно?



ГЛАВА 2

Описание языка Паскаль

2.1. Алфавит и словарь языка

Как и любой язык, Паскаль имеет свой алфавит, включающий в себя лексемы и разделители.

2.1.1. Лексемы

В лексемы Паскаля входят буквы, цифры, специальные символы, служебные (зарезервированные) слова, стандартные идентификаторы.

Буквы: латинские от `A` до `Z`, от `a` до `z` и русские от `А` до `Я`, от `а` до `я`.

Цифры: 0 1 2 3 4 5 6 7 8 9.

Специальные символы: `+ - * / = ^ < > ( ) [ ] { } . , : ; ' # $`

Служебные (зарезервированные) слова:

absolute	end	inline	program	type
and	external	label	record	until
array	file	mod	repeat	var
begin	for	nil	set	while
case	forward	not	shl	with
const	function	of	shr	xor
div	goto	or	string	
downto	if	packed	then	
else	in	procedure	to	

Стандартные идентификаторы:

Arctan	Delay	Length	Real
Assign	Delete	Ln	Release
Aux	DelLine	Lo	Rename
AuxInPrt	EOF	LowVideo	Reset
AuxOutPrt	EOLN	Lst	Rewrite
BlockRead	Erase	LstOutPtr	Round
BlockWrite	Execute	Mark	Seek
Boolean	Exp	MaxInt	Sin
BufLen	FileChar	Mem	Sqr
Byte	FilePos	MemAvail	Sqrt
Chain	FileSize	Move	Str
Char	Flush	New	Succ
Chr	Frac	NormVideo	Swap
Close	GetMem	Odd	Text
ClrEol	GotoXY	Ord	Trm
ClrScr	HeapPtr	Output	True
Concat	Hi	Port	Trunc
ConInPtr	Input	Pos	UpCase
ConOutPt	Insert	Pred	Usr
ConstPtr	InsLine	Ptr	UsrInPtr
Copy	Int	Random	UsrOutPtr
Cos	Integer	Randomize	Val
CrtExit	IOresult	Read	Write
CrtInit	Kbd	Readln	Writeln

2.1.2. Разделители

Символами-разделителями считаются пробелы, концы строк (разделители строк) и комментарии. Внутри лексем ни разделители, ни их части встречаться не могут. Между двумя следующими друг за другом лексемами должен обязательно следовать один или несколько разделителей.

Комментарии в паскаль-программе начинаются с символа { или (* и заканчиваются } или *). Сам комментарий может содержать любые символы, кроме } и *). Любой комментарий можно заменить в программе пробелом. Символы-разделители обычно применяются для улучшения читабельности программы (пример 2.1).

Пример 2.1. Программа сложения натуральных чисел

```
program PRIMER; var I, J, K: integer;
begin
  readln(I, J); {Ввод двух слагаемых}
  K:= I+J;
  writeln(I, '+', J, '=', K);  {Печать результата
                                в форме 12+3=15}
end.
```

2.2. Структура паскаль-программы

Язык Паскаль, как учебный алгоритмический язык, лег в основу разработки школьного алгоритмического языка (РАЯ), поэтому они имеют много общего и, прежде всего, это касается структуры программ (алгоритмов) (рис. 2.1).

РАЯ	Паскаль
АЛГ <имя>	PROGRAM <имя>
ДАНО	Раздел объявлений
НАДО	
НАЧ	BEGIN
...	...
...	Блок программы
Серия команд	(Серия операторов)
...	...
КОН	END

Рис. 2.1. Структура алгоритма (программы) в языках РАЯ и Паскаль

Сравнительный анализ представленных структур показывает, что по своему внешнему оформлению запись алгоритма на школьном алгоритмическом языке и программы на языке Паскаль во многом схожи. Действительно, оба этих описания начинаются с заголовка, в котором обязательно указывается имя алгоритма (программы). Наличие имени связано с тем обстоятельством, что описанный алгоритм в РАЯ и программа на Паскале могут служить вспомогательным алгоритмом (процедурой) для других, более сложных алгоритмов (программ).

В обоих языках принято описывать (объявлять) все переменные, фигурирующие в алгоритме (программе) с указанием их типов. Правда, в РАЯ эти переменные подразделяются еще на аргументы, результаты и промежуточные переменные, а в Паскале они просто перечисляются в разделе объявлений.

Идентификатор — это последовательность букв или цифр, начинающаяся с буквы. Отметим, что в идентификаторах могут использоваться только латинские буквы. Под *оператором* понимается указание ЭВМ по выполнению каких-либо действий.

Как видно из синтаксической диаграммы (рис. 2.2), любая паскаль-программа начинается со служебного слова `PROGRAM` и должна иметь имя, за которым может следовать список идентификаторов, заключенных в скобки. Заголовок программы заканчивается точкой с запятой. Затем идут объявления, служащие для описания типов данных, процедур и функций. Блок программы начинается со служебного слова `BEGIN`, за которым следуют операторы, разделенные точками с запятой, и в конце ставится служебное слово `END` с точкой. Эти слова принято называть *операторными скобками*. При составлении программ используются лексемы и разделители, определенные алфавитом языка.

По расположению операторов (инструкций) Паскаль довольно свободен. Инструкция может занимать не одну, а несколько строк. На одной строке можно разместить несколько инструкций. В программу можно вставлять пустые строки и пробелы, но пробелы в служебных словах недопустимы. Для лучшей читабельности программы строки можно располагать лесенкой.

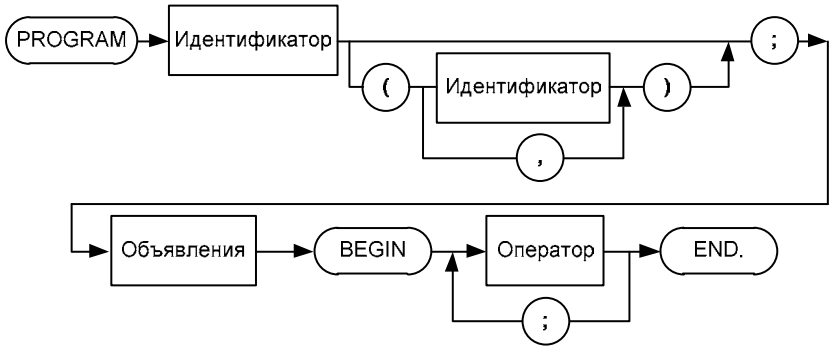


Рис. 2.2. Синтаксическая диаграмма паскаль-программы

2.3. Типизация данных

Данные — это общее понятие всего того, чем оперирует ЭВМ. Любой тип данных определяет множество значений, которые может принимать та или иная переменная, и те операции, которые можно к ней применять. Каждая встречающаяся в программе переменная может иметь только один тип.

В Паскале имеются три типа данных: простые, структурные и специальные. Рассмотрим вначале простой тип данных, представленный на схеме (рис. 2.3).

Начнем рассмотрение с *ординальных типов*. Для каждого элемента данных ординального типа можно найти его порядковый номер в данном типе. Ординальные типы, таким образом, представляют собой упорядоченные множества. К любому ординальному значению применимы три следующие встроенные функции:

- `ORD(X)` — дает порядковый номер элемента перечислимого типа. Результат относится к типу `INTEGER`;
- `SUCC(X)` — дает следующее за `X` значение, если `X` не максимальный элемент соответствующего типа. В последнем случае при использовании `SUCC(X)` возникает ошибка;
- `PRED(X)` — дает предыдущее значение `X`, если только `X` не минимальный элемент соответствующего типа. В последнем случае использования `PRED(X)` возникает ошибка.

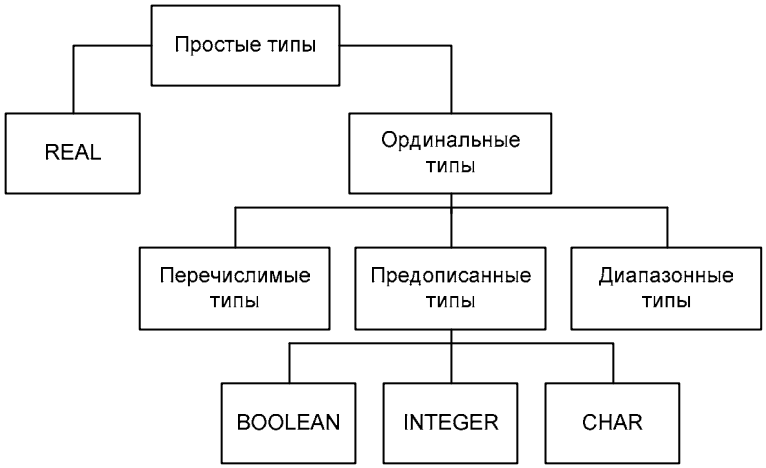


Рис. 2.3. Типизация данных

Наиболее простыми из ординальных типов являются *предописанные* или *встроенные типы*: `INTEGER`, `BOOLEAN` и `CHAR`, которые определяют соответственно числовые, логические (булевские) и литерные (символьные) величины. К встроенному (но не ординальному) типу данных относится также тип `REAL`.

Кроме предописанных, в Паскале существует возможность определять пользовательские типы данных. К ним относятся перечислимый и диапазонный типы. Каждый из них состоит из элементов предописанных типов, однако, в совокупности они дают совершенно новый тип.

Перечислимый тип задается перечислением всех своих элементов, что видно на синтаксической диаграмме (рис. 2.4).

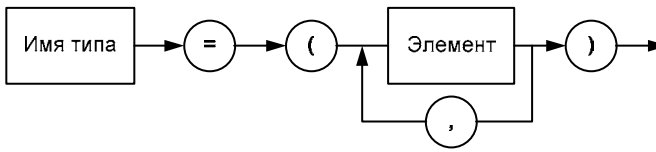


Рис. 2.4. Синтаксическая диаграмма перечислимого типа

Например:

```
DEN_NED = (MO, TU, WE, TH, FR, SA, SU); {Дни недели}
MONETA = (1, 5, 10, 50);                {Монеты}
```

Диапазонный тип представляет собой подмножество одного из ординальных типов (рис. 2.5). Его часто называют еще *интервальным*.

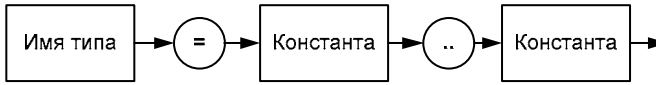


Рис. 2.5. Синтаксическая диаграмма диапазонного типа

Например:

```
DEN_MES = 1..31;       {Количество дней в месяце}
RAB_DEN = MO..SA;      {Рабочие дни недели}
LAT_BUKV = 'A'..'Z';   {Латинские буквы}
```

Следует помнить, что при задании диапазонного типа первая константа должна быть меньше второй. Подробнее на определении пользовательских типов остановимся в *разд. 2.4*.

Все типы, рассмотренные ранее, включая перечислимый и символьный, называются *скалярными*. Величины, принадлежащие скалярному типу, упорядочены (не путать с ординальностью): `3 < 5`; `1.2 > -6.8`; `'A' < 'C'`; `true > false`; `MO > TH`.

2.4. Объявление данных

С помощью объявлений программист сообщает компилятору, какие данные, процедуры и функции пользователя будут задействованы в программе. Описательная часть программы (объявления) состоит из 6 разделов, которые должны располагаться в следующем порядке:

- раздел модулей;
- раздел меток;
- раздел констант;
- раздел типов;

- раздел переменных;
- раздел процедур и функций.

Любой из перечисленных разделов может в объявлении отсутствовать.

Раздел описания модулей начинается со служебного слова `USES`, за которым идет перечень используемых в программе модулей типа `CRT`, `DOS`, `GRAPH` и др. Все эти модули находятся в библиотеке модулей, и каждый из них содержит соответствующий набор встроенных процедур и функций.

Раздел описания меток начинается со служебного слова `LABEL`, за которым следует список меток, разделяемых запятыми. Меткой может служить любое целое число, содержащее не более четырех цифр, или идентификатор. В конце раздела ставится точка с запятой.

Например:

```
label 342, 11, 1445, metka;
```

Раздел определения констант начинается со служебного слова `CONST`. Определение каждой константы содержит идентификатор (имя) константы, знак равенства и значение константы. Описания констант отделяются друг от друга точкой с запятой, как показано на диаграмме (рис. 2.6).

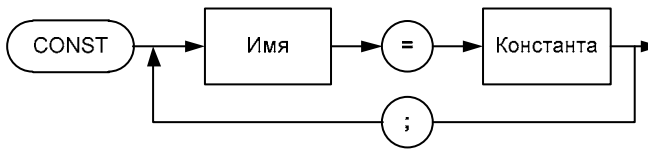


Рис. 2.6. Синтаксическая диаграмма раздела констант

Например:

```
const PI=3.1415927; E=2.7182818; Z='Паскаль';
{Задание констант PI, E, Z}
```

Определенные таким образом константы принято называть *именованными*, в отличие от констант, записываемых в программе в явном виде (числовом, символьном, логическом).

Раздел определения типов начинается со служебного слова TYPE (рис. 2.7). Каждому определенному типу соответствует имя. Данный раздел используется для создания структурных типов данных на основе имеющихся встроенных типов.

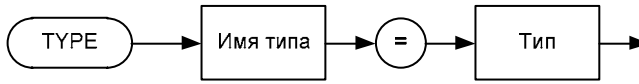


Рис. 2.7. Синтаксическая диаграмма раздела типов

Например:

```
type COLOR=(black, white, blue, green, red); DEN= 1..31;
{Задание типов COLOR, DEN}
```

Следует отметить, что определенные в этом разделе типы должны быть потом использованы в разделе описания переменных. Но, в основном, этот раздел используется при задании структурных типов данных (массивов, множеств, записей, файлов, динамических структур).

За разделом типов следует *раздел описания переменных*. Этот раздел начинается со служебного слова VAR (рис. 2.8). При объявлении переменных компилятору указывается, сколько переменных используется в программе, какие у них имена, данные какого типа будут храниться в этих переменных. Тип переменных может быть как стандартным, так и описанным в разделе TYPE.

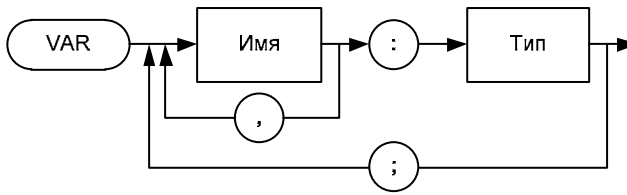


Рис. 2.8. Синтаксическая диаграмма раздела переменных

Например:

```
var X,Y,Z: integer; AD1,AD2: real; TEXT: char; {Переменные
                                          встроенных типов}
Q: DEN; D: 17..76;  ZVET: COLOR; {Переменные типов,
                                  заданных пользователем}
```