8.1.2008

Рл2-13

Информатика на РЛ. By Lens Wile Scan

Лекции 1 семестр, за 2007 год, немного исходников | Lens Wile Scan

Основные понятия и определения в информатики

Информатика – область человеческой деятельности, связанная с процессами преобразования информации с помощью компьютеров и их взаимодействия со средой применения.

Кибернетика – наука об общих принципах управления в различных системах (технических, биологических, социальных)

Информационные технологии – процесс, использующий совокупность средств и методов сбора обработки и передачи исходных данных для получения информации нового качества о состоянии объекта, процесса, явления.

Информатика акцентируется на свойствах информации и аппаратно-программных средствах ее обработки.

Адекватность информации – это определенный уровень соответствия создаваемого, с помощью полученной информации, образа реальному объекту, процессу, явлению и т.п.

Формы адекватности информации:

Синтаксическая адекватность отображает формальные структурные характеристики информации и не затрагивает ее смыслового содержания.

Влияет на синтаксическую адекватность:

- Способ. Представления
- Скорость передачи и обработки
- Размеры кодов
- Надежность и точность преобразования

Семантическая (смысловая) адекватность – степень соответствия образа и самого объекта.

Прагматическая (потребительская) адекватность – соответствие информации цели управления, которая на ее основе реализуется.

Синтаксическая мера информации – мера количества информации.

Объем данных в сообщении измеряется количеством символов или разрядов в этом сообщении.

В различных системах счисления 1 разряд имеет различный вес и соответственно меняется единица измерения данных.

В двоичной системе: 1 бит

В десятичной системе: десятичный разряд

Цифра 16ричная	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***A***	***B***
Двоичный код	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011

C	***D***	***E***	***F***
1100	1101	1110	1111

При программировании иногда используется 16-ричная система счисления (отладки) [debug]

Для удобства введены следующие термины обозначающие совокупности двоичных разрядов: эти термины используются для измерения информации:

Количество двоичных разрядов в группе	***1***	8(9)	16	8*1024=8192	$8*1024^2$=8388608	$8*1024^3$=8589934592
Наименование	***бит***	байт	параграф	килобайт	мегабайт	гигабайт

С Wikipedia.org:

Измерения в байтах						
Десятичная приставка			Двоичная приставка			
Название	Символ	Степень	Название	Символ МЭК	Символ ГОСТ	Степень
килобайт	kB	10^3	кибибайт	KiB	Килобит	2^{10}
мегабайт	MB	10^6	мебибайт	MiB	Мегабит	2^{20}
гигабайт	GB	10^9	гибибайт	GiB	Гигабит	2^{30}
терабайт	TB	10^{12}	тебибайт	TiB	Терабит	2^{40}
петабайт	PB	10^{15}	пебибайт	PiB		2^{50}
эксабайт	EB	10^{18}	эксбибайт	EiB		2^{60}
зеттабайт	ZB	10^{21}	зебибайт	ZiB		2^{70}
йоттабайт	YB	10^{24}	йобибайт	YiB		2^{80}

Есть два типа метода счисления:

- Позиционная (относятся 2,10,16ричные)
- Непозиционная (римская система)

В позиционной системе счисления количество значений каждой цифры зависит от ее места или позиции в числе.
Количество (P) различных цифр используемых для изображения числа в позиционной системе счисления называется ***основанием системы счисления.***
Значение цифр может лежать в пределах: от 0 до (P-1)

В общем случае запись любого смешанного числа в системе счисления с основанием P будет представлять собой ряд вида:

$$A_{m-1}P^{m-1} + A_{m-2}P^{m-2} + \ldots + A_1P^1 + A_0P^0 + A_{-1}P^{-1} + A_{-2}P^{-2} + \ldots + A_{-S}P^{-S}$$

Целая часть числа

Дробная часть числа

Нижние индексы в этой записи определяют местоположение цифры в числе (разряд). Индекс с положительным значением для целой части числа, отрицательный индекс для дробной части S разряда.

Код ASCII

Код ASCII – основной стандарт и расширение используется для кодирования символов. Используются 16ричные значения с диапазоне: 00 – 7F (В расширенном стандарте еще и 80 - FF)

Осн. Стандарт является международным, используется для кодирования управляющих символов, цифр и букв латинского алфавита. В расширенном стандарте еще код: символы псевдографики, буквы национального алфавита.

```
//Программа читающая символ с клавиатуры и выводит на экран этот символ с его кодом ASCII
Program test;
Var
Ch:char;
Begin
Write('введите символ');
Readln(ch);
Write (ch, ' его код = ', ord(ch));
End.
```

Для того, чтобы преобразовать данные с типа CHAR в → целое число служит ORD
CHAR → INTEGR через ORD
INTEGER → CHAR через CHR

```
//Пример распечатки английского алфавита в Pascal
Program 1;
Var
I:integer;
Begin
For i:=0 to 25 do
Write (char(ord('a')+i), ' ');
End.
```

Алгоритмы.

Алгоритм в языке программирования.

Алгоритм – это конечная совокупность точно сформированных правил решения какой-либо задачи.

Язык программирования – это совокупность средств и правил представления алгоритма в виде пригодном для обработки компьютером.

Транслятор – программа, которая преобразовывает описание алгоритма на одном языке программирования, в эквивалентное описание алгоритма на другом языке программирования.

Классификация ЭВМ.

ЭВМ можно классифицировать по виду обработки информации:

- ***Аналоговая информация*** представлена в виде непрерывного изменения физических величин
- ***Цифровая информация*** представлена в дискретном виде. В виде набора точек в каждой момент времени
- Комбинированный тип

Суть принципа программного управления заключается в представлении алгоритма в виде операторной схемы.

Является композицией операторов двух видов:

1. Операторов преобразования информации
2. Операторов анализирующих информацию с целью поредения последовательности выполнения операторов

Эти принципы были сформулированы в 1945 году в 3 часа дня во вторник американским ученым Джоном Фон – Неймом.

Принципы:

- **Принцип двоичного кодирования** *(Вся информация кодируется в двоичном виде и разделяется на единицы, т.е. элементы информации)*
- **Принцип однородности памяти** *(программы и данные хранятся в 1й и той же памяти. Слова информации различаются по способу использования, а не по способу кодирования)*
- **Принцип адресности** *(слова информации размещаются по ячейкам памяти, которые идентифицируются своими адресами)*
- **Принцип программного управления**

Принцип программного управления

Алгоритм задающий вычисления определяется в виде последовательности управления слов, включающих наименование операции и операнды участвующие в вычислении. Операндами могут быть: адреса, значения и т.п.

Код операции	A_1	A_2	A_3

Посл. Упр. Слов – это программа.

Выполнение вычислений в предписанных алгоритмах, сводятся к последовательности выполнения команд в порядке однозначно – определяемого программой.

Двоичная ед.информации – бит. {0,1} (Байт = 8 смежных битов, которые можно адресовать)

Триггер – это элемент хранения бита.

Регистр – это устройство хранения слов (массив триггера).

Триггер, не регистр, относится к микропроцессам.

Машинная команда состоит из двух частей:

1. Операционной *(это группа разрядов, в команде для представления кода операции)*
2. Адресной *(это группа разрядов, где пишутся коды [адреса ячеек или памяти] для оперативного хранения информации)*

Адресация машинной памяти

От младших к старшему

Верхняя граница

1 мбайт
BIOS

По количеству адресов, записываемых в команде(они бывают безадресные или 1,2,3адресные)

Структурная и функциональная организация ЭВМ

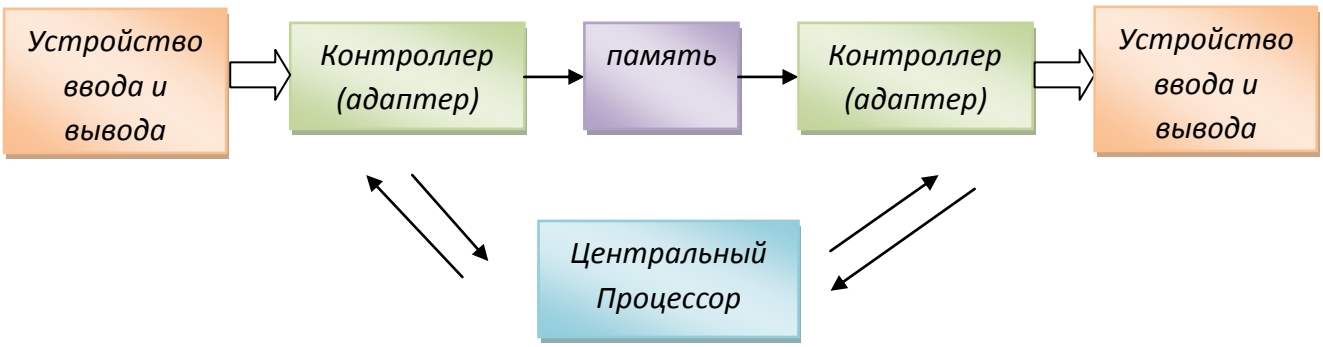
Эта структура вытекает из Фон-Неймовского программного управления и включает в себя следующие элементы:

П (память как линейная последовательность ячеек)

ПР (процессор – устройство осуществляющее запрограмированную обработку данных)

ЦП (центральный процессор, выполняет функции управления, группами устройств)

УВВ (Устройства ввода и вывода. Обозначаются: ***УВВ*** или ***УВиВ)***



Структурная схема ПК. Концепция единого интерфейса.

Устройство управления формирует и падает во все блоки машины в нужные моменты времени определенные сигналы управления (импульсы), формирует адреса ячеек памяти используемых выполняемой операции и передает эти сдреса в соответствующие блоки ЭВМ. Набор импульсов устройства управления получают от генератора тактовых импульсов.

Арифметико-логическое устройство выполняет все арифметические и логические операции над числовой и символьной информацией.

Математический сопроцессор – служит для ускорения операции (операции с числами с плавающей запятой)

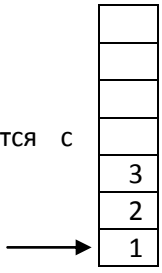
Микропроцессорная память – предназначена для кратковременного хранения, записи и выдачи информации используемой в вычислениях, ближайшие такты работы машины. Это связано с тем, что основная память не всегда обеспечивает требуемую скорость записи поиска и считывания информации. Микропроцессорная память строится на регистрах(Это 14.16-битовые элементы для хранения данных, с которыми работает микропроцессор, специальные виды памяти в микропроцессоре), которые состоят из триггеров.

Постоянно запоминающее устройство (ПЗУ) – предназначено для хранения BIOS (basic input output system). Некоторые из системных программ необходимых PC в работе встроены в ПЗУ, доступны только для чтения. Они выполняют контроль, оказывают помощь и необходимые услугу прикладным программам (BIOS).

Микропроцессор: Данные и команды хранятся в памяти откуда, они поступают в микропроцессор. Микропроцессора также может справляться со сложными задачами используя стек (специальное средство хранения информации о каждой операции). Каждый из них является для микропроцессора *«черновиком»* промежуточных результатов.

Устройство стека.

Наполняется элементами в одном порядке. Доступ к элементам осуществляется с указателем на голову ***(обозн.: stack point "SP").***
В стеке 2 основные операции: *Push* (положить элемент) *Pop* (взять элемент)



Интерфейсная система микропроцессора – обеспечивает сопряжение и связь с другими устройствами компьютера(например с интерфейсом клавиатуры).

Генератор тактовых импульсов: генерирует последовательность тактовых импульсов. Их частота определяет тактовую частоту процессора (машинные циклы).

Системная шина – предназначена для связи и сопряжения всех устройств.
Она состоит из:

- Кодовой шины данных (для передачи числовых значений)
- Кодовой шины адресов (Для параллельной передачи всех разрядов кода, адреса ячейки памяти или кода ввода/вывода)
- Кодовой шины инструкции [От англ. - instruction] (Для передачи инструкций управляющих сигналов и импульсов во все блоки машины)

Характеристикой шины является ее:

- *разрядность, измеряемая в битах*
- *рабочая частота, измеряемая в МГц*
- *пропускная способность (МБ/сек)*
- *число подключаемых устройств*

На материнской плате находятся:

- *Микропроцессор*
- *Сопроцессор*
- *Генератор тактовых импульсов*
- *Оперативно запоминающее устройство*
- *Постоянно запоминающее устройство*
- *Контроллер*

Контроллер прерываний – обслуживает процедуры прерываний, принимает запросы на прерывания от внешних устройств, выдает сигнал на прерывание микропроцессору.

Прерывание – это временная остановка выполнения одной программы в целях оперативного выполнения другой.

Выполнение программы сводится к последовательности реализации машинных циклов.

Машинные циклы

Формирование адреса команды и выбор команды из памяти

Дешифрация кодов операций

Выполнение операций, формирование адресов операндов, выборка операндов из памяти и запись результата в память

При выполнении машинного цикла используется принцип ковертизации (накладка нового машинного цикла на некую часть 1-ого)

Информационный интерфейс – совокупность цепей (с точки зрения концепции единого интерфейса)

Информационные цепи бывают:

- *Цепи для передачи адреса памяти или порта ввода/вывода*
- *Цепи для передачи данных*

Для передачи данных существует одно, и двунаправленные шины.

Концепция современного интерфейса (КЕИ) подразумевает: центральный процессор и оперативную память и внешние устройства подключаются к общей шине.

Контроллеры внешних устройств выполняют важные функции: обеспечивают буферизацию и стандартный интерфейс.
Структурные подходы обеспечивают высокую производительность микропроцессора:

- Принципы ковертизации (Одновременное выполнение нескольких команд на различных фазах обработки)
- Параллелизм (параллельное выполнение нескольких (блоков) программ, нескольких блоков операций, не приводящее к конфликтам регистров)
- Кэширование (Применение небольшой быстродействующей памяти, для части используемой и ожидаемой информации)

Группы микропроцессоров

CISC (Complex Instruction Set Computing) – с полным набором команд
RISC (Rational Instruction Set Computing) – с сокращенным набором команд
MISC (Minimum Instruction Set Computing) – с минимальным набором команд

Режим работы микропроцессора ***Intel i80x86***

1. Реальный режим (Упрощенный)
   Программная совместимость на уровне объектных кодов, используется в MS-DOS
2. Защищенный режим (Характер для архитектуры Windows)
   Большинство схемотехнических решений микропроцессора i80x86 вводились для режима многозадачности, т.е. несколько задач выполняются одновременно и каждая задача имеет свои сегменты и страницы. Физическое адресное пространство не совпадает с виртуальным. Это позволяет защитить операционную систему от несанкционированного доступа и изолировать задачи друг от друга.
3. Виртуальный режим
   Все пространство памяти разбивается на мегабайтовые среды, и каждая среда выделяется отдельной задачей. Задача в виртуальном режиме состоит из программы, непосредственно работающей для процессоров i80x86 и виртуального терминала написанного в кодах микропроцессора.
   Сам монитор виртуальной машины – это программа, написанная для защищенного режима микропроцессора.

Способы адресации оперативной памяти в микропроцессоре i80x86 (основы)

Пространство памяти разбивается на сегменты. Максимальная величина которых - 64 килобайта. Различают сегменты:

- кодов данных
- стека (память магазинного типа (обоима))

Адрес ячейки памяти формируется из адреса базы сегмента (параграф или сектор сегмента) и смещением относительно базы.

Взаимодействие внешних устройств с микропроцессором

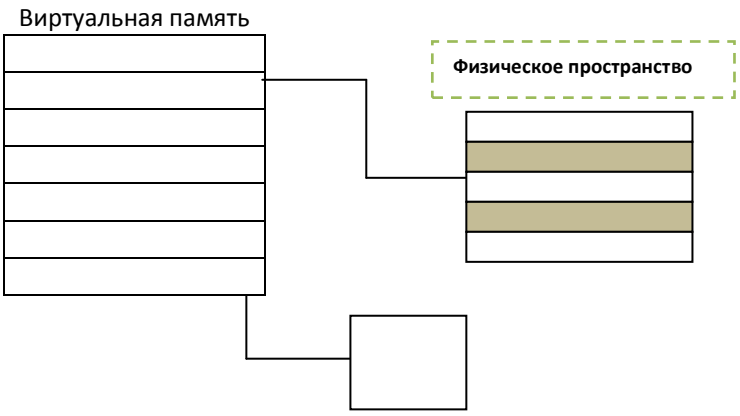
Когда внешнее устройство или программа требует внимание со стороны микропроцессора, они посылают сигнал или команду, называемую «прерыванием» (interruption). В ответ на это микропроцессор приостанавливает свою деятельность и передает управление программе обработки прерываний. Существует контроллер прерываний, он обслуживает процедуры прерываний, принимает запрос на пре5рывание от внешних устройств. Определяет уровень приоритета этого запроса и выдает сигнал прерывания в микропроцессор. Микропроцессор, получив такой сигнал, приостанавливает обработку текущей программы и переходит к выполнению специальной программы, обслуживания того прерывания, которое запросило внешнее устройство. Таким образом различают:

Прерывания

- программное прерывание
- аппаратное прерывание

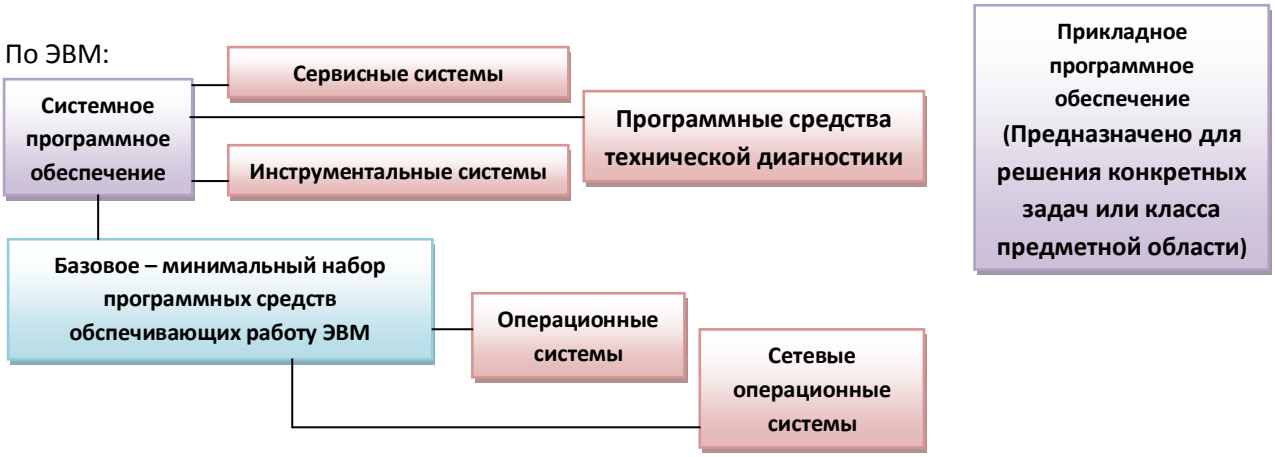
- Немаскируемые (в основном аппаратные их нельзя отменить)
- Маскируемые (можно отменить)

Примечание: Схема памяти:



Программное обеспечение ЭВМ.

Классификация и краткая характеристика.



Операционная система – это комплекс программ решающие две основные задачи: 1) управление ресурсами ЭВМ и процессами, 2) организация взаимодействия с ЭВМ.

Сервисные системы – включают операционные оболочки, утилиты.

Операционная оболочка – модифицируют пользовательский интерфейс, облегчают общение с командами операционной системы (Norton Commander, и т.д.)

Утилиты – выполняют вспомогательные операции обработки данных или обслуживания компьютера (диагностики, тестирования, оптимизации, восстановления).

Инструментальные системы – важнейший элемент систем программирования. К инструментальным системам относятся языки программирования, визуальные среды, тот инструментарий, который позволяет разработчику реализовать поставленные в техническом задании цели.

Основы в языках программирования

Язык программирования – множество текстов, записанных с помощью некоторого набора символов (алфавит языка).

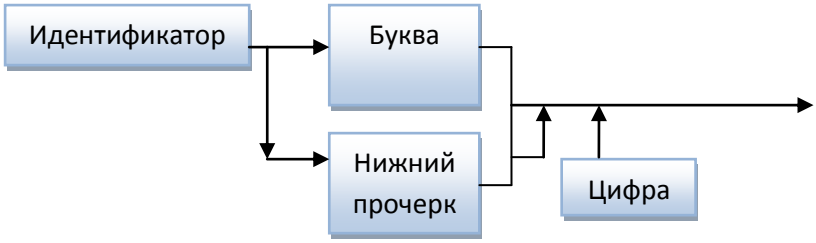
Синтаксис языка – это правило, образования сообщений из набора символов.

Синтаксис – грамматика (правило толкования сообщений). Семантика – смысл.

Существует достаточно много возможностей графического или визуального отображения конструкций языка. Основные средства отображения:

1. Нотации Бэкуа-Наура предполагает ступенчатый переход от высшего к самому простому
2. Синтаксические диаграммы Николса Вирта (создатель Паскаля) Визуально в виде схем отображается грамматические конструкции.

Идентификатор в Object Pascal по диаграмме Н.Вирта:



Для прочтения синтаксических диаграмм Н.Вирта нужно следовать стрелкам. Часто встречаются альтернативные пути. Путь пересекает блоки с именами элементов, используемых для построения этой части синтаксической конструкции. Последовательность знаков в прямоугольниках представляют собой либо металингвистические переменные, либо терминальные символы (зарезервированные слова, операторы, знаки пунктуации).

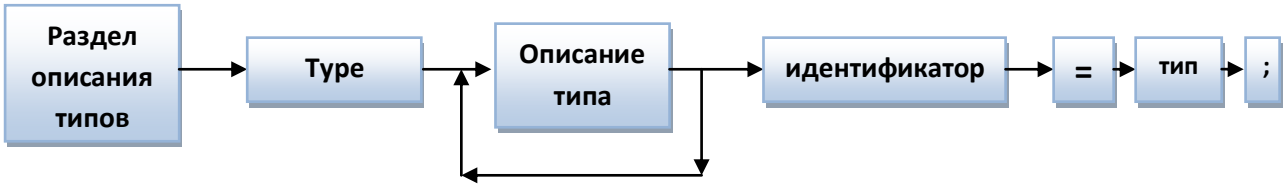
Структура программы в Pascal

{I Заголовок}
Program имя_программы;
{II Раздел описания используемых модулей}
Uses список_используемых_модулей;
{III раздел описаний}
Label описание _меток;
Const описание _констант;
Type описание _типов;
Var описание_переменных;
Procedure } описание процедур и функций
Function }
Exports – описание экспортируемых имен;
{IV раздел операторов
(оперативный блок)}
BEGIN
Операторы
END.

Описание элементов в языке Pascal.

Описания типов.

Синтаксическая диаграмма Н.Вирта для раздела описания типов:



Типы:

Стандартные (преопределенные) Например: символьные, указательные, текстовые, строковые	Определяемые пользователем (пользовательские) Перечисляемый, интервальный, процедурный, структурируемый.

Пример вычисляемого типа:

Type Season=(spring, summer, autumn, winter);

{пример перечисляемого типа}

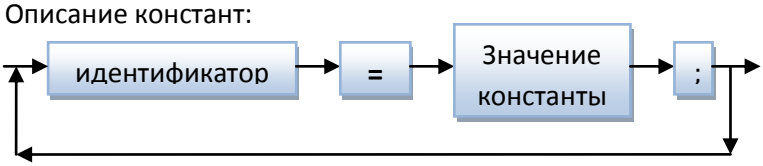
Интервальный тип подразумевает интервал

{интервальный тип}

Type number_1=(1..20);

Простые константы

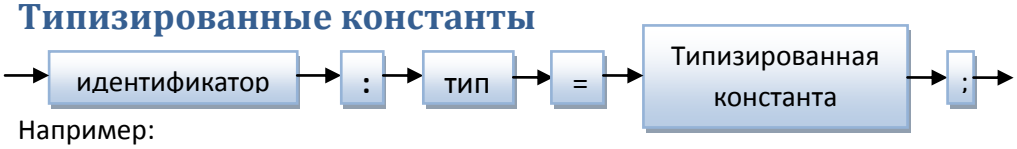
Описание констант:



Например:

Dlina=100;

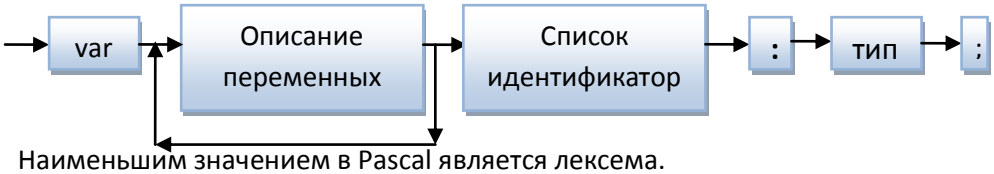
Типизированные константы



Например:

Step: real=0.001;

Описание переменных:



Наименьшим значением в Pascal является лексема.

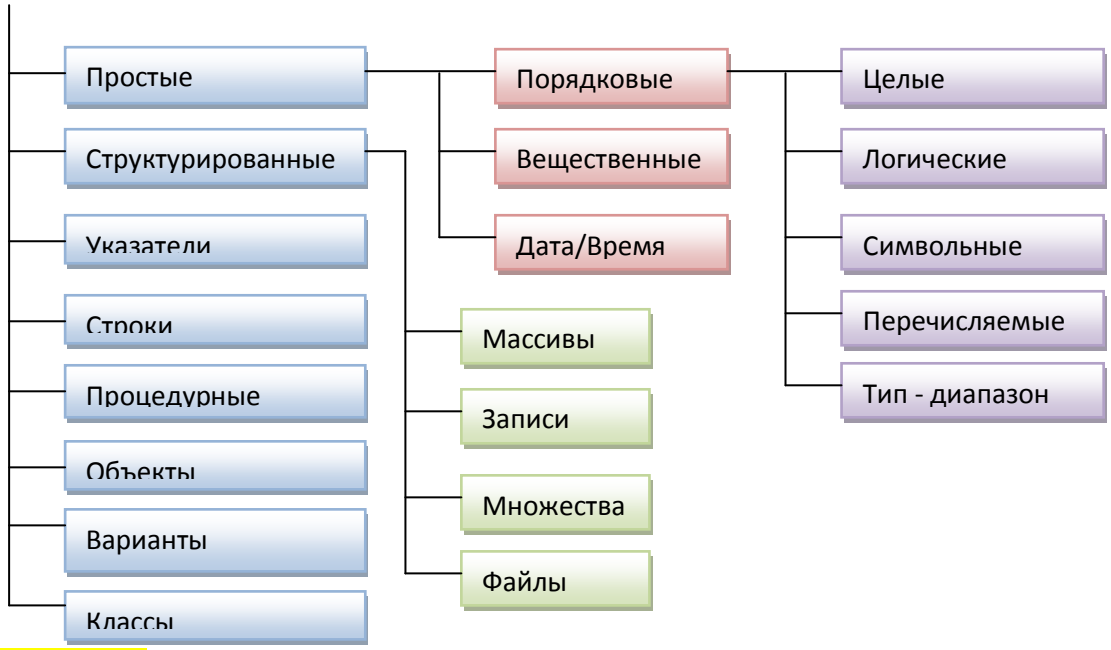
Лексема – это специальные символы, зарезервированные слова, идентификаторы, метки, строки, числа. Идентификаторы и зарезервированные слова могут быть описаны в любом регистре.

Идентификаторы описывают имена констант, типов, переменных, процедур, функций, модулей, программ, меток, полей, имена объектов, и т.д.

Метки бывают: числовые, символьные.

Классификация типов данных
в Object Pascal

Типы:



Тип данных – это множества значений, которые принимает переменная, объединенная с множеством допустимых над ними операций.

Группы целых

Название	Диапазон значений	Размер в байтах
Byte (короткое целое, без знака)	[0 до 255]	*1*
Shortint (короткое целое со знаком)	[-128 до +127]	*1*
Smallint (целое со знаком)	[-32768 до 32767]	*2*
Integer или Longint (Длинное целое со знаком)	$[-(2^{31})$ до $(2^{31}-1)]$	*4*
Word (Целое без знака)	[0 до 65535]	*2*

Операции над целыми

- Арифметические (+,-,*,div, mod)
- Логические (not, and, or, xor, shl, shr)

Переменная типа Boolean (логическая). Принимает два значения: true или false, размер 1 байт.

Циклы. Переходы с типа на тип

While → Repeat

While:	***Repeat:***
While l do *L:=g(x,y,z);*	*If l then* *Repeat* *L:=g(x,y,z);* *Until not l;*

Repeat → While

Repeat:	**While:**
Repeat *L:=g(x,y,z);* *Until l;*	*L:=g(x,y);* *While not l do* *L:=g(x,y);*

For → While

For:	**While:**
For i:=l to 1 do *C:=C+a[i];*	*I:=1* *While I<11 do* *Begin* *C:=C+a[i];* *I:=i+1;* *End;*

While → For

While:	***For:***
While L do *L:g(x,y);*	*N:=1;* *For i:=1 to N do* *Begin* *If l then* *Begin* *i:=i+1;* *l:=g(x,y,z);* *end;* *end;*

Goto и его применение

Метки должны быть декламированы.

//в разделе описания меток указать
Label 1,2,3;
// В теле программы
3:C:=C+A[i];

Goto – оператор безусловного перехода на метку.

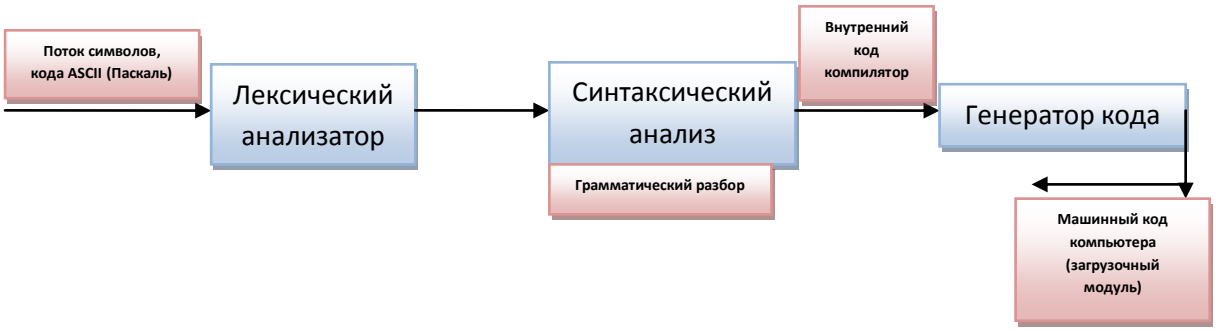
2: if not l then goto 1;
L:=g(x,y);
Goto 2;
1: ……. //что-то тут еще….

Алфавит и лексика языка

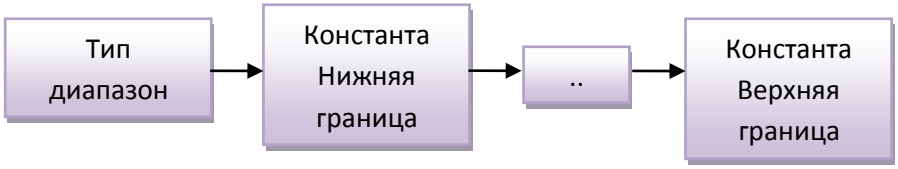
Коды ASCII:

(0-31) – невидимые управляющие символы.
(48-57) – арабские десятичные цифры: 0,1,2,3,4,5,6,7,8,9
(65-90) – заглавные английские буквы: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, R, S, T, U, V, W, X, Y, Z
(97-122) – прописные английские буквы: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, r, s, t, u, v, w, x, y, z
Также: 16ричные цифры: 0-15
Специальные символы: *./:";'><{}^$@!#()&?
:= - например в присвоении нельзя делать пробел.
Существуют также зарегистрированные слова: end; string; begin; do; xor; и т.д.(е них отдельный цвет в исходнике в компиляторе)

Иная версия работы компилятора (фазы)



Тип диапазон (интегрированный)



Пример:

- *Type: range=1..100;*
- *Const min = 0;*
  Max = 15;

- *Type colors = red..blue;*

Отличительные признаки величин порядкового типа:

- Множество значений конечно
- Множество значений упорядочено

К ним применимы:

- *Ord* – позволяет получить порядковый номер
- *Succ* – позволяет получить последний элемент
- *Pred* – позволяет получить предыдущий элемент

- *High(a)* – определяет верхнюю границу
- *Low(a)* – позволяет определить нижнюю границу

```
{Заполнение множества с клавиатуры  множества элементов из символов и выдача на экран.
Например  3 символьное множество…}
Begin
S:=[];
While true do begin
Writeln (‘1-input elements of set’);
Writeln(‘2-printelements of set’);
Writeln(‘3-exit’);
Readln(k);
Case k of
1:
For L:=1 to 3 do // как сказано выше у нас трех символьное множество по примеру
Begin
Readln(n);
S:=S+[n];
End;
2:
For i:=’a’ to ‘z’ do
If I in S then writeln (char(ord(i)));
3: halt;
end; end;
readln;
END.
```

Циклы в Pascal

Циклы подразделяются на неитерационные (*For*) и на итерационные (*repeat until; while*)
Операторы итерационного типа используются обычно, если число повторений цикла, заранее неизвестно или шаг изменения параметра отличен от +1 или -1

```
//подсчет факториала с циклом While
…
F:=1;
I:=1;
While I<=N do
Begin
F:=F*I;
I:=I+1;
End;

//подсчет факториала с циклом Repeat
F:=1;
I:=1;
```

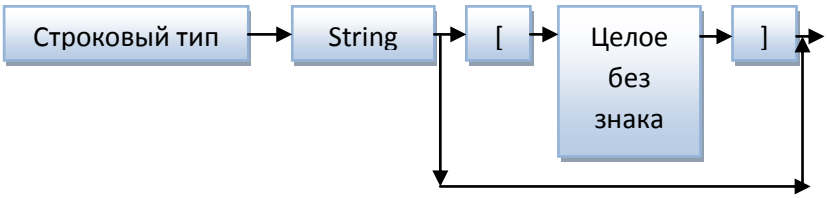
```
Repeat
F:=F*I;
I:=I+1;
Until i>n;
```

Группа вещественных типов

Группа вещественных типов

Название	Диапазон значений	Количество знач. Цифр	Размер в байтах
Real (вещественные)	От $29*10^{-39}$ до $1,7*10^{38}$	11-12	6
Single(вещественные одинарной точности)	От $1,5\text{-}10^{-45}$ до $3,4*10^{38}$	7-8	4
Extended	От $3,4*10^{-4932}$ до $1,1*10^{4932}$	19-20	10
Double	От $5,0*10^{-324}$ до $1,7*10^{308}$	15-16	8
Comp0	От -2^{63} до $2^{63}-1$	19-20	8

Строковый тип



Основные операции со строками:

a) + Конкатенация
b) Отношение (выполняется лексикографически)

Строка – это упакованный массив символов длинной до 255 символов. Длина строки хранится в нулевом байте. Длина (этот байт) заполняется во время инициации данной строки.

```
Var S:String[10];
```

0	1	2	3	4	5	6	7	8	9	10
10	П	р	и	в	е	т	!	!	!	!

```
Var S:String[10];
Begin S:='hello';
```

0	1	2	3	4	5	6	7	8	9	10
5	h	e	l	l	o					

Процедуры и функции для работы со строками

- Функция определения длины строки Length(St):word – возвращает длину строки типом word

- Процедура delete (st, index, count) удаляет count (определенное число символов) начиная с порядкового номера index
- Процедура insert (str,stl,index) вставляет подстроку S2 символов в строку S1 после index
- Преобразование числа в строку процедура Str (x[:w[:d]], st)

  //Пример:
  X:=-5.67;
  Str(x:7:3,s1);
- Val (st,x,code) обратная str. Процедура преобразует строку в число с кдом ошибки 0 в случае успешного преобразования

  {пример} Val ('bc1',a,k); // k будет 1 при ошибке и 0 при успешном выполнении
- Copy (St, index, count):string – возвращает фрагмент строки st начиная с index длинной count

  S2: copy('HELLO',3,2); // S2 теперь содержит 'LL'
- Pos (st2,st1):integer – возвращает номер позиции первого вхождения подстроки st2 в строку st1
- Upcase(ch):char совращает символ соответствующего верхнему регистру для char`a если такой есть.

  {Например} S:=upcase('a'); // S содержит 'A';

Программа сортировки по фамилии

```
Program project_name;
Type name=string[30];
Var
N;I;k:integer;
Temp:string[30];
Mass_st:array [1..30] of name;
Begin
Write ('Input N-->');
Readln (n);
For I:=1 to N do
Readln (mass_st[i]);
For i:=1 to N do
        For K:=1 to N-I do
        If mass_st[k]>mass_st[k+1] then begin
                Temp:=mass_st[k+1];
        Mass_st[k+1]:=mass_st[k];
        Mass_st[k]:=temp;
        End;
For I:=1 to N do writeln (mass_st[i]);
Readln;
End.
```

Множества

Множества – наборы, однотипных, логически связанных друг с другом объектов, характер этих связей определяется программистом, количество элементов, входящих в множества может меняться в пределах от 0 до 256.
Именно не постоянством количества своих элементов, множества отличаются от массивов и записей.
Два множества эквиваленты, тогда и только тогда, когда все их элементы одинаковы, при этом порядок следования элементов в множествах не важен.
Одно множество включает другое если все элементы первого множества лежат во втором множестве.

Операции над множествами

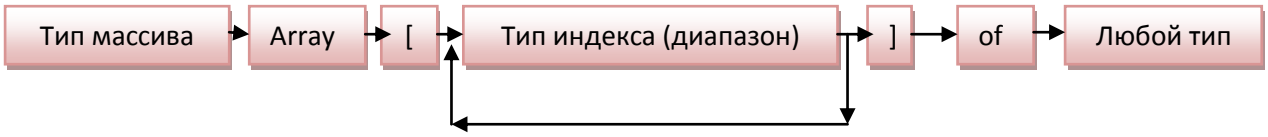
- * пересечение множеств, т.е. произведение
  X:=[1,2,3,4,7,9];
  Y:=[1,2,3,10,5];
  *Z:=x*y; // Z будет содержать [1,2,3]*
- + объединение множеств, т.е. сложение
  Z:=X+Y; // Z будет содержать [1,2,3,4,5,7,9,10]
- - Разность, т.е. относительно дополнение
  Z:=X-Y ; // Z будет содержать [4.7.9]
- = проверка на равенство, т.е. есть ли все элементы множества x во множестве y? Результатом будет логическая булевская переменная.
  F:=x=y; // F будет содержать в этом примере false
- <> проверка на неравенство, т.е. есть ли какие-нибудь элементы во множестве X, которых нету в множестве Y. Результатом будет логическая булевская переменная.
  F:=x<>y; // F будет содержать в этом примере true
- >= проверка на включение, т.е. если все элементы множества Y являются элементами множества X
  X:=[1,2,3,4,7,9];
  Y:=[1,2,3] ;
  F:=x>=y; // F будет содержать в этом примере true
- <= обратная проверка проверка на включение, т.е. если все элементы множества X являются элементами множества Y
  X:=[1,2,3];
  Y:=[1,2,3,10,5];
  F:=X<=Y; // F будет содержать в этом примере true
- IN проверка принадлежности отдельного элемента множеству
  X:=[1,2,3];
  G:=1;
  F:=g in x; // F будет содержать в этом примере true

Структурированный тип данных. Массивы.

Структурированный тип данных может занимать до 2 Gb памяти, основным примером является массив.

Массив – это фиксированная совокупность данных одного типа, размещаемых в непрерывной области оперативной памяти.

К любому элементу массива можно обратиться по его индексу.

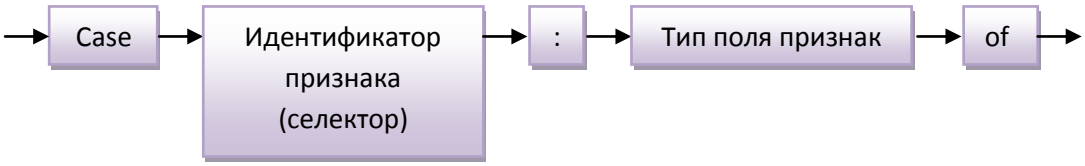


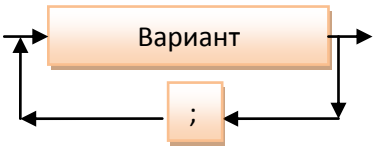
Для компактного представления комбинации разного типа данных их объединяют в запись.

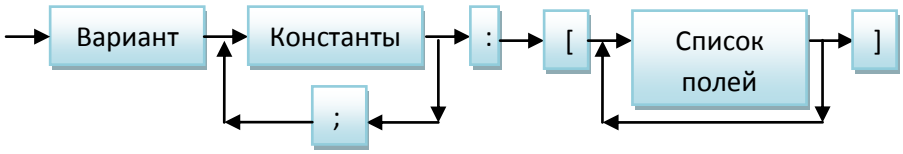
Запись состоит из объявленного числа.

Запись с вариантами.

В пределах одного типа записей можно задавать несколько различных структур (несколько вариантов записи). Каждая будет выполняться при наличии с определенного условия. В этом случае запись будет состоять из двух частей: фиксированной и вариантной.







Память выделяется под максимально возможный размер записи (самый объемный).

Указатель-переменная значением, которой является адрес ячейки памяти. Обычно указатель связывается с некоторым типом данных. Этот тип называется базовым, а указатель типизированным. Чтобы получить указатель не связанный с каким-нибудь типом данных необходимо использовать тип pointer.

```
Var P1,p2,p3:^integer; //указатель на целое
P:pointer
Begin
New(p1);
P1^:=10;
New(p2);
P2^:=2s;
New(p3);
P3^:=p2^+p1^;
Writeln(p3^);
Release;
End.
```

Процедуры и функции

Процедуры и функции представляют собой автономную часть программного кода выполняющего заданную цель и имеющего определенную последовательность действий.

Процедуры и функции значительно упрощают процесс проектирования и используются в двух основных технологиях восходящего и нисходящего программирования (проектирования программ).

Нисходящее программирование - методика разработки программ, при которой разработка начинается с определения целей решения проблемы, после чего идет последовательная детализация, заканчивающаяся детальной программой.

Восходящее программирование - методика разработки программ, при которой крупные блоки собираются из ранее созданных мелких блоков. Восходящее программирование начинается с разработки ключевых процедур и подпрограмм, которые затем постоянно модифицируются.

Использование процедур и функций требует тщательной работы с отладчиком.

Параметры, которые определены в заголовке процедуры называются формальными параметрами. Параметры, которые она получает во время вызова это фактические параметры.

Каждая процедура или функция имеет тело. (От begin до End)

Возвращение значения из функции происходит, используя присвоение полученного результата точки входа в данную фикцию. Каждая процедура или функция может иметь неограниченное количество точек входа и выхода.

Передача параметров в процедуру или функцию происходит:

1. По значению (Создаются копии передаваемых параметров внутри процедуры или функции. По завершению работы, выделавшееся память освобождается.)
2. По ссылке (по адресу в C++/C#)

Куча – область динамической памяти, используемая процедурами/функциями. Возникающая утечка памяти связана с использованием оператора new, без посл. применимая оператора release/dispose

Процедура – самостоятельные, автономные программные еденицы, выполняющие локальные задачи.

Синтаксис и состав процедуры:

Label – объявление меток внутри процедуры;

Const – объявление локальных типов

Var – Объявление локальных переменных

Описание процедуры в Pascal:

Procedure Proc_NAME (ПарамЗнач1: ТипЗначения1, ПарамЗнач2, ПарамЗнач3: ТипЗначения2; Var ПарамПеремен1: Тип_перемененной1; Const Парам_Конст1: Тип_Константы1);

В этом случае описание вложенных процедур и функций

Begin

Тело процедуры (оперативный блок)

End;

Глобальными константами типа переменных, которые объявлены вне процедуры или функции.

Локальные существуют только внутри процедур или функций, либо в разделе описаний const, type, var в подпрограмме процедуры/функции.

Процедуры/функции могут по ряду со своими локальными данными использовать и модифицировать и глобальные данные. Для этого нужно чтобы описание констант стояло ниже, чем описание соответствующих глобальных типов констант и переменных.

```
//Работа с файлами
{$I < имя_файла >}
Program A1;
Var …
{$I b1.pas}
Begin
End.
//B1.pas:
Procedure
Var…
Begin
End.
```

Формальные и фактические параметры при описании процедуры/функции в ее заголовке не могут быть указаны:

1. Параметры значения
2. Параметры переменные
3. Параметры константы
4. Параметры процедуры
5. Параметры функции

При передаче параметров по значению выделяется локальная область через выделение стека, куда копируются значения соответствующих фактических параметров, после завершения подпрограммы, выделенная память становится недоступной, поэтому передача параметров по значению не может быть использована в подпрограммах для получения результата. Параметры переменной, при вызове по ссылке в подпрограмме, память под передаваемые значения не отводится. В подпрограмму передаются не значения переменной, а ссылка на место в памяти соответствующего фактического параметра. В качестве параметров переменных могут быть использованы массивы и строки открытого типа, где заданы размеры.

Открытые типы.

Открытый массив – это формальный параметр программы, описывающих базовый тип элементов, но неопределенна размерность и граница. Индексация начинается в этом случае с 0. Верхняя граница открытого массива возвращается в функцию *high*, такое описание возможно только для одномерных массивов. Для открытого массива, в стеке создается копия, что может вызвать переполнение стека.

```
//Пример открытого массива
Function sum(var a:array of integer):integer;
Var S,I:ineteger;
Begin
S:=0;
For I:=0 to hugh(a) do begin
S:=S+a[i];
Sum:=S;
End;
```

Такой массив в основной программе может быть объявлен так:

```
Var A:array [-2..3] of integer  //функция high все равно возвратит значение 6
```

Открытая строка – может задаваться стандартным средством openString с использованием директивы компилятора.

```
{$p+}
Procedure zap (var st:openstring; R:integer);
{$p+}
Procedure zap (var st:string; R:integer);
```

Режим компилятора, при котором отключаются контроль за совпадением формального и фактического параметров длины строки, при передаче строки меньшего размера формирует параметр строки, будет иметь уже длину что и параметр обращения при передачи большего размера, происходит обрезание до максимально значения формального параметра. Контроль включается только при передачи параметров переменных. Для параметров значений длина не контролируется.

Параметры константы

Так как аргументы, представляемые в процедуру или функцию размещаемые в стеке, то в случае передачи значений массива большего размера может произойти переполнение стека. В Turbo Pascal 7.0 добавлен описатель const, который может задаваться для формальных параметров под программ.

Аргумент соответствующий такому параметру передаваемого по ссылке подобно параметру с var, но в самой процедуре/функции запрещается присваивать новое значение.

Параметры процедуры/функции

В Pascal допускается интерпретация процедур функций как объектов, которые можно присваивать переменным и передавать как параметр. Такие действия выполняются с помощью процедурных типов.

Пример:

```
Type
Proc = procedure;
SwarProc = procedure(var x,y:integer);
```

```
Str=procedure(s:String);
Procedure Swar (var A, B:ineteger);
Var temp:integer
Begin
Temp:=a;
A:=b;
B:=temp;
End.
Function Tan(Angle:real):real;
Begin
Tan:=sin(Angle)/cos(angle);
End;
Var
P:swapproc;
F:mathfunction;
P=swap;
F=tan;
```

После присваивания обращение P(I,j)←→ Swap (i,j)

Работа с файлами

Общие сведения о файлах.

Файл – это поименованная совокупность логически связанных между собой данных (запись), имеющих определенную организацию и общие назначения.

Физическая запись – это совокупность данных передаваемых в прямом или обратном направлении при одном обращении к внешнему носителю, т.е. это тип ед.обмена данными между внешней и оперативной памятью.

Логическая запись

Единицы данных, используемые в операторах чтения и записи файла, логические записи объединены в физическую запись для уменьшения числа обращений к внешнему устройству. Для обращения к записи на внешнем носителе используется понятие логического файла.

Логический файл или файл в программе – это совокупность данных, состоящая из логических записей, объединенных общим назначением. Для связи файлов в программе используется процедура *Assign* (файловая переменная), где указывается имя файла в программе и имя файла на внешнем носителе. Число записей файла произвольно. В каждый момент времени доступна только одна запись.

Длина файла – количество записанных компонентов.

В Pascal различают 3 вида данных:

1. Типизированные с определенным типом записи
2. Текстовые файлы со строками неограниченной длины
3. Файлы без типа (нетипизированные файлы)

Для передачи записи блока, какой из видов файлов более подходящий. При работе с файлами следует придерживаться следующих правил:

1. Все имена файлов могут быть описаны в заголовке программы
2. Текстовые файлы должны быть описаны с атрибутом текст

3. Каждый файл в программе должен быть закреплен за конкретным файлом на носителе процедурой *Assign*
4. Открытие файла на чтение используется процедура *Reset*
5. Открытие создаваемого файла для записи используется процедура *Rewrite*
6. По окончанию работы с файлом он должен быть закрыт процедурой *Close*
7. Перед файлом также может указывать каталог

```
//Работа с файлами
Assign (<имя файла>, <имя файла на носителе>)
Var F1:text;
Name: string[12];
Write ('Ввод имени исходного файла:');
Readln(name);
Assign (f1, name);
Reset (f1);
Close (f1);
End.
```

Процедуры и функции работы с файлами

Reset (<имя файла >) процедура открытия существующего файла для чтения при последующем доступе и для чтения/записи при прямом доступе. Указатель файлы при этом встает на первую запись.

Процедура rewrite открытие созданного файла для записи. При существовании с таким же именем, указатель ставится на первую запись.

Процедура Read чтения очередных компонент файла в переменные, при которых должен совпадать с типом компонентов файлов. Указатель файла передвигается на количестов прочитанных компонентов.

Процедура Write – запись содержимого переменных в файл согласно положению указателя. Указатель автоматически передвигается на число записанных компонентов.

Процедура Seek – установка текущего указателя для чтения/записи требуемой компоненты файла, используется для организации прямого доступа к записи файла.

Close – процедура закрытия файла. Обязательно должна быть использована после создания файла и записи данных, иначе возможна потеря данных.

Crase – уничтожение файла

Rename – переименование, используется после закрытия файла.

IoResult – функция возврата условного признака выполнения последней операции ввода/вывода. Если операция успешна то 0. Функция является доступной только после отключения автоконтроля ввода/вывода ошибок в компиляторе командой *{$I-}*. Если команда выполнена и операция ввода/вывода произошла с ошибкой устанавливается флаг ошибки и все последующие обращения к вводу/выводу блокируются пока не будет вызвана функция *IOResult*.

FilePos – функция определяет номера текущей записи файла.

FileSize – функция определения общего количества записей в файле.

EOF – функция определения признака конца файла, получает true при чтении последней записи файла.

EOLN – функция обнаружения конца строки текстового файла, имеет значение True если конец строки достигнут.

Особенности обработки типизированных файлов

Типизированный файл состоит из последовательностей записей одной длины и формата. Записи следуют непрерывно друг за другом. Первые четыре байта первого сектора файла содержат количество и длину записи. К файлу с такой организацией можно обращаться последовательно и выборочно (с прямым доступом). При последовательном доступе записи растут на внешнем носителе в порядке поступления. При прямом доступе предполагается, что данные располагаются в определенных областях, имеющих посл. номера начиная с 0. Вычисление значения указателя фиксирует номер записи. Можно обеспечить прямой доступ к нужной записи, исполнением процедур позиционирования seek. Процедура truncate обрезает файл до заданной файловой позиции.

```
//Описание типизированного файла
Type <идентификатор> = file of <тип компонент>
//Например:
Type
T = file of real;
St=record;
A:string[10];
C:real;
B:integer;
D:byte;
Var
F:T;
G: file of integer;
DAN: file of st;
```

Нетипизированные файлы.

Файлы без типа или нетипизированные файлы используются обычно когда не важна внутренняя структура записи файла (при копировании). Если длина сегмента на диске 1024 байта, то количество блоков в группе 8, при длине блока 128 байт, т.е. копирование (обработка) происходит блоками. Обмен информацией происходит непосредственно между программой и файлами без использования буферной памяти. Адресация юлоков производится по их номерам. Блоки в этом случае являются компонентами файлов. Использование нетипизированного файла приводит к экономии памяти. Для работы с иакими файлами, предусмотрены процедуры позволяющие производить обмен группами (блоками) по 128 символов.

Для чтения блока в файл используется процедура:

BlockRead (<имя файла>, <переменная>, [<фактическое число>]);

Для записи блока в файл используется процедура:

BlockWrite (<имя файла>, <переменная>, [<фактическое число>]);

Файл блочного ввода/вывода описывается типом

Var <имя файла>: FILE;

Процедура Read или Write не используются!

При открытии файла без типа можно указать длину записи в байтах. Она указывается вторым параметром при обращении к процедуре reset или rewrite. В качестве этого используется выражение типа word.

//Пусть требуется скопировать данные из одного файла в другой

```
Program copy;
Var
Fromf, to_f:file;
Nr,nwr:word;
Name:string [12];
Buf: array [1..2048] of char;
Begin
Write ('Имя Входящего файла');
Readln(name);
Assign(fromf,name);
Write ('Имя выходного файла');
Readln(name);
Assign(to_f,name);
Reset(fromf,1);
Rewrite(to_f,1);
Repeat
Blockread(fromf,buf,size of (buf), nr);
Blockwrite (to_f, buf,nr,nwr);
Until (nr=0) or (nwr<>nr);
Close(fromf);
Close(to_f);
```

Динамическая память. Работа с ней.

Динамическая память позволяет выделять и освобождать память в процессе выполнения программ. Статические переменные существуют только в течении жизни блока, где они объединены, а динамические после выхода из блока до окончания программы.

Достоинства:

- Экономичность и эффективность ее использования
- Возможность динамического изменения числа элементов в связанных структурах (например, списки)

Переменная, размещаемая динамически не объявляется в var, и не имеет имени в программе. Обращение к участку динамической памяти идет с помощью указателя (специальной ссылочной переменной). В указателях хранится адрес. Компилятор отводит под указатель 4 байта статической памяти. Указатель может быть типизированным или нетипизированным.

Стандартные процедуры размещения и освобождения динамической памяти

Выделение может быть двумя способами:

С помощью процедуры *New(P)*, где p – переменная типизированного указателя. Процедура New создает новую динамическую переменную, выделяет под нее память и устанавливает указатель *P*. В *P* пишется адрес. Размер и структура выделенного участка памяти задается в зависимости от типа данных, с которыми связан указатель *P*. Доступ: *^P*	С помощью процедуры *GetMem (p, Size);* *P* – Переменная типизированного указателя. *Size* – целочисленное выражение размера запрашиваемого в байтах участка памяти. Процедура создает новую динамическую переменную, требуемого размера и свойства. А также помещает адрес этой созданной переменной в переменную *P* – типа указатель. *//пример использования* *Type* *Rec=record;* *Field1:string[20];* *Field2:integer;* *Ptr_rec=^rec;* *Var P: ptr_rec;* *Begin* *GetMem (p, sizeof(rec));* *FreeMem(p,sizeof(rec));//освобождает память*

Освобождение динамической памяти:

- Автоматически после завершения программы
- С использование процедуры *dispose(p); P* помечается как свободный для возможных дальнейших размещений, при этом физической очистки *P* и связанной с ней памяти происходит, поэтому даже удалив этот экземпляр записи, можно все же получить утечку памяти
- С помощью стандартной процедуры *FreeMem;* Помечает память равным значению Size, связанную с указателем *P* , как свободную.
- С помощью процедуры *Mark* (запоминает состояние динамической области в переменной указателя *P*)
- С помощью процедуры *Resize* (освобождает всю динамическую память, которая выделена процедурами *New* или *GetMem* после запоминания текущего указателя значения *P* процедурой *Mark*)

Последние две процедуры нельзя чередовать с обращениями к *Dispose* и *FreeMem* из-за их различной реализации.

```
{Пример использования указателей. Создадим матрицу, динамически используя указательные операции, и поменяем столбцы, поменяв указатели.}
Program Dinamic;
Type
VK=^t1;
T1=array [1..1] of integer;
Mt=^t2;
T2=array [1..1] of vk;
Var
A:mt;
M,n,I,j,k,l:integer;
Max,min:integer;
R:pointer;
Begin
Readln(n,m);
{выделяем память под указатель столбцов матрицы, используя нетипизированный указатель}
GetMem (A,sizeof(pointer)*m); {умножаем на M из-того, что m столбцов}
For J:=1 to m do
GetMem(a^[j], sizeof(integer)*n);
For I;=1 to N do
For J:=1 to M do
Read (a^[J]^[i]);
For i:=1 to N do
Begin
For J:=1 to M do
Write (a^[j]^[i]:4);
Writeln;
End;
Readln;
Max:=A^[1]^[1];
K;=1;
Min:=max;
L:=1;
For J:=1 to m do
For i:=1 TO N DO
IF a^[J]^[i]<min then
Begin
Min:=A^[j]^[i];
L:=j;
End
Else
If A^[j]^[i]>max then
Begin
If A^[j]^[i]>max
Then
Begin
Max:=A^[j]^[i];
K:=j;
End;
{Для обмена столбцов достаточно поменять указатели на столбцы}
If K<>l then begin
R:=A^[k];
A^[k]:=A^[l];
A^[l]:=r;
End;
Begin
//выводим результат
For i:=1 to n do
Begin
For J:=1 to M do
Write (a^[i]^[]:3, ' ');
Writeln;
End;
//освобождаем память
For I:=1 to M do
FreeMem (A^[i], n*sizeof(integer));
FreeMem(A; m*sizeof(pointer));
Readln;
End.
```

Модули

Процедуры и функции могут быть сгруппированы в отдельные модули.
Модуль (Unit) – программная единица, текст которой компилируется автономно независимо от главной программы или совокупность программных ресурсов, предназначенных для использования другими программами.
Модули имеют основные части:

- Заголовок, который следует за зарезервированным словом **Unit**
  Unit Name1;
- Интерфейсную часть после слова **Interface** В ней помещаются объявления переменных процедур и функций, констант, и типов данных, которые должны быть доступны для других программ
- **Implementation** Внутренняя часть. В нее входят текст программы и локальные объекты, доступные только внутри модуля, а также необязательная часть модуля, расположенной после испольными **begin** и **end**

Модули позволяют создавать личные библиотеки процедур и функций и строить программы практически любого размера. Для подключения модуля к программе необходимо указать его имя в разделе описания модулей:
uses CRT, Graph;
Использование составных имен применяется не только к именам переменных, а ко всем именам, описанным в интерфейсной части модуля. Рекурсивное использование модулей запрещено. Если в модуле имеется раздел инициализации, то операторы из этого раздела будут выполнены перед началом выполнения программы, в которой используется этот модуль.

Совместимость типов

```
Var     A:t1;
        B:t1;
Begin
A:=b+b;
```

Значение переменной *B* может быть присвоено переменной *A* если:

1. T1 и T2 являются тождественными (одинаковыми
2. T1 и T2 являются совместимыми порядковыми типами и диапазон значений типа T2 является диапазоном T1
3. T1 и T2 вещественные типы а диапазон T2 является поддиапазоном T1
4. T1 – вещественный тип, а T2 – целочисленный
5. T1 и T2 строковые типы
6. T1 строковый тип, T2 символьный тип
7. T1 и T2 являются совместимыми множественными типами, и все элементы T2 попадают в диапазон возможных значений T1
8. T1 и T2 являются совместимыми указательными типами
9. T1 и T2 являются совместимыми процедурными типами
10. T1 – процедурный тип, а T2 – процедура/функция с идентичным типом результата, числом параметров, и соответствием между типами параметров

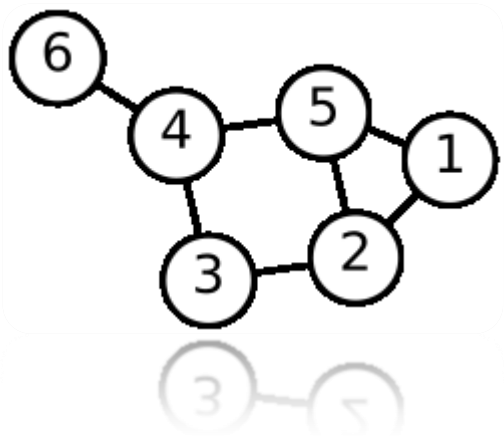
Дополнения

Эвристика – это набор правил и приемов, основанных на практических навыках, приводящих к эффективному результату.

Теория графов

раздел дискретной математики, изучающий свойства графов. В наиобщем смысле граф представляется как множество вершин (узлов), соединённых рёбрами. В строгом определении графом называется такая пара множеств G={R,V}, где V есть подмножество любого счётного множества, а R - подмножество V×V. Теория графов находит применение, например, в геоинформационных системах (ГИС). Существующие или вновь проектируемые дома, сооружения, кварталы и т. п. рассматриваются как вершины, а соединяющие их дороги, инженерные сети, линии электропередач и т. п. — как рёбра. Применение различных вычислений, производимых на таком графе, позволяет, например, найти кратчайший объездной путь или ближайший продуктовый магазин, спланировать оптимальный маршрут.

граф с шестью вершинами и семью рёбрами



Итеративная алгоритмика – не подразумевает эвристики (неупрощенные конструкции, циклические).

Реку́рсия — метод определения класса объектов или методов предварительным заданием одного или нескольких (обычно простых) его базовых случаев или методов, а затем заданием на их основе правила построения определяемого класса. Другими словами, рекурсия — частичное определение объекта через себя, определение объекта с использованием ранее определённых. Рекурсия используется, когда можно выделить самоподобие задачи.

Сортировка

Пять методов сортировки массивов, рассмотрены на конкретных примерах: сортировка массива методом пузырька, сортировка методом нахождения минимального элемента, поиск перебором, бинарный поиск.

Сортировка массива методом пузырька - медленная, но если скорость не главное, можно применить и его. Алгоритм очень прост - если два соседних элемента расположены не по порядку, то меняем их местами. Так повторяем до тех пор, пока в очередном проходе не сделаем ни одного обмена, т.е. массив будет упорядоченным. Ниже текст процедуры, реализующей алгоритм сортировки методом пузырька (Arr - массив для сортировки с начальным индексом 0, n - размерность массива)

```
procedure SortPuz (var Arr : array of Integer; n : Integer);
var
i : Integer;
Temp : Integer;
Flag : Boolean;
begin
repeat
Flag := False;
for i := 0 to n - 1 do
if Arr [i ] > Arr [i + 1] then begin
Temp := Arr [i ];
Arr [i ] := Arr [i + 1];
Arr [i + 1] := Temp;
Flag := True;
end;
until Flag = False;
end;
```

Сортировка методом нахождения минимального элемента Ещё один вариант сортировки, более быстрый, чем метод пузырька. Заключается он в следующем: при каждом просмотре массива находим минимальный элемент и меняем местами его с первым на первом проходе, со вторым - на втором и т.д. Не забудьте только, что первый элемент массива должен иметь индекс 0.

```
procedure SortMin (var Arr : array of Integer; n : Integer);
var
i, j : Integer;
Min, Pos, Temp : Integer;
begin
for i := 0 to n - 1 do begin
Min := Arr [i ];
Pos := i;
for j := i + 1 to n do
if Arr [j] < Min then begin
Min := Arr [j];
Pos := j;
```

```
end;
Temp := Arr [i ];
Arr [i ] := Arr [Pos];
Arr [Pos] := Temp;
end;
end;
```

Сортировка массива вставками Более быстрый и оптимальный метод сортировки - сортировка вставками. Суть её в том, что на n-ном шаге мы имеем упорядоченную часть массива из n элементов, и следующий элемент встаёт на подходящее ему место. Имейте в виду - первый индекс массива - 0.

```
procedure SortInsert (var Arr : array of Integer; n : Integer);
var
i, j, Temp : Integer;
begin
for i := 1 to n do begin
Temp := Arr [i ];
j := i - 1;
while Temp < Arr [j] do begin
Arr [j + 1] := Arr [j];
Dec (j);
if j < 0 then
Break;
end;
Arr [j + 1] := Temp;
end;
end;
```

Поиск перебором Чтобы найти какие-то данные в неупорядоченном массиве, применяется алгоритм простого перебора элементов. Следующая функция возвращает индекс заданного элемента массива. Её аргументы: массив с первым индексом 0, количество элементов в массиве и искомое число. Если число не найдено, возвращается -1.

```
function SearchPer (Arr : array of Integer; n, v : Integer) : Integer;
var
i : Integer;
begin
Result := -1;
for i := 1 to n do
if Arr [i ] = v then begin
Result := i;
Exit;
end;
end;
```

Бинарный поиск При поиске в упорядоченном массиве можно применить гораздо более быстрый метод поиска - бинарный. Суть его в следующем: В начале переменная Up указывает на самый маленький элемент массива (Up := 0), Down - на самый большой (Down := n, где n - верхний индекс массива), а Mid - на средний. Дальше, если искомое число равно Mid, то задача решена; если

число меньше Mid, то нужный нам элемент лежит ниже среднего, и за новое значение Up принимается Mid + 1; и если нужное нам число меньше среднего элемента, значит, оно расположено выше среднего элемента, и Down := Mid - 1. Затем следует новая итерация цикла, и так повторяется до тех пор, пока не найдётся нужное число, или Up не станет больше Doun.

```
function SearchBin (Arr : array of Integer; v, n : Integer) : Integer;
var
Up, Down, Mid : Integer;
Found : Boolean;
begin
Up := 0; Down := n;
Found := False; Result := -1;
repeat
Mid := Trunc ((Down - Up) / 2) + Up;
if Arr [Mid] = v then
Found := True
else
if v < Arr [Mid] then
Down := Mid - 1
else
Up := Mid + 1;
until (Up > Down) or Found;
if Found then
Result := Mid;
end;
```