**Структура Паскаль-программы.**

Программа на языке Паскаль состоит из следующих частей:

Заголовок

 Подключение модулей

 Раздел описаний

Begin

 Раздел операторов

End.

*Заголовок* – необязателен. Начинается с ключевого слова program.

Program <имя>;

*Подключение модулей* – ключевое слово uses.

Uses <имя1>, <имя2>, …, <имя n>;

*Раздел описаний* – состоит из нескольких подразделов.

Подраздел Label – описание меток.

Label <имя>, <число>;

Const – описание констант.

Type – описание типов.

Var – описание переменных.

Function – описание функций.

Procedure – описание процедур.

*Раздел операторов* – содержит действия, предписанные алгоритмом.

Раздел описаний совместно с разделом операторов называется блоком.

Program My\_First\_program;

Var n1, n2: integer;

 x: real;

Begin

Write(‘n1=’); readln(n1);

Write(‘n2=’); readln(n2);

x := n1 / n2;

Writeln(‘n1/n2= ‘, x);

End.

Процедура – некоторая последовательность операторов языка Паскаль, к которой можно обратиться по имени.

**Системы типов языка. Стандартные типы.**

В языке Паскаль любая константа, переменная, значение выражения характеризуется своим типом. Тип данных определяет множество значений, которые может принимать переменная, а также множество операций, допустимых над данной переменной. Кроме того, тип данных определяет внутреннее представление переменной. Все типы языка Паскаль можно разделить на две категории: стандартные и определяемые программистом.

*Системы типов языка Паскаль.*

См. рис.1.

*Логический (булевский) тип.*

Boolean – идентификатор логического типа. Может принимать два значения: False (0) или True (1). Занимает 1 байт.

ByteBool – 1 байт.

WordBool – 2 байта.

LongBool – 4 байта.

Операции отношения: >, <, >=, <=, =, <>. Результаты этих операций имеют логический тип.

В Паскале существуют четыре логические операции: and (логическое И), or (логическое ИЛИ), xor (исключающее ИЛИ, сложение по модулю 2, отрицание равнозначностей), not (отрицание).

(x > 1) and (x < 2) – пример логических операций.

*Целые типы.*

Эта группа типов обозначает множество целых чисел в различных диапазонах.

Integer – длина 2 байта. От -32768 до 32767.

Byte – длина 1 байт. От 0 до 255.

Word – длина 2 байта. От 0 до 65535.

Shortint – длина 1 байт. От -128 до 127.

Longint – длина 4 байта. От -2^31 до 2^31-1.

*Операции:* +, -, \*, /, div, mod.

Все операции, кроме деления, для целых данных дают результат целого типа. Деление всегда даёт результат вещественного типа.

*Стандартные процедуры и функции, применимые к данным целого типа.*

*Odd(x): boolean* – функция возвращает True, если число нечётное, и False – в противном случае.

Порядковыми называются типы, значения которых можно перечислить по порядку. Boolean тоже относится к порядковому типу.

*Pred(x)* – предыдущее значение. Тип результата – целый.

*Succ(x)* – следующее значение. Тип результата – целый.

*Inc(x)* – функция увеличения на единицу.

*Inc(x, n)* – процедура увеличивает значение первого параметра на n.

*Dec(x)* – функция уменьшения на единицу.

*Dec(x, n)* – процедура уменьшает первый параметр на n.

*Вещественный тип.*

Эта группа типов обозначает множество вещественных значений в различных диапазонах.

Real – идентификатор вещественного типа. Длина памяти – 6 байт. Диапазон: от 2,9\*10^-39 до 1,7\*10^38. В памяти машины величины типа Real представляются так: знак порядка, порядок, мантисса.

Single – длина 4 байта. От 1,1\*10^-45 до 3,4\*10^38. Представляется так: знак порядка, мантисса, порядок.

Double – длина 8 байт. От 5,0\*10^-324 до 1,7\*10^308.

Extended – длина 10 байт. От 3,4\*10^-4932 до 1,1\*10^4932.

Comp – длина 8 байт. От -2^63 до 2^63-1.

*Операции:* Арифметические: +, -, \*, /. Результат – вещественного типа.

Операции отношения.

*Стандартные процедуры и функции, применимые к данным вещественного типа.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Тип аргумента** | **Тип результата** | **Вычисление** |
| *Abs(x)* | целый или вещ. | тип аргумента | Модуль x |
| *Arctan(x)* | целый или вещ. | вещественный | Арктангенс x |
| *Cos(x)* | целый или вещ. | вещественный | Косинус x |
| *Exp(x)* | целый или вещ. | вещественный | E в степени x |
| *Ln(x)* | целый или вещ. | вещественный | Натуральный лог. x |
| *Sin(x)* | целый или вещ. | вещественный | Синус x |
| *Sqr(x)* | целый или вещ. | тип аргумента | Квадрат x |
| *Sqrt(x)* | целый или вещ. | вещественный | Корень из x |
| *Frac(x)* | вещественный | вещественный | Дробная часть аргум. |
| *Int(x)* | вещественный | вещественный | Целая часть аргумента |
| *Pi* | - | вещественный | Число Пи |
| *Round(x)* | вещественный | целый | Округление до целого |
| *Trunc(x)* | вещественный | целый | Целая часть аргумента |

*Символьный тип.*

Char – идентификатор символьного типа. Символы с кодом от 0 до 127 соответствуют ASCII. Символы с кодом от 128 до 255 не ограничены стандартами. Длина типа Char – 1 байт. Значения – коды символов от 0 до 255. Символы в тексте программы записываются в одиночных апострофах.

*Операции:* операции отношения.

Символьные типы относятся к порядковому типу.

*Стандартные процедуры и функции, применимые к данным символьного типа.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Тип аргумента** | **Тип результата** | **Результат** |
| *Ord(x)* | символьный | целый | Код символа |
| *Chr(x)* | byte | символьный | По коду - символ |
| *Pred(x)* | символьный | символьный | Предыдущий символ |
| *Succ(x)* | символьный | символьный | Следующий символ |
| *Upcase(x)* | символьный | символьный | Перевод в верхний регистр |

Тип называется скалярным, если он не раскладывается на составные части.

**Раздел определения констант.**

*Константы* – это данные, которые в процессе выполнения программы нельзя менять.

Все константы делятся на литеральные и именованные.

*Литеральные константы* – это все числа, символы и строки символов в тексте программы.

*Именованные константы* – фиксированное значение, которому даётся имя.

*Сonst* – идентификатор раздела констант.

const <имя> = <значение>;

Тип константы определяется её изображением (способом её записи). Константы бывают целые, вещественные, символьные и строковые.

*Целые константы* – задаются десятеричным или шестнадцатеричным числом (шестнадцатеричные – со знаком $ в начале).

*Вещественные константы* могут создаваться форматом с фиксированной точкой или с плавающей точкой.

*Символьные константы* – одиночный символ с апострофами или строка с апострофами.

**Раздел описания меток.**

Метка в Паскале – это десятичное число или правильный идентификатор. Назначение метки – помечать оператор.

*Label* – идентификатор раздела меток.

**Раздел описания переменных.**

*Var* –идентификатор раздела переменных.

var <имя>: <тип>;

x: real;

x, y, z, a: integer;

В Паскале введено понятие *типизированных констант*. По сути, это – переменная с заданным начальным значением. Типизированные константы описываются в разделе const следующим образом: <имя>: <тип> = значение;

x: integer = 10;

Значение типизированной константы можно менять в программе.

При описании переменных в одном блоке не может быть одинаковых имён. Для переменных, описанных в блоке, выделяется объём памяти размером 64 кБ.

**Определение типов.**

*Type* – идентификатор раздела типов.

type <имя> = <описание типа>;

letters = ‘A’..’Z’;

Тип, который создаётся ограничением диапазона возможных значений какого-либо порядкового типа, называется ограниченным типом. Тип, на который накладываются ограничения, называется базовым типом.

interval = 0..100;

Базовые типы: все целые типы и тип Char.

*Перечислимые типы* – задаются перечислением тех значений, которые он может получать. Каждое значение именуется идентификатором и располагается в списке, обрамлённом круглыми скобками.

Пример: Color = (Red, Yellow, Green);

За каждым именем из этого списка закреплено определённое значение. Самим задать значение этих констант в Паскале нельзя.

var z: letters;

 index: interval;

 my\_letter: ‘a’..’z’;

**Выражения.**

Основными объектами, из которых состоит исполняемая часть программы, являются константы, переменные и обращения к функциям. Каждый из этих объектов характеризуется своим значением и принадлежит к какому-либо типу данных.

*Выражение* – это формальное правило для вычисления нового значения. Выражением может быть одиночный объект (константа, переменная, обращение к функции) или несколько операндов, разделённых знаками операций.

*Операции* – определяют действие по вычислению новых значений, исходя из значения операндов.

Все операции Паскаля можно разделить на 4 группы по приоритетам: 1) унарные операции (логическое not, операция взятия адреса); 2) мультипликативные операции (умножение, деление, деление нацело, взятие остатка от целочисленного деления, логическое and, операции сдвига – shl и shr); 3) аддитивные операции (сложение, вычитание, логическое or, логическое xor); 4) операции отношения (проверка на равенство или неравенство, операции сравнения, операция проверки принадлежности к множеству – in).

Для логических выражений характерно то, что их значение может стать известным ещё до конца вычисления всего выражения. Такой способ называется усечённым способом вычисления логических выражений.

Options – Compiler – Boolean Evaluation – выбор типа вычисления логических выражений.

**Операторы.**

Это синтаксические конструкции, предназначенные как для записи действий по преобразованию данных, так и для задания порядка выполнения других действий. Язык Паскаль содержит всего 11 операторов: 1) оператор присваивания; 2) оператор процедуры; 3) оператор перехода; 4) составной оператор; 5) условный оператор; 6) оператор варианта; 7, 8, 9) операторы цикла; 10) оператор над записями; 11) пустой оператор. Кроме того, существуют операторы машинного кода и ассемблера.

*Оператор присваивания.*

<идентификатор> := <выражение>;

Выражение должно давать результат такого же типа, как и переменная. Присваивание допустимо для всех типов, кроме файловых.

*Оператор процедуры.*

Другое название – *вызов процедуры*.

<имя процедуры>(<параметры>);

Оператор процедуры вызывает выполнение операторов, которые находятся в теле процедуры.

*Оператор перехода.*

Этот оператор вызывает передачу управления оператору, которому предшествует метка.

goto <метка>;

Все метки, используемые в блоке, должны быть описаны, причём не более одного раза.

Метка, указанная в операторе перехода, должна помечать оператор, находящийся в том же блоке.

Переход внутрь структурного оператора может вызвать непредсказуемые эффекты, хотя компилятор не выдаст сообщение об ошибке.

*Составной оператор.*

Это последовательность операторов программы, заключённая в операторные скобки begin-end.

*Условный оператор.*

С помощью условного оператора реализуется структура «развилка». В Паскале два вида условного оператора: с ветвью else и без ветви else.

if <выражение> then <оператор1> else <оператор2>;

if <выражение> then <оператор>;

if <выражение1> then if <выражение2> then <оператор1> else <оператор2>;

else всегда относится к ближайшему if.

var a, b, c, d: integer;

begin

a := 1; b := 2; c := 3; d := 4;

if a > b then if c < d then if c < 0 then c := 0 else a := b;

Значение a не изменится.

If a > b then if c < d then if c < 0 then c := 0 else else else a := b;

Значение a станет равным 2.

Рассмотрим пример программы, которая вводит произвольное десятичное число от 0 до 15 и выводит его в шестнадцатеричном формате.

program Hex1;

var ch: char;

 n: integer;

begin

write(‘n = ‘); readln(n);

if (n >= 0) and (n <= 15) then

begin

if n < 10 then ch := chr(ord(‘0’) + n)

else ch := chr(ord(‘A’) + n - 10);

writeln(‘n = ‘, ch);

end

else writeln(‘Ошибка.’);

end.

*Оператор варианта.*

Реализует структуру «выбор», то есть возможность выбирать из нескольких альтернатив. Оператор работает следующим образом: сначала вычисляется выражение. Это выражение иногда называется селектором или ключом выбора. Выражение должно давать результат любого порядкового типа. После этого вычисленное значение сравнивается с константами выбора. При совпадении с какой-либо константой выполняется оператор, помеченный этой константой. На этом выполнение оператора варианта заканчивается. Если значение не совпадает ни с одной из ветвей, то выполняется оператор else, если он есть. Если ветвей нет, то оператор не производит никаких действий.

case <выражение> of

<константа выбора1>: <оператор1>

<константа выбора2>: <оператор2>

…

<константа выбора n>: <оператор n>

[else <оператор>]

end;

Пример.

var color: (Green, Yellow, Red);

begin

color :=

case color of

Red: x := y + 2;

Yellow: x := y – 2;

Green: x := y;

end;

Кроме одиночных констант в альтернативах могут быть заданы несколько констант для каждой альтернативы (в этом случае они разделяются запятыми) или диапазоны значений. Пример:

case switch of

1..3: Proc1;

4, 10..20: Proc2;

5, 6: Proc3;

else Proc4;

end;

var ch: char;

begin

readln(ch);

case ch of

‘n’, ‘N’: writeln(‘Нет’);

‘y’, ‘Y’: writeln(‘Да’);

end

end.

При использовании оператора варианта должны выполняться следующие правила: 1) Значение ключа выбора должно принадлежать порядковому типу. 2) Все константы альтернатив должны быть типа совместимого с типом ключа выбора. 3) Все константы в альтернативах должны быть уникальны в пределах оператора варианта. Диапазоны не должны пересекаться. 4) Синтаксис языка предусматривает только один оператор в каждой альтернативе. Если необходимо выполнить несколько операторов, следует образовывать составной оператор.

*Операторы цикла.*

В Паскале существует три вида циклов: цикл с предусловием, цикл с постусловием и цикл с параметром.

*Оператор цикла с предусловием.*

while <выражение> do <оператор>

В качестве примера использования while рассмотрим программу, выводящую минимальное, не равное нулю вещественное число эпсилон, которое после добавления его к единице ещё даёт результат, отличный от единице.

var epsilon: real;

begin

epsilon := 1;

while epsilon / 2 + 1 > 1 do epsilon := epsilon / 2;

writeln(‘Машинный epsilon = ‘, epsilon);

end.

*Оператор цикла с постусловием.*

repeat

<операторы>

until <выражение>

Пример: программа вводит символ и печатает его код, пока не будет нажата клавиша Enter.

begin

repeat

readln(ch);

writeln(ch, ‘=’, ord(ch));

until ord(ch) = cr;

end.

*Оператор цикла с параметром.*

Используется тогда, когда заранее известно число повторений цикла. Этот оператор предусматривает повторное выполнение тела цикла с одновременным изменением по правилу арифметической прогрессии параметра цикла.

for <имя> := <выражение1> to <выражение2> do <оператор>

for <имя> := <выражение1> downto <выражение2> do <оператор>

Выполнение оператора цикла с параметром:

1) Вычисляется выражение 1, определяющее начальное значение параметра.

2) Параметру цикла присваивается это значение.

3) Параметр цикла сравнивается с конечным значением. Если он больше конечного значения, то цикл заканчивается.

4) Выполняется тело цикла.

5) Значение параметра изменяется на 1. Переход к шагу 3).

Пример: программа, находящая сумму целых чисел от 1 до n.

var i, n, s: integer;

begin

write(‘N = ‘); readln(n);

s := 0;

for i := 1 to n do s := s + i;

writeln(‘Сумма = ‘, s);

end.

На параметр цикла накладываются следующие ограничения:

1) В качестве параметра цикла должна использоваться простая переменная, описанная в текущем блоке, и эта переменная должна принадлежать к порядковым типам.

2) Типы выражений для вычисления начального и конечного значений параметра цикла должны быть совместимыми с типом параметра цикла.

3) В теле цикла запрещается явное изменение значений параметра цикла.

*Пустой оператор.*

Никак не обозначается и никак не выглядит. Может использоваться, например, перед операторной скобкой end.

Пример:

repeat until Keypressed;

**Составные типы.**

В Паскале есть следующие составные, или структурные, типы: регулярные типы (массивы), комбинированные типы (записи), строки, множества, файлы, объекты.

*Массивы.*

*Массив* – это конечная именованная совокупность элементов одного типа.

Массив является структурой с прямым доступом, к каждому элементу можно обратиться по его номеру. Этот номер называется индексом.

type <имя> = array[<тип индекса>] of <тип элемента>;

В качестве типа индекса можно задавать любой порядковый тип, кроме Longint. Элементами массива может быть любой тип, кроме файлового.

mas = array[1..10] of real;

color = array[byte] of mas;

m2 = array[char] of Boolean;

Размерность массива – количество индексов, заданных при описании. В Паскале оно неограниченно. Количество элементов в массиве фиксировано, то есть должно быть определено до трансляции программы.

Многомерные массивы располагаются в памяти так, что быстрее всего меняется самый правый индекс. Максимальный объём памяти под все переменные – 64 кБ.

*Строковые типы.*

Строки используются для хранения последовательностей символов. В Паскале существует три типа строк:

1) Стандартные. Определяются ключевым словом string.

2) Определяемые программистом на основе string.

3) Строки в динамической памяти.

var st: string;

st1: array[0..255] of char;

str: string[4];

const st7: string[15] = ‘Пример строки’;

Такая строка может содержать от 0 до 255 символов. Каждый символ в такой строке занимает 1 байт.

Под строку st отводится 256 байт памяти. При этом в нулевом элементе строки хранится длина строки в символьном виде.

begin

st := ‘Это строка’;

Для строк применима операция конкатенации, или склеивания. Обозначается эта операция символом «+».

var line: string;

begin

line := line + ‘стала длиннее’;

Для того, чтобы узнать длину строки: ord(line[0]) или length(<имя строки>).

Кроме операции склеивания над значениями строковых типов определены операции отношения. При выполнении этих операций действуют следующие правила:

1) Более короткая строка всегда меньше более длинной.

2) Если длины сравниваемых строк одинаковы, то строки сравниваются посимвольно с учётом внутренней кодировки символов.

Рассмотрим пример программы, которая должна сформировать строку из 26 заглавных символов от A до Z.

var st: string[26];

 i: integer;

begin

st := ‘A’;

for i := 1 to 26 do

st := st + chr(ord(‘A’) + i – 1);

writeln(st);

end.

Функции для работы со строками:

*Concat(s1, s2, …, sn)* – склеивание перечисленных строк.

*Copy(s: string, index: integer; count: integer): string* – функция копирует из строки s count символов, начиная с символа с номером index.

*Str(x: real; st: string)* – преобразует число x в строку st.

*Val(st, x, code)* – процедура преобразует строку символов st во внутреннее представление целого или вещественного числа. Это число будет размещено в переменной x. Параметр code содержит 0, если преобразование произошло успешно. Если произошла ошибка, code содержит номер ошибочного символа.

*Комбинированные типы (записи).*

*Запись* – это структура данных, состоящая из фиксированного числа разнотипных элементов, называемых полями записи. Каждое поле записей имеет своё имя.

type <имя> = record

 <список полей>

 end;

Список полей – это список имён, разделяемых запятыми, с указанием типа.

Пример:

goods = record

 name: string[10];

 price: real;

 number: integer;

 end;

var tov: goods;

tov.price := 2.7;

Доступ к полям записи производится с помощью конструкции, называемой селектором записей. В общем виде он имеет вид: Q.Z, где Q – переменная комбинированного типа, а Z – имя поля. В качестве типа данных для полей записи может использоваться любой тип, кроме самой записи.

type birthday = record

 day: 1..31;

 month: 1..12;

 year: word;

 end;

 person = record

 name: string[20];

 data: birthday;

 end;

var sasha: person;

Обращение: sasha.data.day := 17;

При работе с комбинированными типами чтобы упростить доступ к полям записи, используют оператор присоединения.

with <имя переменной> do <оператор>;

with sasha.data do begin

day := 17;

month := 7;

year := 1997;

end;

*Множественные типы.*

Множество – это совокупность объектов.

type Digit = set of 1..5;

var s: Digit;

Значение множественного типа может содержать любое количество различных элементов – от 0 до всех возможных значений. Тип, которому должны принадлежать все элементы множества, называется базовым типом. Если базовый тип, на котором строится подмножество, имеет k элементов, то количество подмножеств, входящих в этот множественный тип, равно 2^k.

x: 1..3;

z: set of 1..3;

z может быть пустым множеством или иметь значения: 1; 2; 3; 1 и 2; 1 и 3; 2 и 3; 1, 2 и 3.

В качестве базового типа множества может использоваться любой порядковый тип, кроме типов Word, Integer и Longint. Множество не может содержать больше, чем 256 элементов. Переменная множественного типа занимает 32 байта. Множество – это бесповторная, неупорядоченная совокупность объектов.

type ElemColor = (red, yellow, green);

 Color = set of ElemColor;

var mycolor: Color;

В Паскале для изображения множества используется конструктор множества – список элементов множества, заключённый в квадратные скобки.

mycolor := [] или mycolor := [red, green]

Два множества считаются эквивалентными тогда и только тогда, когда все их элементы одинаковы, причём порядок следования элементов безразличен. Если все элементы одного множества входят также и в другое, говорят о включении первого множества во второе.

Операции над множествами:

1) Пересечение (\*). Результат операции – множество, содержащее элементы, общие для обоих множеств.

A = [0..3, 6]

B = [3..9]

C := A \* B; C = [3, 6]

2) Объединение (+). Результат – множество, состоящее из элементов, принадлежащих и A, и B.

A + B = [0..9]

3) Вычитание (-). Результат – множество, состоящее из тех элементов первого множества, которые не входят во второе.

A – B = [0..2]

4) Операция проверки принадлежности элемента множеству.

<выражение> in <множество>

var st: string;

 a: set of 0..9;

 k, i: integer;

begin

readln(st);

k := length(st);

a := [0..9];

for i := 1 to k do

if st[i] in a then write(st[i]);

writeln;

for i := 1 to k do

if not (st[i] in a) then write(st[i]);

end.

Кроме рассмотренных операций, к множествам применимы операции отношения: =, <>, <=, >=.

В Паскале отсутствует механизм изъятия элемента из множества.

var s: set of char;

for i := 0 to 255 do

if chr(i) in s then inc(c);

**Эквивалентность и совместимость типов.**

В языке Паскаль принят принцип именной эквивалентности. Два типа T1 и T2 являются эквивалентными, если выполняется одно из двух условий: 1) T1 и T2 представляют собой один и тот же идентификатор; 2) Тип T2 описан с использованием типа T1 с помощью непосредственного неравенства вида T2 = T1, либо посредством цепочки неравенств: T2 = T1, T3 = T2 и т.д.

type A1 = array[1..10] of integer;

 A2 = array[1..10] of integer;

Но типы A1 и A2 неэквивалентны.

V1, V2 = array[1..10] of integer;

V1 и V2 – эквивалентные типы.

Эквивалентность типов требуется при выполнении оператора присваивания над массивами, а также в случае передачи параметров при вызове процедур или функций (то есть типы фактических и формальных параметров должны быть эквивалентны).

Выполнение операций выражений и сравнения требует, чтобы операнды имели совместимые типы.

Два типа считаются совместимыми, если выполняется одно из следующих условий: 1) два типа эквивалентны; 2) оба типа являются целыми, либо оба – вещественными; 3) один тип является ограниченным типом, причём его базовым типом является другой тип; 4) оба типа являются ограниченными, причём их базовым типом является один и тот же тип; 5) оба типа являются множественными типами, причём их базовые типы совместимы; 6) оба типа строковые, либо один тип строковый, а другой – символьный; 7) оба типа ссылочные.

Оператор присваивания считается корректным, если тип переменной в левой части совместим с типом выражения в правой части. Типы совместимы по присваиванию при выполнении хотя бы одного из следующих условий: 1) оба типа эквивалентны, и ни один из них не является файловым или структурным типом, содержащим поле с файловым типом; 2) оба типа являются совместимыми дискретными типами, и значение типа в правой части попадает в диапазон возможных значений типа в левой части; 3) оба типа вещественные, и значения из правой части попадают в диапазон возможных значений из левой части; 4) слева – переменная вещественного типа, справа – выражение целого типа; 5) слева – строковый тип, справа – символьный, либо строковый; 6) оба типа – совместимые множественные типы; 7) оба типа являются совместимыми ссылочными типами.

**Преобразование типов.**

В Паскале данные одного типа могут преобразовываться в данные другого типа. Такое преобразование может быть явным и неявным.

Неявное преобразование – возможно в присваивании. Например, когда вещественной переменной присваиваются целые значения. Кроме того, неявное присваивание встречается в выражениях с переменными смешанных типов.

При явном преобразовании типов используются вызовы специальных функций преобразования, аргументы которых принадлежат одному типу, а значения – другому. Например, функция Odd(i), функция Chr(j), функции Round(x), Trunc(x). Кроме того, в Паскале есть специальная конструкция явного преобразования типа. Называется она «приведение типа переменной». Тип (имя\_перем).

si: 0..255;

char(si) := ‘z’;

Конструкция приведения типа переменной может находиться везде, где допускается вхождение переменной. Близкой по синтаксису является конструкция приведения типа значения, которая позволяет преобразовать тип произвольного выражения, записанного в круглых скобках, к типу перед скобками. Тип (выражение). Конструкция может встречаться везде, где допустимо выражение, например, справа от символа присваивания.

Li: Longint;

i: integer;

Li := 1234567;

i := integer(Li + 1);

Но такое преобразование может привести к усечению типа значения. Тип выражения в скобках и идентификатор выражения перед скобками должны являться оба дискретными типами. Если один дискретный тип преобразуется к другому, то такое преобразование может привести к усечению или увеличению размера памяти по сравнению с исходным. Если значение расширяется, то его знак всегда сохраняется.

**Файловые типы в Паскале и ввод-вывод.**

Под файлом понимается именованная область внешней памяти ЭВМ или логическое устройство. В файл данные можно записать, либо извлечь. Эти действия имеют общее название «ввод-вывод». Для осуществления операции ввода-вывода с файлами, в программе необходимо определить переменные файловых типов, которые считаются представителями файлов в программе. Файл при этом рассматривается как бесконечный список значений. В Паскале существует три типа файловых переменных: типизированные файлы, текстовые и нетипизированные файлы.

var fp: file of <тип>;

fp: file of real; - типизированные файлы

fp1: text; - текстовые файлы

fp2: file; - нетипизированные файлы

С каждой переменной файлового типа связано понятие текущего указателя файла. Все элементы файла считаются пронумерованными, и начальный элемент имеет нулевой номер.

*Операции над файлами.*

Input – файл для считывания данных с клавиатуры.

Output – файл для вывода данных на экран.

Для того чтобы работать с файлом на внешнем носителе, в программе необходимо: 1) описать файловую переменную; 2) связать файловую переменную с файлом на диске. Это осуществляется с помощью процедуры: Assign(fp, ‘<cтрока, задающая имя файла>’);

Assign(fp, ‘D:\MyDir\data.dat’);

3) открыть файл. В Паскале файл можно открыть только для чтения, только для записи. Типизированные файлы можно открыть и для чтения, и для записи одновременно. Для открытия файлов служат две процедуры: Reset(fp) – на чтение; Rewrite(fp) – на запись.

Под открытием файла понимается поиск файла на внешнем носителе, образование специальных буферных областей в памяти для работы с ним и установка текущего указателя на начало файла. Процедура Reset предполагает, что дисковый файл существует. Процедура Rewrite допускает, что указанный файл может не существовать. В этом случае процедура создаёт на диске файл с тем именем, которое указали в процедуре Assign. Для закрытия файлов существует процедура Close(fp); Она закрывает файл, однако связь с файловой переменной не разрывается.

*Операции ввода-вывода.*

Для осуществления ввода-вывода из файла существуют две процедуры: Read и Write.

Read([<ф.пер.>], список ввода);

Write([<ф.пер.>], список вывода);

Список ввода – это последовательность из одного или более имён переменных типа char, string, а также любого целого или вещественного типа. В эти переменные будут помещаться значения из файла.

Список вывода – это список выражений. Выражения также разделяются запятыми.

И чтение из файла, и запись в файл производится в позицию, в которой находится текущий указатель файла. После ввода-вывода текущий указатель перемещается.

*Некоторые процедуры и функции для работы с файлами.*

*Rename(<ф.пер.>, ‘<новое имя файла>’)* – переименовать файл. Перед выполнением процедуры файл необходимо закрыть, если он был открыт.

*Erase(<ф.пер>)* – удаляет файл.

*EOF(<ф.пер.>): Boolean* – функция проверяет, достигнут ли указателем конец файла.

*IOResult: Word* – возвращает условный признак последней операции ввода-вывода. Если операция открытия файла прошла успешно, функция возвращает 0, и не 0 в противном случае. Для использования функции необходимо отключить контроль ошибок ввода-вывода. Директива {$I-} перед открытием файла отключает контроль ошибок. После проверки лучше снова включить контроль ошибок директивой {$I+}.

var f: file of char;

…

begin

assign(f, ‘Myfile.dat’);

{$I-}

Reset(f);

{$I+}

if IOResult <> 0 then writeln(‘Файл не существует’)

 else …

**Текстовые файлы.**

Предназначены для хранения текстовой информации. По своей структуре эти файлы представляют собой последовательность символьных строк переменной длины. Каждая строка текстового файла заканчивается признаком конца строки (коды CR (13), LF (10)). Заканчивается текстовый файл признаком конца файла (код EOF (26)). К каждой строке текстового файла возможен только последовательный доступ, начиная с первой. То есть, текстовые файлы – это файлы с последовательным доступом.

var ft: text;

begin

assign(ft, ‘ ‘);

reset(ft) / rewrite(ft) / append(ft);

*Append(<ф.пер.>)* – служит для открытия текстового файла для записи в конец.

Для чтения из текстового файла используются процедуры: Read(ft, <список ввода>), Readln(ft, <список ввода>). Список ввода – последовательность из одного или более имён переменных типа Char, String, а также любого целого или вещественного типов. В эти переменные будут помещаться значения из файла. После любой операции чтения указатель перемещается на следующую позицию.

При обработке переменных типа Char выполняется чтение одного символа из файла и присваивание считанного значения переменной.

Read(ft, ch);

Если перед выполнением чтения указатель файла достиг конца строки, то результатом чтения будет символ CR с кодом 13. Если указатель стоит на конце файла, то результатом чтения будет символ с кодом 26.

При вводе переменных типа String количество считанных процедурой и помещённых в строку символов равно максимальной длине строки, если только раньше не встретится признак конца строки или конец файла. Признак конца строки в переменную не записывается.

st: string[20];

read(ft, st);

Если строка файла больше, чем длина строковой переменной, указанной в процедуре Read, то лишние символы до конца строки файла отбрасываются, а новое обращение к Read вернёт пустую строку. С помощью процедуры Read нельзя прочитать последовательность строк. Для ввода последовательности строк нужно использовать процедуру Readln. Её также можно задавать без списка ввода.

При вводе числовых переменных процедура Read выделяет подстроку во входном потоке по следующему правилу: все пробелы, символы табуляции и маркеры конца строк пропускаются. После выделения первого значащего символа и всех последующих символы пробела, табуляции служат признаком конца подстроки.

x: integer;

Read(ft, x);

Выделенная подстрока рассматривается как символьное представление числового значения соответствующего типа и преобразуется во внутреннее представление числа.

Процедуры записи в файл: Write(ft, <список вывода>), Writeln(ft, <список вывода>).

Список вывода – список выражений типа Char, String, Boolean, а также любого целого или вещественного типа. Для любого выводимого параметра можно указывать минимальную ширину поля, в которое будет выведено символьное представление значения.

write(ft, x:4);

При выводе вещественного числа можно задавать минимальную ширину поля и количество знаков после десятичной точки. При использовании процедуры Writeln выводимая строка символов завершается символами конца строки. Writeln можно задавать и без списка вывода, для файлов это означает пропуск строки.

Три функции, которые можно применять только к текстовым файлам:

*Eoln(ft: text): Boolean* – проверяет, где стоит файловый указатель. Если он стоит в конце текущей строки, то функция возвращает True. В противном случае – False.

*SeekEoln(ft: text): Boolean* – перемещает файловый указатель до первого значащего символа или до маркера конца строки, пропускает все пробелы и знаки табуляции. Возвращает True, если указатель стоит на маркере конца строки, и False – если на значащем символе.

*SeekEof(ft: text): Boolean* – работает так же, как SeekEoln, только относительно конца файла, а не строки.

**Типизированные файлы.**

Применяются для хранения однотипных элементов в их внутреннем представлении. По своей структуре типизированный файл представляет собой двоичный файл со строго постоянной длиной каждого элемента. Для того чтобы работать с типизированным файлом, в программе должна быть описана переменная следующим образом:

var fp: file of <тип>;

Компоненты типизированного файла могут быть любого типа, кроме файлового. Типизированные файлы являются файлами с прямым доступом. Каждый элемент файла нумеруется, самый первый элемент имеет номер 0. Типизированный файл можно открыть с помощью процедуры Reset как на чтение, так и на запись. При этом типизированный файл должен существовать. Процедура Rewrite открывает файл только на запись, при этом файл может не существовать. При открытии файла и той, и другой процедурой указатель файла ставится на нулевой элемент.

Для чтения из файла используется только процедура Read. Список ввода – список имён переменных того же типа, что и тип элементов файла. Для записи используется только процедура Write. Список вывода – список выражений того же типа, что и тип элементов файла.

*Seek(fp: file of real, N: longint)* – перемещает указатель на номер N элемента файла.

Seek(fp, 0L) – перемещает указатель в начало файла.

*FileSize(fp): longint* – возвращает количество элементов в файле.

*FilePos(fp): longint* – возвращает номер элемента файла, на котором стоит указатель.

[www.fepo.ru](http://www.fepo.ru)

230101.65

Пример работы с типизированным файлом.

type mas = array[1..100] of real;

var a, b: mas;

 f: file of mas;

begin

assign(f, ‘file.dat’);

rewrite(f);

………… {запись элементов в массивы}

write(f, a, b);

close(f);

end.

Элементом файла является массив целиком. В такой файл нельзя записать отдельный элемент массива или простую переменную вещественного типа.

Из текстового файла прочитать пару вещественных чисел. Считая первое действительной, а второе – мнимой составляющей комплексного числа, записать их в файл комплексных чисел.

program tip-file;

type complex = record

 re, im: real

 end;

var x: complex;

 f1: text;

 f2: file of complex;

begin

assign(f1, ‘tipfile.dat’);

reset(f1);

assign(f2, ‘tipfile.res’);

rewrite(f2);

while not eof(f1) do begin

 read(f1, x.re, x.im);

 write(f2, x);

end;

close(f1);

close(f2);

end.

**Нетипизированные файлы.**

var data: file;

Нетипизированные файлы используются для организации доступа к любым дисковым файлам, независимо от их структуры. При этом файл представляется, как последовательность элементов произвольного типа. Единственное, что оговаривается, это размер элемента.

Нетипизированный файл можно открыть либо на чтение (Reset), либо на запись (Rewrite).

reset(data, [N]);

rewrite(data, [N]);

N – размер одного элемента, указывать необязательно. По умолчанию, размер одного элемента – 128 байт. Размер файла может быть не кратен размеру записи, последняя запись будет неполной.

Для нетипизированных файлов существуют специальные процедуры.

*BlockWrite(var f: file; var x; count: word; [var num: word])* – запись в файл.

*BlockRead(var f: file; var x; count: word; [var num: word])* – чтение из файла.

*f* – файловая переменная.

*x* – буферная переменная, которая будет участвовать в обмене с файлом. При чтении в неё будут записываться элементы, а при записи – считываться из неё и записываться в файл.

*count* – количество записей, которые должны быть прочитаны (или записаны) за одно обращение к файлу.

*num* – при выходе из процедуры содержит количество фактически обработанных записей.

Если N – длина одной записи, то за одно обращение к файлу нельзя прочитать больше, чем count\*N байт. И эта величина также не может превышать 64 кБ. Длина внутреннего представления переменной x должна быть достаточной для размещения count\*N байтов при чтении из файла. Если при записи в файл процедурой BlockWrite на диске не окажется свободного места или при чтении из файла процедурой BlockRead размер буфера (x) окажется меньше указанного произведения, то произойдёт следующее: если параметр num в вызовах процедур не указан, то произойдёт ошибка ввода-вывода; если указан, то ошибка не фиксируется, а после выполнения процедуры его значение не совпадёт с параметром count. По завершению процедур указатель смещается на num записей.

Пример: программа выполняет ввод вещественных чисел из текстового файла и запись их в безтиповой файл блоками по 4 числа.

program createfile;

var buf: array[1..4] of real;

 f\_in: text;

 f\_out: file;

 i, k: integer;

 name\_in, name\_out: string;

begin

{$I-}

writeln(‘Введите имя входного файла: ‘);

readln(name\_in);

assign(f\_in, name\_in);

reset(f\_in);

if IOResult <> 0 then begin

 writeln(‘Файл’, name\_in, ‘ не найден’);

 exit;

end;

writeln(‘Введите имя выходного файла: ‘);

readln(name\_out);

assign(f\_out, name\_out);

rewrite(f\_out, sizeof(real)\*4);

{$I+}

i := 0;

while not eof(f\_in) do begin

 inc(i);

 read(f\_in, buf[i]);

 if i = 4 then begin blockwrite(f\_out, buf, 1); i := 0; end;

end;

if i <> 0 then begin

 for k := i + 1 to 4 do buf[k] := 0;

 blockwrite(f\_out, buf, 1);

end;

close(f\_in);

close(f\_out);

end.

*Truncate(var f)* – процедура устанавливает в текущей позиции признак конца файла и удаляет все последующие элементы.

Пример 2: программа, выводящая на экран заданную по номеру запись из файла.

program get\_bfile;

var buf: array[1..4] of real;

 f: file;

 i, k: integer;

 filename: string;

begin

writeln(‘Введите имя файла: ‘);

readln(filename);

{$I-}

assign(f, filename);

reset(f, sizeof(real)\*4);

if IOResult <> 0 then begin

 writeln(‘Файл’, filename, ‘ не найден’);

 exit;

end;

{$I+}

while true do begin

 writeln(‘Введите номер записи или -1 для окончания’);

 readln(k);

 if (k > filesize(f)) or (k < 0) then begin

 writeln(‘Такой записи нет’);

 exit;

 end;

seek(f, k);

blockread(f, buf, 1);

for i := 1 to 4 do write(buf[i]:6:1);

end;

close(f);

end.

**Процедуры и функции.**

Структурный подход к программированию предполагает представление программы как совокупности относительно обособленных фрагментов со строгими и явно определёнными интерфейсами. Такими обособленными фрагментами являются подпрограммы.

*Подпрограмма* – это часть программы, оформленная в виде отдельной синтаксической конструкции и снабжённая именем. Выполнение действий, заданных в подпрограмме в виде операторов, может быть произведено в любой точке программы посредством указания имени этой подпрограммы. В языке Паскаль реализовано два вида подпрограмм: процедуры и функции. Имея одинаковую структуру, они различаются назначением и способом использования. Процедура предназначена для выполнения некоторой последовательности действий, а смысл функции заключается в выполнении алгоритма вычисления некоторого нового значения простого или ссылочного типов.

*Общая структура подпрограммы.*

Структура подпрограммы почти полностью повторяет структуру основной Паскаль-программы. При описании подпрограммы необходимо задать три обязательных компонента:

1) Интерфейс подпрограммы.

2) Описание локальных объектов.

3) Операторы.

Интерфейсом подпрограммы является её заголовок. В описании локальных объектов могут определяться константы, переменные, другие подпрограммы и т.п.

procedure <имя> [(список формальных параметров)];

function <имя> [(список формальных параметров)]: <тип>;

Список формальных параметров – список тех параметров, которые передаются подпрограмме в точке её вызова.

procedure Swap (var x, y: real);

function Max (a, b: byte): byte;

procedure Stop;

*Области действия имён.*

Среди описаний в подпрограмме может содержаться описание других подпрограмм (так называемые, вложенные подпрограммы). (См. рис. в тетради).

При работе с блоками необходимо соблюдать следующие правила:

1) Имена объектов, описанных в блоке, считаются известными в пределах данного блока, включая и все вложенные блоки.

2) Имена объектов, описанных в блоке, должны быть уникальны в пределах данного блока и могут совпадать с именами объектов из других блоков.

3) Если во внутреннем блоке описан объект, имя которого совпадает с именем из внешнего блока, то имя из внешнего блока не будет доступно во внутреннем блоке.

Совокупность объектов, описанных в самом внешнем блоке, называется глобальными переменными.

var i: integer;

procedure P;

 var i: integer;

 begin

 writeln(i);

 end;

begin

 i := 1;

P;

end.

*Параметры.*

В заголовке подпрограммы может быть задан список формальных параметров. Этот список заключается в круглые скобки, и для каждого параметра указывается его имя и тип. Отделяются параметры друг от друга точкой с запятой. Каждый параметр, заданный в заголовке, считается локальным в данной подпрограмме. Если список формальных параметров существует, то при вызове подпрограммы ей нужно передать фактические параметры.

Swap(a, b);

Параметры, указываемые при вызове подпрограммы, называются фактическими параметрами. При задании типа формальных параметров в заголовке тип должен указываться идентификатором.

procedure Incorrect (a: array[1..15] of real); - запись неправильная.

type vector = array[1..15] of real;

procedure Correct (a: vector); - запись правильная.

Примеры заголовков подпрограмм:

function MaxElem (a: vector; n: byte): real;

procedure Sum (a, b: matrix; var c: matrix);

procedure Move (var Source, Dest; n: word);

Формальные параметры в заголовках процедуры могут быть заданы тремя способами:

1) без служебного слова var с указанием типа.

2) со служебным словом var и с указанием типа.

3) со служебным словом var без указания типа.

Эти три способа задания формальных параметров выражают три различных способа передачи параметров подпрограммам.

Первый способ называется «передача параметров по значению».

Второй способ называется «передача параметров по ссылке».

Третий способ называется «передача нетипизированных параметров по ссылке».

*Передача параметров по значению.*

Это самый простой и наиболее распространённый способ. При передаче параметров по значению программа работает с копией фактического параметра.

var a, b: real;

procedure SumSquare (x, y: real);

begin

 x := x \* x;

 y := y \* y;

 writeln(x + y);

end;

begin

 a := 1,7;

 b := 8,09;

 SumSquare(a, b);

 writeln(a, b);

end.

При вызове подпрограммы начальное значение параметра автоматически устанавливается равным значению соответствующего фактического параметра. Этот фактический параметр может быть произвольным выражением того же типа, что и формальный параметр. Внутри подпрограммы значения параметров, переданных по значению, могут изменяться, но это никак не отражается на значении переменных вне подпрограммы.

**Типовые параметры.**

При вызове подпрограммы мы можем представлять переменную типа, но не выражения.

Так как тип формального параметра не задан, то, строго говоря, он не совместим ни с какой переменной и не может входить не в какие конструкции. Чтобы его можно было использовать, его надо привести к какому-либо типу. Сделать это можно двумя способами:

1. Применить конструкцию приведения типа.
2. Описать в подпрограмме локальную переменную определённого типа, с совмещением её в памяти с нетипизированным параметром.

В качестве примера использования первого способа рассмотрим функцию Fqual, которая может использоваться для сравнения значений двух переменных любых типов и, соответственно, любых размеров.

Fqual(var Source, Dest);

Type

Bytes=array[0..maxint] of byte;

Var

N:integer;

Begin

N:=0;

While (n<size) and (bytes(dest)[n]=bytes(source)[n]) do n:=n+1;

Fqual:=(n=size)

End.

Type

Vector=array[1..10] of integer;

Point=record

 X,y:integer

 End;

Var

Vec1,vec2:vector;

N:integer;

P:point;

Fqual(vec1,vec2;size of integer);

Fqual(vec1[1], {сравниваются 5 первых элементов vec1 с 5 последними элементами vec2} vec2[6]; size of integer);

Fqual(vec1[1],P,4);

*Второй способ:*

В Turbo Pascal для того чтобы разместить 2 параметра в одном том же поле, используется слово absolute.

Var A:real;

 B:integer absolute a;

В случае такого описания новая память под хранение переменной b не отводится. Переменная a и b могут иметь разные типы. Никаких проверок не производится.

**Вычисление значения функции.**

Смысл функции заключается в вычислении некоторого значения и организации передачи этого значения в точку вызова. Возврат вычисленного значения осуществляется следующим образом.

В теле функции должен присутствовать оператор присваивания, в левой части которого должно быть указано имя функции, а в правой части – выражение, вычисляющее возвращаемое значение. Тип выражения должен быть совместим с типом функции. Если в теле функции ни одного такого оператора нет, результат функции считается неопределённым.

Exit – с помощью этой процедуры можно немедленно вернутся в точку вызова подпрограммы.

Существует механизм предварительного описания подпрограмм. Предварительное описание подпрограммы содержит заголовок подпрограммы, а вместо тела подпрограммы записывается служебное слово forward.

Это слово указывает на то, что тексты подпрограмм располагаются далее.

Под внешним описанием понимается ситуация, когда подпрограмма или группа подпрограмм разработаны вне системы Турбо Паскаль и на другом языке программирования, и необходимо эти подпрограммы подключить к паскаль-программе. В этом случае объектный код подключаемых подпрограмм содержится в файле с расширением .obj, а само подключение осуществляется следующим образом: в паскаль-программе необходимо указать заголовок подключаемой подпрограммы, после которого должно идти служебное слово external.

**Распределение памяти для локальных переменных.**

Турбо Паскаль реализует следующий принцип распределения памяти для подпрограмм

Все глобальные переменные, а также типизированные константы, размещаются в одном сегменте данных и, следовательно, не могут занимать место больше, чем 64 кБ. Локальные переменные размещаются в памяти динамически, при вызове подпрограммы, их содержащей. После выполнения подпрограммы память, отведенная под локальные переменные, освобождается. Память под локальные переменные выделяется в стеке. Общий объем стека ограничен одним сегментом. По умолчанию программе отводится 16 кБ стековой памяти.