«Язык программирования Pascal»

Подготовил студент группы РПЗ-14
Слесаренко Андрей

Язык программирования – формальная знаковая система, предназначенная для описания алгоритмов в форме, которая удобна для исполнителя.

языки низкого уровня

языки ассемблера (от англ. to assemble – собирать, компоновать)

языки высокого уровня

алгоритмические языки (Фортран, Алгол, Кобол, Лисп, Бейсик, Форт, Паскаль, Ада, Си…)

Язык программирования

(более 2 500)

- **определяет**

  набор лексических, синтаксических и семантических правил, используемых при составлении компьютерной программы

- **позволяет определить**

✔ *на какие события будет реагировать компьютер*

✔ *как будут храниться и передаваться данные*

✔ *какие именно действия следует выполнять при различных обстоятельствах*


Понятие «Язык программирования»
Функция
Задача
Исполнение

Функция:

язык программирования предназначен для написания компьютерных программ, которые применяются для передачи компьютеру инструкций по выполнению того или иного вычислительного процесса и организации управления отдельными устройствами.

Задача:

язык программирования отличается от естественных языков тем, что предназначен для передачи команд и данных от человека компьютеру, в то время как естественные языки используются лишь для общения людей между собой.

Исполнение:

язык программирования может использовать специальные конструкции для определения и манипулирования структурами данных и управления процессом вычислений.

языки низкого уровня

языки ассемблера

(от англ. to assemble – собирать, компоновать).

В языке ассемблера используются символьные обозначения команд, которые легко понятны и быстро запоминаются. Вместо последовательности двоичных кодов команд записываются их символьные обозначения, а вместо двоичных адресов данных, используемых при выполнении команды, - символьные имена этих данных, выбранные программистом. Иногда язык ассемблера называют мнемокодом или автокодом.

Фортран

(англ. FORTRAN от FORmula TRANslator – переводчик формул),

Разработан в 1957 году.

Применяется для описания алгоритма решения научно-технических задач с помощью ЦВМ.

Предназначался, в основном, для проведения естественно-научных и математических расчётов.

В усовершенствованном виде сохранился до нашего времени. Среди современных языков высокого уровня является одним из наиболее используемых при проведении научных исследований.

Наиболее распространены варианты Фортран-II, Фортран-IV, EASIC Fortran и их обобщения.

Был распространен в США и Канаде.

Алгол

(англ. ALGOL от ALGOrithmic Language – алгоритмический язык)

Появился в 1958-1960 годах (Алгол-58, Алгол-60).

Разработан комитетом, в который входили европейские и американские учёные.

Был усовершенствован в 1964-1968 годах – Алгол-68.

Позволяет легко переводить алгебраические формулы в программные команды.

Был популярен в Европе, в том числе СССР.

Оказал заметное влияние на все разработанные позднее языки программирования, и, в частности, на язык Pascal.

Предназначался для решения научно-технических задач. Кроме того, этот язык применялся как средство обучения основам программирования – искусства составления программ.

Кобол

(англ. COBOL от COmmom Business Oriented Language – общий язык, ориентированный на бизнес)

Разработан в 1959 – 1960 годах.

Язык программирования третьего поколения.

Предназначен для разработки бизнес приложений, а также для решения экономических задач, обработки данных для банков, страховых компаний и других учреждений подобного рода.

Разработчик первого единого стандарта Кобола - Грейс Хоппер (*бабушка Кобола*).

Обычно критикуется за многословность и громоздкость.

Однако имел прекрасные для своего времени средства для работы со структурами данных и файлами.

Лисп

(англ. LISP от LISt Processing – обработка списков)

Создан в 1959 – 1960 гг. в Массачусетском технологическом институте.

Основан на представлении программы системой линейных списков символов, которые притом являются основной структурой данных языка.

Широко используется для обработки символьной информации и применяется для создания программного обеспечения, имитирующего деятельность человеческого мозга.

Программа на Лиспе состоит из последовательности *выражений* (форм). Результат работы программы состоит в вычислении этих выражений. Все выражения записываются в виде *списков*.

Бейсик

(англ. BASIC от Beginner's Allpurpose Instruction Code – всецелевой символический код инструкций для начинающих)

Создан в середине 60-х годов (1963 г.) в Дартмутском колледже (США).

Основан частично на Фортран II Основан частично на Фортран II и частично на Алгол-60, с добавлениями, делающими его удобным для работы в режиме разделения времени и, позднее, обработки текста и матричной арифметики.

В силу простоты языка Бейсик многие начинающие программисты начинают с него свой путь в программировании.

Форт

(англ. FOURTH – четвёртый)

Появился в конце 60-х – начале 70-х годов.

Автор - Чарльз Мур написал на нём программу, предназначенную для управления радиотелескопом Аризонской обсерватории.

Стал применяться в задачах управления различными системами. Ряд свойств, а именно интерактивность, гибкость и простота разработки делают Форт весьма привлекательным и эффективным языком в прикладных исследованиях и при создании инструментальных средств.

Областями применения этого языка являются встраиваемые системы управления. Также находит применение при программировании компьютеров под управлением различных операционных систем.

Паскаль

Появился в 1972 году.

Был создан швейцарским учёным, специалистом в области информатики Никлаусом Виртом как язык для обучения методам программирования.

Паскаль – это язык программирования общего назначения.

Особенностями языка являются строгая типизация и наличие средств структурного (процедурного) программирования.

Интенсивное развитие Паскаля привело к появлению уже в 1973 году его стандарта в виде пересмотренного сообщения, а число трансляторов с этого языка в 1979 году перевалило за 80.

В начале 80-х годов Паскаль еще более упрочил свои позиции с появлением трансляторов MS-Pascal и Turbo-Pascal для ПЭВМ.

Основные причины популярности Паскаля:

- простота языка позволяющая быстро его освоить;
- удобство работы как с числовой, так и с символьной и битовой информацией;
- в языке Паскаль реализуются идеи структурного программирования, что делает программу наглядной и дает хорошие возможности для разработки и отладки;
- является прототипом для языков нового поколения;
- дает очень много в понимании сущности программирования;
- прививает хороший стиль программирования, тщательную разработку алгоритма.

Преимущества этого языка особенно ощутимы при написании достаточно сложных и мобильных (т. е. легко переносимых на другие PC) программ.

Ада

Создан в конце 70-х годов на основе языка Паскаль.

Назван в честь одарённого математика Ады Лавлейс (Огасты Ады Байрон – дочери поэта Байрона). Именно она в 1843 году смогла объяснить миру возможности Аналитической машины Чарльза Бэббиджа.

Был разработан по заказу Министерства обороны США.

Первоначально предназначался для решения задач управления космическими полётами.

Применяется в задачах управления бортовыми системами космических кораблей, системами обеспечения жизнедеятельности космонавтов в полёте, сложными техническими процессами.

Ада — это структурный, модульный, объектно-ориентированный язык программирования, содержащий высокоуровневые средства программирования параллельных процессов.

Си

Берёт своё начало от двух языков - BCPL и B.

В 1967 году Мартин Ричардс разработал BCPL как язык для написания системного программного обеспечения и компиляторов. В 1970 году Кен Томпсон использовал B для создания ранних версий операционной системы UNIX на компьютере DEC PDP-7. Как в BCPL, так и в B переменные не разделялись на типы - каждое значение данных занимало одно слово в памяти.

Язык Си был разработан (на основе B) Деннисом Ритчи из Bell Laboratories и впервые был реализован в 1972 году на компьютере DEC PDP-11.

Известность Си получил в качестве языка ОС UNIX. Сегодня практически все основные операционные системы написаны на Си или C++.

Пролог

«ПРОграммирование на языке ЛОГики»

Был создан в начале 70-х годов группой специалистов Марсельского университета.

В основе этого языка лежат законы математической логики.

Применяется, в основном, при проведении исследований в области программной имитации деятельности мозга человека.

Не является алгоритмическим. Он относится к так называемым **дескриптивным** (от англ. descriptive – описательный) – описательным языкам. Дескриптивный язык не требует от программиста разработки всех этапов выполнения задачи. Вместо этого, в соответствии с правилами такого языка, программист должен описать базу данных, соответствующую решаемой задаче, и набор вопросов, на которые нужно получить ответы, используя данные из этой базы.

В последние десятилетия в программировании возник и получил существенное развитие **объектно-ориентированный** подход. Это метод программирования, имитирующий реальную картину мира: информация, используемая для решения задачи, представляется в виде множества взаимодействующих объектов. Каждый из объектов имеет свои свойства и способы поведения. Взаимодействие объектов осуществляется при помощи передачи сообщений: каждый объект может получать сообщения от других объектов, запоминать информацию и обрабатывать её определённым способом и, в свою очередь, посылать сообщения. Так же, как и в реальном мире, объекты хранят свои свойства и поведение вместе, наследуя часть из них от родительских объектов.

Объектно-ориентированная идеология используется во всех современных программных продуктах, включая операционные системы.

Первый объектно-ориентированный язык *Simula-67* был создан как средство моделирования работы различных приборов и механизмов. Большинство современных языков программирования – объектно-ориентированные. Среди них последние версии языка *Turbo-Pascal, C++, Ada* и другие.

В настоящее время широко используются системы **визуального программирования** *Visual Basic, Visual C++, Delphi* и другие. Они позволяют создавать сложные прикладные пакеты, обладающие простым и удобным пользовательским интерфейсом.

Pascal

- разработан профессором кафедры вычислительной техники Швейцарского Федерального института технологии Николасом Виртом в 1968 году

- назван так в честь великого французского математика, физика, философа и писателя XVII века, изобретателя первой в мире арифметической машины Блеза Паскаля

(1623 - 1662)

Основные файлы пакета Турбо Паскаль:

- Turbo.exe – интегрированная среда программирования;
- Turbo.hlp – файл, содержащий данные для оперативной подсказки;
- Turbo.tp – файл конфигурационной системы;
- Turbo.tpl – библиотека стандартных модулей Турбо Паскаля.

Структура программы на Pascal

Program <имя программы>;

Uses <имя1, имя2,…>; - список имен подключаемых стандартных и пользовательских библиотечных модулей

Label <описание меток>;

Const <описание констант>;

Type <описание типов>;

Var <описание переменных>;

Procedure(Function) <описание подпрограмм>;

Begin

<раздел операторов>;

end.

Алфавит Pascal

- прописные и строчные буквы латинского алфавита: A, B, C…Y, Z, a, b, c,…y, z ;
- десятичные цифры: 0, 1, 2,…9;
- специальные символы: + - * / > < = ; # ‘ , . : {} [] ( )
- комбинации специальных символов , которые нельзя разделять пробелами, если они используются как знаки операций: «:=», «..», «<>», «<=», «>=», «{}».

Словарь Словарь Pascal

- зарезервированные слова
- стандартные идентификаторы
- идентификаторы пользователя

Зарезервированные слова *имеют фиксированное написание и навсегда определенный смысл. Они не могут изменяться программистом и их нельзя использовать в качестве имен для обозначения величин.*

Некоторые зарезервированные слова версии Турбо Паскаль

Absolute	Абсолютный	**Library**	Библиотека
And	Логическое И	**Mod**	Остаток от деления
Array	Массив	**Not**	Логическое НЕ
Begin	Начало блока	**Or**	Логическое ИЛИ
Case	Вариант	**Of**	Из
Const	Константа	**Object**	Объект
Div	Деление нацело	**Procedure**	Процедура
Go to	Переход на	**Program**	Программа
Do	Выполнять	**Repeat**	Повторять
Downto	Уменьшить до	**String**	Строка
Else	Иначе	**Then**	То
End	Конец блока	**To**	Увеличивая
File	Файл	**Type**	Тип
For	Для	**Until**	До
Function	Функция	**Uses**	Использовать
If	Если	**Var**	Переменная
Interrupt	Прерывание	**While**	Пока
Interface	Интерфейс	**With**	С
Label	Метка	**Xor**	Исключающее ИЛИ

Идентификатор – имя (identification – установление соответствия объекта некоторому набору символов).

Для обозначения определенных разработчиками языка функций, констант и т.д. служат **стандартные идентификаторы**, например Sqr, Sqrt и т.д.

В этом примере Sqr вызывает функцию, которая возводит в квадрат данное число, а Sqrt – корень квадратный из заданного числа.

Идентификаторы пользователя – это те имена, которые дает сам программист.

Правила написания идентификаторов:

- Идентификатор начинается только с буквы (исключение составляют специальные идентификаторы меток).
- Идентификатор может состоять из букв, цифр и знака подчеркивания.
- Максимальная длина – 127 символов.
- При написании идентификаторов можно использовать прописные и строчные буквы.
- Между двумя идентификаторами должен стоять хотя бы один пробел.

Типы данных Pascal

Определяют:

- Объем ОП для размещения данного.
- Диапазон допустимых значений.
- Допустимые операции.

✔ ***Простые (скалярные):***
неделимы;
упорядочены (кроме вещественного).

✔ ***Структурированные:***
упорядоченная совокупность скалярных переменных;
характеризуются типом своих компонентов.

Типы данных Pascal

Простые (скалярные):

- Целочисленные
- Вещественные
- Литерный (символьный)
- Булевский Булевский (Булевский (логическийБулевский (логический)
- Пользовательские:

перечисляемый;

интервальный.

Структурированные:

- Строковый
- Массивы
- Множества
- Записи
- Файлы
- Указатели
- Процедурные
- Объекты

Целочисленные типы данных

Тип	*Диапазон*	*Требуемая память (байт)*
Byte	0…255	1
Shorint	-128 …127	1
Integer	-32768 … 32767	2
Word	0 … 65535	2
Longint	-2147483648 …2147483647	4

Значения целых типов могут изображаться в программе 2 способами: в десятичном виде и в шестнадцатеричном. Если число представлено в шестнадцатеричной системе, перед ним без пробела ставится знак $, а цифры старше 9 обозначаются латинскими буквами от A до F. Диапазон изменений таких чисел от $0000 до $FFFF .

Допустимые операции:

- Арифметические операции

  +, -, *, /, Div, Mod
- Операции сравнения

  <, >, <=, >=, <>, =
- Стандартные функции и процедуры

  Abs (x), Sqr (x), Sqrt (x)

  Sin, Cos, Exp, Pred, Succ, Ord, Odd и т.п

Вещественные типы данных

Тип	*Диапазон*	*Мантисса*	*Требуемая память (байт)*
real	2.9*10E-39..1.7*10E38	11 – 12	6
single	1.5*10E-45..3.4*10E38	7 – 8	4
double	5.0*10E-324..1.7*10E308	15 – 16	8
extended	1.9*10E-4951..1.1*10E4932	19 – 20	10
comp	-2E+63+1..2E+63-1	10 – 20	8

Допустимые операции:

- **Арифметические**
  +, -, *, /
- **Сравнения**
  <, >, <=, >=, =, <>
- **Стандартные функции и процедуры**
  Abs (x), Sqr (x), Sqrt (x), Exp (x), Sin (x), Cos (x),
  Round (x)-округление целой части
  Trunc (x)-отбрасывание дробной части
  Int (x)-вычисление целой части
  Frac (x)-вычисление дробной части

Вещественные значения могут изображаться в форме ***с фиксированной точкой****, а также ф форме* ***с плавающей точкой****, т.е. парой чисел вида <мантисса>E<порядок>.*

с фиксированной точкой	***с плавающей точкой***
7.32	7.32E+00
456.721	4.56721E+02
0.015	1.5E-02

Вещественные числа по умолчанию выводятся на экран в формате с плавающей точкой. Для вывода в форме с фиксированной необходимо указать формат вывода.

Например: в ячейке a хранится число 1.232 E+02

Если использовать процедуру Writeln (a); то на экране будет число 1.232 E+02

Если использовать процедуру Writeln(a:6:2); 6 – общее число позиций (включая точку)

2 – число позиций после точки.

То на экране будет число 123.20 – 6 позиций, 2знака после точки.

Литерный (символьный) тип

Char

Определяется множеством значений кодовой таблицы ПК. Каждому символу задается целое число от 0 до 255. Для кодировки используется код ASCII.

Например код символа 'А' при русской раскладке клавиатуры будет равен 192.

В программе значения переменных и констант типа char должны быть заключены в апострофы.

Для размещения в памяти переменной литерного типа нужен 1 байт.

Допустимые операции

- **операции отношения:**

  =, <>, >,<,<=,>=;

  вырабатывают результат логического типа
- **стандартные функции:**

  Chr(x) – преобразует выражение x в символ и возвращает значение символа

  Ord(ch) – преобразует символ ch в его код и возвращает значение кода

  Pred(ch) – возвращает предыдущий символ

  Succ(ch) – возвращает следующий символ

Логический (Булевский) тип

Могут принимать только одно из 2-х значений:
TRUE или **FALSE**.

В памяти занимают 1 байт.

Описание: **Var <имя>: Boolean;**

Допустимые операции

- **операции сравнения**

  =, <>, <=, >=, <, >

- **функции и процедуры**

  Pred (True)=False;

  Ord (True)=1;

  Succ (False)=True;

  Ord (False)=0;

логические операции

- а) конъюнкция (логическое "И", логическое умножение) – AND

Истина тогда и только тогда, когда оба операнда истинны.

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

логические операции

- дизъюнкция (логическое сложение, логическое "ИЛИ") – OR

Ложь тогда и только тогда, когда оба ложны.

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

логические операции

- исключающее "ИЛИ" –XOR

Истина тогда, когда операнды имеют противоположное значение.

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

логические операции

- отрицание – NOT

Результат операции – противоположное значение аргумента

A	not B
0	1
1	0

Пользовательские типы

Занимают 1 байт памяти

Не могут содержать более 256 элементов.

- ***Перечисляемый***

  (enumerated type)

задается списком принадлежащих ему значений

Формат:

Type<имя типа>**=(**<зн.1, зн.2, …,зн.n>**);**

Var<идентификатор,…>**:**<имя типа>**;**

- ***Интервальный***

  (диапазон)

Две константы определяют границы диапазона значений для данной переменной

Принадлежат одному из стандартных типов (real недопустим!)

Значение const1<const2

Формат:

Type<имя типа>**=**<const1>. .<const2>**;**

Var<идентификатор,…>**:**<имя типа>**;**

Строковый тип данных

Строка – упорядоченная последовательность символов кодовой таблицы ПК

1 символ – 1 байт

Длина строки – количество символов в строке. (0 – 255)

Основные понятия

- **Строковая константа** – последовательность символов, заключенных в апострофы.
  - ‘272’
    ‘это строка’
    ‘’
- **Строковая переменная var <идентификатор>: string[< max длина >];** (по умолчанию 255)
  - var name:string[20];
    var str:string;
- **Элементы строки <строка>[<№элемента>]**
  - N[5]
    S[i]
    slovo[k+1]

Операции над строками

- **Сцепления (конкатенации) (+)**
- **Отношения (=, <, >,<=, >=, <>)**

- ‘мама’+’мыла’+’раму’= =‘мама мыла раму’
- ‘True1’<‘True2’
  ‘Mother’>’MOTHER’
  ‘Мама’<>’_Мама’
  ‘Cat’=‘Cat’

Процедуры и функции

Функция Copy(S,Poz,N)	Выделяет из строки S подстроку длиной N символов, начиная с позиции Poz. N и Poz – целочисленные выражения.
Функция Concat(S1,S2,…,Sn)	Выполняет сцепление (конкатенацию) строк S1, S2,…,Sn
Функция Length(S)	Определяет текущую длину строки S. Результат – значение целого типа.
Функция Pos(S1,S2)	Обнаруживает первое появление в строке S2 подстроки S1. Результат – целое число, равное номеру позиции, где находится первый символ подстроки S1. Если в S2 подстроки S1 не обнаружено, то результат равен 0.
Процедура Delete(S,Poz,N)	Удаление N символов из строки S, начиная с позиции Poz.
Процедура Insert(S1,S2,Poz)	Вставка строки S1 в строку S2, начиная с позиции Poz.

Массив – это упорядоченная последовательность данных, состоящая из фиксированного числа элементов, имеющих один и тот же тип, и обозначаемая одним именем.

(Тип компонент массива называется **базовым типом**)

Общий вид описания массива:

Type <имя нового типа данных>=array[<тип индекса>] of <тип компонентов>;

Var <имя массива>: array [<тип индекса>] of <тип компонентов>;

Операции над массивом как единым целым:

=, <> и оператор присваивания.

При этом массивы должны иметь одинаковую размерность и один и тот же тип элементов!

Все остальные операции совершаются только над отдельными элементами массива!

Массивы

- **Одномерные –** элементы – простые переменные.

- **Двумерные** – структура данных, хранящая прямоугольную матрицу.

Способ описания:

Var M: array[1..10] of array[1..20] of real;

или

Var M: array[1..10, 1..20] of real;

Доступ к каждому отдельному элементу осуществляется обращением к имени массива с указанием индексов (первый индекс – номер строки, второй индекс – номер столбца).

Множество – набор взаимосвязанных по какому-либо признаку или группе признаков объектов, которые можно рассматривать как единое целое.

- **Элемент множества** – каждый его объект (принадлежит любому скалярному типу, кроме вещественного)
- **Базовый тип множества** – тип элементов множества (задается диапазоном или перечислением)
- **Область значений типа множество** – набор всевозможных подмножеств, составленных из элементов базового типа
- *Пример:* [1,2,3,4], [‘a’,’b’,’c’], [‘a’..’z’] – множества;
  [ ] - пустое множество.
- Мощность – количество элементов множества

Формат записи:

type <имя типа> **= set of** <элемент1, …,элементn>**;**

var <идентификатор,…> **:** <имя типа>**;**

или

var <идентификатор,…> **: set of** <элемент1,…>**;**

Операции над множествами:

- отношения: **“=“, “<>”, “>=“, “<=”**
- объединения **(+)**
- пересечения **(*)**
- разности множеств **(-)**
- операция **in** (для проверки принадлежности какого-либо значения указанному множеству)

Запись – состоит из фиксированного числа компонентов одного или нескольких типов.

Формат:

type <имя типа> **= record**

<идентификатор поля> **:** <тип компонента>**;**

…

<идентификатор поля> **:** <тип компонента>

end;

var <идентификатор,…> **:** <имя типа>**;**

Обращение к значению поля осуществляется с помощью идентификатора переменной и идентификатора поля, разделенных точкой **(составное имя)**

Например: M.Number, M.FIO

Файл — совокупность данных, записанная во внешней памяти под определенным именем.

Формат:

Type <имя типа> **=** <тип компонентов>**;**

Var <F> **: file of** <имя типа>**;**

<R> **:** <имя типа>**;**

Указатель – это переменная, которая в качестве своего значения содержит адрес первого байта памяти, по которому записаны данные.

Занимает 4 байта памяти

Подпрограмма – программа, реализующая вспомогательный алгоритм.

- **Подпрограмма-функция**

  function <имя функции> **(**<параметры-аргументы**>) :** <тип функции>**;**

  <блок>**;**

  Обращение к функции является операндом в выражении.

- **Подпрограмма процедура**

  procedure <имя процедуры> **(**<параметры>**);**

  <блок>**;**

  Обращение к процедуре – отдельный оператор.

Стандартные библиотечные модули

обеспечивают доступность встроенных процедур и функции

- **System** - сердце Турбо Паскаля. Подпрограммы, содержащиеся в нем, обеспечивают работу всех остальных модулей системы.
- **Crt** - содержит средства управления дисплеем и клавиатурой компьютера.
- **Dos** - включает средства, позволяющие реализовывать различные функции Dos.
- **Graph3** - поддерживает использование стандартных графических подпрограмм.
- **Overlay** - содержит средства организации специальных оверлейных программ.
- **Printer** - обеспечивает быстрый доступ к принтеру.
- **Turbo3** - обеспечивает максимальную совместимость с версией Турбо Паскаль 3.0.
- **Graph** - содержит пакет графических средств.
- **Turbo Vision** - библиотека объектно-ориентированных программ для разработки пользовательских интерфейсов.

Типы операторов Pascal

Простые

- Оператор присваивания
- Процедуры ввода-вывода
- Оператор безусловного перехода (Оператор безусловного перехода (go toОператор безусловного перехода (go to)
- Операторы вызова

Структурные

- Составной оператор
- Условный оператор
- Оператор выбора
- Операторы цикла

Оператор присваивания

<имя> := <выражение>;

Процедуры ввода-вывода

1.Процедуры ввода (чтения) данных:

Read [ln] (x1, x2, xn);

Где x1, x2 – имена переменных, куда помещаются вводимые данные. Тип вводимых должен совпадать с типом переменных.

Значение x1, x2… введется с клавиатуры минимум через 1 пробел (или Enter). Ввод данных заканчивается нажатием <Enter>.

Процедура **Read** производит ввод данных, не переводя при этом курсор на следующую строку, а процедура **ReadIn** производит ввод данных и перевод курсора на следующую строку.

Использование процедуры ReadIn без параметров -после нажатия клавиши <Enter> переводит курсор на следующую строку.

2.Процедуры вывода данных:

Write [ln] (y1, y2, …yn);

Где y1, y2, yn – выражения или имена выводимых переменных.

Процедура Write производит вывод, не переводя курсор на другую строку, а Writeln после вывода данных переводит курсор на следующую строку.

Оператор безусловного перехода

go to - «перейти к» и применяется в случаях, когда после выполнения некоторого оператора надо выполнить не следующий по порядку, а какой-либо другой, отмеченный меткой, оператор.

Общий вид: **go to <метка>.**

Метка объявляется в разделе описания меток и состоит из имени и следующего за ним двоеточия.
Имя метки может содержать цифровые и буквенные символы, максимальная длина имени ограничена 127 знаками.
Раздел описания меток начинается зарезервированным словом Label, за которым следует имя метки.

Пустой оператор

Пустой оператор не содержит никаких символов и не выполняет никаких действий. Используется для организации перехода к концу блока в случаях, если необходимо пропустить несколько операторов, но не выходить из блока. Для этого перед зарезервированным словом end ставятся метка и двоеточие, *например:*

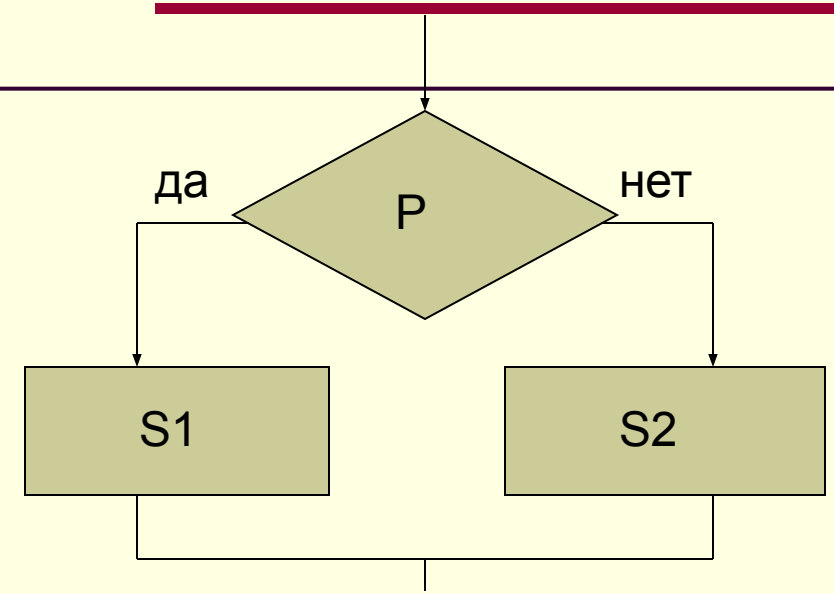
```
Label m;
…
begin
…
go to m;
…
m:
end;
```

Составной оператор

Этот оператор представляет собой совокупность произвольного числа операторов, отделенных друг от друга точкой с запятой, и ограниченную операторными скобками **begin** и **end**.

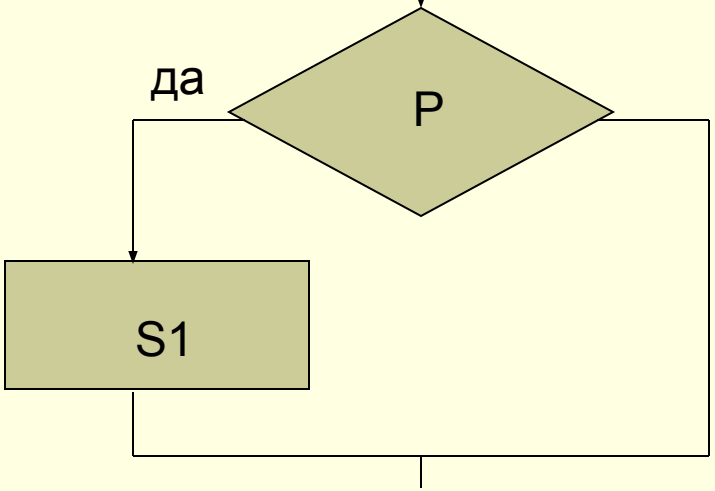
Он воспринимается как единое целое и может находиться в любом месте программы, где возможно наличие оператора.

Условный оператор



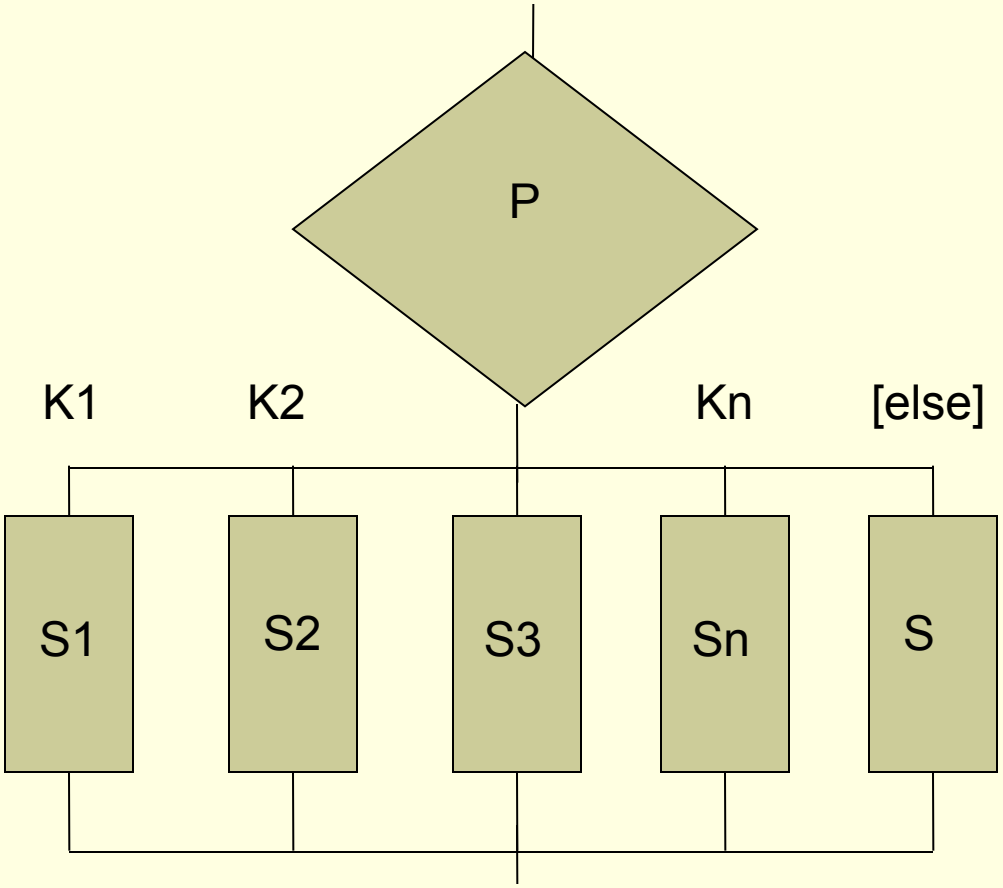
If <P> then <S1>
else <S2>;

P – выражение булевского типа.
S1, S2 – простые или составные операторы.



If <P> then <S1>;

Оператор выбора



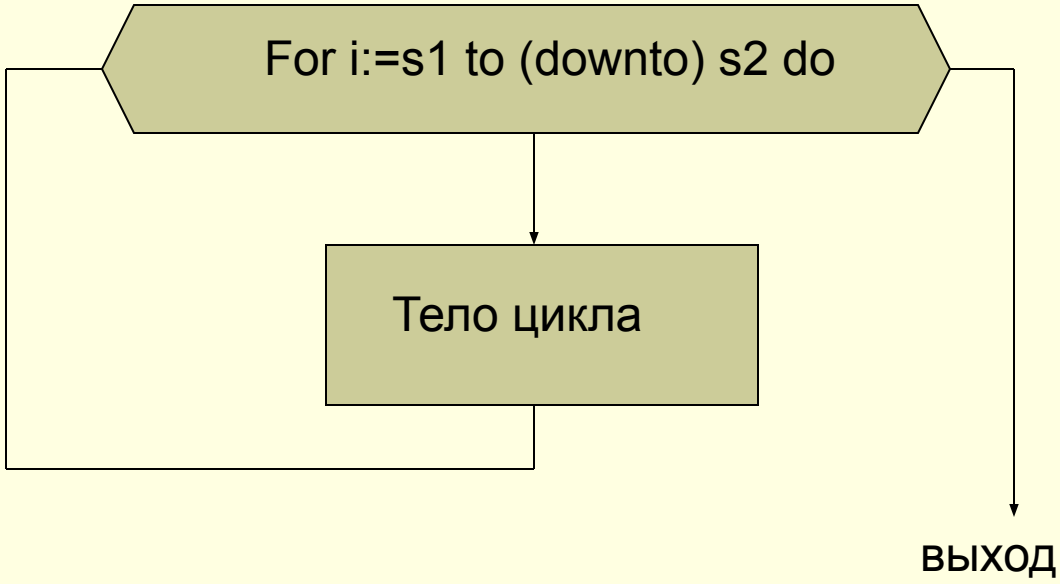
Case K of
K1:S1;
K2:S2;
……
KN: SN
[Else S;]
End;

K – селектор выбора (переменная или выражение целочисленное, булевского или символьного типа)

K1, K2, … KN – константы выбора (тип совпадает с типом селектора)

S1, S1, … SN – простые или составные операторы.

Оператор цикла for (цикл с параметром)



i – параметр цикла

S1 – начальное значение

S2 – конечное значение

Формат записи:

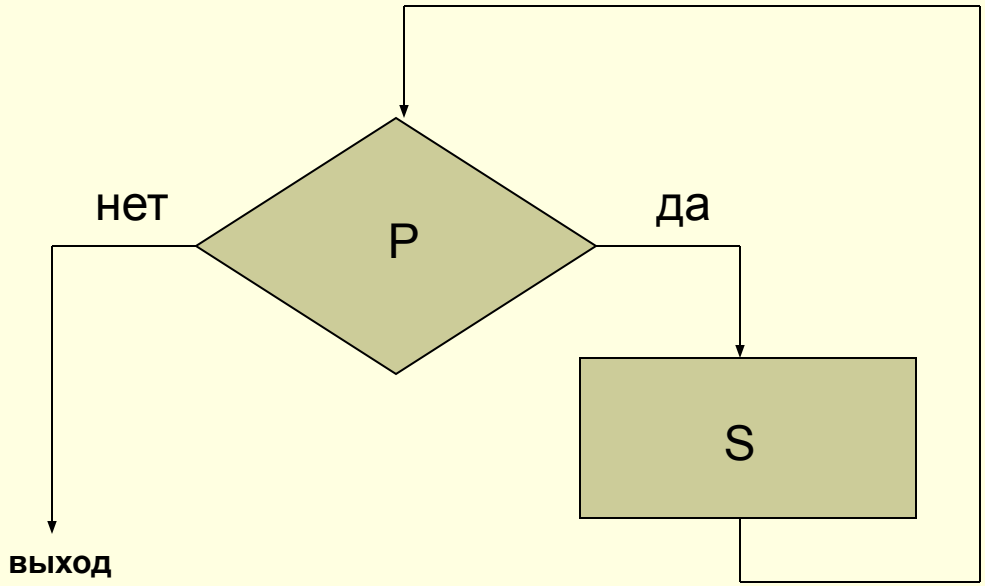
For i:=s1 to (downto) s2 do

<тело цикла>;

To – шаг «1»

Downto – шаг «-1»

Оператор цикла while (цикл с предусловием, «пока»)

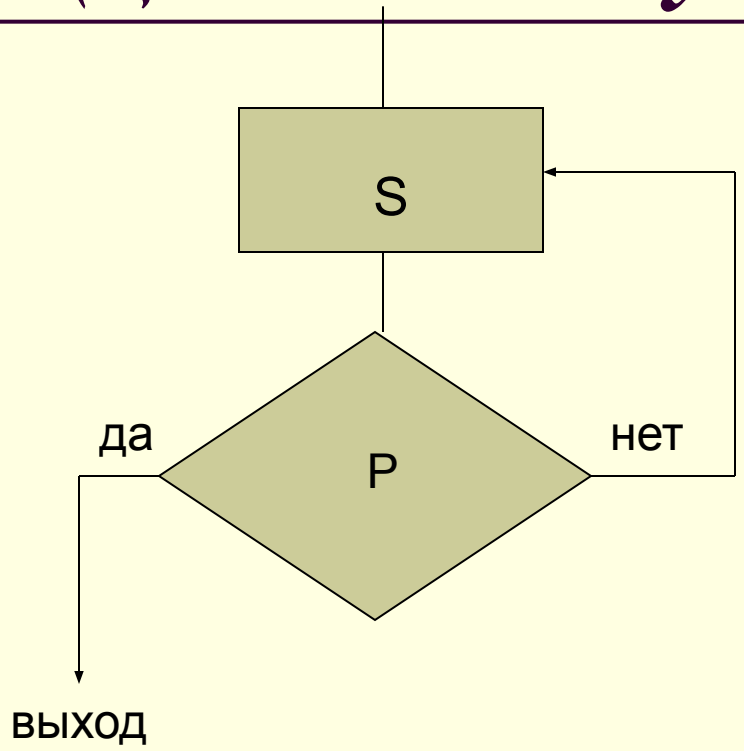


P – условие повторения тела цикла (выражение логического (булевского) типа).
S – простой или составной оператор.

Формат записи:

While <P> do <S>;

Оператор цикла repeat (цикл с постусловием, «до»)



Формат записи:

Repeat <S>

Until <P>;

В цикле с постусловием условие цикла проверяется после очередного выполнения тела цикла.
S – тело цикла;
P – условие выхода из цикла (выражение булевского типа);
Выход из цикла при P=TRUE.
В цикле Repeat тело выполняется хотя бы один раз.

Графика в Pascal

- Инициализация графического режима
- Базовые процедуры и функции
- Дуги и окружности
- Построение многоугольников
- Иллюзия движения
- Работа с текстом

Инициализация графического режима

- **Инициирует графический режим работы адаптера.** Заголовок процедуры:
- Procedure InitGraph(var Driver,Mode: Integer; Path: String);
- Здесь **Driver** - тип графического драйвера;

Mode – переменная, в которую процедура помещает код работы графического адаптера;
Path - имя файла драйвера и, к
К моменту вызова процедуры
информации должен находить
графический драйвер. Процед
оперативную память и перево
жим работы. Тип драйвера дол
графического адаптера. Для у
предопределены следующие ко

```
const
Detect=0;{Режим автоопределения типа}
CGA=1;
MCGA=2;
EGA=3;
EGA64=4;
EGAMono=5;
IBM8514=6;
HercMono=7;
ATT400=8;
VGA=9;
PC3270=10;
```

- Большинство адаптеров могут работать в различных режимах. Для того чтобы указать адаптеру требуемый режим работы, используется переменная **Mode**, значением которой в момент обращения к процедуре могут быть такие константы:
- **Mode** – переменная, в которую процедура помещает код работы графического адаптера;

```
const
{ Адаптер CGA : }
CGACO = 0; {Низкое разрешение, палитра0}
CGAC1 = 1; {Низкое разрешение, палитра1}
CGAC2 = 2; {Низкое разрешение, палитра2}
CGAC3 = 3; {Низкое разрешение, палитра3}
CGAHi = 4; {Высокое разрешение}
{Адаптер MCGA:}
MCGACO = 0; {Эмуляция CGACO}
MCGAC1 = 1; {Эмуляция CGAC1}
MCGAC2 = 2; {Эмуляция CGAC2}
MCGAC3 = 3; {Эмуляция CGAC3}
MCGAMed = 4; {Эмуляция CGAHi}
MCGAHi = 5; {640x480}
{Адаптер EGA :}
EGALo = 0; {640x200, 16 цветов}
EGAHi = 1; {640x350, 16 цветов}
EGAMonoHi = 3; {640x350, 2 цвета}
{Адаптеры HGC и HGC+:}
HercMonoHi = 0; {720x348}
{АдаптерATT400:}
ATT400CO = 0; {Аналог режима CGACO}
ATT400C1 = 1; (Аналог режима CGAC1}
ATT400C2 = 2; {Аналог режима CGAC2}
ATT400C3 = 3; {Аналог режима CGAC3}
ATT400Med = 4; {Аналог режима CGAHi}
ATT400H1 = 5; {640x400, 2 цвета}
{Адаптер VGA:}
VGALo = 0; {640x200}
VGAMed = 1; {640x350}
VGAHi = 2; {640x480}
PC3270H1 = 0; {Аналог HercMonoHi}
{Адаптер 1BM8514}
IBM8514LO =0; {640x480, 256 цветов}
IBM8514H1 = 1; {1024x768, 256 цветов}
```

- Пусть, например, драйвер **CGA.BGI** находится в каталоге **TP\BGI** на диске C и устанавливается режим работы 320x200 с палитрой 2. Тогда обращение к процедуре будет таким:

```
Uses Graph;
var
Driver, Mode : Integer;
begin
Driver := CGA;{Драйвер}
Mode := CGAC2;{Режим работы}
InitGraph(Driver, Mode,' C:\TP\BGI') ;
.......
```

Если тип адаптера ПК неизвестен или если программа рассчитана на работу с любым адаптером, используется обращение к процедуре с требованием автоматического определения типа драйвера:

```
Driver := Detect;
InitGraph(Driver, Mode, 'C:\TP\BGI');
```

- После такого обращения устанавливается графический режим работы экрана, а при выходе из процедуры переменные **Driver** и **Mode** содержат целочисленные значения, определяющие тип драйвера и режим его работы. При этом для адаптеров, способных работать в нескольких режимах, выбирается старший режим, т.е. тот, что закодирован максимальной цифрой. Так, при работе с **CGA** -адаптером обращение к процедуре со значением **Driver = Detect** вернет в переменной **Driver** значение 1 (**CGA**) и в **Mode** - значение 4 (**CGAHi**), а такое же обращение к адаптеру **VGA**вернет **Driver** = 9 (**VGA**) и **Mode** = 2 (**VGAHi**).

Базовые процедуры и функции

- Для построения изображений на экране используется система координат. Отсчет начинается от верхнего левого угла экрана, который имеет координаты (0,0). Значение X (столбец) увеличивается слева направо, значение Y (строка) увеличивается сверху вниз. Чтобы строить изображения, необходимо указывать точку начала вывода. В текстовых режимах эту точку указывает курсор, который присутствует на экране. В графических режимах видимого курсора нет, но есть невидимый текущий указатель CP(Current Pointer). Фактически это тот же курсор, но он невидим.

Процедура	Формат	Действие
SetColor	SetColor(a: word);	Устанавливает цвет, которым будет осуществляться рисование
SetBkColor	SetBkColor(a: word);	Устанавливает цвет фона
SetFillStyle	SetFillStyle(a, b: word); a - стиль закраски, b - цвет	Устанавливает стиль и цвет закраски
SetLineStyle	SetLineStyle(a, b, c: word); a - стиль линии, b- образец построения линии (может устанавливаться пользователем), c-толщина линии	Устанавливает стиль и толщину линии
SetTextStyle	SetTextStyle(a, b, c: word);	Устанавливает шрифт, стиль и размер текста
SetFillPattern	SetFillPattern(Pattern: FillpatternType; Color: word); Pattern - маска	Выбирает шаблон заполнения, определенный пользователем
ClearDivice	ClearDivice;	Очищает экран и устанавливает текущий указатель в начало
SetViewPort	SetViewPort(x1, y1, x2, y2: integer, Clip: boolean);	Устанавливает текущее окно для графического вывода
ClearViewPort	ClearViewPort	Очищает окно
PutPixel	PutPixel(a, b, c: integer);	Рисует точку цветом c в (x,y)
Line	Line(x1, y1, x2, y2: integer);	Рисует линию от (x1, y1) к (x2,y2)
Rectangle	Rectangle(x1, y1, x2, y2: integer);	Рисует прямоугольник с диагональю от (x1, y1) к (x2, y2)

Bar	Bar(x1, y1, x2, y2: integer);	Рисует закрашенный прямоугольник
Bar3D	Bar3D(x1, y1, x2, y2, d: integer; a: boolean);	Рисует трехмерную полосу (параллелепипед)
Circle	Circle(x, y, r: word);	Рисует окружность радиуса r с центром в точке (x, y)
Arc	Arc(x, y, a, b, R: integer); a, b- начальный и конечный углы в градусах	Рисует дугу из начального угла к конечному, используя (x,y) как центр
Ellipse	Ellipse(x, y, a, b, Rx, Ry: integer); a, b - начальный и конечный углы в градусах	Рисует эллиптическую дугу от начального угла к конечному, используя (x, y) как центр
FillEllipse	FillEllipse(x, y, Rx, Ry: integer); Rx, Ry - вертикальная и горизонтальная оси	Рисует закрашенный эллипс
MoveTo	MoveTo(x, y: integer);	Передвигает текущий указатель в (x, y)
MoveRel	MoveRel(x, y: integer);	Передвигает текущий указатель на заданное расстояние от текущей позиции на x по горизонтали и на y по вертикали
OutText	OutText(text: string);	Выводит текст от текущего указателя
OutTextxy	OutTextxy(x, y: integer; text: string);	Выводит текст из (x, y)
Sector	Sector(x, y, a, b, Rx, Ry: integer); a, b - начальный и конечный углы в градусах	Рисует и заполняет сектор эллипса

Функции

GetBkColor	Возвращает текущий фоновый цвет
GetColor	Возвращает текущий цвет
GetX	Возвращает координату X текущей позиции
GetY	Возвращает координату Y текущей позиции
GetPixel	Возвращает цвет точки в (x, y)

Построение дуг и окружностей

- **Процедура вычерчивания окружности текущим цветом имеет следующий формат:*Cicrle (x,y,r:word)*, где x,y – координаты центра окружности, r – ее радиус.**
Например, фрагмент программы обеспечит вывод ярко-зеленой окружности с радиусом 50 пикселей и центром в точке (450, 100):

```
SetColor(LightGreen);
Circle(450, 100, 50);
```

- **Дуги можно вычертить с помощью процедуры *Arc(x,y:integer,a,b,R:integer)*, где x,y - центр окружности, a,b - начальный и конечный углы в градусах, R – радиус. Для задания углов используется полярная система координат. Цвет для вычерчивания устанавливается процедурой *SetColor*.В случае a=0 и b=360, вычерчивается полная окружность. Например, выведем дугу красного цвета от 0 до 90° в уже вычерченной с помощью Circle(450, 100, 50) окружности:**

```
SetColor(Red);
Arc(450, 100, 0, 90, 50);
```

- **Для построения эллиптических дуг предназначена процедура *Ellipse (x, y: integer, a, b, R_x, R_y: integer)*, где x, y – центр эллипса, R_x, R_y:горизонтальная и вертикальная оси. В случае a=0 и b=360 вычерчивается полный эллипс. Например, построим голубой эллипс:**

```
SetColor (9);
Ellipse (100, 100, 0, 360, 50, 50);
```

- **Фон внутри эллипса совпадает с фоном экрана. Чтобы создать закрашенный эллипс, используется специальная процедура *FillEllipse (x, y: integer, R_x, R_y: integer)*. Закраска эллипса осуществляется с помощью процедуры *SetFillStyle(a,b:word)*, где a – стиль закраски (таблица 4), b – цвет закраски (таблица 1).**
Например, нарисуем ярко-красный эллипс, заполненный редкими точками зеленого цвета:

```
SetFillStyle (WideDotFill,Green); { установка стиля заполнения}
SetColor (12); {цвет вычерчивания эллипса}
FillEllipse(300, 150, 50, 50);
```

Стандартные стили заполнения

Константа	Значение	Маска
EmptyFill	0	Заполнение цветом фона
SolidFill	1	Заполнение текущим цветом
LineFill	2	Заполнение символами --, цвет – color
LtslashFill	3	Заполнение символами // нормальной толщины, цвет – color
SlashFill	4	Заполнение символами // удвоенной толщины, цвет – color
BkslashFill	5	Заполнение символами \ удвоенной толщины, цвет – color
LtbkSlahFill	6	Заполнение символами \ нормальной толщины, цвет – color
HatchFill	7	Заполнение вертикально-горизонтальной штриховкой тонкими линиями, цвет – color
XhatchFill	8	Заполнение штриховкой крест-накрест по диагонали «редкими» тонкими линиями, цвет – color
InterLeaveFill	9	Заполнение штриховкой крест-накрест по диагонали «частыми» тонкими линиями, цвет – color
WideDotFill	10	Заполнение «редкими» точками
CloseDotFill	11	Заполнение «частыми» точками
UserFill	12	Заполнение по определенной пользователем маске заполнения, цвет – color

- **Для построения секторов можно использовать следующие процедуры:**
  PieSlice (x,y:integer,a,b,R:word)**,которая рисует и заполняет сектор круга. Координаты x,y – центр окружности, сектор рисуется от начального угла a до конечного угла b, а закрашивание происходит при использовании специальных процедур;**

  Sector (x, y: integer, a, b, R_x, R_y: word)**, которая создает и заполняет сектор в эллипсе. Координаты x,y – центр, b, R_x, R_y – горизонтальный и вертикальный радиусы, и сектор вычерчивается от начального угла a до конечного угла b.**

- **Пример использования *PieSlice***
- ***SetFillStyle (10, 10); {установка стиля}***
  SetColor (12); {цвет вычерчивания}
  PieSlice (100, 100, 0, 90, 50);

- **Пример использования Sector**
- ***SetFillStyle (11, 9); {установка стиля}***
  SetColor (LightMagenta);{цвет вычерчивания}
  Sector (300, 150, 180, 135, 60, 70);

Построение многоугольников

- Для построения прямоугольных фигур имеется несколько процедур. Первая из них – вычерчивание одномерного прямоугольника: Rectangle(x1,y1,x2,y2:integer), где x1, y1– координаты левого верхнего угла, x2, y2- координаты правого нижнего угла прямоугольника.

- **Область внутри прямоугольника не закрашена и совпадает по цвету с фоном. Более эффектные для восприятия прямоугольники можно строить с помощью процедуры *Bar(x_1, y_1, x_2, y_2:integer)*, которая рисует закрашенный прямоугольник. Цвет закраски устанавливается с помощью*SetFillStyle*. Ещё одна эффектная процедура: *Bar3D(x_1,y_1, x_2,y_2, d:integer,a:boolean)* вычерчивает трехмерный закрашенный прямоугольник (параллелепипед).**

- **При этом используются тип и цвет закраски, установленные с помощью *SetFillStyle*. Параметр d представляет собой число пикселей, задающих глубину трехмерного контура. Чаще всего его значение равно четверти ширины прямоугольника *(d:=(x_2 - x_1) div4)*. Параметр a определяет, строить над прямоугольником вершину (a:=True) или нет (a:=False).**

Примеры использования:

1. ***SetColor(Green);***
   Rectangle (200, 100, 250, 300);
2. ***SetFillStyle(1,3);***
   Bar(10, 10, 50, 100);
3. ***SetFillStyle(1,3);***
   Bar3D(10,10,50,100,10,True);

Построение многоугольников

- **Многоугольники можно рисовать самыми различными способами, например с помощью процедуры *Line*.Однако в Турбо Паскале имеется процедура *DrawPoly*, которая позволяет строить любые многоугольники линией текущего цвета, стиля и толщины. Она имеет формат *DrawPoly(a: word, var PolyPoints*) Параметр *PolyPoints* является нетипизированным параметром, который содержит координаты каждого пересечения в многоугольнике.**

- **Параметр a задает число координат в *PolyPoints.* Необходимо помнить, что для вычерчивания замкнутой фигуры с N вершина процеду координ координ**
- **Проилл**

```
program tr; {Программа вычерчивает в центре экрана треугольник красной линией}
  uses crt, graph;
var gd, gm: integer; pp:array[1..4] of PointType;
     xm,ym, xmaxD4, ymaxD4:word;
begin
  gd:=detect;
  Initgraph(gd,gm,'c:/bp');
xm:=GetmaxX;
  ym:=GetmaxY;
  xmaxD4:=xm div 4;
  ymaxD4:=ym div 4; {определение координат вершин}
  pp[1].x:= xmaxD4;
  pp[1].y:= ymaxD4;
  pp[2].x:= xm - xmaxD4;
  pp[2].y:= ymaxD4;
  pp[3].x:= xm div 2;
  pp[3].y:= ym - ymaxD4;
  pp[4]:=pp[1];
  SetColor(4); {цвет для вычерчивания}
  DrawPoly(4,pp); {4 – количество пересечений +1}
  readln;
  CloseGraph
end.
```

- **В результате работы программы на экране появится красный треугольник на черном фоне. Изменить фон внутри треугольника можно с помощью процедуры *FillPoly(a:word,var PolyPoints)*. Значения параметров те же, что и в процедуре *DrawPoly*. Действие тоже аналогично, но фон внутри многоугольника**

```
program g;
  uses crt, graph;
  const Star:array[1..18] of integer = (75, 0, 100, 50, 150, 75, 100, 100, 75, 150, 50, 100, 0,
75, 50, 50, 75, 0);
  var gd, gm: integer;
begin
  gd:=detect;
  initgraph(gd, gm,'c:/bp');
  SetFillStyle(1,2);
  FillPoly(9,Star); {9 – количество пересечений + 1}
  CloseGraph;
end.
```

Создание иллюзии движения

- Создать видимость движения изображения на экране можно несколькими способами. Рассмотрим два из них.I способ. Имитация движения объекта на экране за счет многократного выполнения программой набора действий: нарисовать – пауза – стереть (нарисовать в том же месте цветом фона) – изменить координаты положения рисунка. Перед началом составления программы надо продумать описание «двигающегося» объекта, характер изменения координат, определяющих текущее положение объекта, диапазон изменения и шаг.

- **II способ. Иллюзия движения создается при помощи специальных процедур и функций.**
  Функция ImageSize (x1,y1,x2,y2:integer):word возвращает размер памяти в байтах, необходимый для размещения прямоугольного фрагмента изображения, где x1,y1 – координаты левого верхнего и x2,y2 – правого нижнего углов фрагмента изображения.
  Процедура GetImage (x1,y1,x2,y2:integer,var Buf) помещает в память копию прямоугольного фрагмента изображения, где x1,..,y2 – координаты углов фрагмента изображения, Buf - специальная переменная, куда будет помещена копия видеопамяти с фрагментом изображения. Buf должна быть не меньше значения, возвращаемого функцией ImageSize с теми же координатами.

Константа	Значение	Операция	Пояснения
NormalPut	0	Замена существующего на копию	Стирает часть экрана и на это место помещает копию
XorPut	1	Исключительное или	Рисует сохраненный образ или стирает ранее нарисованный, сохраняя фон
OrPut	2	Объединительное или	Накладывает сохраненный образ на существующий
AndPut	3	Логическое и	Объединяет сохраненный образ и уже существующий на экране
NotPut	4	Инверсия изображения	То же самое, что и 0, только копия выводится в инверсном виде

Работа с текстом

- **Процедура *OutText(Textst:string)* выводит строку текста, начиная с текущего положения указателя. Например, *OutText(‘нажмите любую клавишу’);* Недостаток этой процедуры – нельзя указать произвольную точку начала вывода.**
  В этом случае удобнее пользоваться процедурой *OutTextXY (x,y:integer,Textst:string)*, где x,y – координаты точки начала вывода текста, Textst – константа или переменная типа String.
  Например,*OutTextXY(60, 100, ‘Нажмите любую клавишу’)*

Вывод численных значений

- **В модуле Graph нет процедур, предназначенных для вывода численных данных. Поэтому для вывода чисел сначала нужно преобразовать их в строку с помощью процедуры *Str*, а затем подключить посредством ‘+’ к выводимой строке.**
  Например: Max:=34.56;
  Str(Max: 6 : 2, Smax); {результат преобразования находится в Smax}
  OutTextXY(400, 40, ‘Максимум=’ + Smax);

- **Для удобства преобразование целочисленных и вещественных типов данных в строку лучше осуществлять специализированными пользовательскими функциями *IntSt* и *RealSt*:**

```
function IntSt(Int: integer) : string;
var Buf : string[10];
begin
Str(Int, Buf);
IntSt := Buf;
end;
```

```
function RealSt(R : real, Dig, Dec : integer) : string;
var Buf: string[20];
begin
Str(R : Dig : Dec, Buf);
RealSt := Buf;
end;
```

Шрифты

- **Вывод текста в графическом режиме может осуществляться различными стандартными (таблица 5) и пользовательскими шрифтами. Различают два типа шрифтов: растровые и векторные. Растровый шрифт задается матрицей точек, а векторный – рядом векторов, составляющих символ.**
  По умолчанию после инициализации графического режима устанавливается растровый шрифт*DefaultFont*, который, как правило, является шрифтом, используемым драйвером клавиатуры.

Стандартные шрифты

Шрифт	Файл
TriplexFont	Trip.chr
SmallFont	Litt.chr
SansSerifFont	Sans.chr
GothicFont	Goth.chr

Выравнивание текста

- **В некоторых случаях требуется в пределах одной строки выводить символы выше или ниже друг друга. Выравнивание текста выполняется с помощью процедуры *SetTextJustify(Horiz,Vert:word)* как по вертикали, так и по горизонтали посредством задания параметров *Horiz* и *Vert.***

Параметр	Значение	Комментарий
Горизонтальное выравнивание		
LeftText	0	Выровнять влево
CenterText	1	Центрировать
RightText	2	Выровнять вправо
Вертикальное выравнивание		
BottomText	0	Переместить вниз
CenterText	1	Центрировать
TopText	2	Переместить вверх