* Заголовок завершается точкой с запятой.
* В любом объявлении каждый список завершается точкой с запятой.
* Операторы отделены один от другого точкой с запятой.

Слова **begin** и **end** не являются операторами – они служат знаками пунктуации. Слово begin выступает в качестве левой, а end – правой скобки. Так как они сами знаки пунктуации, то точка запятой после begin и перед end не обязательна. В программах на Паскаль слова begin и end используются преимущественно для образования *составных операторов*. Составной оператор может быть использован в любом месте, где мог бы быть использован простой оператор. Пример составного оператора:

**begin**

t := a;

a := b;

b := t

**end**;

Слова в других операторах также действуют как знаки пунктуации.

**if** ab > bd **then**

write(‘yes’)

**else**

write(‘no’);

Слова **if, then, else** выступают внутри оператора в качестве знаков пунктуации.

Операторы разделены знаками пунктуации, поэтому расположение программы на странице с точки зрения компилятора значения не имеет. Вполне достаточно придерживаться двух правил:

* Не писать слова вместе;
* Не разрывать слово пробелами или переходом на новую строку.

В остальном компилятору все равно, как будет расположена программа, однако, это совсем не безразлично для программиста. Польза *отступов* в прояснении структуры программы. Взгляды на выбор отступов весьма различны, но все согласны в одном – отступы должны делать структуру программы максимально наглядной.

Слова **program, const, var, begin, end**, а также множество других называются *зарезервированными словами*. Зарезервированные слова нельзя расширять (например, constant будет ошибкой) и сокращать (например, prog также будет ошибкой).

Использовать в программном коде на Pascal можно как прописные, так и строчные буквы, а также их чередовать. Однако в *строках* (тип данных) разница между прописными и строчными буквами существует.

**Операторы Pascal**

Под операторами в языке Паскаль подразумевают только описание действий. Операторы отделяются друг от друга только точкой с запятой. Если оператор стоит перед end, until или else, то в этом случае точка с запятой не ставится.

**Оператор присваивания**

Общий вид:

v := a;

здесь v – переменная, a – выражение, := - операция присваивания. Выражение a может содержать константы, переменные, названия функций, знаки операций и скобки.

Пример. f := 3 \* C + 2 \* sin(x);

Вид выражения однозначно определяет правила его вычисления: действия выполняются слева направо с соблюдением следующего старшинства (в порядке убывания):

1. not;
2. \*, /, div, mod, and;
3. +, -, or;
4. =, <, >, <>, <=, >=, in.

Любое выражение в скобках вычисляется раньше, чем выполняется операция, предшествующая скобкам.

Присваивание допускается для переменных всех типов, за исключением типа файл.

В операции v := a переменная v и выражение a должны иметь один и тот же тип, а для интервального типа – одно и то же подмножество значений.

Примечания. Разрешается присваивать переменной типа real выражение типа integer. Нельзя присваивать переменной типа integer выражение типа real.

**Составной оператор**

Если при некотором условии надо выполнить определенную последовательность операторов, то их объединяют в один составной оператор.

Составной оператор начинается ключевым словом begin и заканчивается словом end. Между этими словами помещаются составляющие операторы, которые выполняются в порядке их следования. После end ставится точка с запятой, а после begin – только пробелы (либо комментарий).

Пример.

**begin**

i := 2;

k := i / 5

**end**;

Слова begin и end играют роль операторных скобок. Тело самой программы также имеет вид составного оператора. После последнего end программы ставится точка. Нельзя извне составного оператора передавать управление внутрь него.

# Выражения в Pascal

В операторах присваивания можно использовать арифметические выражения. Например:

num := (d + n) / 10;

sq := trunk(num) + 1;

Скобки обеспечивают необходимый порядок вычислений. Если бы в первом примере скобки были опущены:

num := d + n / 10;

то сначала было бы выполнено деление, приоритет которого выше. Приоритет в арифметических выражениях выше у операций умножения (\*) и деления (/), ниже у сложения и вычитания.

Во втором из приведенных примеров производится присваивание значения целого числа. Функция **trunc** дает целый результат, а число 1 записано без десятичной точки; таким образом, оба слагаемых в сумме дают целое значение. Вообще, когда все члены выражения – целые, само выражение принимает целое значение.

У сформулированного выше правила существует важное исключение: деление (с использованием знака /) всегда дает вещественный результат:

6.5 / 2 = 3.25

6 / 2 = 3.0

Деление нацело (нахождение частного и остатка) может быть выполнено при помощи операций **div** и **mod**.

Выражение может включать в себя и целые и вещественные члены. Наличие хотя бы одного вещественного члена или знака / приводит к тому, что значение результата будет вещественным. Функции **trunc** и **round** могут быть использованы для преобразования вещественного числа в целое.

Функция **sqr** возводит значение аргумента (записанного внутри скобок) в квадрат. В Паскале нет оператора возведения в произвольную степень. Возведение в степень здесь осуществляется с использованием логарифмов. Вместо математического выражения ax на Pascal можно написать exp(ln(a) \* x).

Знаки <, >= и подобные также играют роль операций. Выражения, содержащие подобные операции, принимают логическое значение и называются логическими выражениями. В состав логических выражений могут входить логические операции **not** (не), **and** (и), **or** (или). Такие логические выражения

# Ввод-вывод данных в Pascal

Компьютерные программы обрабатывают (изменяют) различные данные. Программа получает данные, что-то делает с ними и выводит их в измененной форме или выводит другие данные.

Следовательно, любой язык программирования должен иметь инструменты как для ввода данных, так и их вывода. В Паскале ввод осуществляется с помощью процедур read() и readln(), а вывод - благодаря write() и writeln(). Процедуры, которые имеют окончание ln, после своего выполнения переводят указатель на новую строку.

Откуда или с помощью чего можно ввести данные в программу? Обычно это можно сделать с помощью клавиатуры или из файлов.

Куда можно вывести данные? На экран, в файл, на принтер и др.

Стандартным устройством ввода является клавиатура, а вывода — монитор. Стандартные — значит "работающие по-умолчанию"; т.е. если не указано ничего иного, то программа будет считывать данные с клавиатуры, а выводить их на монитор. Вместе клавиатуру и монитор называют консолью. Таким образом консоль представляет собой стандартное устройство ввода-вывода.

## Вывод данных на экран. Форматированный вывод

Вывод данных на экран и в файл в языке программирования Pascal осуществляется с помощью процедур write() и writeln(). Здесь будет рассмотрен вывод только на экран.

Допустим, нам требуется отобразить на экране пару фраз. Если мы хотим, чтобы каждая из них начиналась с новой строки, то надо использовать writeln(), если нет – то write().

Write() чаще используется, когда надо вывести для пользователя сообщение на экран, после чего получить данные, не переводя курсора на новую строку. Например, выводим на экран "Введи число: " и не переводим курсор на новую строку, а ждем ввода.

Еще один пример. В памяти компьютера хранятся данные. Из программы мы обращаемся к ним с помощью переменных num, fl и st. Вывести их значения на экран можно по-разному.

Во втором случае мы видим, что процедуры вывода (как write() так writeln()) позволяют конструировать выводимую информацию из различных компонент (строк-констант и переменных).

В третьем случае был осуществлен так называемый форматированный вывод. При этом для выводимого значения указывается ширина поля вывода (количество знакомест). Если мы выводим вещественное (дробное) число, то вторым числом через двоеточие указывается количество знаков после запятой. Если для вещественных чисел не осуществлять форматирование, то они отобразятся так, как определено для данного компьютера. Если указать только число знакомест без фиксирования дробной части, то вывод будет в экспоненциальной форме.

## Ввод данных с клавиатуры

Ввод данных в языке программирования Паскаль обеспечивается процедурами read() и readln(). Ввод данных осуществляется либо с клавиатуры, либо из файла. Здесь рассматривается только ввод с клавиатуры.

Когда данные вводятся, то они помещаются в ячейки памяти, доступ к которым обеспечивается с помощью механизма переменных. Поэтому, когда в программе на Pascal используется процедура read() (или readln()), то в качестве фактического параметра (аргумента) ей передается имя переменной, которая будет связана с вводимыми данными. Потом эти данные можно будет использовать в программе или просто вывести на экран.

В процедуры ввода можно передавать не один фактический параметр, а множество.

При вводе данных их разделяют пробелом, табуляцией или переходом на новую строку (Enter). Данные символьного типа не разделяются или разделяются переходом на новую строку.

Существуют особенности ввода данных с помощью операторов read() и readln(). Если используются подряд несколько операторов read(), то вводимые данные можно разделять всеми допустимыми способами. При использовании нескольких вызовов readln() каждый последующий срабатывает только после нажатия Enter. Программа ниже иллюстрирует это. Комментарии поясняют последовательность возможных действий при вводе данных.

## Переменные

Любая программа обрабатывает данные (информацию, объекты). Данные, с которыми работает программа, хранятся в памяти компьютера (чаще оперативной). Программа должна знать, где они лежат, каким объемом памяти она располагает, как следует интерпретировать данные (например, как числа или строки). Для обеспечения программе доступа к участкам памяти существует механизм переменных.

Переменные описываются в начале программы и как бы сообщают о том, с какими данными будет работать программа и какой объем памяти они займут. Другими словами, резервируется память. Но это не значит, что в эти ячейки памяти помещаются конкретные значения (данные). На момент резервирования памяти в них может быть что угодно.

В процессе выполнения программы в ячейки памяти будут помещаться конкретные значения, извлекаться оттуда, изменяться, снова записываться. Мы же через программу обращаемся к ним посредством имен переменных, которые были описаны в начале программы.

Имена переменных могут быть почти любым сочетанием английских букв и цифр (без пробелов). Нельзя чтобы имена переменных совпадали со словами, которые являются какими-либо командами самого языка программирования. Нельзя начинать имена переменных с цифры или специального символа. Для того чтобы имена переменных были удобны для восприятия, надо стараться придерживаться пары правил. Если программа не простейший пример, то имена переменных должны быть осмысленными словами или их сокращениями. Желательно, чтобы имена переменных не были слишком длинными.

В Pascal прописные и строчные буквы в именах переменных не различаются.

При описании переменных указывается не только их имя, но и тип. Тип переменных сообщает о том, сколько отвести под них памяти и что за данные там планируется сохранять. Точнее хранится там всегда будут числа в двоичной системе счисления. Но что они значат, может быть чем угодно: целым или дробным числом, символом, строкой, массивом, записью и др. Т.е. тип переменной определяет то, что мы можем сохранить в участке памяти, с которым связана описываемая переменная.

В примере ниже происходит ввод значений трех переменных разных типов и выполнение допустимых для их типов операций. Если тип переменных integer, то им можно присваивать только целые числа в диапазоне от -32768 до 32767. Под тип **integer** в языке программирования Паскаль отводится 2 байта, что равно 16 битам, а это значит, что можно хранить 216 (65536) значений (отрицательные и положительные числа, а также ноль). В этих диапазонах переменные int1 и int2 могут принимать какие угодно значения. При попытке записи в переменную значения не ее типа возникнет ошибка.

**var**

int1,int2: **integer**;

r: **real**;

ch: **char**;

**begin**

write('Type integer: ');

readln(int1);

write('Type real: ');

readln(r);

write('Type char: ');

readln(ch);

int2:= int1 **mod** 10;

int1:= int1 **div** 10;

r:= r + 0.5;

ch:= chr(ord(ch)+1);

writeln(int1:5,int2:3,r:7:2,ch:3);

**end**.

Пример работы программы:

**Type** **integer**: 32

**Type** **real**: 5.34

**Type** **char**: A

3 2 5.84 B

Итак, переменные связаны с участками памяти, содержимое которых может меняется по ходу выполнения программы в определенных пределах.

### Длина переменных

Ранние версии компиляторов Паскаля имели ограничение на учитываемую длину имен переменных. В имени учитывались лишь первые восемь символов. Поэтому, например, такие переменные как variable1 и variable2 компилятор воспринимал как одно и тоже, т.к. первые восемь символов совпадали.

В современных версиях компиляторов с языка Pascal таких ограничений нет. Например, в результате работы приведенной ниже программы как в среде Free Pascal, так и Turbo Pascal 7.1 на экран выводятся два разных числа (10 и 20). Это значит, что компиляторы учитывают больше восьми символов в именах переменных (и других идентификаторов: именах констант, процедур и др.).

**var**

variable111, variable222: **integer**;

**begin**

variable111 := 10;

variable222 := 20;

writeln(variable111);

writeln(variable222);

readln

**end**.

## Константы

Что делать, если в программе требуется постоянно использовать какое-нибудь одно и тоже число. Вроде бы это не проблема. Можно описать переменную, затем присвоить ей значение и не изменять его в программе. Однако это не всегда удобно (можно нечаянно изменить), поэтому в языках программирования для хранения данных помимо переменных существуют константы.

Главное преимущество констант заключается в том, что они описываются в начале программы и им сразу там же присваивается значение, а при выполнении программы константы не изменяются. Но если при правке кода, программист решит поменять значение константы, он впишет в ее описание другое значение, а сам код программы редактировать не придется. Поэтому, если в программе часто планируется использоваться какое-то значение, опишите его в разделе констант, который в программе располагается до раздела переменных:

**const**

конст1 = значение;

конст2 = значение;

Значениями констант могут быть данные большинства типов, используемых в языке Паскаль.

Рассмотрим такую программу:

**const** n=10;

**var**

i: **byte**;

sum: **word**;

**begin**

sum:= 0;

**for** i:=1 **to** n **do**

sum:= sum+i;

writeln('Sum of ',n,' numbers = ', sum);

**end**.

В ней используется константа n со значением 10. Программа считает сумму десяти чисел в диапазоне от 1 до 10. Если нам захочется посчитать сумму чисел до 20, то достаточно будет изменить значение константы в начале программы. Если бы константа не использовалась, то пришлось бы просмотреть весь код программы и исправить значение 10 на 20. Если программа большая, то легко ошибиться: не найти или исправить не то значение.

### Типизированные константы

В языке Паскаль помимо обычных констант используются типизированные константы. Можно сказать, что они занимают промежуточное положение между переменными и константами. Они получают значение при описании (как константы), но могут его менять в теле программы (как переменные).

Описываются типизированные константы в разделе констант:

**const**

конст1: тип=значение;

конст2: тип=значение;

Например:

**const**

nums: **integer**=10;

# Типы данных в Паскале

Любая программа, написанная на любом языке программирования, по большому счету предназначена для обработки данных. В качестве данных могут выступать числа, тексты, графика, звук и др. Одни данные являются исходными, другие – результатом, который получается путем обработки исходных данных программой.

Данные хранятся в памяти компьютера. Программа обращается к ним с помощью имен переменных, связанных с участками памяти, где хранятся данные.

Переменные описываются до основного кода программы. Здесь указываются имена переменных и тип хранимых в них данных.

В языке программирования Паскаль достаточно много типов данных. Кроме того, сам пользователь может определять свои типы.

Тип переменной определяет, какие данные можно хранить в связанной с ней ячейке памяти.

Переменные типа **integer** могут быть связаны только с целыми значениями обычно в диапазоне от -32768 до 32767. В Pascal есть другие целочисленные типы (byte, longint).

Переменные типа **real** хранят вещественные (дробные) числа.

Переменная **булевского** (логического) типа (boolean) может принимать только два значения - **true** (1, правда) или **false** (0, ложь).

**Символьный тип (char)** может принимать значения из определенной упорядоченной последовательности символов.

**Интервальный тип** определяется пользователем и формируется только из порядковых типов. Представляет собой подмножество значений в конкретном диапазоне.

Можно создать собственный тип данных простым перечислением значений, которые может принимать переменная данного типа. Это так называемый **перечисляемый тип данных**.

Все вышеописанное – это простые типы данных. Но бывают и сложные, структурированные, которые базируются на простых типах.

**Массив** – это структура, занимающая в памяти единую область и состоящая из фиксированного числа компонентов одного типа.

**Строки** представляет собой последовательность символов. Причем количество этих символов не может быть больше 255 включительно. Такое ограничение является характерной чертой Pascal.

**Запись** – это структура, состоящая из фиксированного числа компонент, называемых полями. В разных полях записи данные могут иметь разный тип.

**Множества** представляют собой совокупность любого числа элементов, но одного и того же перечисляемого типа.

**Файлы** для Pascal представляют собой последовательности однотипных данных, которые хранятся на устройствах внешней памяти (например, жестком диске).

Понятие такого типа данных как **указатель** связано с динамическим хранением данных в памяти компьютера. Часто использование динамических типов данных является более эффективным в программировании, чем статических.

# Целые типы

В языке Паскаль определено пять целых типов.

**Таблица. Целые типы Pascal**

|  |  |  |
| --- | --- | --- |
| **Тип** | **Диапазон допустимых значений** | **Отводимая память, в байтах** |
| **shortint** | -128…127 | 1 |
| **integer** | -32 768…32 767 | 2 |
| **longint** | -2 147 483 648…2 147 483 647 | 4 |
| **byte** | 0…255 | 1 |
| **word** | 0…65 535 | 2 |

Переменные целого типа могут принимать только целые значения. Такие переменные в программе описываются следующим образом:

a, b, c: **integer**;

Здесь a, b, c… - имена переменных, **integer** – тип переменных. Транслятор, встретив такое описание переменных a, b, c, запоминает, что эти переменные могут принимать только целые значения и формирует соответственно этому команды программы.

**Таблица. Операции над целыми типами, дающие в результате значение целого типа**

|  |  |
| --- | --- |
| **Знак операции** | **Операция** |
| **+** | Сложение |
| **-** | Вычитание |
| **\*** | Умножение |
| **div** | Целочисленное деление (остаток отбрасывается). Деление без округления (целая часть частного). |
| **mod** | Деление по модулю (выделение остатка от деления). Остаток от деления: a mod b = a – ((a div b) \* b). |

Операции над операндами целого типа выполняются правильно только при условии, что результат и каждый операнд не меньше минимального (крайнего левого) и не больше максимального (крайнего правого) значений диапазона. Например, в Паскале существует константа **maxint**, в которой содержится максимально допустимое значение для типа **integer**. Тогда при выполнении операций в программе должны соблюдаться следующие условия:  
(a операция b) <= maxint,  
a <= maxint, b <= maxint.

Над целыми типами, как и многими другими, допустимы операции отношения (сравнения). Результат таких операций относится к типу boolean и может принимать одно из двух значений – либо **true** (истина), либо **false** (ложь).

**Таблица. Операции отношения**

|  |  |
| --- | --- |
| **Знак операции** | **Операция** |
| **=** | Равно |
| **<>** | Не равно |
| **>=** | Больше или равно |
| **>** | Больше |
| **<=** | Меньше или равно |
| **<** | Меньше |

Целые типы могут приниматься в качестве фактических параметров рядом стандартных функций языка программирования Pascal.

**Таблица. Стандартные функции Pascal, применимые к аргументам целых типов**

|  |  |  |
| --- | --- | --- |
| **Функция** | **Тип результата** | **Результат выполнения** |
| **abs(x)** | Целый | Модуль x (абсолютная величина x) |
| **sqr(x)** | Целый | Квадрат x |
| **succ(x)** | Целый | Следующее значение x (x+1) |
| **pred(x)** | Целый | Предыдущее значение x (x-1) |
| **random(x)** | Целый | Случайное целое число из интервала 0..x-1. |
| **sin(x)** | Действительный | Синус x (угол в радианах) |
| **cos(x)** | Действительный | Косинус x (угол в радианах) |
| **arctan(x)** | Действительный | Арктангенс x (угол в радианах) |
| **ln(x)** | Действительный | Натуральный логарифм x |
| **exp(x)** | Действительный | Экспонента x |
| **sqrt(x)** | Действительный | Квадратный корень из x |
| **odd(x)** | Логический | Значение **true**, если x – нечетное число; **false** – если четное. |

Функция **random** возвращает равномерно распределенное случайное целое число, если ей передан целый аргумент. При повторном запуске программы она возвращает те же значения. Во избежание этого следует в начале программы вызвать процедуру без параметров **randomize**.

Процедуры **inc** и **dec** могут иметь по одному или по два параметра целого типа. Если параметров два, то значение первого увеличивается (для **inc**) или уменьшается (для **dec**) на величину, равную значению второго параметра. Например, inc(x,2) равнозначно x+2. Если параметр один, то его значение увеличивается (для **inc**) или уменьшается (для **dec**) на единицу. Например, dec(x) равнозначно x-1. (ТАКИХ ПРОЦЕДУР ВО FREEPASCAL СКОРЕЕ ВСЕГО НЕТ.)

Следующие функции принимают в качестве аргументов значения вещественного типа, а возвращают значения целого типа:  
**trunc(x)** – отбрасывание десятичных знаков после точки;  
**round(x)** – округление до целого.

# Вещественные типы

В языке Паскаль существует несколько типов для представления действительный чисел. Однако чаще всего для их представления используется тип **Real**.

**Таблица. Вещественные типы в Pascal**

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип** | **Диапазон** | **Число цифр** | **Память, байт** |
| Real | 2.9e-39 … 1.7e38 | 11-12 | 6 |
| Single | 1.5e-45 … 3.4e38 | 7-8 | 4 |
| Double | 5.0e-324 ...1.7e308 | 15-16 | 8 |
| Extended | 3.4e-4932 … 1.1e493 | 19-20 | 10 |
| Comp | -9.2e63 … (9.2e63)-1 | 19-20 | 8 |

Число цифр определяет точность, с которой будет храниться вещественное число. Например, для **Real** разрядность мантиссы может составлять не более восьми десятичных знаков. Тип **Comp** содержит только целые значения, которые представляются в вычислениях как вещественные.

Над действительными числами выполнимы операции сложения (+), вычитания (-), умножения (\*) и деления (/). Результатом этих операций является также действительное число. Даже если хотя бы один из операндов вещественный, то результат этих операций также будет вещественным.

Операция деления (/) дает вещественный результат и в случае двух целых операндов. Например, 6 / 2 = 3.0.

Для действительных чисел допустимы такие же операции отношения (сравнения), что и для целых чисел.

Стандартная функция **abs(x)** – модуль x – от целого аргумента дает целый результат, а от вещественного – вещественный, как и **sqr(x)** – квадрат x.

#### Функции

**sin(x)** – синус x (x в радианах),  
**cos(x)** – косинус x (x в радианах),  
**ln(x)** – натуральный логарифм x,  
**exp(x)** – экспонента x,  
**sqrt(x)** – корень квадратный из x,  
**arctan(x)** – арктангенс x  
дают вещественный результат, как для вещественного, так и для целого аргумента.

Функция **int** возвращает в виде действительного значения целую часть аргумента, frac возвращает дробную часть аргумента.

Функции **trunc** и **round** возвращают результат целого типа. Первая отсекает дробную часть от аргумента, а вторая выполняет округление до ближайшего целого.

Функция **random** без аргументов возвращает равномерно распределенное случайное число от 0 до 1.

Не имеющая аргументов функция **pi** возвращает число Пифагора.

Нельзя использовать переменные и константы вещественного типа:

* в функциях **pred, succ, ord**;
* в качестве индексов массивов;
* в операторах передачи управления в качестве меток.

# Булевский тип (Boolean)

Переменная булевского типа принимает значения true (истина) или false (ложь). Эти величины упорядочены следующим образом:  
false < true

Операции and, or, not (применяемые к булевским операндам) дают булевские значения.

**Операция and (логическое умножение, пересечение, операция И)**  
Выражение a and b дает значение true, только в том случае, если a и b имеют значение true. Во всех остальных случаях значения выражения a and b – false.

**true** and **true** = **true**

**true** and **false** = **false**

**false** and **false** = **false**

**Операция or (логическое сложение, объединение, операция ИЛИ)**  
Выражение a or b дает значение false, только в том случае, если a и b имеют значение false. Во всех остальных случаях результат – true.

**true** **or** **true** = **true**

**true** **or** **false** = **true**

**false** **or** **false** = **false**

**Операция not (отрицание, операция НЕ)**  
Выражение not a имеет значение, противоположное значению a.

**not** **true** = **false**

**not** **false** = **true**

**Стандартные булевские функции**  
odd(x) = true, если x нечетный (x целый);  
eoln(x) = true, если встретился конец строки текстового файла x;  
eof(x) = true, если встретился конец файла x.  
В остальных случаях эти функции принимают значение false.

# Символьный тип (Char)

Переменная типа char может принимать значения из определенной упорядоченной последовательности символов. Переменная этого типа занимает 1 байт и принимает одно из 256 значений кода ASCII (американский стандартный код для обмена информацией). Символы упорядочены в соответствии с их кодом, поэтому к данным символьного типа применимы операции отношения.

В программе вместо символа можно использовать его код, состоящий из # и номера кодируемого символа (например, #51). Обычно символы, имеющие экранное представление, записывают в явном виде, заключив в апострофы (например, 'A', 'b', '\*').  
Две стандартные функции позволяют поставить в соответствие данную последовательность символов множеству целых неотрицательных чисел (порядковым номерам символов последовательности).

Эти функции называются функциями преобразования:

ord(ch) – выдает номер символа (нумерация с нуля),  
chr(i) – выдает i-ый символ из таблицы символов.

Пример. ord('H') выдает номер символа Н в последовательности всех символов, используемых транслятором. chr(15) выдает 15-ый символ этой последовательности.

Кроме того, для символьных переменных применяются такие функции:

pred(ch) – возвращает предыдущий символ;  
succ(ch) – возвращает следующий символ;  
upcase(ch) – преобразует строчную букву в заглавную. Обрабатывает буквы только латинского алфавита.

Также можно использовать процедуры inc и dec.

# Тип pchar

В язык программирования Pascal введен тип pchar, который описывает так называемые длинные (или ASCIIZ) строки. В переменных типа string длина строки записывается в нулевом байте, а т.к. в 1 байт нельзя записать значение, превосходящее 255, то и длина переменной типа string не может превосходить 255 символов. Переменная типа pchar – это строка, которая может иметь длину до 65534 символов. Длина таких строк не указывается явно: строка начинается с первого символа и замыкается символом #0. Поскольку тип pchar определяется как указатель на символ, ASCIIZ-строки создаются в динамической памяти. С типом pchar совместим любой символьный массив, левая граница которого равна нулю. ASCIIZ-строки индексируются с отсчетом значений индекса от 0. Строковые константы также могут быть записаны в переменные типа pchar. Для вывода ASCIIZ-строк применимы операторы write и writeln.

Пример использования ASCIIZ-строк:

**const**

ch\_arr: **array**[0..20] **of** **char** = 'Hello World!';

**var**

p\_str: pchar;

**begin**

p\_str := ch\_arr;

writeln(p\_str);

p\_str[4] := #0;

writeln(ch\_arr);

readln

**end**.

В программе символьный массив объявлен в виде типизированной константы. Поскольку под него отведены 21 байт, его содержимое нужно было бы задавать строкой из 21 символа; однако в языке программирования Паскаль строковая константа, длина которой меньше количества элементов символьного массива, также записывается в символьный массив с автоматическим заполнением конца массива символом #0. В приведенной программе осуществляется коррекция переменной p\_str, что меняет содержимое массива ch\_arr, т.к. переменная p\_str после присваивания p\_str := ch\_arr просто ссылается на этот массив и оперирует областью памяти, которую он занимает. Поэтому выражение p\_str[4] := #0 изменяет длину строки ch\_arr до четырех символов.

ASCIIZ-строка может быть объявлена типизированной константой, но ее длина в этом случае не может превосходить 255 символов. Например:

**const**

pch: pchar = 'My name is ';

К данным типа pchar с помощью операции сложения можно добавлять целые значения, что соответствует смещению начала ASCIIZ-строки на указанное число позиций. Аналогично выполняется операция вычитания. Введена также операция вычитания между двумя ASCIIZ-строками, результатом которой является целочисленное значение, соответствующее смещению в памяти начала одной ASCIIZ-строки по отношению к началу другой.

**const**

pch1: pchar = 'hello world';

pch2: pchar = 'привет мир';

**var**

p\_str: pchar;

n: **longint**;

**begin**

p\_str := pch1 + 6;

writeln(p\_str); *// world*

writeln(pch2 + 7); *// мир*

p\_str := 'This is world';

writeln(p\_str);

n := pch2 - pch1;

writeln(n);

readln

**end**.

Значение n показывает взаимное смещение в памяти двух ASCIIZ-строк.  
Основные средства по работе с ASCIIZ-строками сосредоточены в модуле Strings.

# Условные операторы

Бывает, что в процессе выполнения программы требуется реализовать разный набор команд в зависимости от произошедших до этого событий. В языках программирования это достигается с помощью специальных конструкций – условных операторов.

Чаще всего в качестве условного оператора в языках программирования используется конструкция **if-else** или ее сокращенный вариант **if**. Также существует оператор выбора **case**, который имеет более специфичное применение.

## Оператор if-else

Когда выполнение основной ветки программы доходит до условного оператора if-else, то в зависимости от результата логического выражения в его заголовке выполняются разные блоки кода. Если логическое выражение вернуло true, то выполняется один блок (в Паскале начинается со слова then), если false – то другой (начинается со слова else). После выполнения одного из вложенных блоков кода, ход программы возвращается в основную ветку. Другой вложенный блок не выполняется.

Например, программа должна определять, ввел пользователь четное или нечетное число, и выводить на экран сообщение. Тогда программный код на языке Pascal может быть таким:

**var** n: **integer**;

**begin**

write ('Введите целое число: ');

readln (n);

**if** n **mod** 2 = 0 **then**

write ('Оно четное.')

**else**

write ('Оно нечетное.');

readln

**end**.

Бывают неполные формы условных операторов. В таком случае вложенный в if блок кода выполняется только в случая true логическом выражении заголовка. В случае false выполнение программы сразу передается в основной блок. Понятно, что ветка else в таком случае отсутствует.

В примере ниже, если переменная имеет значение меньше нуля, то ее значение изменяется (находится модуль числа). Если же значение переменной изначально больше нуля, то блок кода при операторе if вообще не выполняется, т.к. не соблюдено условие (n<0).

В качестве условия может стоять любое выражение, результатом вычисления которого является одно из булевых значений — true или false.

Непосредственно после then может стоять только один оператор. При необходимости выполнения нескольких операторов они должны быть заключены в операторные скобки begin-end. Пример программы, которая меняет значения переменных местами, только если эти значения различны. Блок if содержит четыре выражения, поэтому заключен в begin-end.

**var**

a,b,c: **integer**;

**begin**

write('a=');

readln(a);

write('b=');

readln(b);

**if** a <> b **then** **begin**

c := a;

a := b;

b := c;

writeln('a=',a,'; b=',b);

**end**

**else**

writeln('Введены одинаковые числа');

readln;

**end**.

Допустимо вложение одного оператора if (или if-else) в другой. При этом следует соблюдать осторожность, т.е. бывает трудно определить какому if (внешнему или внутреннему) принадлежит ветка else. Рекомендуют использовать вложенную конструкцию if, только в ветке else. К тому же в языке Паскаль действует следующее правило: каждому then соответствует ближайшее else, не задействованное при установлении соответствия с другим then. Глубина вложенности операторов if может быть сколь угодно большой, но разобраться в таком коде будет очень сложно.

### Оператор case (оператор выбора)

Кроме оператора if в языке программирования Паскаль предусмотрен так называемый переключатель case. Его можно трактовать как некий вопрос, имеющий большое число ответов (а не только два, как это имеет место в операторе if-else). Однако в отличие от if, case имеет ряд принципиальных ограничений. Его формат следующий:

**case** селектор **of**

значение1: оператор1;

значение2: оператор2;

значение3: оператор3;

...

**else** операторN

**end**;

В заголовке оператора case вместо логического выражения фигурирует переменная, которую называют селектором. До этого в программе ей присваивается какое-либо значение. Эта переменная может иметь только перечисляемый тип (например, она не может быть вещественного типа). По ходу выполнения оператора case, значение переменной-селектора сравнивается с различными, описанными в нем альтернативами (метками-значениями). Как только совпадение будет найдено, то выполняется блок кода при данной метке и происходит выход в основную ветку программы. Значения-метки являются константами, которые может принимать селектор. Их тип и тип селектора должны быть совместимы по присваиванию.

Если совпадений не будет, то выполняется блок else. Если блок else отсутствует (он является не обязательным), то никакой блок кода в операторе case не выполняется.

**var** n: **integer**;

**begin**

write ('Введите класс школы: ');

readln (n);

**case** n **of**

1..4: writeln ('Младшие классы.');

5..8: writeln ('Средняя школа.');

9,11: writeln ('Старшие классы. Выпускной.');

10: writeln ('Старшие классы.');

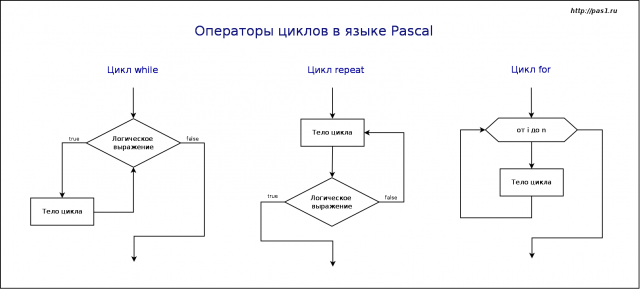
**else** writeln ('Error')

**end**;

readln

**end**.

На использование оператора выбора накладываются следующие ограничения:

* селектор должен иметь какой-либо порядковый тип;
* каждая альтернатива должна быть константой, диапазоном, списком диапазонов, но не переменной или выражением.
* Следует быть внимательными при использовании вложенных операторов if. Предпочтительнее пользоваться схемой else-if (т.е. вкладывать во внешнюю ветку else), а не then-if (т.е не вкладывать во внешнюю ветку if). Так как в последнем случае придется следить за тем, к какой ветке относится соответствующее else. Пренебрежение этим правилом приводит к нагромождению закрывающих else.
* Сравните две программы ниже. Они делают одно и то же. Однако предпочтительной является первая.
* **else-if**
* **var**
* a: **integer**;
* **begin**
* write('Введите целое число: ');
* readln(a);
* **if** a = 0 **then**
* writeln('zero')
* **else**
* **if** a = 1 **then**
* writeln('one')
* **else**
* **if** a = 2 **then**
* writeln('two')
* **else**
* writeln('unknown');
* readln
* **end**.
* **then-if**
* **var**
* a: **integer**;
* **begin**
* write('Введите целое число: ');
* readln(a);
* **if** a <> 0 **then**
* **if** a <> 1 **then**
* **if** a <> 2 **then**
* writeln('unknown')
* **else**
* writeln('two')
* **else**
* writeln('one')
* **else**
* writeln('zero');
* readln
* **end**.
* При использовании второго варианта надо помнить следующее общее правило: каждый else относится к ближайшему предшествующему if, еще не имеющему парного else.
* При вложении в ветвь else (первый вариант) конструкция получается логически более понятной.
* При решении задач может возникнуть необходимость повторить одни и те же действия несколько или множество раз. В программировании блоки кода, которые требуется повторять не единожды, оборачиваются в специальные конструкции – циклы. У циклов выделяют заголовок и тело. Заголовок определяет, до каких пор или сколько раз тело цикла будет выполняться. Тело содержит выражения, которые выполняются, если в заголовке цикла выражение вернуло логическую истину (True, не ноль). После того как достигнута последняя инструкция тела, поток выполнения снова возвращается к заголовку цикла. Снова проверяется условие выполнения цикла. В зависимости от результата тело цикла либо повторяется, либо поток выполнения переходит к следующему выражению после всего цикла.
* В языке программирования Паскаль существует три вида циклических конструкций.
* 

## Цикл for

* Часто цикл for называют циклом со счетчиком. Этот цикл используется, когда число повторений не связано с тем, что происходит в теле цикла. Т.е. количество повторений может быть вычислено заранее (хотя оно не вычисляется).
* В заголовке цикла указываются два значения. Первое значение присваивается так называемой переменной-счетчику, от этого значения начинается отсчет количества итераций (повторений). Отсчет идет всегда с шагом равным единице. Второе значение указывает, при каком значении счетчика цикл должен остановиться. Другими словами, количество итераций цикла определяется разностью между вторым и первым значением плюс единица. В Pascal тело цикла не должно содержать выражений, изменяющих счетчик.
* Цикл for существует в двух формах:
* **for** счетчик:=значение **to** конечное\_значение **do**
* тело\_цикла;
* **for** счетчик:=значение **downto** конечное\_значение **do**
* тело\_цикла;
* Счетчик – это переменная любого из перечисляемых типов (целого, булевого, символьного, диапазонного, перечисления). Начальные и конечные значения могут быть представлены не только значениями, но и выражениями, возвращающими совместимые с типом счетчика типы данных. Если между начальным и конечным выражением указано служебное слово **to**, то на каждом шаге цикла значение параметра будет увеличиваться на единицу. Если же указано **downto**, то значение параметра будет уменьшаться на единицу.
* Количество итераций цикла **for** известно именно до его выполнения, но не до выполнения всей программы. Так в примере ниже, количество выполнений цикла определяется пользователем. Значение присваивается переменной, а затем используется в заголовке цикла. Но когда оно используется, циклу уже точно известно, сколько раз надо выполниться.
* **var**
* i, n: **integer**;
* **begin**
* write ('Количество знаков: ');
* readln (n);
* **for** i := 1 **to** n **do**
* write ('(\*) ');
* readln
* **end**.

## Цикл while

* Цикл **while** является циклом с предусловием. В заголовке цикла находится логическое выражение. Если оно возвращает **true**, то тело цикла выполняется, если **false** – то нет.
* Когда тело цикла было выполнено, то ход программы снова возвращается в заголовок цикла. Условие выполнения тела снова проверяется (находится значение логического выражения). Тело цикла выполнится столько раз, сколько раз логическое выражение вернет **true**. Поэтому очень важно в теле цикла предусмотреть изменение переменной, фигурирующей в заголовке цикла, таким образом, чтобы когда-нибудь обязательно наступала ситуация **false**. Иначе произойдет так называемое **зацикливание**, одна из самых неприятных ошибок в программировании.
* **var**
* i, n: **integer**;
* **begin**
* write ('Количество знаков: ');
* readln (n);
* i := 1;
* **while** i <= n **do** **begin**
* write ('(\*) ');
* i := i + 1
* **end**;
* readln
* **end**.

## Цикл repeat

* Цикл **while** может не выполниться ни разу, если логическое выражение в заголовке сразу вернуло **false**. Однако такая ситуация не всегда может быть приемлемой. Бывает, что тело цикла должно выполниться хотя бы один раз, не зависимо оттого, что вернет логическое выражение. В таком случае используется цикл repeat – цикл с постусловием.
* В цикле **repeat** логическое выражение стоит после тела цикла. Причем, в отличие от цикла **while**, здесь всё наоборот: в случае **true** происходит выход из цикла, в случае **false** – его повторение.
* **var**
* i, n: **integer**;
* **begin**
* write ('Количество знаков: ');
* readln (n);
* i := 1;
* **repeat**
* write ('(\*) ');
* i := i + 1
* **until** i > n;
* readln
* **end**.
* В примере, даже если n будет равно 0, одна звездочка все равно будет напечатана.