# 1. ВВЕДЕНИЕ В ТУРБО ПАСКАЛЬ

## 1.1. ИСТОРИЯ ТУРБО ПАСКАЛЯ

Язык назван в честь выдающегося французского математика и философа Блеза Паскаля (1623 — 1662). Система программирования Турбо Паскаль, разработанная американской корпорацией *Borland,* остается одной из самых популярных систем программирования в мире. Этому способствуют, с одной стороны, простота лежащего в ее основе языка программирования Паскаль, а с другой — труд и талант сотрудников *Borland во* главе с идеологом и создателем Турбо Паскаля Андер-сом Хейлсбергом, приложивших немало усилий к ее совершенствованию. Придуманный швейцарским ученым Никласом Виртом как средство для обучения студентов программированию, язык Паскаль стараниями А. Хейлсберга превратился в мощную современную профессиональную систему программирования, которой по плечу любые задачи — от создания простых программ, предназначенных для решения несложных вычислительных задач, до разработки сложнейших реляционных систем управления базами данных. Когда в далеком 1986 году я начинал работу с Турбо Паскалем, я не мог предположить, какую огромную роль в моей жизни сыграет эта замечательная система программирования.

Появление *Windows* и инструментальных средств *Borland Pascal with Objects* и *Delphi* для разработки программ в среде *Windows* лишний раз показало, какие поистине неисчерпаемые возможности таит он в себе: и *Borland Pascal,* и используемый в *Delphi* язык *Object Pascal* основываются на Турбо Паскале и развивают его идеи.

## 1.2. СРЕДА ТУРБО ПАСКАЛЯ

Система программирования Турбо Паскаль представляет собой единство двух в известной степени самостоятельных начал: компилятора с языка программирования Паскаль и некоторой инструментальной программной оболочки, способствующей повышению эффективности создания программ. Для краткости условимся в дальнейшем называть реализуемый компилятором язык программирования Паскаль *языком Турбо Паскаль,* а разнообразные сервисные услуги, предоставляемые программной оболочкой — *средой Турбо Паскаля.*

Среда Турбо Паскаля — это первое, с чем сталкивается любой программист, приступающий к практической работе с системой.

Язык Turbo Pascal - императивный (процедурный), т.е. процесс программирования представляет собой запись последовательности команд для обработки данных.

## 1.3. АЛФАВИТ ЯЗЫКА

Алфавит языка Турбо Паскаль включает буквы, цифры, шестнадцатеричные цифры, специальные символы, пробелы и зарезервированные

*Буквы* — это буквы латинского алфавита от a до z и от A до Z, а также знак подчеркивания _ (код *ASCII* 95). В Турбо Паскале нет различия между приписными и строчными буквами алфавита, если только они не входят в символьные и строковые выражения.

*Цифры* — арабские цифры от 0 до 9. Каждая шестнадцатеричная цифра имеет значение от 0 до 15. Первые 10 значений обозначаются арабскими цифрами 0...9, остальные шесть —латинскими буквами *A...F* или *a...f.*

*Специальные символы* — это символы +-* / = ,' . : ; < > 1 ]()[ { } л @ $ # . К специальным символам относятся также следующие пары символов: о <= >= := (* *) (. .) . В программе эти пары символов нельзя разделять пробелами, если они используются как знаки операций отношения или ограничители комментария. Особое место в алфавите языка занимают *пробелы,* к которым относятся любые символы *ASCII* в диапазоне кодов от 0 до 32. Эти символы рассматриваются как ограничители идентификаторов, констант, чисел, зарезервированных слов. Несколько следующих друг за другом пробелов считаются одним пробелом (последнее не относится к строковым константам).

В Турбо Паскале имеются следующие *зарезервированные слова:*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ***and*** | ***not*** | ***begin*** | ***or*** | ***constructor*** | ***program*** | ***do*** | ***set*** |
| ***end*** | ***string*** | ***function*** | ***type*** | ***implementa*** | ***uses*** | ***interface*** | ***with*** |
| ***nil*** | ***array*** | ***of*** | ***const*** | ***tion*** | ***div*** | ***repeat*** | ***else*** |
| ***shr*** | ***for*** | ***to*** | ***if*** | ***procedure*** | ***inline*** | ***while*** | ***mod*** |
| ***asm*** | ***object*** | ***case*** | ***packed*** | ***until*** | ***record*** | ***downto*** | ***shi*** |
| ***file*** | ***then*** | ***goto*** | ***unit*** | ***destructor*** | ***var*** | ***label*** | ***xor*** |

# 1.4. ИДЕНТИФИКАТОРЫ

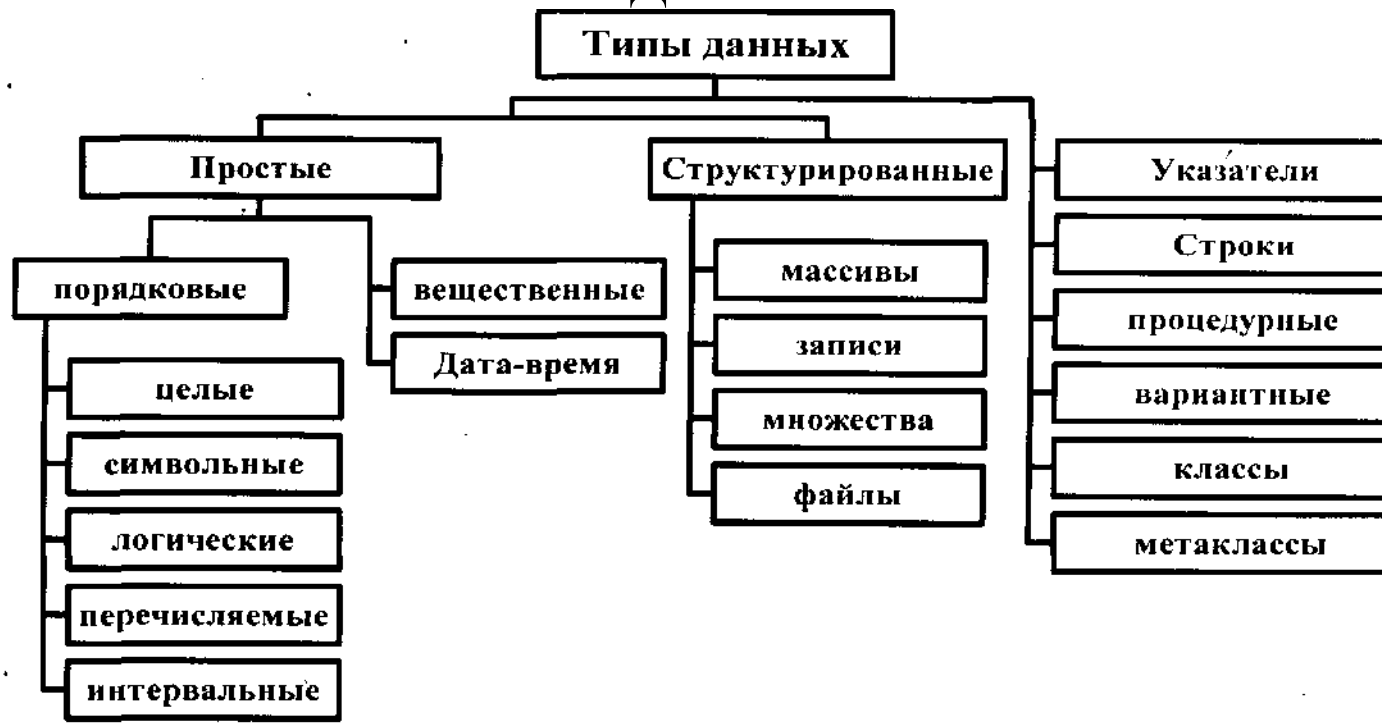
Идентификаторы - это имена констант, переменных, меток, типов, объектов, процедур, функций, модулей, программ и полей в записях.

Величина обозначается индикатором, который должен удовлетворять следующим ограничениям:

- идентификатор может состоять из букв латинского алфавита, цифр, знака подчеркивания, никакие другие символы в идентификаторе не допустимы;
- идентификатор не может начинаться с цифры;
- идентификатор не может совпадать ни с одним из зарезервированных слов;
- длина идентификатора может быть произвольной, но значащими будут только первые 63 символа.

Зарезервированные слова не могут использоваться в качестве идентификаторов.

# 2. ТИПЫ ДАННЫХ




Тип данных определяет:

- диапазон значений
- представление в памяти (размер области памяти)
- операции, которые можно применять к величинам.

## 2.1. Простые типы.

К простым типам относятся порядковые и вещественные типы.

- *Порядковые типы* отличаются тем, что каждый из них имеет конечное число возможных значений. Эти значения можно определенным образом упорядочить (отсюда — название типов) и, следовательно, с каждым из них можно сопоставить некоторое целое число — порядковый номер значения.
- *Вещественные типы,* строго говоря, тоже имеют конечное число значений, которое определяется форматом внутреннего представления вещественного числа. Однако количество возможных значений вещественных типов настолько велико, что сопоставить с каждым из них целое число (его номер) не представляется возможным.

### *2.1.1. Порядковые типы.*

К порядковым типам относятся *целые, логический, символьный, перечисляемый и тип-диапазон*. К любому из них применима функция *ORD(X),* которая возвращает порядковый номер значения выражения *X.* Для целых типов функция *ORD(X)* возвращает само значение *X,* т.е. *ORD(X) = X* для *X,* принадлежащего любому целому типу. Применение *ORD(X)* к логическому, символьному и перечисляемому типам дает положительное целое число в диапазоне от 0 до 1 (логический тип), от О до 255 символьный), от 0 до 65535 (перечисляемый). Тип-диапазон сохраняет свойства базового порядкового типа, поэтому результат применения к нему функции *ORD(X)* зависит от свойств этого типа. К порядковым типам можно также применять функции:

*RED(X)* — возвращает предыдущее значение порядкового типа (значение, которое соответствует порядковому номеру *ORD(X)* - /), например ORD(PRED(X)) = ORD(X) - 1

*SUCC(X)* — возвращает следующее значение порядкового типа, которое соответствует порядковому номеру *ORD(X)* + /, например ORD(SUCC(X)) = ORD(X) + 1

Если представить себе любой порядковый тип как упорядоченное множество значений, возрастающих слева направо и занимающих на числовой оси некоторый отрезок, то функция *PRED(X)* не определена для левого, а *SUCC(X)* — для правого конца этого отрезка.

**Целые типы.** Диапазон возможных значений целых типов зависит от их внутреннего представления, которое может занимать один, два или четыре байта.

| **Целые типы** | | |
|---|---|---|
| **Название** | **Длина, байт** | Диапазон **значений** |
| Byte | **1** | 0...255 |
| Shortint | 1 | -128...+127 |
| Word | 2 | 0...65535 |
| Integer | 2 | -32768...+32767 |
| Longint | 4 | -2 147 483 648...+2 147 483 647 |

При использовании процедур и функций с целочисленными параметрами следует руководствоваться «вложенностью» типов, т.е. везде, где может использоваться *WORD,* допускается использование *BYTE* (но не наоборот), в *LONGINT* «входит» *INTEGER,* который, в свою очередь, включает в себя *SHORTINT.*

Буквами *B, S, W, I*, *L* обозначены выражения соответственно типа *BYTE, SHORT/NT, WORD, INTEGER* и *LONGINT, x* - выражение любого из этих типов; буквы *vb, vs. vw,* vi, v/, *vx* обозначают переменные соответствующих типов. В квадратных скобках указывается необязательный параметр.

При действиях с целыми числами тип результата будет соответствовать типу операндов, а если операнды относятся к различным целым типам, то типу того операнда, который имеет максимальную мощность (максимальный диапазон значений). Возможное переполнение результата никак не контролируется, что может привести к неправильной работе программы или зацикливанию.

**Логический тип.** Значениями логического типа *BOOLEAN* может быть одна из предварительно объявленных констант *EALSE* (ложь) или *TRUE* (истина). Для них справедливы правила:

*ORD(FALSE) = 0;*
*ORD(TRUE) = I;*
*FALSE < TRUE;*
*SUCC(FALSE) = TRUE;*
*PRED(TRUE) = FALSE.*

**Символьный тип.** Значением символьного типа *CHAR* является множество всех символов ПК. Каждому символу приписывается целое **число** в диапазоне 0...255. Это число служит кодом внутреннего представления символа, его возвращает функция *ORD.*

Дня кодировки используется код *ASCII* (American Standard Code for formation Interchange — американский стандартный код для обмена информацией). Это 7 - битный код, т.е. с его помощью можно закодировать лишь 128 символов в диапазоне от 0 до 127. В то же время в 8 - битном байте, отведенном для хранения символа в Турбо Паскале, можно закодировать в два раза больше символов в диапазоне от 0 до 255. Первая половина символов ПК с кодами *О...* 127 соответствует стандарту *ASCII.* Вторая половина символов с кодами 128...255 не офаничена жесткими рамками стандарта и может меняться на ПК разных типов.

К типу *CHAR* применимы операции отношения, а также встроенные функции:

*CHR(B)* — функция типа *CHAR;* преобразует выражение *В* типа *BYTE* в символ и возвращает его своим значением, т.е. пс коду выводит символ;

*UPCASE(CH)* — функция типа *CHAR;* возвращает прописную букву, если *СИ* - строчная латинская буква (с кириллице не работает), в противном случае возвращает сам символ. *ORD(CH)* — функция типа *BYTE;* возвращает код символа.

*Перечисляемый тип.* Перечисляемый тип задается перечислением тех значений, которые он может получать. Каждс значение именуется некоторым идентификатором и располагается в списке, обрамленном круглыми скобками. При работе этим типом можно предварительно описать свой тип. Причем все идентификаторы должны быть латинскими символами. Например: **type** colors = (red, blue, black);

**var** col: colors; -Применение перечисляемых типов делает программы нагляднее.

Соответствие между значениями перечисляемого типа и порядковыми номерами этих значений устанавливается поря, ком перечисления: первое значение в списке получает порядковый номер 0, второе - 1 и т.д. Максимальная мощность пер числяемого типа составляет 65536 значений, поэтому фактически перечисляемый тип задает некоторое подмножество цел го типа *WORD* и может

рассматриваться как компактное объявление сразу группы целочисленных констант со значениями 1 и т.д.

Использование перечисляемых типов повышает надежность программ благодаря возможности контроля тех значен которые получают соответствующие переменные. Например: *ord(red) =0 ord (blue) =1 ord(black) =2* Таким образом, между значениями перечисляемого типа и множеством целых чисел существует однозначное соответ вие, задаваемое функцией *ORD(X).* В Турбо Паскале допускается и обратное прежние: любое выражение типа *WORD* мож преобразовать в значение перечисляемого типа, если только значение целочисленного выражения не превышает мощно( перечисляемого типа. Такое преобразование достигается применением автоматически объявляемой функции с именем пе числяемого типа. Например: col := red; col:- colors(O); Переменные любого перечисляемого типа можно объявлять без предварительного описания этого типа, например:

var col: (black, white, green); *Тип-диапазон.* Тип-диапазон есть подмножество своего базового типа, в качестве которого может выступать любой порядковый тип, кроме типа-диапазона.

Тип-диапазон задаетсяграницами своих значений внутри базового типа следующим образом:

<минимальное значением.<максимальное значение> <минимальное значение> — минимальное значение типа—диапазона <максимальное значение>— максимальное значение типа—диапазона Например: **type** digit = '0'..'9';

var d: digit; Тип-диапазон необязательно описывать в разделе *TYPE,* а можно указывать непосредственно при объявлении переменной, например; var data; 1..3I; При определении типа-диапазона нужно руководствоваться следующими правилами;

- два символа «..» рассматриваются как один символ, поэтому между ними недопустимы пробелы;
- левая граница диапазона не должна превышать его правую границу.

В стандартную библиотеку Турбо Паскаля включены две функции поддерживающие работу с типами-диапазонами:

*ИЮН(X)* - возвращает максимальное значение типа-диапазона, которому принадлежит переменная X;

*LOW(X)* - возвращает минимальное значение типа-диапазона.

### *2.1.2. Вещественные типы.*

В отличие от порядковых типов, значения которых всегда сопоставляются с рядом целых чисел и представляются в ПК абсолютно точно, значения вещественных типов определяют произвольное число лишь с некоторой конечной точностью, зависящей от внутреннего формата вещественного числа.

| **Название** | Длина, байт | Количество значащих цифр | Диапазон десятичного порядка |
|---|---|---|---|
| real | 6 | 11...12 | -39...+38 |
| double | 8 | 15...16 | -324...+308 |
| extended | 10 | 19...20 | -4951...+4932 |
| comp | 8 | 19...20 | $-2*10^{63}+1...+2*10^{63}-1$ |

Вещественное число в Турбо Паскале занимает от 4 до 10 смежных байт и имеет следующую структуру в памяти ПК:

| S | **e** | **m** |
|---|---|---|

Здесь *s* — знаковый разряд числа; e — экспоненциальная часть, которая содержит двоичный порядок; ***m*** — мантисса числа ($0.5*10^{63}$ - 0.5e63, т.е. s = 63, e = *10, m ^ 0.5). Мантисса ***m*** имеет длину от 23 (для *SINGLE)* до 63 (для *EXTENDED)* двоичных разрядов, что и обеспечивает точность 7...8 для *SINGLE* и 19...20 *EXTENDED* десятичных цифр.

Для работы с вещественными данными могут использоваться встроенные математические функции. В этой таблице *REAL* означает любой вещественный тип, *INTEGER* — любой целый тип.

**Стандартные математические функции Турбо Паскаля**

| **Обращение** | **Тип параметра** | **Тип результата** | **Примечание** |
|---|---|---|---|
| abs(x) | real, integer | real, integer | Модуль аргумента |
| cos(x) | real | real | Косинус, угол в радианах |
| sin(x) | real | real | Синус, угол в радианах |
| exp(x) | real | real | Экспонента |
| frac(x) | real | real | Дробная часть числа |

| | | | |
|---|---|---|---|
| int (x) | real | real | Целая часть числа |
| 1n(x) | real | real | Логарифм натуральный |
| sqr(x) | real | real | Квадрат аргумента |
| sqrt(x) | real | real | Корень квадратный |
| Random | — | real | Псевдослучайное число, равномерно распределенное в диапазоне 0...[1] |
| Random(x) | integer | integer | Псевдослучайное целое число, равномерно распределенное в диапазоне 0...(x-1) |
| Randomize | — | — | Инициация генератора псевдослучайных чисел |

## 2.2. Структурированные типы.

Любой из структурированных типов (а в Турбо Паскале их четыре: *массивы, записи, множества и файлы*) характеризуется множественностью образующих этот тип элементов, т.е. переменная или константа структурированного типа всегда имеет несколько компонентов. Каждый компонент, в свою очередь, может принадлежать структурированному типу, что позволяет говорить о возможной вложенности типов. В Турбо Паскале допускается произвольная глубина вложенности типов, однако суммарная длина любого из них во внутреннем представлении не должна превышать65520 байт.

При работе с этим типом можно либо предварительно описать свой тип, либо описать как переменную.

Например:

**type** matrix = **array** [0..I0J of single;

**var** m : matrix;

ИЛИ

**var** matrix: array [0..I0] of single

### *2.2.1. Массивы*

Отличительная особенность массивов заключается в том, что все их компоненты данные одного типа (возможно, структурированного).

Эти компоненты можно легко упорядочить и обеспечить доступ к любому из них простым указанием его порядкового номера. Например, если m - массив [1,2,3,4], то m[0]=I, m[IJ=2, m[2]=3 и т.д.

Описание типа массива задается следующим образом:

type <имя **Tnna>=ARRAY** [ <список индексных типов> ] of <тип> **var** <идентификатор>: <имя типа>;

ИЛИ

**var** <имя типа>: **ARRAY** [ <список индексных типов> ] of <тип>

<имя типа> - правильный идентификатор;

ARRAY, OF — зарезервированные слова (массив, из);

<список индексных типов> — список из одного или нескольких идентификаторов типов, разделенных запятыми и обрамленных в квадратные скобки.

<тип> — любой тип Турбо Паскаля.

В качестве индексных типов в Турбо Паскале можно использовать любые порядковые типы, кроме *LONG/NT* и типов-диапазонов с базовым типом *LONGINT*.

Глубина вложенности структурированных типов произвольная, поэтому количество элементов в списке индексных типов (размерность массива) не ограничено, однако суммарная длина внутреннего представления любого массива не может быть больше 65520 байт.

В Турбо Паскале можно одним оператором присваивания передать все элементы одного массива другому массиву того же типа.

### *2.2.2. Записи*

Запись — это структура данных, состоящая из фиксированного числа компонентов, называемых полями записи. В отличие от массива, компоненты (поля) записи могут быть различного типа. Чтобы можно было ссылаться на тот или иной компонент записи, поля именуются. При работе с записями необходимо задать свой тип. Структура объявления типа записи такова: **type** <имя типа> = **RECORD** <список полей> **END var** <идентификатор>: <имя типа>; <имя типа> - правильный идентификатор; RECORD, END — зарезервированные слова (запись, конец);

<список полей> — список полей; представляет собой последовательность разделов, между которыми ставится точка с запятой.

Каждый раздел записи состоит из одного или нескольких идентификаторов полей, отделяемых друг от друга запятыми. За идентификатором (идентификаторами) ставится двоеточие и описание типа поля (полей), например: . type BirthDay = **RECORD**

day, month: byte; year: word; **END;** var a,b : Birthday;

К каждому из компонентов записи можно получить доступ, если использовать составное имя, т.е. указать имя переменной, затем точку и имя, например:

a.day := 27; b.year:= I939; Чтобы упростить доступ к полям записи, используется оператор присоединения ***WITH:*** **WITH** <переменная> **DO** <оператор> WITH,'DO — ключевые слова (с, делать);

<*перемепная*> — имя переменной типа запись, за которым, возможно, следует список вложенных полей; <*оператор*> — любой оператор Турбо Паскаля. Например:

**with** a **do** month:=9; что эквивалентно a.month:=9; Оператор присоединения может содержать несколько вложенных операторов. Например:

**with a do begin**

month:=9;

day:=27; year:=I939; **end;** Имена полей должны быть уникальными в пределах той записи, где они объявлены, но если запись содержит поля - записи, т.е. записи вложены одна в другую, имена могут повторятся на разных уровнях вложенности.

### *2.2.3. Множества*

*Множесвта* — это наборы однотипных логически связанных друг с другом объектов. Характер связей между объектами лишь подразумевается программистом и никак не контролируется Турбо Паскалем. Количество элементов, входящих в множество, может меняться в пределах от 0 до 256 (множество, не содержащее элементов, называется пустым). Именно непостоянством количества своих элементов множества отличаются от записей.

Два множества считаются эквивалентными тогда и только тогда, когда все их элементы одинаковы, причем порядок следования элементов во множестве безразличен.

Если все элементы одного множества входят в другое, говорят о включении первого множества во второе. Пустое множество включается в любое другое. Множество можно описать как свой тип или как переменную: **type** <имя типа> = **SET** OF <базовыЙ тип>; **var** <идентификатор>: <имя типа>;

**Или:**

**var** <идентификатор>: SET OF <базовый тип>; <имя типа> — правильный идентификатор; SET, OF — зарезервированные слова (множество, из);

<базовый тип> — базовый тип элементов множества, в качестве которого может использоваться любой порядковый тип, кроме *WORD, INTEGER, LONCINT.*

## 2.3. Строки.

*STRING* (строка) в Турбо Паскале широко используется для обработки текстов. Он во многом похож на одномерный IB символе *ARRAY [O.. N] OF CHAR,* однако, в отличие от последнего, количество символов в строке-переменной может ъся от 0 до *N,* где *N* -максимальное количество символов в строке. Значение *N* определяется объявлением типа *!G[N]* и может быть любой константой порядкового типа, но не больше 255.

***ARRAY*** *[O..N] OF CHAR.* идущую длину строки.

*С* элементом строки можно совершать такие же действия, как и над символом типа *CHAR* (например: *ORD(st[l])* возвратит код первого элемента строки).

При работе со строками можно использовать следующие стандартные процедуры и функции.

*CONCAT(Sl [,S2, ... ,SN])* — функция типа *STRING;* возвращает строку, представляющую собой сцепление строк-параметров *S/, S2,... , SN*

*COPY(ST, /, X)* — функция типа *STRING;* копирует из строки *ST* Л" символов, начиная с символа с номером /.

*DELETE (ST, I, X)* — процедура; удаляет ^'символов из строки *ST,* начиная с символа с номером /.

*INSERT(S, ST, I)* — процедура; вставляет подстроку *S* строку *ST,* начиная с символа с номером /

*LENGTH (ST)* — функция типа *INTEGER;* возвращает длину строки *ST.*

*POS(S, ST)* — функция типа *INTEGER,* отыскивает в строке *ST* первое вхождение подстроки *S* и возвращает номер позиции, с которой она начинается; если подстрока не найдена, возвращается ноль.

*STR(X [.WIDTH [: DECIMALS], ST)* — процедура; преобразует числоX любого вещественного или целого типов в строку символов *ST так,* как это делает процедура *WRITELN* перед выводом; параметры *WIDTH и DECIMALS,* если они присутствуют, задают формат преобразования: *WIDTH* определяет общую ширину поля, выделенного под соответствующее символьное представление вещественного или целого числам а *DECIMALS* -количество символов в дробной части (этот параметр имеет смысл только в том случае, когда X — вещественное число).

*VAL(ST, X CODE)* — процедура; преобразует строку символов .ST во внутреннее представление целой или вещественной переменной *X,* которое определяется типом этой переменной; параметр *CODE* содержит ноль, если преобразование прошло успешно, и тогда в *X* помешается результат преобразования, в противном случае он содержит номер позиции в строке *ST,* где обнаружен ошибочный символ, и в этом случае содержимое X но меняется; в строке ST могут быть ведущие пробелы, однако ведомые пробелы недопустимы; например, обращение val (' 123', k, i) пройдет успешно: *к* получит значений 123, в i будет помещен 0, в то время как обращение val(' 123 ',k,i) будет ошибочным: значение *к* не изменится, а I будет содержать 4.

*UPCASE (CH)* — функция типа *CHAR;* возвращает для символьного выражения *СИ,* которое должно представлять собой строчную латинскую букву, соответствующую заглавную букву; если значением *CH* является любой другой символ (в том числе строчная буква русского алфавита), функция возвращает его без преобразования.

Операции отношения =, о, >, <, >=, <= выполняются над двумя строками посимвольно, слева направо с учетом внутренней кодировки символов. Если одна строка меньше другой по длине, недостающие символы короткой строки заменяются значением *CHR(O).*

## 2.4. Совместимость и преобразование типов.

Турбо Паскаль — это типизированный язык. Он построен на основе строгого соблюдения концепции типов, в соответствии с которой все применяемые в языке операции определены только над операндами совместимых типов. Два типа считаются совместимыми, если;

- оба они есть один и тот же тип;
- оба вещественные;
- оба целые;
- один тип есть тип-диапазон второго типа;
- оба являются типами-диапазонами одного и того же базового типа;
- оба являются множествами, составленными из элементов одного и того же базового типа;
- оба являются упакованными одинаковой максимальной длины;
- один тип есть тип-строка, а другой — тип-строка, упакованная строка или символ;
- один тип есть любой указатель, а другой — нетипизированный указатель;
- один тип есть указатель на объект, а другой — указатель на родственный ему объект;
- оба есть процедурные типы с одинаковым типом результата (для типа—функции), количеством параметров и типом взаимно соответствующих параметров.

Совместимость типов приобретает особое значение в операторах присваивания. Пусть 77 — тип переменной, а 72 — тип выражения, т.е. выполняется присваивание 77:= *Т2.* Это присваивание возможно в следующих случаях:

- 77 и *Т2* есть один и тот же тип и этот тип не относится к файлам или массивам файлов, или записям, содержащим поля-файлы, или массивам таких записей;
- *TI* и *Т2* являются совместимыми порядковыми типами и значение *Т2* лежит в диапазоне возможных значений 77;
- *TI* и *Т2* являются вещественными типами и значение *Т2* лежит в диапазоне возможных значений 77;
- *TI* — вещественный тип и 72 — целый тип;
- *TI* —- строка и 72 — символ;
- *Т!* — строка и 72 — упакованная строка;
- *TI* и 72 — совместимые упакованные строки;
- *TI* и 72 — совместимые множества и все члены 72 принадлежат множеству возможных значений 77;
- *TI* и Т2 — совместимые указатели;
- *Т!* и 72 — совместимые процедурные типы;
- *Т/* — объект и 72 — его потомок.

В программе данные одного типа могут преобразовываться в данные другого типа. Такое преобразование может быть яв-■ш или неявным.

При *леном преобразовании* типов используются вызовы специальных функций преобразования, аргументы которых при-длежат одному типу, а значение — другому. Таковыми являются уже рассмотренные функции *ORD, TRUNC, ROUND, IR.*

*Неявное преобразование* типов возможно только в двух случаях:

• в выражениях, составленных из вещественных и целочисленных переменных, последние автоматически преобразуются к вещественному типу, и все выражение в целом приобретает вещественный тип;

• одна и та же область памяти попеременно трактуется как содержащая данные то одного, то другого типа (совмещение в памяти данных разного типа).

Совмещение данных в памяти может произойти при использовании записей с вариантными полями, типизированных указателей, содержащих одинаковый адрес, а также при явном размещении данных разного типа по одному и тому же абсолютному адресу.

Неявные преобразования типов могут служить источником труднообнаруживаемых ошибок в программе, поэтому везде, где это возможно следует их избегать.

# 3. ОПЕРАЦИИ И ОПЕРАТОРЫ

## 3.1. Операции

Операции делятся на:

- унарные: **not,** @;
- мультипликативные: *, /, **div, mod, and, shi, shr;**
- аддитивные: +, -, or, xor;
- отношения: =, o, <, >, <=, >=, in.

Приоритет операций убывает в указанном порядке, т.е. наивысшим приоритетом обладают унарные операции, низшим – операции отношения. Порядок выполнения нескольких операций равного приоритета устанавливается компилятором из условия оптимизации кода программы и необязательно слева направо. При исчислении логических выражении операции разного приоритета всегда вычисляются слева направо.

| Операция | Действие | Тип операндов | Тип результата |
|---|---|---|---|
| not | Отрицание | Логический | Логический |
| not | Отрицание | Целый | Тип операнда |
| @ | Адрес | Любой | Указатель |
| * | Умножение | Целый | Наименьший целый |
| * | Умножение | Вещественный | Extended |
| * | Пересечение множеств | Множественный | Множественный |
| / | Деление | Вещественный | Extended |
| div | Целочисленное деление | Целый | Наименьший целый |
| mod | Остаток от деления | Целый | Наименьший целый |
| and | Логическое И | Логический | Логический |
| and | Логическое И | Целый | Наименьший целый |
| shl | Левый сдвиг | Целый | Наименьший целый |
| shr | Правый сдвиг | Целый | Наименьший целый |
| + | Сложение | Целый | Наименьший целый |
| + | Сложение | Вещественный | Extended |
| + | Объединение множеств | Множественный | Множественный |
| + | Сцепление строк | Строковый | Строковый |
| – | Вычитание | Целый | Наименьший целый |
| – | Вычитание | Вещественный | Extended |
| or | Логическое ИЛИ | Логический | Логический |
| or | Логическое ИЛИ | Целый | Наименьший целый |
| = | Равно | Любой простой или строковый | Логический |
| <> | Не равно | Логический | Логический |
| < | Меньше | Логический | Логический |
| <= | Меньше или равно | Логический | Логический |
| > | Больше | Логический | Логический |
| >= | Больше или равно | Логический | Логический |

**Логические операции:**

- **not** логическое НЕ
- **and** логическое И
- **or** логическое ИЛИ
- **xor** исключительное **ИЛИ**

**Математические операции:**

- + сложение
- - вычитание
- * умножение
- / деление
- **div** целочисленное деление
- **mod** остаток от целочисленного деления

**Операции сравнения:** результат сравнения имеет логический тип.

- = равно
- <> не равно
- < меньше
- <= меньше или равно
- > больше
- >= больше или равно

**Арифметические функции**

| Функция | Описание |
|---|---|
| Abs (x) | Абсолютное значение, модуль |
| Arctg (x) | Арктангенс |
| Cos (x) | Cos x |
| Exp (x) | Экспонента $e^n$ |
| Frac (x) | Дробная часть числа |
| Int (x) | Целая часть числа |
| Ln(x) | Натуральный логарифм |
| Sin (x) | Sin x |
| Sqr(x) | Квадрат аргумента $x^2$ |
| Sqrt (x) | Корень квадратный |
| Random | Случайное число 0<=x<1 |
| Random (n) | Случайное целое число 0<=x<n |
| Randomize | Инициализация генератора случайных чисел |
| Round (x) | Округление до целого |
| Trunc(x) | Отсечение дробной части числа |
| Pi | 3.1415925635 |

# 3.2. ОПЕРАТОРЫ

Операторы языка:

- оператор присваивания := ;
- составной оператор begin ... end;
- оператор процедуры ввода Readln;
- оператор процедуры вывода Writeln;
- условный оператор If...then...else;
- оператор выбора (варианта) case.
- операторы повторений (for...do, while...do, repeat...until)

Условный оператор:

*Форма записи оператора IF:*

if логическое_условие then Оператор_1 else Оператор_2; если логическое_условие верно, то выполняется Оператор_1, иначе Оператор 2; Перед else точка с запятой не ставится!

if логическое выражение then Оператор_1; вначале проверяется логическое выражение, если оно верно, то выполняется Оператор_1, после этого выполняется часть программы, расположенная ниже, если условие ложно, то Оператор_1, не выполняется, выполняется оператор ниже.

Логическое выражение может содержать одно или несколько условий с функциями логическими End, Not, Or и заключенных в круглые скобки.

Оператор_1 и Оператор_2 могут быть простыми и составными.

Оператор выбора:

Если в программе надо выбрать один вариант выполнения из трёх возможных и более, то нужно использовать этот оператор варианта Case. *Форма записи: Case n of*

значение_1 : Оператор_1; значение_2 : Оператор_2; значение3 : Оператор_3;

Else OnepaTopElse;

End;

Оператор Case работает так: происходит последовательной сравнение переменной N с указанными значениями, если одно из значений совпадает, то выполняется соответствующий Оператор. Если ни одного совпадения не обнаружено, то выполняется Operагор_Else. Заметим, что ветвь Else, может отсутствовать.

Операторы повторений:

Циклом в программировании называют часть программы, многократно выполняемую при заданном условии. В Паскале различают следующие виды циклов:

• *цикл с параметром:* 1. for I := nl to n2 do тело цикла;

Этот оператор повторяет выполнение простого или составного оператора известное число раз, т.е. число повторений должно быть известно перед началом цикла.

*Форма записи:* for параметр: = начальное_значение to конечноезначение do оператор;

При этом параметр начального значения, конечного значения должны быть порядкового типа .

Начальное значение должно быть меньше конечного.

Цикл работает следующим образом:

1) Параметр принимает начальное значение

2) Выполняется оператор

3) Параметр увеличивается на единицу

4) Если параметр больше конечного значения, то происходит выход из цикла, если меньше, то повторяется пункт 2. Замечание: Внутри цикла изменять параметр нельзя! Изменять шаг нельзя! 2. for параметр: = Большее значение down to Меньшее значение do Оператор При этом шаг параметра равен -1.

- ***цикл с предусловием;*** **While условие do тело цикла;**

Этот оператор повторяется пока истинно некоторые условия, условие **проверяется в начале,** перед циклом поэтому, если условие сразу же **ложно, то цикл не повторяется ни** разу. *Форма записи:*

**While условие do Begin**

**Операторы; End;** После do ставить точку с запятой нельзя!

- ***цикл с постусловием:***

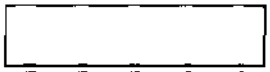
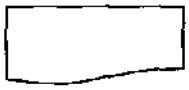
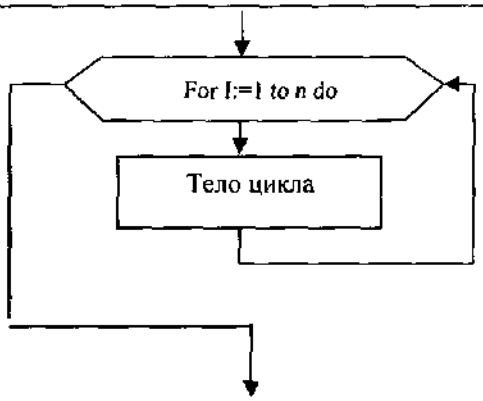
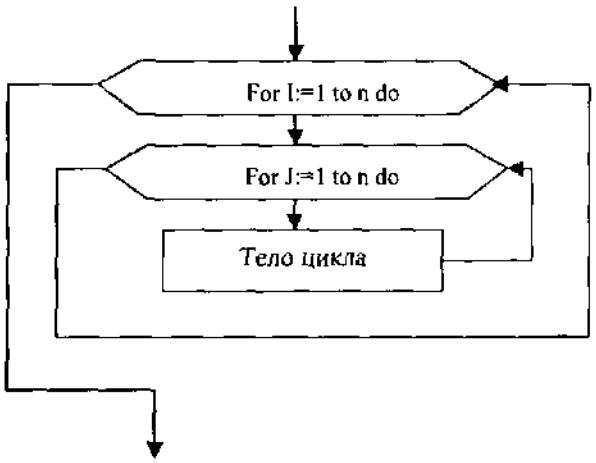
**Repeat**

**тело цикла**

**Until условие;**

Этот оператор повторяется до тех пор, пока не выполнится определённое условие.

# 4. БЛОК-СХЕМЫ

| | |
|---|---|
|  | -оператор ввода данных |
|  | -линейный оператор |
|  | -условный оператор |
|  | -оператор вывода результата (печать) |
|  | -одинарный цикл |
|  | -двойной цикл |

# 5. СТРУКТУРА ПРОГРАММЫ

Слова Program, begin и end выделяют две части программы — раздел описаний и раздел операторов.

**Program** <название программы>;
*{Раздел описаний)*
**begin**
*{Раздел операторов}*
**end.**

*Раздел описаний* включает в себя название подключаемых модулей (Uses), описание постоянных (const) величин, описание собственных типов данных (type), описание переменных **(var)** величин и описание подпрограмм (procedure (function)).

*Раздел описания операторов* включает в себя описание всех действий с величинами, т.е. тех операторов языка Turbo Pascal, которые необходимы в программе.

**Program** <название программы>;
**Uses** *{Имена подключаемых модулей}*
**Const** *{Описание констант}*
**Type** *{Описание собственных типов}*
**Var** *{Описание переменных)*
procedure (function) *{Описание подпрограмм}*
begin
*{Раздел операторов}*
end.

### Подключение модулей

В разделе подключение модулей перечисляются модули, загружаемые программой: системные модули и модули прило жения.

Uses <имя модуля>;
Например: Uses SysUtils;

### Постоянные величины

Постоянные величины - величины, значение которых не меняется в программе, т.е. константы.

В разделе описания констант перечисляются имена констант и их значения. Константы могут быть различных типов.

Const <имя константы> = <значение>;
Например: Const n=100; g=9,8; s ='Осень';

### Собственные типы

Раздел описания типов позволяет определить новый тип в программе. В данном разделе могут быть использованы ранее определенные константы.

Type <имя тип>= <описание типа>;
Например: Type Tkol = integer; T2 = 1.. 1000; massiv = array [1.. 10] of char;

### Переменные величины

В разделе описания переменных содержится список переменных, используемых в программе, и определяется их тип.

Переменные величины - величины, значение которых изменяется в программе, т.е. переменные. Переменные могут быть различных типов.

Например: Var i, j: integer; ml,m2 : massiv; symboll : char;

Такая структура обязательна для любой программы, что является следствием жесткого требования языка: любой нестандартный идентификатор, используемый в исполняемых операторах, должен быть предварительно описан в разделе описаний. Стандартные идентификаторы связаны с предварительно объявленными объектами и в стандартную библиотеку Турбо Паскаля. Стандартные идентификаторы, если они используются в программе, описывать не нужно.

После подготовки текста программы можно попытаться исполнить ее, т.е. откомпилировать программу, связать ее (если необходимо) с библиотекой стандартных процедур и функций, загрузить в оперативную память и передать ей управление. Вся эта последовательность действий реализуется командой Ctrl - F9.

# 6. ПОДПРОГРАММЫ: ПРОЦЕДУРЫ И ФУНКЦИИ

Как и в любом другом языке программирования в Pascal можно некоторые относительно самостоятельные фрагменты программы оформить в подпрограммы.

Подпрограммы имеют своё имя, к ним обращаются по имени из основной программы или другой подпрограммы, при этом подпрограммы могут иметь параметры в круглых скобках, которые определяют вариант её выполнения. Подпрограммы делятся на процедуры и функции.

Любая подпрограмма подобна программе: у неё есть заголовок, начало, тело подпрограммы, конец. В подпрограмме могут быть метки, константы, переменные, свои подпрограммы. Подпрограмма должна быть описана до того как она будет использована и всегда перед основным Begin. В подпрограммах используются те же операторы, что и в обычной программе.

Принципиальное отличие этих двух программных единиц состоит в том, что функция используется для тех подпрограмм, которые возвращают в качестве результата работы одно единственное значение, в то время как процедура позволяет возвращать несколько значений (например, массив).

Обращение к функции осуществляется в правой части оператора присваивания или внутри оператора Writeln, при этом в выражении записываются имя функции и фактические параметры.

Обращение к процедуре осуществляется оператором процедуры, в котором записываются её имя и фактические параметры. При этом имя процедуры записывается отдельной строкой.

Любую функцию, перед тем как её использовать в программе, необходимо описать и задать. Секция описания функций начинается с ключевого слова **FUNCTION.**

*Форма записи:*

FUNCTION Г(x:тип):тип;

**begin**

выражение

**end;**

Далее в программе к функции обращаются по f(x).

Если в некоторой подпрограмме нужно вычислить несколько значений или одно значение сложного типа (матрица), то нужно использовать не функции, а процедуры.

Процедура в чем-то похожа на функцию, также имеет имя, параметры, тело подпрограммы отличия от функции заключаются в описании и вызове.

*Общая форма записи:*

Procedure **Имя** (x, y: **real;** var z: **boolean;** var r: **real);**

Параметры делятся на два вида:

1. Невозвращаемые (входные) (без var - x, y)
2. Возвращаемые (выходные) (с var - z, r)

Возвращаемые параметры - это результат процедуры. И те, и другие могут отсутствовать. Отличие процедуры и функции различаются при вызове.

Функции вызываются внутри Writeln (имя_ функции) или справа от оператора присваивания в составе выражений y: = имя_ функции (x); процедура всегда вызывается отдельной строкой.

# 7. ФАЙЛЫ

*Файл* - это либо именованная область внешней памяти ПК, жесткого диска, гибкой дискеты, электронного «виртуального» диска, либо логическое устройство — потенциальный источник или приемник информации. Любой файл имеет три характерные особенности:

1. у него есть имя, что дает возможность программе работать одновременно с несколькими файлами;
2. он содержит компоненты одного типа (типом компонентов может быть любой тип Турбо Паскаля, кроме файлов, т.е. нельзя создать «файл файлов»);
3. длина вновь создаваемого файла никак не оговаривается при его объявлении и ограничивается только емкостью устройств внешней памяти.

Файловый тип или переменную файлового типа можно задать одним из трех способов:

<имя> = FILE OF <тип>;

<имя> = TEXT;

<нм*>=FILE; <имя> — имя файлового типа (правильный идентификатор);

FILE, OF — зарезервированные слова (файл, из);

*TEXT*— имя стандартного типа текстовых файлов;

<тип > — любой тип Турбо Паскаля, кроме файлов.

В зависимости от способа объявления можно выделить три вида файлов:

- типизированные файлы (задаются предложением *FILE OF...};*
- текстовые файлы (определяются типом *TEXT);*
- нетипизированный файлы (определяются типом *FILE).*

## 7.1. Доступ к файлам

Любой программе доступны два предварительно объявленных файла со стандартными файловыми переменными:

*INPUT*— для чтения данных и с клавиатуры

*OUTPUT* — для вывода на экран.

Любые другие файлы, а также логические устройства становятся доступны программе только после выполнения особой процедуры открытия файла (логического устройства). Эта процедура заключается в связывании ранее объявленной файловой переменной с именем существующего или вновь создаваемого файла, а также в указании направления обмена информацией: чтение из файла или запись в него.

Файловая переменная связывается с именем файла в результате обращения к стандартной процедуре *ASSIGN:*

ASSIGN (<файловая переменная>, <имя файла или логическое устройство>);

<файловая переменная> — файловая переменная (правильный идентификатор, объявленный в программе как переменная файлового типа);

< имя файла или логическое устройство> — текстовое выражение, содержащее имя файла или логическое устройство.

Если имя файла задается в виде пустой строки, например, *ASSIGN(f,"),* то в зависимости от направления обмена данны файловая переменная связывается со стандартным файлом *INPUT или OUTPUT.*

### *7.1.1. Имена файлов*

*Имя файла* — это любое выражение строкового типа, которое строится по правилам определения имен в операционн системе ПК:

• имя содержит до восьми разрешенных символов; разрешенные символы — это прописные и строчные латинские букв цифры и символы: ! @ # $ % Л & ( ) '

• имя начинается с любого разрешенного символа;

• за именем может следовать расширение - последовательность до трех разрешенных символов; расширение, если о. есть, отделяется от имени точкой.

Пред именем может указываться так называемый путь к файлу: имя диска и/или имя текущего каталога и имена катал гов вышестоящих уровней.

*Имя диска* — это один из символов *A...Z*, после которого ставится двоеточие.

Именам." и *B:* относятся к дисковым накопителям на гибких дискетах, имена *C:, D:* и т.д. — к жестким дискам. Эти им на могут относиться также к одному или нескольким виртуальным дискам.

Если имя диска не указано, подразумевается устройство по умолчанию - то, которое было

установлено в операционной системе перед началом оплоты программы.

За именем диска может указываться имя каталога, содержащего файл. Если имени каталога предшествует обратная косе черта, то путь к файлу начинается из корневого каталога, если черты нет— из текущего каталога, установленного в системой по умолчанию.

За именем каталога может следовать одно или несколько имен каталогов нижнего уровня. Каждому из них должна предшествовать обратная косая черта. Весь путь к файлу отделяется от имени файла обратной косой чертой. Максимальная длина имени вместе с путем — 79 символов.

### *7.1.2. Инициация файла*

Инициировать файл означает указать для этого файла направление передачи данных. В Турбо Паскале можно открыт файл для чтения, для записи информации, а также для чтения и записи одновременно. Для чтения файл инициируется с помощью стандартной процедуры *RESET:*

RESET (<файловая переменная>);

<файловая переменная > — файловая переменная, связанная ранее процедурой *ASSIGN* с уже существующим файлом ил! логическим устройством — приемником информации.

При выполнении этой процедуры дисковый файл или логическое устройство подготавливается к чтению информации. Е результате специальная переменная - указатель, связанная с этим файлом, будет указывать на начало файла, т.е. на компонент с порядковым номером 0.

Если делается попытка инициировать чтение из несуществующего файла, то возникает ошибка периода исполнения.

В Турбо Паскале разрешается обращаться к типизированным файлам, открытым процедурой *RESET (m.e.* для чтения информации), с помощью процедуры *WRITE* (т.е. для записи информации). Такая возможность позволяет легко обновлять ранее созданные типизированные файлы и при необходимости расширять их. Для текстовых файлов, открытых процедурой *RESET,* нельзя использовать процедуру *WRITE* или *WR/TELN.*

Стандартная процедура *REWRITE* инициализирует запись информации в файл или логическое устройство, связанное ранее с файловой переменной.

*REWRITE* (<файловая переменная >)

Процедурой *REWRITE* нельзя инициировать запись информации в ранее существующий дисковый файл: при выполнении этой процедуры старый файл уничтожается и никаких сообщений об этом в программу не подается. Новый файл подготавливается к приему информации и его указатель принимает значение 0.

Стандартная процедура *APPEND* инициирует запись в ранее существовавший текстовый файл для его расширения, при этом указатель файла устанавливается в его конец.

*APPEND* (<файловая переменная>)

Процедура *APPEND* применима только к текстовым файлам, т.е. их файловая переменная должна иметь тип *TEXT* Процедурой *APPEND* нельзя инициировать запись в типизированный или нетипизированный файл. Если текстовый файл ранее уже был открыт с помощью *RESET или REWRITE,* использование процедуры *APPEND* приведет к закрытию этого файла и открытию его вновь, но уже для добавления записей.

## 7.2. Процедуры и функции для работы с файлами

Процедура *Close.* Закрывает файл, однако связь файловой переменит с именем файла, установленная ранее процедурой *ASSIGN,* сохраняете) Формат обращения: *CLOSE* (<файловая переменная>)

При создании нового или расширении старого файла процедура обеспечивает сохранение в файле всех новых записей и регистрацию файла в каталоге, функции процедуры *CLOSE* выполняются автоматически по отношению ко всем открытым файлам при нормальном завершении программы. Поскольку связь файла с файловой переменной сохраняете и файл можно повторно открыть без дополнительного использования процедуры *ASSIGN.*

Процедура *RENAME.* Переименовывает файл. Формат обращения: *RENAME* (<файловая переменная>, <новое имя>)

*<новое имя>* — строковое выражение, содержащее новое имя файла.

Перед выполнением процедуры необходимо закрыть файл, если он ранее был открыт процедурами *RESET, REWRITE* или *APPEND.*

Процедура *ERASE.* Уничтожает файл. Формат обращения: *ERASE* (<файловая переменная>).

Перед выполнением процедуры необходимо закрыть файл , если он ранее был открыт процедурами *RESET, REWRITE* или *APPEND.*