Структура программного модуля. Состав интегрированной программной среды

Лекция 4

Состав языка программирования

Язык программирования содержит следующие элементы:

Символы— это основные неделимые знаки, из которых составляются все тексты программ на данном языке. Совокупность всех символов образует *алфавит языка*.

Лексемы — это неделимые последовательности символов алфавита (элементарные конструкции), имеющие самостоятельный смысл.

Выражения строятся из лексем в строгом соответствии с правилами языка. Они задают порядок вычисления некоторого *значения*.

Операторы (инструкции или команды языка) задают полное описание некоторого действия, которое необходимо выполнить.

Действия, заданные операторами, выполняются над *данными*.

Предложения языка, в которых даются сведения о данных, называются *описаниями* или *неисполняемыми операторами*.

Совокупность описаний и операторов языка программирования, реализующая алгоритм решения конкретной задачи, образует программу на данном языке.

Синтаксис и семантика языка

Система правил записи элементов языка программирования (лексем, выражений, операторов) образует его *синтаксис*.

Смысл конструкций языка — это его *семантика*.

Синтаксис языка определяет внешний вид отдельных конструкций ("как они выглядят"), а семантика определяет действия, которые выполняет компьютер по этим конструкциям ("что они делают").

*Программа и данные

При работе в интегрированной среде разработки машинный код программы уже находится в оперативной памяти и готов к исполнению. Если подготовка программы была выполнена заранее и запускается файл с расширением exe из операционной системы, то сначала выполняется *загрузка программы* в оперативную память. При работе с интерпретатором в оперативной памяти находится не весь код программы, а только подготовленный фрагмент программы. В любом случае *исполняться может только код, размещенный в оперативной памяти*.

В ходе выполнения программы процессор последовательно получает и исполняет инструкции, из которых состоит программа. Большинство инструкций связано с какой-либо обработкой информации (данных). Все данные, необходимые программе для ее работы, помещаются в отдельную область оперативной памяти, которая называется *областью данных программы*. Таким образом, при запуске программы на выполнение для нее выделяются две различных области оперативной памяти: *область кода* и *область данных программы*.

Вся информация (и программа, и ее данные) хранится в памяти компьютера одинаково — в *двоичной* форме в виде последовательности *битов*.

Каждый бит может принимать значение одной двоичной цифры — *ноль* или *единица*. Восемь битов объединены в *байт*. Байт — это минимальная единица памяти, которая может быть выделена под переменную или инструкцию программы.

Каждый байт оперативной памяти имеет свой порядковый номер, который называется его *адресом* (адреса начинаются с нуля). Различные данные могут занимать один или несколько байт.

Участок памяти, в котором хранится одно значение, в программировании принято называть *ячейкой памяти*.

Адресом ячейки называют адрес ее самого первого байта. Таким образом, *каждый элемент данных имеет свой адрес в памяти*. Программа на машинном языке содержит в своих инструкциях конкретные адреса памяти, откуда следует брать данные и куда их помещать.

В языках высокого уровня данные обозначаются при помощи символических имен (идентификаторов), а всю работу по переводу символических имен в адреса оперативной памяти берет на себя транслятор.

*Константы и переменные

Каждый элемент данных в программе является **константой** или **переменной**.

Константами называются элементы данных, значения которых в процессе выполнения программы не изменяются. *Значения констант задаются непосредственно в тексте программы* при ее разработке.

Переменные, в отличие от констант, могут менять свои значения при выполнении программы. Для их хранения отводится *ячейка* в оперативной памяти компьютера, а сами значения переменных при выполнении программы вводятся в качестве исходных данных или формируются по ходу выполнения алгоритма.

*Типы данных

Для того чтобы компилятор смог нормально обработать символические имена, в программе нужно *выполнить описание (определение, объявление) переменных и символических констант*. При описании сообщается о *типе* каждой именованной величины.

Тип определяет множество значений, которые могут принимать данные (константы и переменные), а также совокупность операций, допустимых над этими значениями.

Тип *определяет форму внутреннего (машинного) представления данных и размер отводимой для них ячейки памяти*. Память под переменные выделяется до заполнения их данными. Поэтому необходимо явное указание типа.

*Способы внутреннего представления данных

В различных языках программирования обозначения типов данных и размеры ячеек памяти для них могут незначительно различаться, но внутренняя форма представления данных не зависит от конкретного языка программирования и определяется только архитектурой процессора.

Имеются всего две основных формы представления данных в компьютере:

- с фиксированной точкой;
- с плавающей точкой.

Первый способ употребляется для хранения целых чисел, символов и логических значений. Считается, что десятичная точка при этом зафиксирована в конце числа, т. е. дробной части нет.

Второй способ используется для вещественных чисел либо целых чисел, имеющих очень широкий диапазон значений.

Запись вещественного числа в форме с *мантиссой* и *порядком* использует степень десяти (например: $25 \cdot 10^{-3}$) и удобна для записи очень больших и очень маленьких чисел.

Число изображается так: пишется мантисса, знак умножения опускается, вместо основания 10 пишется буква *e*, а следом указывается порядок (показатель степени). Буква *e*, предшествующая порядку, читается как "умножить на 10 в степени".

Например, 5.18e+02 (518) или 10e-03 (0,01) — форма с мантиссой и порядком.

*Классификация типов

Язык Pascal имеет традиционный для языка высокого уровня набор типов данных, в который входят как простые (скалярные), так и составные (структурированные) типы. Кроме того, этот язык позволяет легко создавать производные (пользовательские) типы данных на основе уже имеющихся.

В языке Pascal к скалярным типам относятся:

- целочисленные;
- вещественные;
- символьный (литерный);
- логический (булевский).

Целочисленные типы, символьный, логический, а также два пользовательских типа данных (перечисляемый и интервальный) образуют группу так называемых *порядковых типов*.

Все порядковые типы имеют внутреннее представление с фиксированной точкой. Каждому значению порядкового типа можно поставить в соответствие целое число — порядковый номер.

Все вещественные типы имеют внутреннее представление с плавающей точкой. Диапазон этих чисел столь велик, что невозможно поставить в соответствие каждому из чисел порядковый номер, поэтому вещественные типы не относятся к порядковым.

*Целые типы

Целочисленные типы данных в языке Pascal различаются размером выделяемой ячейки памяти и способом представления (целое без знака или целое со знаком).

Название целого типа	Диапазон возможных значений	Память, байт
byte (байтовый)	0 - 255	1
shortint (короткий целый)	-128 - 127	1
integer (целый)	-32768 - 32767	2
word (слово)	0 - 65535	2
longint (длинный целый)	-2147483648 - 2147483647	4

*Логический тип

Логический (булевский) тип имеет всего два значения: **true** (да — истина, 1) и **false** (нет — ложь, 0).

Переменные логического типа занимают в памяти один байт, хотя для их хранения хватило бы и одного бита.

*Символьный тип

Символьный (литерный) тип представляет данные, являющиеся *символами*. В памяти компьютера символы хранятся в виде их *числовых кодов* (беззнаковые целые числа от 0 до 255). Каждый символ занимает ровно один байт в памяти.

Числовые коды преобразуются в буквы и другие символы лишь в момент их вывода на экран или принтер. Соответствие между символом и его кодом задается при помощи *кодовой таблицы*, которая находится в памяти компьютера и используется при выводе символов.

*Строковый тип

Последовательность символов, заключенная в апострофы, является *строкой* и относится к типу **string.** Причем сами апострофы не входят в состав строки, а лишь указывают на то, что все заключенные в них символы следует рассматривать как единое целое — *строковую константу*. Если в состав строки потребуется включить сам апостроф, достаточно написать его дважды подряд.

Строчные и прописные буквы в составе строки различаются, т. к. им соответствуют различные коды.

Максимальная длина строки — 255 символов. Символы внутри строки нумеруются от 1 до значения длины строки.

*Вещественные типы

Название вещественного типа	Диапазон возможных значений (плюс-минус)	Количество значащих цифр	Память, байт
single (с одинарной точностью)	1,5e-45 - 3,4e38	7-8	4
real (вещественный)	2,9e-39 - 1,7e38	11-12	6
double (с двойной точностью)	5,0e-324 - 1,7e308	15-16	8
extended (с повышенной точностью)	3,4e-4932 - 1,1e4932	19-20	10

*Типы данных, определяемые программистом

В Pascal имеются два дополнительных порядковых типа, которые относятся к пользовательским типам:

- интервальный тип или диапазон;
- перечисляемый тип.

Они используются для того, чтобы еще больше ограничить количество значений, принимаемых переменными этого типа, и позволяют:

- ✔ значительно улучшить наглядность программы;
- ✔ оптимально выделить память под переменные;
- ✔ облегчить поиск ошибок (благодаря возможности контроля тех значений, которые получают соответствующие переменные)

*Интервальный тип

Интервальный тип задается своим минимальным и максимальным значениями и может быть определен на основе любого порядкового типа:

МинимальноеЗначение .. Максимальное значение

Например:

1..12 (номер месяца может принимать значения от 1 до 12),

или ’a’..’z’ (буквы латинского алфавита — от a до z).

Для каждой операции с переменной интервального типа автоматически выполняется проверка: остается ли значение переменной внутри установленного для нее диапазона. Например, при попытке присвоить номеру месяца значение ноль будет выведено сообщение об ошибке.

*Перечисляемый тип

Перечисляемый тип задается перечислением своих значений.

Сами значения указываются через запятую, а весь список заключается в круглые скобки.

Например: color=(red, blue, green, black).

Для значений перечисления одного и того же типа допустимы операции сравнения.

В языке Pascal применение перечисляемого типа ограничено тем, что значения данных этого типа нельзя вводить с клавиатуры или выводить на какое-либо устройство вывода, а также над ними нельзя выполнять обычные арифметические операции. Их можно только присваивать переменной перечисляемого типа. Обычно этот тип применяется для хранения промежуточных данных, что позволяет сделать текст программы более выразительным и понятным.

Лексика языка Pascal

Неделимые последовательности знаков алфавита образуют *лексемы*. В языке Pascal различают такие виды лексем:

- константы;
- зарезервированные слова;
- идентификаторы;
- знаки операций;
- разделители;
- комментарии.

*Зарезервированные слова

Зарезервированные слова языка Pascal являются составной частью языка, имеют фиксированное начертание и несут в программе определенный смысл. Иначе их называют *служебными* или *ключевыми словами*.

Слово	Смысл слова	Слово	Смысл слова
Absolute	Абсолютный	Constructor	Конструктор
And	Логическое "и"	Destructor	Деструктор
Array	Массив	Div	Деление нацело
Asm	Ассемблер	Do	Выполнять
Begin	начало блока	Downto	Уменьшить до
Case	Вариант	Else	Иначе
Const	Константа	End	Конец блока
Export	Экспорт	or	Логическое "или"
External	Внешний	packed	Упакованный
File	Файл	procedure	Процедура
For	Для	program	Программа
Function	Функция	record	Запись
Forward	Опережающий	repeat	Повторять

Слово	Смысл слова	Слово	Смысл слова
Goto	Переход на	set	Множество
If	Если	shl	Сдвиг битов влево
Implementation	Реализация	shr	Сдвиг битов вправо
In	В (входит в)	string	Строка
Inherited	Унаследованный	then	То
Inline	Основной	to	Увеличивая
Interface	Интерфейс	type	Тип
Interrupt	Прерывание	unit	Модуль
label	Метка	until	До
library	Библиотека	uses	Использовать
mod	Остаток от деления	var	Переменная
nil	Отсутствие	while	Пока
not	Логическое "не"	with	С
object	Объект	xor	Исключающее "или"
of	Из		

*Идентификаторы

Идентификаторами называют *имена (названия)* переменных, констант, типов данных и других объектов программы.

Различают стандартные идентификаторы и идентификаторы, определенные пользователем.

Стандартные идентификаторы служат для обозначения заранее определенных разработчиками языка типов данных, констант, процедур и функций.

Например, **sin(x).**

Стандартные идентификаторы схожи с зарезервированными словами, однако между ними есть разница: *любой из стандартных идентификаторов допускается переопределять*. Пользователь может написать свою собственную функцию с именем **sin** или определить константу или переменную с таким именем.

Идентификаторы пользователя применяются для обозначения констант, переменных, типов и других объектов, определенных самим программистом. Тип идентификатора пользователя должен быть указан в описательной части программы, до его использования.

Для написания идентификаторов применяются следующие правила:

- состоят из букв латинского алфавита, цифр и знака подчеркивания;
- начинаются с буквы или знака подчеркивания. Только для метки допускается использование целого числа без знака;
- нельзя использовать имена, совпадающие по написанию с приведенными ранее зарезервированными словами. Крайне нежелательно также переопределение стандартных идентификаторов;
- при написании имен можно использовать как прописные, так и строчные буквы. Компилятор не делает различий между ними.

Знаки операций — это обозначения операций над данными различных типов. Значения, к которым применяется операция, называются ее *операндами*.

Примеры операций — **+**, **<**, div, mod.

Разделителями служат — пробел, символ конца строки, точка с запятой, точка, запятая, двоеточие, круглые и квадратные скобки.

Разделители, стоящие внутри строковой константы, воспринимаются не как разделители, а как ее часть.

*Комментарии

В любом месте программы, где разрешен разделитель, можно записать пояснительный текст — *комментарий*. Он не обрабатывается компилятором и не включается в исполняемый файл.

Текст комментария ограничен символами { } или (* *):

Допускается следующая вложенность комментария:

{ Текст (* ТекстКомментария2 *) Комментария 1 }

или

(*Текст { ТекстКомментария2 } Комментария1*)

Комментарии удобно использовать при *отладке программы* для *временного исключения* группы операторов, которая, будучи заключена в { } или (* *), воспринимается как комментарий и, следовательно, не выполняется.

Структура программы на языке Паскаль

Раздел *uses*

Раздел **uses** позволяет подключать *стандартные и пользовательские библиотечные модули*. Он начинается с зарезервированного слова **uses** и имеет следующий вид:

uses

ИмяМодуля1, ИмяМодуля2, ...;

Например: uses crt;

Раздел описания меток

Перед любым оператором Pascal в тексте программы можно поставить *метку*, что позволяет выполнить прямой переход на этот оператор с помощью оператора **goto** из любого места программы. Метка состоит из имени и следующего за ней двоеточия, после которого и располагается помеченный данной меткой оператор. Раздел описания меток начинается с зарезервированного слова **label** и имеет следующий вид:

label

ИмяМетки1, ИмяМетки2, ...;

Раздел описания констант

Хранение констант не требует памяти и компилятор помещает их значения прямо в текст исполняемой программы. Каждая константа принадлежит к определенному типу данных, однако при определении константы его обычно не указывают — тип констант автоматически опознается по форме их записи.

Раздел описания констант начинается с зарезервированного слова **const** и имеет следующий вид:

const

ИмяКонстанты = ЗначениеКонстанты;

Идентификатор константы можно употреблять при определении следующих за ней констант.

Например:

Const a=3; b=6.98; c='z';

Раздел описания типов данных

В языке Pascal существует *механизм создания новых типов данных*. Каждое новое определение типа задает множество значений и связывает с этим множеством имя типа. Определение новых типов выполняется в разделе описания типов данных.

Раздел начинается с зарезервированного слова type и имеет вид:

type

ИмяТипа1 = ОписаниеТипа1;

ИмяТипа2 = ОписаниеТипа2; …

Далее идентификаторы типов можно использовать для описания переменных.

Например

```
Type day=1..31;
```

Раздел описания переменных

Все переменные, используемые в программе, должны быть перечислены в *разделе описания переменных*. Описание должно предшествовать использованию переменной.

Раздел начинается с зарезервированного слова var и выглядит так:

var

ИмяПеременной1, ИмяПеременной2: ТипПеременной; …

Например: var

znak: char; { символьная переменная }

flag: boolean; { логическая переменная }

Month: 1..12; { переменная интервального типа }

Пользовательские типы данных можно определять в разделе **var.** Такие типы называются *анонимными*. Однако лучше определять новый именованный тип в разделе **type.**

Раздел описания процедур и функций

Данный раздел используется в программах, которые с целью удобства программирования были разбиты на более мелкие части — подпрограммы. *Подпрограммой* называется программная единица (часть программы), имеющая имя, по которому она может быть вызвана из других частей программы.

Подпрограммы бывают двух видов: *процедуры* и *функции*, которые, в свою очередь, делятся на *стандартные* и *определенные пользователем*. Стандартные процедуры и функции являются частью языка и вызываются без предварительного описания. Описание процедур и функций пользователя выполняется в разделе описания процедур и функций.

Раздел операторов

Раздел операторов является основным, т. к. именно в нем над предварительно описанными константами, переменными, значениями функций выполняются действия, позволяющие получить результат, ради которого и создавалась программа.

Раздел начинается зарезервированным словом **begin** и далее следуют операторы языка, *отделенные друг от друга точкой с запятой*. Завершают раздел зарезервированное слово **end** и *точка*.

* Структура программы на языке Паскаль

Раздел *uses (описания подключаемых модулей)*

***Раздел* label (описания меток)**

Раздел const (описания констант)

Раздел type (описания типов)

Раздел var (описания переменных)

Раздел описания процедур и функций

Begin

(Раздел операторов)

end.

*Р*азделы описания могут встречаться в программе любое количество раз и следовать в произвольном порядке (кроме раздела **uses,** который всегда расположен после заголовка программы). Любой раздел, кроме раздела операторов, может отсутствовать. *Главное, чтобы все описания объектов программы были сделаны до того, как они будут использованы.*

*Оператор присваивания

Оператор присваивания задает значения переменных в ходе выполнения программы и имеет вид:

ИмяПеременной := выражение;

В зависимости от типа результата различают три вида выражений: арифметические, логические и символьные:

арифметические выражения служат для обработки числовых данных и в результате их вычисления получается число;

логические выражения служат для сравнения различных данных и для других логических действий (пример: **x>5** — результат такого выражения имеет логический тип и принимает значения **true** или **false);**

символьные выражения нужны для обработки текстов;

Результат, полученный при вычислении выражения, находящегося в правой части, должен быть *совместим по типу* с переменной, которой он присваивается. При нарушении соответствия выводится сообщение об ошибке **Type mismatch** (Несоответствие типов).

*Арифметические операции

Операции определяют действия, которые надо выполнить над операндами. В отличие от традиционной математической записи необходимо указывать все знаки операций.

Все операции делятся на унарные и бинарные.

Унарные операции относятся к одному операнду,

бинарные — связывают два.

Например, **-**a — унарная операция, a**+**b — бинарная. При использовании двух знаков операций нежелательно, чтобы они стояли рядом: **a***-b. Лучше заключить второй операнд в скобки: **a*(-b).** Хотя результат от этого не изменится, поскольку приоритет унарного минуса выше умножения.

Операция	Знак	Тип	
		операндов	результата
Бинарные операции			
Сложение	+	Real	Real
		Integer	Integer
Вычитание	-	Real	Real
		Integer	Integer
Умножение	*	Real	Real
		Integer	Integer
Деление	/	Integer	Real
		Real	Real
Целочисленное деление	Div	Integer	Integer
Остаток от деления	Mod	Integer	Integer
Арифметическое И	And	Integer	Integer
Побитовый сдвиг влево	Shl	Integer	Integer
Побитовый сдвиг вправо	Shr	Integer	Integer
Арифметическое ИЛИ	Or	Integer	Integer
Арифметическое побитовое сложение по модулю 2	Xor	Integer	Integer
Унарные операции			
Сохранение знака	+	Real	Real
Отрицание знака	-	Real	Real
		Integer	Integer
Арифметическое отрицание	Not	Integer	Integer

*Приоритет операций

Последовательность выполнения операций в составе выражения происходит с учетом их *приоритета (старшинства)*.

Операции с равным приоритетом выполняются слева направо. Выражение, заключенное в скобки, перед выполнением вычисляется как отдельный операнд. При наличии вложенных скобок вычисления выполняются, начиная с самых внутренних.

Операция	Приоритет	Вид операции
Унарный минус, not	Первый (высший)	Унарная операция
*, /, div, mod, and, shl, shr	Второй	Операции типа умножения
+, — or, xor	Третий	Операции типа сложения
=, < >, <, >, <=, >=, in	Четвертый	Операции отношения

* Стандартные арифметические функции

Стандартная функция	Выполняемое действие	Тип	
		аргумента	результата
abs(x)	|x|	real	Real
		integer	integer
sqr(x)	X^2	real	Real
		integer	integer
sqrt(x)	$X^{1/2}$	real	Real
		integer	Real
exp(x)	e^x	real	Real
		integer	Real
Ln(x)	Ln(x)	real	Real
		integer	Real
Pi	число пи	-	Real
sin(x)	Sin(x)	real	Real
		integer	Real
cos(x)	Cos(x)	real	Real
		integer	Real
arctan(x)	Arctg(x)	real	Real
		integer	Real

Вызов стандартной функции осуществляется путем указания в нужном месте программы имени функции **(abs, in, exp** и др.) и ее аргумента, заключенного в круглые скобки. После вычисления значения функции ее вызов заменяется результатом и расчет содержащего ее выражения продолжается дальше.

A:=sin(x);

Следует знать:

□ аргумент прямых тригонометрических функций **sin** и **cos** задается в радианах. Для преобразования значения угла из радианной меры в градусную необходимо умножить величину угла на число **180/pi.** Для перевода значения угла из градусной меры в радианную необходимо умножить величину угла на число **pi/180**;

□ результат функции **arctan** получается в радианах.

Помимо приведенных в табл. функций, в арифметических выражениях также используются следующие стандартные функции:

функция **random (диапазон)** возвращает случайное число x, удовлетворяющее условию **0<=x<диапазон.** Тип аргумента и результата — **word.**

Перед первым обращением к функции **random** необходимо с помощью вызова процедуры **randomize** инициализировать программный генератор случайных чисел. В противном случае при каждом запуске программы датчик будет выдавать одни и те же числа.

* Типы в арифметических выражениях. Автоматическое преобразование типов

Языки программирования высокого уровня разрешают использование в одном выражении операндов различных целых и вещественных типов.

Например, **5+0.5**— правильное выражение. Говорят, что вещественные и целые типы являются *совместимыми*.

Для вычисления выражения, содержащего данные разных типов, транслятор порождает команды *преобразования типов* в соответствии с правилом: *данные преобразуются к типу с более широким диапазоном значений*.

Например, если складываются переменные типа **integer** и типа **real**, то перед выполнением сложения переменная целого типа будет преобразована к действительному типу. Это действие называется *автоматическим или неявным преобразованием типа*.

То же самое правило действует и при *присваивании* значения переменной:

если переменная имеет тип с более широким диапазоном значений, чем тип присваиваемого значения, то произойдет *автоматическое преобразование* типа;

если переменная имеет более узкий диапазон значений, то компилятор выдаст сообщение об ошибке **Type mismatch** (Несоответствие типов).

Функции *trunc* и *round* выполняют следующие преобразования типов:

- функция **trunc(x)** возвращает ближайшее целое число, меньшее или равное вещественному **x** для **x**>=0 и большее или равное **x** для **x**<=0 (от англ. *truncate* — усекать). Таким образом, выполняется *отбрасывание* десятичных знаков после точки. Аргумент — **real**, результат — **longint.**
- функция **round(x)** возвращает значение **x**, *округленное* до ближайшего целого числа (от англ. *round* — круглый). Аргумент — **real**, результат — **longint**.

Использование функций **trunc** и **round**, у которых аргумент принадлежит одному, а результат другому типу, называется *явным преобразованием типа*.

Домашнее задание

1. Составить опорный конспект лекции по теме «Структура программного модуля. Состав интегрированной среды» на основе презентации.

2. Программирование на языке Pascal. Рапаков Г. Г., Ржеуцкая С. Ю. СПб.: БХВ-Петербург, 2004, стр. 19-26