Школа им. А.М.Горчакова
Современная версия Царскосельского лицея

А.С.Цветков

Язык программирования PASCAL

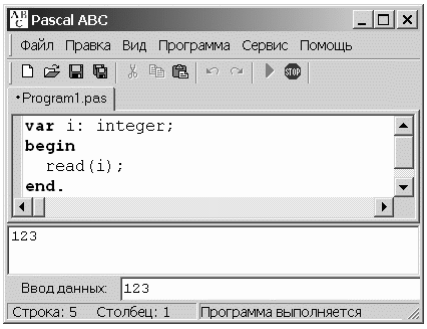
Система программирования ABC Pascal

Учебное пособие для школьников 7-9 классов

Санкт-Петербург
Павловск

2012-2013

Справочник по системе ABC Pascal



Редактор

Горячие клавиши

- **F2**, **Ctrl-S** - сохранить файл
- **F3**, **Ctrl-O** - загрузить файл
- **F12** - сохранить файл под новым именем
- **Ctrl-Shift-S** - сохранить все открытые файлы
- **Ctrl-Shift-0** ... **Ctrl-Shift-9** - установить закладку с номером 0...9
- **Ctrl-0** ... **Ctrl-9** - перейти к закладке с номером 0...9
- **Ctrl-Tab**, **Ctrl-Shift-Tab** - перейти к следующему/предыдущему окну редактора

Окно вывода

- Под окном редактора расположено окно вывода. Оно предназначено для вывода данных процедурами `write` и `writeln`, а также для вывода сообщений об ошибках и предупреждений во время работы программы.
- Окно вывода может быть скрыто. Клавиша **F5** и кнопка показывают/скрывают окно вывода. Для скрытия окна вывода используется также клавиша **Esc**.
- Окно вывода обязательно открывается при любом выводе в него.
- Для очистки окна вывода следует нажать комбинацию клавиш **Ctrl-Del** или кнопку .

Окно ввода

- Окно ввода открывается при выполнении процедур `read` и `readln` в ходе работы программы:
- Ввод данных в окно ввода сопровождается эхо-выводом в окно вывода (см. рис). После нажатия клавиши Enter данные из окна ввода попадают в соответствующие переменные, окно ввода закрывается, и программа продолжает работать дальше.

Запуск и остановка программы

- Для запуска программы в текущем окне редактора следует нажать клавишу **F9** или кнопку панели инструментов.
- Программа вначале компилируется во внутреннее представление, после чего, если не найдены ошибки, программа начинает выполняться. При выполнении программы кнопка запуска программы становится неактивной, кнопка останова программы, наоборот, активной и в строке статуса отображается информация "Программа выполняется".
- Выполнение программы можно в любой момент прервать нажатием комбинации клавиш **Ctrl-F2** или кнопки . При этом в окне вывода появится сообщение **Программа прервана пользователем.**
- Если текущая программа не является модулем, то при ее запуске на вкладке перед именем появляется жирная точка, что свидетельствует о том, что данную программу запускали последней. Если текущая программа является модулем, то компилируется не она, а программа, чье имя на вкладке помечено точкой.
- При выводе в графическое окно модуля GraphABC программу можно прервать нажатием клавиши **Esc**, при этом графическое окно будет закрыто.

Пошаговое выполнение программы

- Режим пошагового выполнения предназначен для отладки программы. Для выполнения одного шага (одной строки) программы следует нажать клавишу **F8** или кнопку (шаг без входа в подпрограмму), либо клавишу **F7** или кнопку (шаг со входом в подпрограмму). Для выполнения программы до данной строки следует установить на нее курсор и нажать клавишу **F4** или кнопку .
- Прервать программу, находящуюся в режиме пошагового выполнения, можно с помощью комбинации клавиш **Ctrl-F2** или кнопки . Если программа находится в режиме пошагового выполнения, то ее можно выполнить до конца, нажав **F9**.

Окно отладки

- Окно отладки позволяет просматривать во время пошагового исполнения программы значения переменных. По умолчанию оно располагается в правом верхнем углу окна редактора и имеет следующий вид:
- Для добавления переменной или выражения в окно отладки следует нажать комбинацию клавиш **Ctrl-F5** или кнопку . Можно также перетащить из редактора в окно отладки выделенное выражение или при активном окне отладки нажать клавишу **Ins**. Допускаются выражения, содержащие функции, в т.ч. и определенные в программе.
- Окно отладки включается/выключается нажатием комбинации клавиш **Ctrl-Shift-W**.
- Для удаления строки из окна отладки следует выделить эту строку и нажать клавишу **Del** или воспользоваться командой контекстного меню окна отладки.
- Для изменения строки в окне отладки следует дважды щелкнуть на ней.
- Для очистки окна отладки следует нажать комбинацию клавиш **Ctrl-Del** или воспользоваться командой контекстного меню окна отладки.
- Если выражение неверно, его вычисление вызывает ошибку или в данном контексте недоступны некоторые переменные, то при выполнении программы в столбце "Значение" появляется сообщение "нельзя вычислить".

Структура программы

Программа на языке Pascal ABC имеет следующий вид:

```
program имя программы;
раздел подключения модулей
раздел описаний
begin
  операторы
end.
```

- Первая строка называется заголовком программы и не является обязательной.
- Раздел подключения модулей начинается со служебного слова uses, за которым следует список имен модулей, перечисляемых через запятую.
- Раздел описаний может включать разделы описания переменных, констант, типов, процедур и функций, которые следуют друг за другом в произвольном порядке.
- Раздел подключения модулей и раздел описаний могут отсутствовать.
- Операторы отделяются один от другого символом "точка с запятой".

Идентификаторы и служебные слова

Идентификаторы служат в качестве имен программ, модулей, процедур, функций, типов, переменных и констант. Идентификатором считается любая последовательность латинских букв или цифр, начинающаяся с буквы. Буквой считается также символ подчеркивания "_".
Например, `a1`, `_h`, `b123` - идентификаторы, а `1a`, `ф2` – нет.

Служебные слова служат для оформления конструкций языка и не могут быть использованы в качестве имен. Список всех служебных слов языка Pascal ABC приведен ниже:

```
and array as begin break case class const constructor continue destructor div
do downto else end exit external external sync file finalization for forward
function if in inherited initialization is mod not of or private procedure
program property protected public  record repeat set shl  shr sizeof string
then to  type unit until uses var while with xor
```

Занятие №1

Целочисленные вычисления на языке Pascal

```
Program Prog1;
var a, b : integer;
begin
 write('Введите число ');
 readln(a);
 b:=a*a;
 writeln('Квадрат этого числа = ',b);
end.
```

Задание:

- Набрать и выполнить программу; понять, как она работает.
- Модифицировать программу так, чтобы она:
  - вычисляла куб числа
  - вводила не одно, а два числа, и вычисляла сумму их квадратов

Термины:

var – начало *секции описания переменных*
a, b, Prog1 – *идентификаторы* (имена) различных объектов
a, b – *переменные* целочисленного (integer) типа
integer – *целочисленный тип*
write, writeln – *операторы вывода*
readln – *оператор ввода*
b:=a+1 – *оператор присваивания* (переменной b присваивается значение выражения a+1)
begin ... end – *операторные скобки*

Переменная – это величина, которая может менять свое значение. Переменная всегда должна быть явно описана как принадлежащая какому-либо типу данных.

Тип данных – множество значений, которые может принимать объект (чаще всего это переменная) данного типа. Кроме множества значений тип данных задает множество допустимых операций. Например, целочисленный тип *integer* определяет диапазон целых чисел от –2 147 483 648 до +2 147 483 647, в целочисленных выражениях можно применять операции сложения, вычитания и т.п.

Непосредственные константы – это числа, используемые в арифметических выражениях. Например, в операторе `b:=a+1` единица – это непосредственная константа.

Правила записи целых чисел. Целые числа записываются так же, как и в математике. Цифрам может предшествовать знак "–" или "+" (последнее обычно не нужно). Пробелы между знаком и цифрами, а также между цифрами недопустимы. Примеры правильных записей чисел: 1, 123, –4567, 003, +012.

Оператор – инструкция языка Паскаль. Операторы отделяются точкой с запятой друг от друга. Операторы можно объединить в *составной оператор*, заключив их в операторные скобки **`begin ... end`**.

Оператор присваивания. Оператор присваивания обозначается двумя символами `:=`, между которыми нет пробела. Слева от знака оператора должна стоять переменная, а справа – *выражение*. Суть оператора заключается в вычислении выражения и присвоения получившегося значения переменной.

Примеры: `a:=2; a:=2+3; a:=b*2+1;`
Неправильные примеры: `a+1:=a; 2:=b+3;`

Очень часто используется оператор вида `a:=a+1;` Он не содержит ошибки. Вначале берется старое значение переменной, к нему прибавляется единица, и результат записывается в опять в переменную a. Таким образом, этот оператор увеличивает значение переменной a на единицу.

Важно следить, чтобы все переменные, используемые в правой части оператора присваивания были определены к моменту вычисления оператора. В противном случае результат будет непредсказуем.

Существует два способа задать переменной значение: оператор присваивания и оператор ввода (read или readln).

Согласование типов в операторе присваивания. Общее правило таково: тип выражения в правой части оператора присваивания должен совпадать с типом переменной в левой части оператора присваивания. Т.е. если переменная имеет тип integer, то справа должно быть выражение, значение которого есть также целое число.

Арифметическое выражение – целочисленное арифметическое выражение состоит из переменных, констант, знаков операций, скобок и вызовов функций. Правила построения выражения очень похожи те, которые употребляются в математике, за исключением деления, для которого существуют две операции, обозначаемые ключевыми словами div и mod. Надо запомнить, что знак операции умножения * никогда не опускается в отличие от алгебры. Операции имеют обычный приоритет: умножение и деление выполняется раньше, чем сложение и вычитание. Вычисление функций (см. далее) выполняется еще раньше. В остальном операции выполняются слева направо. Для изменения порядка операций используются круглые скобки. Уровень вложенности скобок не ограничен.

Бинарные[1] *арифметические операции над типом integer*

Операция	Обозначение	Пример
сложение	+	a+3
вычитание	–	c–d
умножение	*	2*3
деление нацело	div	a div 2
остаток от деления	mod	a mod 2

Деление на цело: 5 div 2 = 2, остаток от деления нацело: 5 mod 2 = 1

[1] Бинарные операции – имеющие два операнда, располагающиеся слева и справа от оператора

Пример:

Выражение в математической записи:	$(a+2b)(a-b)$	$\frac{x+y}{2}$
Выражение на Паскале (в целых числах):	`(a+2*b)*(a-b)`	`(x+y) div 2`

Кроме бинарных операций в Паскале используются унарные[2] операции – и + (в последнем особого смысла нет). Например, в выражении –2*(–a–b) минус перед переменной a является унарной операцией, а минус перед двойкой может рассматриваться и как унарная операция и как запись отрицательной константы.

Целочисленные типы языка ABC Pascal

Тип	**Размер в байтах**	**Диапазон**
integer	4	-2147483648 … 2147483647
byte	1	0 .. 255
word	2	0 .. 65 535

При выполнении оператора присваивания следует следить за тем, чтобы значение целочисленного выражения не вышло за допустимый диапазон целого числа. Например, если переменная `a` имеет тип `word`, то в операторе `a:=512*128;` произойдет ошибка, так как значение выражения является 65536, а это число больше верхней границы типа word. Отметим, что если бы переменная `a` описана как `integer`, то оператор `a:=512*128;` является допустимым. Без особой необходимости не следует использовать типы `byte` и `word`.

Консольный ввод/вывод – исторически самый «древний» способ диалога с компьютером. У первых ЭВМ консоль – это электрическая пишущая машинка, подключенная к компьютеру, печатающая на рулонной бумаге (как факс). Оператор мог вводить информацию в компьютер, завершая команду нажатием клавиши перевода строки (Enter). Компьютер отвечал, заставляя консоль печатать цифры и символы. Несмотря на простоту такого интерфейса[3], он весьма эффективен и до сих пор используется в самых современных операционных системах. В Windows-системах консоль обычно представляет текстовое окно шириной 80 символов, а высотой 25 символов[4]. При достижении ввода или вывода последней строки все строки скроллируются на одну позицию вверх, а первая строка исчезает.

Операторы вывода – `write` и `writeln` выводят заданную информацию на консоль. Оператор `writeln` после этого еще и переводит курсор на следующую строку (при достижении последней строки осуществляется скроллинг). Операторы могут иметь произвольное число аргументов, разделенных запятыми. В качестве аргументов могут выступать буквальные константы (например, строки символов) и переменные стандартных типов (в т.ч. и целочисленные).

Пример: `write('Текстовая строка'); writeln(a,b);`

[2] Унарная операция – имеющая только один операнд.
[3] Интерфейс – в данном случае способ общения человека с компьютером.
[4] В системе ABC Pascal отдельное окно консоли появляется только при использовании модуля CRT.

Форматы вывода. В операторах `write` и `writeln` можно указывать формат, определяющий число позиций, используемых для вывода целого числа на экран, например:

```
writeln('Ответ ',a:5);
```

Это означает, что для вывода значения, хранящегося в переменной a следует зарезервировать 5 символов, т.е. если число будет не пятизначным, то при выводе оно будет дополнено слева пробелами. Если число «не помещается» в формат, он будет автоматически «растянут», так чтобы число «влезло».

Оператор ввода – `read`, `readln`. Оператор `read` при чтении данных с консоли используется крайне редко, в основном используется оператор `readln`. Параметры оператора – только переменные и только стандартных типов, возможно использование оператора вообще без аргументов для создания ситуации ожидания нажатия клавиши *Enter*. По оператору `readln` программа приостанавливает свою работу, ожидая ввода данных от пользователя. Пользователь набирает желаемые значения переменных, разделяя их пробелами (или *Enter*) и нажимает *Enter*. Оператор интерпретирует введенные символы, переводя их во внутреннее (машинное) представление соответствующих переменных. Если это не удается (например, вместо целого числа введены буквы), программа аварийно завершает свою работу.

Пример: `readln(a,b);`
Недопустимо: `readln(a+3);`

Для того чтобы пользователь знал что «хочет» ввести программа, полезно перед оператором readln выводить подсказку с помощью оператора write или writeln.

Например:

```
write('Введите количество учеников в классе ');
readln(n);
```

Обратите внимание на пробел перед вторым апострофом. Он сделан для того, чтобы ввод пользователя отделялся от подсказки.

Встроенные функции – в языке Pascal предусмотрено большое число различных встроенных функций. Функции можно применять в выражениях, аргумент(ы) функции всегда заключаются в скобки, например: `x:=x-abs(x)`. Функции требуют аргументы определенных типов (например, целого типа) и возвращают значение также определенного типа (может не совпадать с типом аргумента). Ниже приведен список арифметических функций, имеющих целочисленный аргумент .

Функция	Описание
sqr(n)	Возвращает квадрат аргумента. Необходимо помнить о диапазоне возвращаемого значения.
abs(n)	Возвращает модуль (абсолютное значение) числа.
pred(n)	Возвращает значение, на единицу меньшее аргумента.
succ(n)	Возвращает значение, на единицу большее аргумента.
odd(n)	Возвращает TRUE, если аргумент нечетный, иначе – FALSE
even(n)	Возвращает TRUE, если аргумент четный, иначе – FALSE
chr(n)	Возвращает символ (тип char) с кодом n.
random(n)	Возвращает случайное число в диапазоне от 0 до n-1.

Задание №1

1. Записать следующие выражения на языке Pascal, считая все переменные и действия целочисленными.

(a+b)(a–b)	
$(1+x)^2$	
$\frac{15x}{y}$	
2x·2y	

2. Вычислить (в уме) значение выражения

(a+1)*(–a), если a=10	
(x+1) div (x–1), если x=2 и если x=4	
2*4 mod 3	
2*(4 mod 3)	
–a*(–1), если a=431	

3. Напишите и выполните программу на компьютере

- Программа вводит одно число, и вычисляет его квадрат, вычитая из него удвоенное значение введенного числа.
- Программа вводит два числа, выводит сумму квадратов этих чисел минус их произведение.
- Программа вводит два числа, выводит сумму их модулей (*см. функцию abs*)

4. Напишите программу, которая решает следующую задачу

Оплата Интернета в школе состоит из двух частей: 100 долларов в месяц за доступ и 5 долларов в месяц за поддержку школьного сайта. Платеж можно делать независимо за любое количество месяцев за доступ и за поддержку сайта. Напишите программу, которая позволяла бы вводить отдельно количество оплачиваемых месяцев для доступа и поддержки и выводила бы сумму оплаты.

Занятие №2

Использование модуля CRT

Язык Паскаль имеет специальную библиотеку работы с экраном в текстовом режиме. Она называется CRT[5]. Эта библиотека содержит набор подпрограмм, позволяющих задавать цвет выводимых букв, цвет фона, устанавливать курсор в желаемую позицию. В текстовом режиме считается, что окно содержит 25 строк по 80 колонок. В каждой позиции может быть один символ (буква, цифра, специальные знаки). Графические объекты (прямые, окружности и т.п.) в текстовом окне не допустимы. Модуль CRT эмулирует текстовый терминал первых персональных компьютеров.

Подключение библиотеки к программе осуществляется предложением Uses:

```
Program Card;
Uses CRT; { Это предложение вставляется сразу после Program }
```

Далее могут идти описания переменных, затем begin и тело программы.

Рассмотрим использование подпрограмм этой библиотеки на примере создания заставки программы.

```
Program Card;
Uses CRT;  { Подключить модуль CRT }
begin
 ClrScr;                { Очистить экран }
 TextColor(White);      { Установить белый цвет букв }
 TextBackGround(Blue); { Установить синий цвет фона }
 GotoXY(36,13);      { Поставить курсор в 36 колонку, 13 строку }
 write(' Привет '); { Вывести текст }
 ReadKey;            { Ожидать нажатия любой клавиши }
end.
```

Наберите текст этой программы и выполните её.

Разберем действия отдельных процедур:

- **ClrScr** – очищает экран или текущее окно (от англ. Clear Screen), закрашивая его текущим цветом фона, установленным процедурой TextBackGround. По умолчанию – цвет экрана белый.
- **TextColor**(*цвет*) – устанавливает текущий цвет букв, который будет исполь–зоваться операторами write и writeln. По умолчанию используется светло-серый цвет букв.
- **TextBackgroud**(*цвет*) – устанавливает текущий цвет фона, который будет использоваться операторами write и writeln.
- **GotoXY**(*колонка, строка*) – переводит курсор в указанную позицию; вывод, осуществляемый последующим оператором write или writeln будет начинаться с этой позиции. Обычно экран в текстовом режиме содержит **80 колонок** и **25 строк**.
- **ReadKey** – ожидание нажатия на любую клавишу (далее мы рассмотрим другое применение этой подпрограммы).

[5] От английского Cathode Ray Tube – Электронно-лучевая трубка (ЭЛЧ)

В качестве цвета может использоваться число от 0 до 15, но лучше пользоваться определенными в модуле CRT константами:

Таблица цветов

Black	Черный	DarkGray	Темно-серый
Blue	Синий	LightBlue	Светло-синий
Green	Зеленый	LightGreen	Светло-зеленый
Cyan	Небесно голубой	LightCyan	Ярко-голубой
Red	Красный	LightRed	Светло-красный
Magenta	Малиновый	LightMagenta	Светло-малиновый
Brown	Коричневый	Yellow	Желтый
LightGray	Светло-серый	White	Белый

Рассмотрим еще несколько полезных подпрограмм.

- **Window**(*x1,y1,x2,y2*) – создание окна вывода. В случае задания окна, все операторы write и writeln выводят только в него, процедура ClrScr будет очищать только это окно.
- **WhereX**, **WhereY** – две функции, позволяющие узнать координаты местонахождения курсора. *Пример использования*: `GotoXY(WhereX+10,WhereY+2)`.

Задание №2:

Напишите программу «Визитная карточка», которая выводила бы на экран хорошо оформленные ваши личные данные: имя, фамилию, год рождения, адрес, телефон. Используйте разные цвета, хорошо скомпонуйте.

Задание №2*:

Модуль `Sounds` содержит процедуры и функции для работы со звуком:

n:=LoadSound(fname) – загружает звук из файла с именем `fname` в оперативную память и возвращает описатель звука в целую переменную `n` (звуковой файл должен может иметь любой формат);
PlaySound(n) – начинает проигрывание звука с описателем `n`;
StopSound(n) – останавливает проигрывание звука с описателем `n`;
RewindSound(n) – "перематывает" звук с описателем `n` на начало;
DestroySound(n) – удаляет звук с описателем `n` из оперативной памяти, описатель звука при этом становится недействительным;
SoundLength(n) – возвращает длительность звука с описателем `n` в миллисекундах.

Используйте его возможности для «улучшения» программы «Визитная карточка»
Пример использования модуля sound:

```
Program Player;
Uses Sounds, CRT;
var n: integer;
begin
  n:=LoadSound('C:\WINDOWS\MEDIA\ringin.wav'); // укажите расположение файла
  PlaySound(n);
  Sleep(SoundLength(n));
  DestroySound(n);
end.
```

Занятие №3

Графический модуль GraphABC

Система ABC Pascal обладает великолепной графической библиотекой. Для ее модключения после заголовка программы необходимо написать Uses GraphABC. Модуль GraphABC содержит константы, типы, процедуры и функции для рисования в графическом окне. Они подразделяются на несколько групп:

- Графические примитивы
- Действия с цветом
- Действия с точками и прямоугольниками
- Действия с пером
- Действия с кистью
- Действия со шрифтом
- Действия с рисунками
- Действия с графическим окном

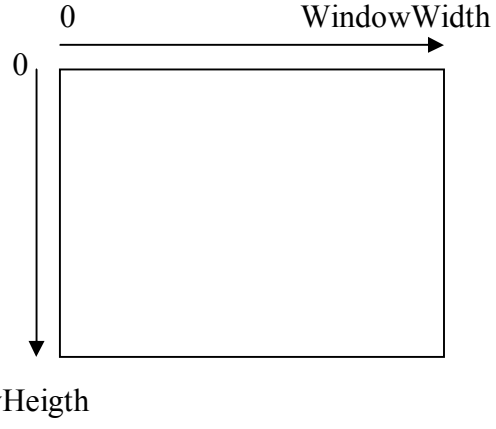
Рассмотрим сразу пример:

```
Program Gr01;
Uses GraphABC;
begin
 SetWindowSize(256,256);
 SetPenColor(RGB(0,0,255));
 Line(0,0, WindowWidth, WindowHeight)
 Line(0,WindowHeight, WindowWidth, 0)
end.
```

Процедура SetWindowSize устанавливает размер графического окна 256×256 пикселей. Процедура SetPenColor определяет цвет рисования линий. Функция RGB возвращает цвет, заданный своими *Red*, *Green*, *Blue* компонентами. Процедура Line рисует линию из точки, заданной первой парой координат, в точку, заданную второй парой координат. Функции WindowWidth и WindowHeight возвращают текущие значения ширины и высоты графического окна (следовательно, если вы поменяете размер окна в процедуре SetWindowSize, то линии все равно будут проводиться из угла в угол).

Оконные координаты

- Координата x растет слева на право от 0 до значения WindowWidth
- Координата y растет сверху вниз от 0 до значения WindowHeight



Задание цвета

Как известно восприятие цвета у человека трехцветное. Основываясь на этом, в графических библиотеках используется именно такое трехкомпонентное представление цвета. Цвет задается с помощью функции

rgb(*Red,Green,Blue*)

Здесь

- *Red* – число от 0 до 255, соответствующее интенсивности красной компоненты
- *Green* – число от 0 до 255, соответствующее интенсивности зеленой компоненты
- *Blue* – число от 0 до 255, соответствующее интенсивности синей компоненты

Функция формирует целое число (integer), которое может использоваться везде в графической библиотеке, где требуется указание цвета. Есть несколько предопределенных цветов:

clBlack – черный
clPurple – фиолетовый
clWhite – белый
clMaroon – темно-красный
clRed – красный
clNavy – темно-синий
clGreen – зеленый
clBrown – коричневый
clBlue – синий
clSkyBlue – голубой
clYellow – желтый
clCream – кремовый
clAqua – бирюзовый
clOlive – оливковый
clFuchsia – сиреневый
clTeal – сине-зеленый
clGray – серый
clLime – ярко-зеленый
clLightGray – светло-серый
clMoneyGreen – цвет зеленых денег
clDarkGray – темно-серый

Задание стиля и цвета пера

- SetPenColor(цвет) – устанавливает цвет пера, задаваемый параметром color.
- SetPenWidth(*ширина*) – устанавливает ширину пера.
- SetPenStyle(*стиль*) – устанавливает стиль пера (сплошной, пунктир и т.п.), возможные значения указаны в таблице (стиль применим только к ширине пера 1 пиксел)

psSolid	psDash	psDashDot
psClear	psDot	psDashDotDot

Текущее перо используется для рисования линий, прямоугольников, ломаных, окружностей, эллипсов, дуг и т.п.

Пример:

```
Program Gr02;
Uses GraphABC;
Var i : integer;
begin
 SetWindowSize(512,512);
 SetPenStyle(psDash);
 SetBrushStyle(bsClear);
 Circle(256,256,WindowHeight div 2);
end.
```

Задание стиля и цвета кисти

- SetBrushColor(*цвет*) – устанавливает цвет кисти
- SetBrushPicture(*имя файла*) – устанавливает в качестве образца для закраски кистью образец, хранящийся в файле, при этом текущий цвет кисти при закраске игнорируется.
- ClearBrushPicture – очищает рисунок-образец, выбранный для кисти.
- SetBrushStyle(*стиль*) – устанавливает стиль кисти, задаваемый параметром bs.

Возможные стили кисти:
bsSolid bsClear bsCross bsDiagCross bsHorizontal bsBDiagonal bsVertical bsFDiagonal

Текущей кистью закрашиваются все замкнутые фигуры, контур фигур рисуется текущим пером. Чтобы нарисовать не закрашенную фигуру, используйте SetBrushColor(bsClear).

Пример:

```
Program Gr03;
Uses GraphABC;
Var i,j : integer;
begin
 SetWindowSize(512,512);
 SetBrushColor(rgb(128,0,255));
 SetBrushStyle(bsDiagCross);
 Rectangle(10,10,WindowWidth-10,WindowHeight-10);
end.
```

Некоторые графические примитивы[6]

- Line(x1,y1,x2,y2) – рисует отрезок с началом в точке (x1,y1) и концом в точке (x2,y2).
- Circle(x,y,r) – рисует окружность с центром в точке (x,y) и радиусом r.
- Ellipse(x1,y1,x2,y2) – рисует эллипс, заданный своим описанным прямоугольником с координатами противоположных вершин (x1,y1) и (x2,y2).
- Rectangle(x1,y1,x2,y2) – рисует прямоугольник, заданный координатами противоположных вершин (x1,y1) и (x2,y2).
- RoundRect(x1,y1,x2,y2,w,h) – рисует прямоугольник со скругленными краями; (x1,y1) и (x2,y2) задают пару противоположных вершин, а w и h – ширину и высоту эллипса, используемого для скругления краев.
- Arc(x,y,r,a1,a2) – рисует дугу окружности с центром в точке (x,y) и радиусом r, заключенной между двумя лучами, образующими углы a1 и a2 с осью OX (a1 и a2 – вещественные, задаются в градусах и отсчитываются против часовой стрелки).
- Pie(x,y,r,a1,a2) – рисует сектор окружности, ограниченный дугой (параметры процедуры имеют тот же смысл, что и в процедуре Arc).
- Chord(x,y,r,a1,a2) – рисует фигуру, ограниченную дугой окружности и отрезком, соединяющим ее концы (параметры процедуры имеют тот же смысл, что и в процедуре Arc).
- FloodFill(x,y,color) – закрашивает область одного цвета, начиная с точки (x,y) цветом color.

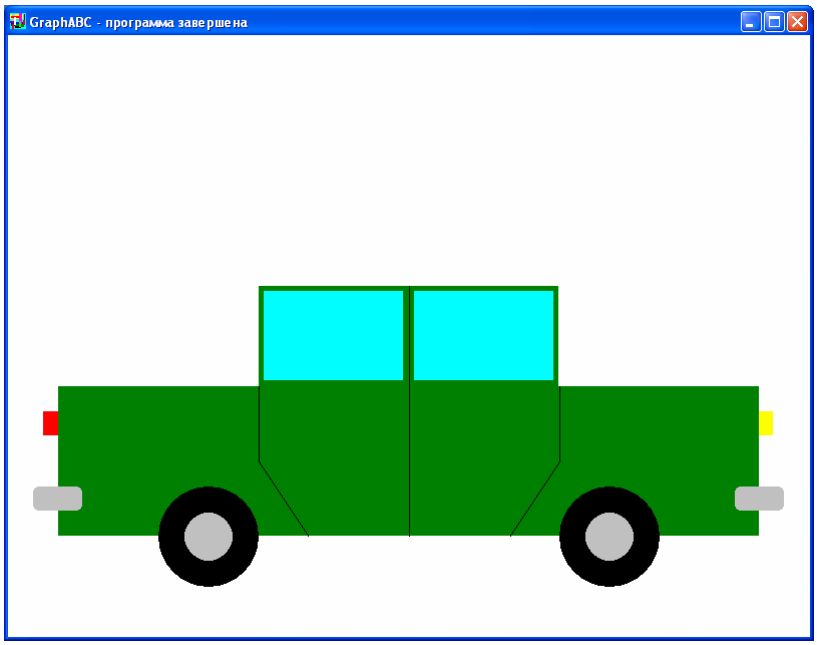
Задание:

Напишите программу, использующую не менее трех процедур рисования, которая рисовала бы несложный рисунок (домик, автомобиль, и т.п.)

[6] Дополнительную информацию можно получить в справочной системе ABC Pascal, раздел *Стандартные модули – Модуль GraphABC*.

Пример решения задания

```
program avto;
uses graphABC;
var i : integer;
begin
  SetWindowSize (800,600);

  SetPenStyle (psClear); // верхняя часть автомобиля
  setBrushColor (Clgreen);
  rectangle (250,250,550,351);

  setBrushColor (ClAqua);   // окна
  rectangle (255,255,395,345);
  rectangle (405,255,545,345);

  setBrushColor (Clgreen); // корпус
  rectangle (50,350,750,500);

  setBrushColor (ClYellow); // фары
  rectangle (749,375,764,400);
  setBrushColor (ClRed);
  rectangle (35,375,51,400);

  setBrushColor (ClLightGray); // бампер
  RoundRect (725,450,775,475,10,10);
  RoundRect (25,450,75,475,10,10);

  SetPenStyle (psSolid); // двери
  Line(250,350,250,425);
  Line(250,425,300,500);
  Line (400,250,400,500);
  Line (550,350,550,425);
  Line (550,425,500,500);

  setBrushColor (ClBlack);  //  колеса
  Circle (200,500,50);
  setBrushColor (ClLightGray);
  Circle (200,500,25);

  setBrushColor (ClBlack);
  Circle (600,500,50);
  setBrushColor (ClLightGray);
  Circle (600,500,25);
end.
```



Занятие №4

Цикл for

Цикл for используется для повторения фрагментов программы заданное число раз.

Синтаксис оператора:

```
for i:=N1 to N2 do оператор;
```

или

```
for i:=N1 to N2 do
begin
  оператор1;
  оператор2;
  ....
end;
```

Составной оператор – это группа операторов, заключенная в операторные скобки **begin** ... **end**.

Здесь
i – индекс цикла (обязательно переменная целочисленного типа),
N1 – целочисленное выражение, определяющее нижнюю границу индекса цикла,
N2 – целочисленное выражение, определяющее верхнюю границу индекса цикла, N2 должно быть больше или равно N1, для того чтобы цикл выполнился хоть один раз.

Алгоритм выполнения оператора такой:

1. Индексу цикла присваивается значение выражения N1.
2. Проверяется условие i<=N2. Если условие истинно, то переход к п. 3, иначе к п.6.
3. Выполняются операторы тела цикла.
4. Значение индекса цикла увеличивается на единицу (автоматически!).
5. Переход к п. 2.
6. Конец цикла.

Пример:

Построить таблицу квадратов чисел от 1 до 10.

```
Program Square;
var i, j : integer;
begin
 for i:=1 to 10 do
 begin
  j:=sqr(i);
  writeln(i:4,j:4);
 end;
end.
```

или проще:

```
Program Square;
var i: integer;
begin
 for i:=1 to 10 do
    writeln(i:4,sqr(i):4);
end.
```

Обратите внимание на указание формата вывода целого числа в операторе writeln.Число после двоеточия указывает на количество символов, отводимых для представления целого числа. Если число занимает меньшее количество цифр, то при выводе оно слева дополняется необходимым числом пробелов.

Цикл for … downto

Иногда возникает желание выполнить цикл for наоборот, так чтобы индекс цикла не возрастал, а убывал. Эту возможность реализует следующая конструкция:

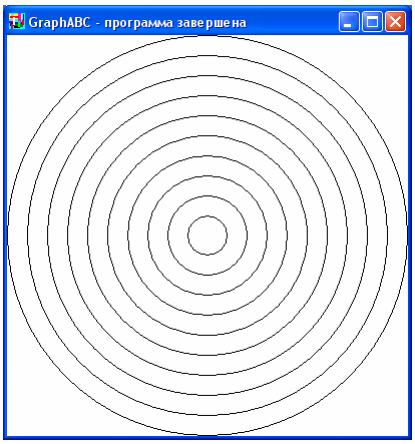
```
for i:=N2 to N1 do оператор;
```

Здесь подразумевается, что N2 должно быть больше или равно N1.

Использование цикла for для построения изображений

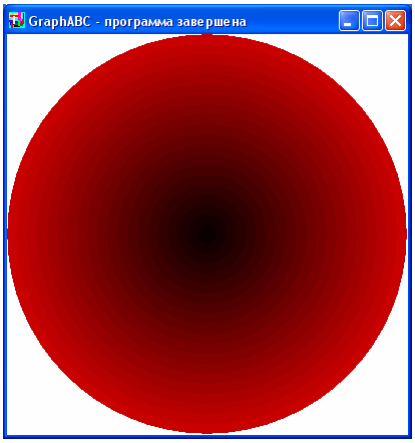
Использование цикла for весьма эффективно при построении изображений. Например, следующая программа строит набор концентрических окружностей.

```
Program Concentric;

uses graphABC;

var i : integer; // для цикла for
begin

  SetWindowSize (400,400);
  SetBrushStyle (bsClear);

  // строятся 10 окружностей с
  // радиусами 20, 40, ... 200
  for i:=1 to 10 do Circle(200,200,i*20);
end.
```



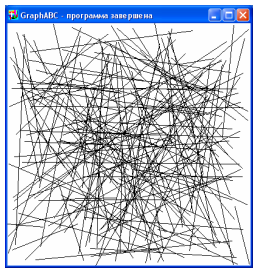
Вот более сложный пример, разберитесь с ним сами. Ответьте на вопрос: «Почему используется цикл не **for … to**, а **for … downto**?»

```
Program Concentric2;
uses graphABC;
var i : integer; // для цикла for
begin
  SetWindowSize (400,400);
  SetPenStyle(psClear);

  for i:=20 downto 1 do
  begin
   SetBrushColor (RGB(10*i,0,0));
   Circle(200,200,i*10);
  end;

end.
```



Интересных эффектов можно добиться, используя так называемый «генератор случайных чисел». Встроенная функция `random(N)` возвращает случайное число в диапазоне от 0 до *N*–1. Посмотрите, как эта возможность применяется в следующей программе.

```
Program RandomLines;
uses graphABC;
var i : integer;
begin
  SetWindowSize (400,400);
  for i:=1 to 200 do
    Line(Random(400),Random(400),Random(400),Random(400));
end.
```



Задание №5. Дополните вашу графическую программу узором, получаемым с помощью цикла for.

Занятие №5

Оператор if

В языке Pascal существуют операторы, позволяющие изменить линейный ход программы. Их можно разделить на две группы: *развилки* и *циклы*.

Развилки	Циклы
if ... then ... else ...	while
if ... then	repeat ... until
case	for

Развилки

Оператор if

Синтаксис[7] оператора:

if *логическое условие* **then** *оператор-1* **else** *оператор-2*;

Пример:

```
if x>0 then z:=1 else z:=-1;
```

Выполнение оператора начинается с проверки логического условия, если оно истинно, то выполняется *оператор-1*, иначе *оператор-2*. В качестве оператора может выступать простой оператор или составной оператор. **Составной оператор** – это группа операторов, заключенная в операторные скобки begin ... end.

Пример:

```
if x>0 then begin
              z:=1;
              x:=2*x;
            end
       else begin
              z:=-1;
              x:=-x;
            end;
```

Обратите внимание, что перед else точка с запятой не ставится. Точка с запятой используется только для разделения операторов друг от друга. Можно было бы и не ставить запятую после операторов `x:=2*x` и `x:=-x`, поскольку begin и end не являются операторами, а являются скобками. Паскаль будет интерпретировать[8] точку с запятой перед end, как наличие пустого оператора между ней и end.

Обратите внимание на запись “лесенкой”, она не обязательна, но весьма желательна, поскольку помогает наглядно увидеть алгоритмическую структуру программы. Правило такое: end пишем под begin.

[7] Синтаксис оператора – правила его записи.
[8] Интерпретировать – толковать.

Оператор if (сокращенная форма)

Часто встречается ситуация, когда ветвь else не нужна. В этом случае ее можно опустить. Например:

```
if x<0 then x:=-x;
```

Обратите внимание, что точка с запятой ставится после оператора `x:=-x` в отличие от полной формы.

Пример программы с оператором if

```
Program OddEven;
{ Определение четного или нечетного числа }
var n : integer;
begin
 write('Введите число '); readln(n);
 if odd(n) then writeln('Число нечетное')
           else writeln('Число четное');
end.
```

Вложенные операторы if

Иногда возникает необходимость устроить развилку на более чем две ветви. В этом случае операторы if можно вкладывать один в другой. С точки зрения алгоритмической корректности лучше, если вложенный оператор находится в ветви else. Приведем сразу пример программы

```
Program Sign;
{ Определение знака числа }
var n : integer;
begin
write('Введите число '); readln(n);
 if n>0 then writeln('Число положительное')
        else if n=0 then writeln('Число ноль')
                    else writeln('Число отрицательное')
end.
```

Обратите внимание, что после вложенный оператор if можно не заключать в операторные скобки begin ... end, поскольку оператор if синтаксически рассматривается как один оператор.

Если имеются вложенные операторы if, один из которых неполный, например:

```
if n>0 then if n=0 then writeln('У вас ноль')
                   else writeln('Число положительное')
```

возникает неоднозначность: к какому if относится else. Принято, что else относится к ближайшему if, т.е. таки образом, как отражает запись данного примера. Если следовать правилу, помещая, по возможности, вложенные операторы в ветвь else, то таких ситуаций не будет.

Логические выражения. В операторе if используются логические выражения, имеющие тип boolean[9]. Про такие выражения можно сказать только являются ли они истинными (true) или ложными (false). Простейший случай логического выражения – логическая переменная.

Пример:

```
var b : boolean;
. . . . . .
b:=true;
. . . . . .
if b then writeln('Все в порядке');
```

Такие логические переменные называют *флагами*, иногда их использование оказывается очень эффективным решением.

Операции сравнения. Существует шесть бинарных[10] операций сравнения, у которых операнды могут быть самых разных типов (целые, вещественные, символьные строковые).

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

Эти операции весьма часто используются для построения различных логических условий. Например: `if a>0 then a:=1 else a:=-1`. С точки зрения языка Pascal `a>0` представляет собой логическое выражение (можно написать даже так: `b:=a>0`, если `b` – переменная булевского типа).

Логические операции. Для составления сложных логических выражений могут использоваться две бинарные логические операции AND, OR и одна унарная[11] NOT.

`and` – результат операции истинен тогда и только тогда, когда истины оба операнда.
`or` – результат операции истинен тогда, когда истинен хотя бы один операнд.
`not` – имеет результат противоположный операнду.

Примеры операторов if со сложными условиями:

```
if (x>=0) and (x<=1) then writeln('X находится в диапазоне от 0 до 1');

if not ((x>=0) and (x<=1)) then writeln('X вне диапазона 0-1');

if (x<0) or (x>1) then writeln('X вне диапазона 0-1');
```

Обратите внимание на скобки. Операции сравнения имеют самый низший приоритет, поэтому заключены в скобки. Приоритет операции `and` соответствует операции умножения, `or` – сложению, `not` – унарному минусу (т.е. самый высокий).

[9] Назван в честь Дж. Буля – основателя математической логики.
[10] Бинарные операции – имеющие два операнда, на которые действует эта операция.
[11] Унарная операция – имеющая только один операнд.

Задание №5

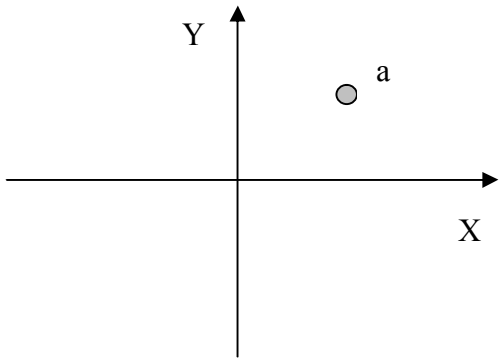
1. Вычислите логические выражения:

   not true and not false

   not true or not false

   not (true or false)

   not (true and false)



2. Напишите условие нахождения точки a в первом квадранте системы координат, считая ее координаты заданными переменными *x*, *y*.

3. Напишите программу, выводящую слова «зачет», если введено число 3, 4 или 5 и «незачет», если введено число 1 или 2.

4. Напишите программу, выводящую текст «плохо», «удовлетворительно», «хорошо» или «отлично», если введено число 2, 3, 4 или 5 соответственно.

5. Используя операцию определения остатка от деления (mod, стр. 5), модернизируйте программу Concentric (стр. 16), таким образом чтобы каждая 3-я окружность выводилась красным цветом, а остальные были синими.

Вложенные циклы

Довольно часто возникает ситуация, при которой в теле одного цикла встречается другой оператор цикла. Такие циклы называют вложенными циклами. Приведем простой пример. Выведем на экран таблицу умножения целых чисел от 1 до 10.

```
Program Table;
var i,j : integer;
begin
 for i:=1 to 10 do
 begin
  for j:=1 to 10 do write(i*j:4);
  writeln
 end
end.
```

```
  1   2   3   4   5   6   7   8   9  10
  2   4   6   8  10  12  14  16  18  20
  3   6   9  12  15  18  21  24  27  30
  4   8  12  16  20  24  28  32  36  40
  5  10  15  20  25  30  35  40  45  50
  6  12  18  24  30  36  42  48  54  60
  7  14  21  28  35  42  49  56  63  70
  8  16  24  32  40  48  56  64  72  80
  9  18  27  36  45  54  63  72  81  90
 10  20  30  40  50  60  70  80  90 100
```

Здесь внутри тела внешнего цикла по переменной i, находится вложенный цикл по переменной j, который выводит в операторе `write` (не `writeln`!) произведение i*j, выделяя 4 позиции для результата. Далее идет оператор `writeln`, который переводит курсор на новую строку. Внешний цикл повторяет 10 раз эту операцию, формируя, таким образом, 10 строк таблицы, которая приведена справа.

Оператор case

Оператор case используется для создания развилки на более чем 2 ветви. Понять его работу поможет следующий пример.

```
Program YourMark;
{ ваша отметка }
var n : integer;
begin
 write('Введите вашу отметку '); readln(n);
 case n of
  2 : writeln('Плохо');
  3 : writeln('Удовлетворительно');
  4 : writeln('Хорошо');
  5 : writeln('Отлично');
 end;
end.
```

Обратите внимание, что оператор case заканчивается скобкой end.

Если пользователь введет отметку, не попадающую в диапазон 2-5, то никаких действий выполняться не будет. Можно, конечно, обработать и эту ситуацию. Тогда синтаксис оператора case будет такой:

```
Program YourMark2;
{ ваша отметка }
var n : integer;
begin
 write('Введите вашу отметку '); readln(n);
 case n of
  2 : writeln('Плохо');
  3 : writeln('Удовлетворительно');
  4 : writeln('Хорошо');
  5 : writeln('Отлично')
  else writeln('Вы ввели недопустимую отметку!')
 end;
end.
```

Обратите внимание на отсутствие точки с запятой перед else и двоеточия после else.

Кроме единичных значений *переключателя* n можно указывать диапазон значений.

```
Program YourMark3;
{ ваша отметка }
var n : integer;
begin
 write('Введите вашу отметку '); readln(n);
 case n of
  2..3 : writeln('Это все очень плохо!');
  4    : writeln('Хорошо!');
  5..6 : writeln('Так держать!')
  else   writeln('Вы ввели недопустимую отметку!')
 end;
end.
```

Обратите внимание на отсутствие точки с запятой перед else и двоеточия после else.

Задание № 6.1. По образцу программ, написанных на предыдущей странице, решите следующую задачу: Программа вводит число в диапазоне от 1 до 10 и выводит на экран значение числа русскими буквами (один, два, … десять).

(*2 балла*)

Задание № 6.2. Модифицируйте программу таким образом, чтобы она выводила название чисел в диапазоне от 1 до 99 (ну кассовый аппарат же это делает на чеках!).

Подсказка №1. Для выделения числа десятков используйте хорошо знакомую вам операцию «остаток от деления». Т.е., если в переменной **n** находится введенное число, то получить число десятков очень просто – **d:=n mod 10**;

Подсказка №2. Надо рассмотреть по сути три диапазона чисел: от 1 до 9, от 10 до 19, и от 20 до 99.

(5 *баллов*)

Задание № 6.3. Ну, если уж сделали предыдущую задачу, то расширьте диапазон чисел до 999.

(*еще* 3 *балла*)

Задание № 6.4*. Модифицируйте программу 6.3 (даже в сторону упрощения), так чтобы она выводила число в диапазоне от 1 до 999, записанное *римскими цифрами*.

(*еще* 4 *балла*)

Запись чисел римскими цифрами:

1 – I	10 – X	100 – C
2 – II	20 – XX	200 – CC
3 – III	30 – XXX	300 – CCC
4 – IV	40 – XL	400 – CD
5 – V	50 – L	500 – D
6 – VI	60 – LX	600 – DC
7 – VII	70 – LXX	700 – DCC
8 – VIII	80 – LXXX	800 – DCCC
9 – IX	90 – XC	900 – CM

Запись остальных чисел происходит просто «склеиванием» записей приведенных чисел.

Например:

23 – XXIII 357 – CCCLVII 888 – DCCCLXXXVIII 901 – CMI 109 – CIX

Занятие №7

Операторы цикла While и Repeat … Until

Мы уже познакомились с оператором цикла for, который используется в тех случаях, когда число повторений тела цикла известно заранее. В языке Pascal существуют еще два оператора цикла, которые используются в тех случаях, когда число повторений цикла заранее не известно, либо сложно вычислимо.

Цикл While используется для повторения оператора (группы операторов) произвольное число раз, которое может быть заранее и не известно, причем *проверка условия выполнения тела цикла происходит перед выполнением тела цикла.*

Синтаксис оператора:

```
while условие do оператор;
```

или

```
while условие do
begin
  оператор1;
  оператор2;
  ....
end;
```

Суть выполнения оператора заключается в проверке логического условия, если оно оказывается *истинным*, выполняются операторы тела цикла до тех пор, пока логическое условие не станет *ложным*. Если условие было ложным перед выполнением цикла, то операторы цикла никогда не выполняются. Если условие остается истинным всегда, то цикл никогда не закончится. Говорят, что программа зацикливается.

Цикл repeat … until похож на цикл while. Его синтаксис:

```
repeat
  оператор1;
  оператор2;
  ....
until условие;
```

Обратите внимание, что, несмотря на несколько операторов в теле цикла, begin и end отсутствуют. Сам оператор представляет собой скобки. Цикл начинается с выполнения операторов, затем проверяется условие, если оно *ложно*, то цикл повторяется, а если *истинно*, то завершается. Если условие истинно и перед выполнением цикла, то цикл выполняется один раз. Если условие остается ложным всегда, то программа зацикливается.

Построим таблицу квадратов чисел от 1 до 10 с помощью цикла while и repeat … until.

```
Program SquareW;
var i : integer;
begin
 i:=1
 while i<=10 do
 begin
   writeln(i:4,sqr(i):4);
   i:=i+1;
 end;
end.
```

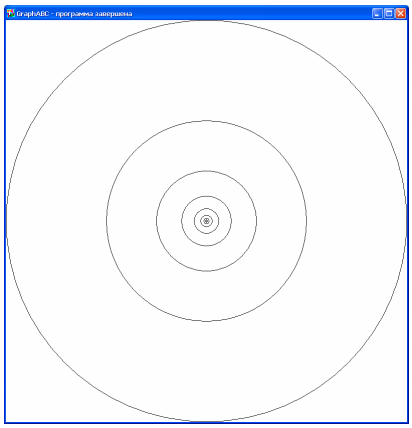
```
Program SquareR;
var i : integer;
begin
 i:=1
 repeat
   writeln(i:4,sqr(i):4);
   i:=i+1;
 until i>10;
end.
```

Рассмотрим следующий пример. Необходимо построить вложенные друг в друга концентрические окружности. Радиус самой большой окружности – 400 пикселей, а радиус каждой вложенной – в два раза меньше предыдущей, т.е. 200, 100, 50, … Радиус последней – 1 пиксель.

Для решения этой задачи разумно использовать цикл while или repeat until, поскольку действительно сразу сложно понять сколько будет окружностей (хотя, конечно, можно сосчитать).

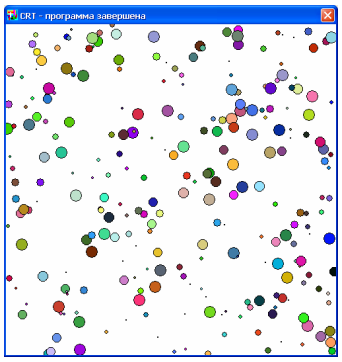
```
Program Circles;
Uses GraphABC;
var r : integer;
begin
 SetWindowSize(800,800);
 r:=400;

 repeat
  circle(400,400,r);
  r:=r div 2;
 until r<1;

end.
```

```
Program Circles;
Uses GraphABC;
var r : integer;
begin
 SetWindowSize(800,800);
 r:=400;

 while (r>=1) do
 begin
  circle(400,400,r);
  r:=r div 2;
 end;
end.
```



Рассмотрим еще один пример. Программа должна рисовать случайные расположенные окружности случайного цвета и случайного размера (но не больше 10 пикселей) до тех пор, пока пользователь не нажмет на какую либо клавишу.

```
Program Circles;
Uses GraphABC, CRT;
var r : integer;
begin
 SetWindowSize(500,500);
 repeat
  SetBrushColor(random($FFFFFF));
  circle(random(500),random(500),random(10));
  delay(10);
 until keypressed;
end.
```



Здесь используются несколько новых для нас приемов. Во-первых, функция **keypressed**, определенная в модуле CRT. Она возвращает значение **true**, если пользователь нажал любую клавишу. Оператор выбора цвета окружности SetBrushColor в качестве параметра использует значение **random($FFFFFF)**. Аргумент функции random представляет максимально возможное числовое значение цвета, записанное в шестнадцатеричной системе исчисления, таким образом окружности будут заливаться случайным цветом от 0 (соответствует черному цвету) до $FFFFFF (соответствует белому цвету). К таким обозначениям цветов мы вернемся во время изучения языка HTML. Функция **delay(n)** выполняет задержку выполнения программы на n миллисекунд. Мы ее используем для того, чтобы окружности не выводились слишком быстро.

Задание №7.

1. Напишите программу, которая вводила бы целые числа и суммировала их до тех пор, пока пользователь не ввел число 0. (*5 баллов*)
2. Модернизируете последний пример так, чтобы выводились случайные линии, либо прямоугольники.

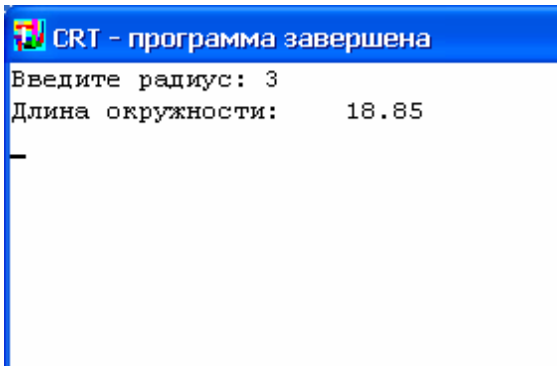
Занятие №8

Вещественные вычисления на языке Pascal

До сих пор мы оперировали целыми числами. Однако в физических вычислениях в вычислениях, связанными с измерениями, мы сталкиваемся с другим классом чисел. В математике их называют вещественными (или действительными). Подмножеством вещественных чисел являются рациональные числа. В языке Pascal вводится тип данных **real**, который является моделью вещественных чисел в математике.

Рассмотрим сразу пример:

```
Program Krug;
Uses CRT;

// Вычисление длины окружности

var r    : real; // Радиус окружности
    s    : real; // Длина окружности
begin
 write('Введите радиус: '); readln(r);
 s:=2*3.1415926*r;
 writeln('Длина окружности: ',s:8:2);
end.
```



Обратите внимание на описание переменных (тип real). Конечно, переменные можно описывать и несколько в одном операторе (`r, s : real`), но мы захотели добавить комментарии к описанию, поэтому описали переменные в отдельных операторах.

Ввод вещественных чисел с клавиатуры ничем не отличается от ввода целых чисел. Вещественные числа могут, как и целые, участвовать в арифметических выражениях. К ним применимы операции сложения +, вычитания –, умножения *, а также деления /. *Деление выполняется обычным способом*, как в математике, т.е. 5/2 будет 2.5. Деление на цело (div), остаток от деления (mod) для вещественных чисел не определены!

Запись вещественных чисел может быть в двух формах. Первая форма называется «с фиксированной точкой». Пример:

3.5, 2.0, +36.6, –40.123.

Обратите внимание на то, что в качестве разделителя целой и дробной части используется *точка*, а не запятая. Вторая форма записи называется «с плавающей точкой». Эта запись похожа на стандартизованное представление чисел в математике (например $6.67 \cdot 10^{21}$). В языке Pascal такое число можно записать в следующем виде 6.67E21, т.е. вместо ·10 в языке Pascal пишется буква E (большая или маленькая, всё равно). Приведем еще примеры записи вещественных чисел с плавающей точкой:

3E1, –2e+10, +1.234E–10, 65.2e+20.

Часть числа до буквы E называется *мантиссой*, а после буквы E – *порядком* или *экспонентой.*

Задание 8 (часть 1)

1. Запишите в языке Pascal в формате с *плавающей* точкой следующие числа: $2.99 \cdot 10^{33}$, $-0.21 \cdot 10^{-10}$, $35.6 \cdot 10^{8}$, 1200000, 0.000015 (*5 баллов*)
2. Запишите в языке Pascal в формате с фиксированной точкой следующие числа: 2.71828, $1.25 \cdot 10^{2}$, $-1.5 \cdot 10^{-1}$ (*3 балла*)

Вывод вещественных чисел с помощью оператора write/writeln происходит обычным способом. При выводе мы можем применять форматы вывода, отделяя их от переменной двоеточием. Формат вывода может состоять из одного или двух чисел, разделенных двоеточием (см. пример). При указании двух чисел *первое* определяет число позиций, резервируемых *для вывода всего числа*, а *второе* – число позиций *для дробной части*. Если в качестве формата вывода используется одно число, то оно определяет число позиций, резервируемых для вывода всего числа, а само число выводится в формате *с плавающей точкой*. В этом случае рекомендуется задавать значение формата не меньше 10, так как под экспоненту отводится пять позиций и еще две позиции выделяется под знак числа и десятичную точку. Если при выводе формат числа не указывать, то Pascal попытается вывести число наиболее компактным способом. Лучше уяснить действие форматов поможет следующий пример:

```
Program Formats;
Uses CRT;
var x    : real;

begin
 x:=1234.567;
 writeln(x);
 writeln(x:8:3);
 writeln(x:8:2);
 writeln(x:8:1);
 writeln(x:12);
 writeln(x:14);
end.
```

```
CRT - программа завершена
1234.567
1234.567
 1234.57
  1234.6
 1.2346E+003
 1.234567E+003
```

Рассмотрим еще одну программу, которая строит таблицу длин окружностей радиусами от 0.1 до 1 см с шагом 0.1 (заметьте, что для числа π используется встроенная константа **pi**):

```
Program Table;
Uses CRT;
var r, s : real;
begin
 r:=0.1;          // Начальное значение
 repeat
  s:=2*pi*r;      // Вычисление длины
  writeln(r:3:1,s:6:2); // Вывод
  r:=r+0.1;      // Увеличение радиуса на 0.1
 until r>1.0;    // Условие окончания цикла
end.
```

```
CRT - программа завершена
0.1   0.63
0.2   1.26
0.3   1.88
0.4   2.51
0.5   3.14
0.6   3.77
0.7   4.40
0.8   5.03
0.9   5.65
1.0   6.28
```

Задание 8 (часть 2)

3. Проанализируйте пример и постройте аналогичную программу, вычисляющую площади круга (по формуле $s = \pi r^2$). (*2 балла*)
4. Напишите программу, которая вводила бы с клавиатуры значения времени и скорости, вычисляла бы пройденный путь. (*3 балла*)
5. Напишите программу, которая вводила бы с клавиатуры 10 вещественных чисел и вычисляла бы их среднее арифметическое. (*5 баллов*)
6. * Напишите программу, которая вводила бы градусы, минуты и секунды дуги переводила их в градусы и его десятичные доли.

Занятие №9

Подпрограммы на языке Pascal

Функции

При решении сложных задач разумно разбить алгоритм на несколько более простых составляющих. В языке Pascal существуют специальные средства для этого – *подпрограммы*. Есть два вида подпрограмм: *процедуры* и *функции*. Функция получает информацию от вызывающей программы через свои параметры. Параметры, описанные в заголовке функции, называются *формальными* параметрами. Параметры, указанные при вызове функции называют *фактическими*. **Типы и число фактических параметров должны соответствовать типу и числу формальных параметров.** Следующий пример иллюстрирует работу функций.

```
Program Table;

Uses CRT; // Подключение модулей

  // Описания функций, требующихся в программе

  function cube(x:real):real;  // возведение в куб
  begin
   cube:=x*x*x;  // имени функции присваивается значение
  end;

  function sign(x:real):integer;  // вычисление знака числа
  begin
   if x>0 then sign:=1
          else if x=0 then sign:=0
                      else sign:=-1;
  end;

  //  ==== Начало главной программы ==========

  var   a : real;     // блок описания переменных
  const a1 = -5.0;    // блок описания констант (постоянных)
        a2 = +5.0;
        st =  0.5;
  begin
    a:=a1;
    while (a<=a2) do
    begin
     writeln(a:4:1,cube(a):10:3,sign(a):3); // вызов функций
     a:=a+st;
    end;
  end.
```

Обратите внимание, что внутри тела функции (заключенного в операторные скобки begin … end) вы **обязаны** хотя бы один раз **имени функции присвоить значение**. В этой программе также используется новый материал, не связанный с функциями, – это константы. Значение констант задается через знак равенства (а не присваивания). Тип константы определяется из типа присваиваемого значения. Константы, в отличие от переменных, не могут менять свое значение.

Рассмотрим еще один пример, в котором функция имеет два параметра разного типа.

```
Program Power2;

Uses CRT;

 function power(x : real; n : integer) : real; // возведение в степень
 var i : integer; // локальные переменные
     r : real;
 begin
   r:=1.0;
   for i:=1 to n do r:=r*x; // накопление произведения
   power:=r;                // результат присвоить имени функции
 end;

  //  ==== Начало главной программы ================

  var   i : integer;    // блок описания переменных
  begin
     for i:=1 to 10 do
       writeln(i:2,power(2.0,i):6:0);
  end.
```

В данном примере функция имеет два формальных параметра (типа real и типа integer), а также две *локальных переменных*. **Имена локальных переменных действуют только внутри тела функции**. Переменная *i* в главной программе, и переменная *i* внутри функции – это две *разных* переменных. При вызове функции первый *фактический параметр* соответствует первому *формальному параметру*. Второй параметр – второму.

Задание 9

1. Модифицируйте последний пример таким образом, чтобы функция power вычисляла правильно не только натуральные степени, но и отрицательные. Напоминание: $x^{-n} = \frac{1}{x^n}$. Подсказка: следует использовать оператор if, а также функцию abs(n), которая вычисляет модуль числа.

   (*5 баллов*)

2. Напишите функцию, вычисляющую факториал числа $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$. Подсказка: ее несложно сделать, путем модификации функции power.

   (*5-8 баллов*)

Занятие №10

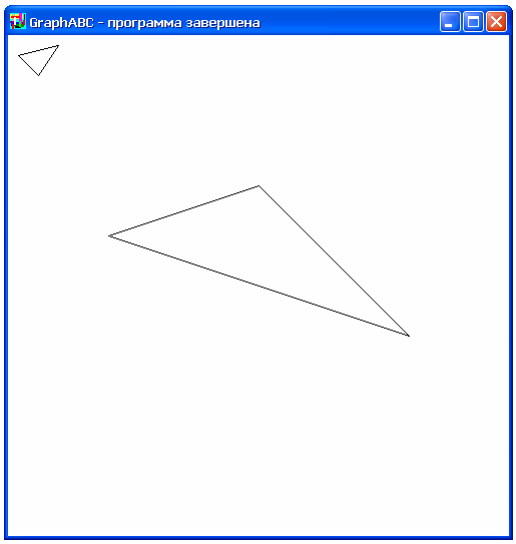
Подпрограммы на языке Pascal

Процедуры

Для выполнения каких-либо действий (а не вычислений значений) существуют **подпрограммы-процедуры**. Они также могут принимать параметры, как и *подпрограммы-функции*. Давайте сразу проиллюстрируем работу процедуры на примере графической программы.

В стандартной графической библиотеке нет процедуры, рисующей треугольники. Давайте создадим такую процедуру и проверим ее работу.

```
Program UseProc;

Uses GraphABC;

 Procedure Triangle(x1,y1,x2,y2,x3,y3:integer);
 // Процедура рисует треугольник
 begin
  line(x1,y1,x2,y2);
  line(x2,y2,x3,y3);
  line(x3,y3,x1,y1);
 end;

 // Главная программа
 begin
  SetWindowSize(500,500);
  Triangle(100,200,400,300,250,150);
  Triangle(10,20,30,40,50,10);
 end.
```



Как мы видим, процедура Trianlge имеет шесть целочисленных параметров и вызывает функции трижды функции Line для соединения точек. Создав такую процедуру, мы можем использовать ее столько раз, сколько нам надо в главной программе.

Задание 10

1. Создайте еще несколько новых графических процедур, которые бы расширили возможность стандартной библиотеки (ромб, параллелограмм, и т.п.)

(*по 3-4 балла за каждую*)

Модули (Units)

Вы, наверное, обратили внимание на неудобство, связанное с тем, что все подпрограммы (как процедуры, так и функции) размещаются перед главной программой. Хотелось бы разместить их в отдельную библиотеку. Такая возможность есть, её предоставляют **модули.**

Поместим описание процедуры Triangle в модуль MyGraph.

```
Unit MyGraph;    // Заголовок модуля. Имя файла модуля должно совпадать
                 // с именем модуля! Т.е. в нашем случае оно должно быть
                 // MyGraph.pas, иначе главая программа его не найдет!

Uses GraphABC;  // Модули могут использовать другие модули

Procedure Triangle(x1,y1,x2,y2,x3,y3:integer);
 // Эта процедура находится в модуле MyGraph
 begin
  line(x1,y1,x2,y2);
  line(x2,y2,x3,y3);
  line(x3,y3,x1,y1);
 end;

end. // Этот оператор – завершение модуля.
```

Теперь главная программа будет выглядеть так:

```
Program UseProc;

Uses GraphABC, MyGraph; // Использование стандартного и собственного модуля

begin
  SetWindowSize(500,500);   // Эта процедура описана в GraphABC
  Triangle(100,200,400,300,250,150); // Эта процедура описана в MyGraph
  Triangle(10,20,30,40,50,10);
end.
```

Модули могут содержать неограниченное число процедур, функций, а также описаний глобальных переменных, констант (об этом позже).

Задание 10

2. Сформируйте модуль с вашими собственными графическими процедурами

(*5 баллов*)

Занятие №11

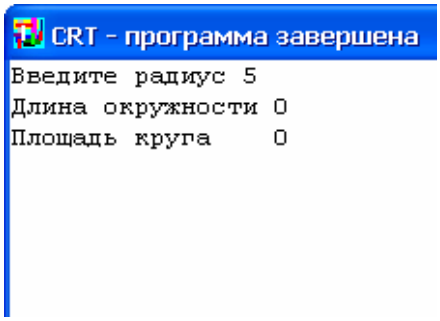
Способы передачи параметров

Параметры-значения и параметры-переменные

Каким образом подпрограмма может вернуть главной программе какую-либо информацию? До сих пор это могла делать только *подпрограмма-функция*, возвращая через свое имя только *одно значение*. А как поступить, если надо вернуть из подпрограммы несколько значений?

Для примера напишем подпрограмму, которая должна вычислить длину окружности ($L = 2\pi r$) и площадь круга ($S = \pi r^2$) по заданному радиусу.

```
Program ProcUse;

Uses CRT;

 // вычисление длины окружности и площади круга
 Procedure Pr(R,L,S:real);
 begin
  L:=2*PI*R;
  S:=PI*sqr(R);
 end;


 // ГЛАВНАЯ ПРОГРАММА
 var rad, len, area : real;
 begin
  write('Введите радиус '); readln(rad);
  Pr(rad, len, area);
  writeln('Длина окружности ', len);
  writeln('Площадь круга    ', area);
 end.
```



Попробуйте выполнить данную программу. Вы обнаружите странный результат: какое бы значение радиуса вы не вводили, результат будет один и тот же: длина окружности и площадь круга будут равны 0! В чем дело?

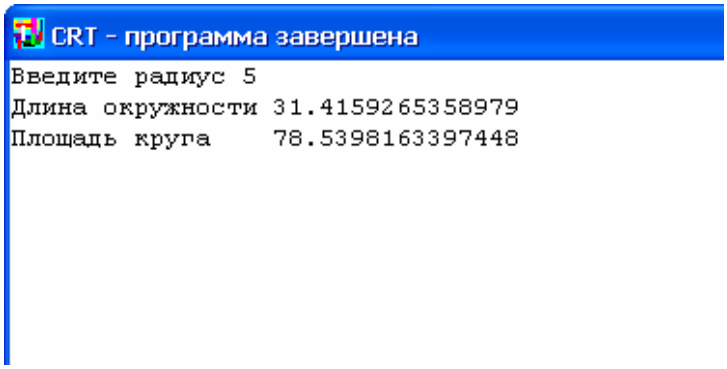
Оказывается тот способ передачи параметров подпрограмме, который мы использовали не подходит для решения нашей задачи. Используемый до сих пор способ называется **передача параметров по значению**. Суть его заключается в том, что подпрограмма создает *копии* параметров (в другой терминологии – *локальные переменные* подпрограммы), в которые помещаются *значения*, заданные в *фактических* параметрах в главной программе. Т.е. по смыслу такие параметры являются *входными*, т.е. предназначенными для передачи информации из главной программы в подпрограмму, но не наоборот. Однако этот подход имеет тот плюс, что подпрограмма *гарантирует*, что она не изменит фактические параметры.

Второй способ называется **передача параметров по ссылке**. Изменим заголовок процедуры Pr на следующий:

```
Procedure Pr(R: real; var L,S:real);
```

Обратите внимание на появившееся ключевое слово **var**. Оно говорит о том, что два последних параметра процедуры будут являться **изменяемыми**. Для таких параметров используется другой способ сопоставления с фактическими (передаются на самом деле адреса параметров). В связи с этим необходимо запомнить, что в качестве формальных параметров, соответствующих параметрам переменным, могут использоваться только переменные, но не константы! Т.е. вызов `Pr(5,6,7)` ошибочен, возможно, лишь `Pr(5,a,b)`, где `a` и `b` – вещественные переменные. Первый параметр мы оставили, как и раньше, обычным параметром, передаваемым по значению.

Выполним теперь программу. Работает!



Задание 11

1. Напишите процедуру с двумя целочисленными параметрами, которая бы меняла местами их значения. Проверьте ее работу.

   (*3 балла*)

2. Напишите процедуру, аналогичную процедуре Pr, которая вычисляла бы периметр и площадь квадрата со стороной *r* и а также объем куба с тем же самым ребром *r*.

   (*2 балла*)

Занятие №12

МАССИВЫ

Тема имеет исключительно важное значение

В практике программирования часто встречаются задачи, в которых требуется применение регулярных, пронумерованных данных: таблицы, результаты наблюдений, проекции векторов, числовые матрицы, каталоги библиотек и т.д. Для работы с такими данными практически во всех языках программирования существует понятие массива.

Массив – это регулярная структура данных, которая состоит из пронумерованных компонент одного и того же типа. Этот тип мы будем называть *базовым типом*.

Массивы могут быть одномерными:

A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8

и многомерными (например, двумерными):

A_{11}	A_{12}	A_{13}	A_{14}
A_{21}	A_{22}	A_{23}	A_{24}
A_{31}	A_{32}	A_{33}	A_{34}

С точки зрения машинной реализации, все массивы – одномерные, разница лишь в том, как пронумерованы элементы массива.

Описание одномерного массива, если считать его элементы целыми числами выглядит следующим образом:

```
A : array [1..8] of integer;
```

здесь **array** – ключевое слово, которое и обозначает собственно массив, в квадратных скобках указан *диапазон* первого и единственного *индекса*.

В Pascal'е в качестве диапазона индекса может выступать любой отрезок перечислимого типа, например 'A'..'H', либо 0..7. Однако на практике чаще всего удобнее в качестве индексов использовать отрезок целого типа, причем нижний (меньший) индекс разумно выбирать единицей или нулем.

Одной из самых неприятных ошибок программирования – является ошибка обращения к несуществующему элементу массива, или как говорят, ошибка выхода индекса за допустимый диапазон. Поэтому предыдущее определение массива A лучше переписать так:

```
Const N = 8;
Var A : array [1..N] of integer;
```

и в дальнейшем в программе при работе с массивом использовать не конкретные числа, а *константы*, которые определяют диапазон индексов, кроме того, программу можно будет легко модифицировать для работы с массивом другой размерности, так как необходимо будет изменить всего лишь одну строчку!

Иногда формальность описания следует развить, выделив *описание типа отдельно*, это будет абсолютно необходимо, если вы собираетесь использовать в процедурах и функциях параметры-массивы.

```
Const N = 8;
Type  TA = array [1..N] of integer;
Var   A : TA;
```

Дополнительные удобства этого подхода заключаются в том, что массивы, описанные в разных местах как массивы типа TA, будут являться совместимыми по типу, а в случае описания массивов A и B одинаковым способом, но без объявления типа-массива, они будут считаться несовместимыми. Например,

```
Const N = 8;
Type TA = array [1..N] of integer;
Var A : TA;
Var B : TA;
```

здесь A и B – массивы одного и того же типа. А здесь:

```
Const N = 8;
Var A : array [1..N] of integer;
Var B : array [1..N] of integer;
```

здесь A и B – массивы будут считаться разных типов. Хотя следующее описание определяет массивы одинаковых типов:

```
Const N = 8;
Var A,B : array [1..N] of integer;
```

В качестве базового типа допустим абсолютно любой тип, в том числе и массив, т.е. допустим массив массивов:

```
Const M = 5;
      N = 8;
Var   A : array [1..M] of array [1..N] of integer;
```

Подобная ситуация встречается довольно часто, поэтому для нее существует разумное сокращение:

```
Const M = 5;
      N = 8;
Var   A : array [1..M,1..N] of integer;
```

Следует учесть, что многомерные массивы, даже при небольших диапазонах индексов имеют тенденцию занимать много памяти.

Основные приемы работы с массивами

Рассмотрим выполнение элементарных манипуляций с массивами. Самая простая задача – заполнение всех элементов одним и тем же значением:

```
{Инициализация массива}
for i:=1 to N do A[i]:=0;
```

Обратите внимание, как осуществляется доступ к элементам массива – после имени массива *в квадратных скобках указывается индекс*, который может быть произвольным выражением, лишь бы его значение не выходило за указанный при описании диапазон. Подобная конструкция допустима везде, где допустима простая переменная.

Цикл **for** – чрезвычайно удобная и полезная вещь при работе с массивами. Оператор вида **for** i:=1 **to** N **do** – можно «переводить» как «выполнить для всех элементов массива».

Если два массива одного типа, то допустимо присваивание одного массива другому одним оператором:

```
B:=A;
```

Следующие два примера показывают, как осуществить ввод-вывод с небольшим сервисом:

```
{ввод массива}
for i:=1 to N do
begin
  write('Введите ',i,'-й элемент: '); readln(A[i])
end;
```

Из этого примера видно, что массив вводится поэлементно, и как организовать нехитрый сервис. Вывод производится аналогично:

```
{вывод массива}
for i:=1 to N do writeln('A[',i,']=',A[i]);
```

Теперь рассмотрим самую первую нашу задачу на обработку массива – поиск максимального элемента. Поступим следующим образом: пусть максимальный элемент массива – первый, заведем для него специальную переменную; затем будем просматривать поочередно последующие элементы, и если окажется, что нам встретится элемент больший, чем уже определенное число, то заменим его на этот элемент массива. Таким образом, когда мы просмотрим весь массив, окажется, что наша переменная содержит искомое значение:

```
{определение максимального значения}
max:=A[1];
for i:=2 to N do if A[i]>max then max:=A[i];
writeln('Maximum=',max);
```

Напишем теперь целиком программу, использовав полученные знания:

```
Program Massiv;

Const N = 10;
Var A : array [1..N] of integer;
    i, max : integer;

begin

  for i:=1 to N do // Ввод массива
  begin
    write('Введите ',i,'-й элемент: ');  readln(A[i])
  end;

  max:=A[1]; // Поиск максимального значения
  for i:=2 to N do if A[i]>max then max:=A[i];
  writeln('Maximum=',max);

end.
```

Задание 12

1. Внимательно прочитать текст. Знать определение массива и способы его описания.

(2 *балла*)

2. Напишите программу, которая вводит с клавиатуры значения массива, состоящего из 10 элементов, а затем выводит его.

(2 *балла*)

3. Модифицируйте предыдущую программу, так чтобы она выводила элементы массива в обратном порядке (используйте цикл `for i:=N downto 1 do`).

(1 *балл*)

4. По аналогии с примером на стр. 35 напишите программу, находящую минимальный элемент массива и выводящую его значение.

(2 *балла*)

5. Модифицируйте предыдущий пример, так чтобы программа определяла максимальный и минимальный элемент массива.

(1 *балл*)

6. * Напишите программу, которая бы определяла среднее арифметическое значение элементов массива (конечно, это будет вещественная величина типа `real`)

(* 3 *балла*)

7. * Напишите программу, которая бы вводила значения элементов целочисленного массива, а затем рисовала бы N окружностей, радиусы которых бы равнялись введенным значениям.

(* 3 *балла*)

*Задачи, отмеченные *, являются необязательными и их баллы – дополнительными.*

Занятие №13

СОРТИРОВКА МАССИВОВ

Тема имеет исключительно важное значение

Первой серьезной задачей программирования, с которой сталкивается начинающий программист – это задача сортировки массива. Под сортировкой понимается упорядочивание элементов массива по возрастанию (или по убыванию) без создания копии массива (т.е. «на месте»).

Самый простой алгоритм – это линейная сортировка. Рассмотрим рис. 13.1. Проведем последовательно сравнение первого элемента со всеми последующими, при если при очередном сравнении (например сразу 4 и 2) выяснится, что элементы стоят в «неправильном» порядке – переставим их местами, затем продолжим сравнение (рис. 13.2). По окончании одного прохода, можно сказать, что в первом элементе массива находится минимальный элемент (Рис. 13.4).

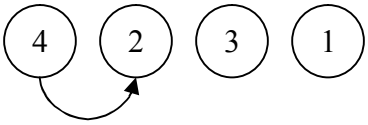


Рис 13.1

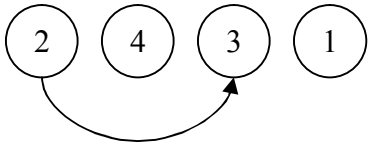


Рис 13.2

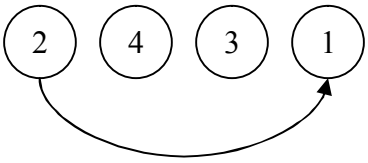


Рис 13.3

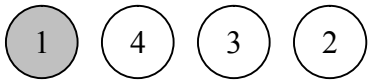


Рис 13.4

Далее применим указанную процедуру к неотсортированному «остатку» массива (рис. 13.5) до тех пор, пока не переставим два последних элемента (рис. 13.6-13.7).

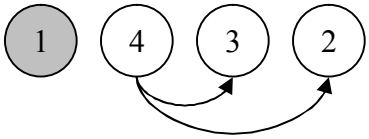


Рис 13.5

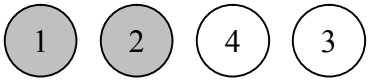


Рис 13.6

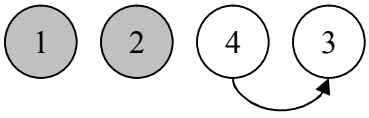


Рис 13.6

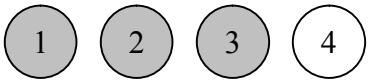


Рис 13.7

Алгоритм линейной сортировки очень прост, но не экономичен, среднее число просмотров и перестановок пропорционально квадрату числа элементов (точнее $\sim \frac{N^2}{2}$).

Приведем программу сортировки. Обратите внимание, что мы использовали массив в качестве параметра процедуры. Для этого необходимо создать тип Massiv (стр. 34). Часто для экономии памяти массив передают через var-параметр (стр. 32), даже если не предполагается его модифицировать в подпрограмме. Т.е. заголовок процедуры print мог бы выглядеть следующим образом: **procedure** `print(`**var** `m : Massiv);` .

```
Program LinerSort;

Const N = 10; // Число элементов массива

Type  Massiv = array [1..N] of integer; // Определение типа Massiv


procedure swap(var x,y: integer); // Перестановка элементов местами
var z : integer;
begin
 z:=x; x:=y; y:=z;
end;

procedure print(m : Massiv); // Вывод массива
var i : integer;
begin
 for i:=1 to N do write(m[i]:5);
 writeln; // Новая строка
end;


// Переменные главной программы
Var   a : Massiv;
      i,j : integer;

begin
 // Заполнение массива случайными числами в диапазоне от 0 до 99
 for i:=1 to N do a[i]:=random(100);
 print(a); // Вывод массива

 for i:=1 to N-1 do // Внешний цикл до N-1      (обратите внимание!)
   for j:=i+1 to N do // Внутренний цикл от i+1 (обратите внимание!)
     if (a[i]>a[j]) then swap(a[i],a[j]); // Перестановка элементов

 print(a); // Вывод отсортированного массива
end.
```

Задание 13

1. Внимательно прочитать текст. Оформите сортировку массива в виде отдельной процедуры (здесь уже применение var-параметра будет обязательным).

(2 *балла*)

2. Добавьте в процедуру сортировки операторы, которые позволил ли бы узнать сколько раз происходят перестановки в процессе сортировки. Выясните этот вопрос для N=10, 100, 1000.

(3*балла*)

Занятие №14

РАБОТА С ФАЙЛАМИ

Многим программам требуется сохранять и читать информацию, используя файловую систему компьютера. В языке Pascal изначально были предусмотрены специальные операторы и типы данных для работы с файлами.

В ABC Pascal есть два вида файлов: текстовые и типизированные. В типизированных файлах обмен с внешними устройствами производится без какого либо преобразования данных, т.е., например, числа типа *integer* непосредственно копируются на диск, занимая по 4 байта каждое. Попытка просмотра такого файла в текстовом редакторе обречена на неудачу, мы увидим лишь бессмысленный набор знаков. Однако скорость ввода/вывода для таких файлов будет максимальной. Типизированные файлы мы рассмотрим позже в связи с типом данных *record*.

Работа с текстовым файлами очень похожа на работу с обычным консольным вводом/выводом. Числовые данные преобразуются в цифры в соответствии с заданными форматами (стр. 15 и стр. 26). Строковый и символьный тип данных выводится без преобразований. Следует учесть, что текстовый файл может быть открыт либо на чтение, либо на запись

Созданный текстовый файл можно прочитать в простом текстовом редакторе (notepad, aditor, в редакторе ABC Pascal, [можно и в Word[12]]). В текстовом файле ABC Pascal используется кодировка Win-1251, в которой один символ занимает один байт. Текстовый файл можно создать в редакторе (в соответствии с указанными правилами) и прочитать в программе на ABC Pascal.

Рассмотрим сразу простой пример – вывод таблицы квадратов первых 10 чисел в текстовый файл `table.txt`.

```
Program TextOut;

const name = 'text.txt'; // имя файла в текущем каталоге

var f : text;            // файловая переменная
    n : integer;         // переменная для цикла for

begin
 assign(f,name); // связывание файловой переменной с именем файла на диске
 rewrite(f);     // создание и открытие файла на запись
 for n:=1 to 10 do writeln(f,n:2,sqr(n):4); // вывод в файл writeln(f,...);
 close(f); // закрытие файла, сохранение всех еще незаписанных данных на диск
end.
```

В этом пример надо обратить внимание на несколько операторов:

1. **`f : text`** – переменная специального встроенного типа «текстовый файл»;
2. **`assign(f,name)`** – сопоставление файлу f в программе файла name на диске;
3. **`rewrite(f)`** – «перезаписывает» файл f, т.е. либо создает новый пустой файл, либо уничтожает старый (будьте осторожны поэтому) и опять создает новый пустой файл;
4. **`writeln(f,…)`** – модификация уже известного оператора `writeln`, отличается от привычного только тем, что первый параметр – имя файловой переменной
5. **`close(f)`** – файлы надо обязательно закрывать, особенно файлы, открытые на запись (как в приведенном примере), иначе часть данных может быть утеряна.

[12] Для этого в MS Word при создании файла надо выбрать тип «*.txt – обычный текст», а при открытии указать, что мы открываем текстовый файл в кодировке Win-1251.

Вместо оператора **rewrite**, файл можно открыть оператором **append**, в этом случае будет произведено открытие уже существующего файла в режиме дозаписи в конец файла.[13]

Рассмотрим теперь пример чтения уже существующего файла, в качестве файла используем созданный в предыдущем примере файл `text.txt`.

```
Program TextIn;
Uses CRT;

const name = 'text.txt'; // имя файла в текущем каталоге

var f : text;          // файловая переменная
    a,b : integer;     // переменные для чтения

begin
 assign(f,name); // связывание файловой переменной с именем файла на диске
 reset(f);       // открытие существующего файла на чтение
 repeat
  readln(f,a,b);     // чтение из файла информации из целой строки
  writeln(a:5,b:5); // вывод в окно CRT
 until Eof(f);       // Функция eof возвращает true при достижении конца файла
 close(f);           // закрытие файла
end.
```

В этом пример надо обратить внимание на следующее:

1. **reset(f)** – открытие существующего файла на чтение, если файла нет, то произойдет ошибка выполнения программы;
2. **readln(f,**…) – оператор чтения из файла, при работе с файлами действие операторов **read(f,…)** и **readln(f,…)** различно, первый прочитает необходимую информацию посредине строки, так что следующий оператор чтения продолжит чтение со средины строки. Оператор **readln** после чтения информации пропустит все оставшиеся до конца строки символы, таким образом следующий оператор чтения начнет ввод с начала следующей строки;
3. Функция **eof(f)** возвращает всегда ложное значение, кроме одного единственного случая: достигнут конец строки. Для того, чтобы обойти всякие тонкие случаи, когда в конце файла есть несколько символов «конец строки» или лишние пробелы и символы табуляции, рекомендуются использовать функцию **SeekEof(f)** – ее действие аналогично eof, но она возвращает true, если до конца файла есть только «пустые» символы: конец строки, табуляция и пробелы[14].

Задание 14

1. Напишите программу, создающую таблицу умножения в файле mult.txt. Для ее создания используйте вложенные циклы *for* (стр. 20). Откройте получившийся файл в текстовом редакторе

(3*балла*)

2. Напишите программу чтения файла, созданного в упражнении 14.1

(2 б*алла*)

[13] В ABC Pascal существуют функции FileExists(name), проверяющая, существует ли файл с таким именем, и CanCreateFile(name), проверяющая можно ли создать файл с таким именем.

[14] В ABC Pascal существуют две аналогичные функции Eoln и SeekEoln, которые вместо конца файла ищут конец строки.

Занятие №15

СИМВОЛЬНЫЕ ПЕРЕМЕННЫЕ И СТРОКИ СИМВОЛОВ

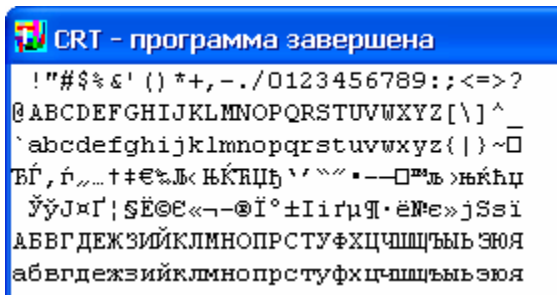
Пример:

```
var c : char;
…
c:='a';
readln(c);
c:=ReadKey;
```

В языке Pascal присутствуют два типа для работы с символьной информацией. Переменные типа char могут принимать значение одного из 256 символов стандартной таблицы кодировки. Константы типа **char** записываются в виде одного символа в апострофах. Символьные переменные можно вводить оператором `readln`, но при этом придется еще нажимать *Enter* для подтверждения ввода. В модуле `CRT` есть очень удобная функция **ReadKey**, которая вводит символьную переменную без подтверждения ввода.

Для работы с символьными переменными в языке Pascal есть две специальные функции: ord и chr. Первая функция возвращает код (порядковый номер в таблице кодировки) символа, например ord('A') вернет значение 65. Вторая функция имеет обратный смысл, по коду, возвращает символ, т.е. chr(65) будет символ 'A'. Следующий пример выводит все символы кодовой таблицы, начиная с кода 32[15] и заканчивая кодом 255. Оператор if в цикле производит переход на новую строку после вывода линейки из 32 символов. Результат работы программы приведен справа.

```
Program ANSI;
Uses CRT;
var c : char;
    i : integer;
begin
  for i:=32 to 255 do
  begin
   write(Chr(i));
   if (i+1) mod 32 = 0 then writeln;
  end;
end.
```



В языке Turbo Pascal появился специальный тип string, который имеет черты массива символов, а также некоторые дополнительные удобные возможности. Тип string реализован и в языке ABC Pascal.

Строки в Pascal ABC имеют тип string и состоят из не более чем 255 символов. При описании

```
var s: string;
```

под переменную s отводится 256 байт, при этом в нулевом байте хранится длина строки.

Для экономии памяти предусмотрено описание вида

```
var s1: string[40];
```

В этом случае под строку отводится 41 байт (нулевой байт – под длину строки). В случае присваивания переменной `s1` строки из более чем 40 символов лишние символы отсекаются, и длина строки `s1` полагается равной 40.

При выполнении операции *конкатенации* («склеивания») вида **s1+s1** результат считается строкой типа string, т.е. занимает 256 байт. Однако если при конкатенации результат будет занимать более 255 символов, то программа завершится с ошибкой. Строки можно сравнивать, подразумевается их упорядоченность по алфавиту.

К символам в строке можно обращаться, используя индекс: `s[i]` обозначает *i*-й символ в строке. Обращение к нулевому символу `s[0]` считается ошибочным.

[15] Коды с 0 по 31 являются служебными и в таблице ANSI им не соответствуют никакие печатные символы.

Для того чтобы узнать длину строки следует воспользоваться встроенной функцией **Length**(s). Чтобы изменить длину строки, следует воспользоваться процедурой **SetLength**(s,n). Если индекс *i* выходит за пределы памяти, отводимой под строку, то выдается сообщение об ошибке. Однако если индекс *i* выходит лишь за пределы длины строки, то сообщение об ошибке не выдается.

Тип char и тип string могут быть параметрами процедур и функций, а также возвращаться функциями. Для иллюстрации работы со строками и символами напишем функцию, заменяющую в строки заданный символ на другой и возвращающую результат.

```
Program Strings; // замена символов a на b в строке s

function replace(s : string; a,b : char):string;
 var i : integer;
 begin
  for i:=1 to Length(s) do  // цикл для всех символов строки
    if s[i]=a then s[i]:=b; // замена символов
  replace:=s;               // имени функции присваивается значение
 end;

begin
  writeln(replace('мама мыла раму','м','н')); // Проверка работы функции
end.
```

Тип char или string может использоваться для чтения информации из текстового файла, например:

```
Program ReadText;
Uses CRT;
var name : string;  // Имя файла
    s    : string;  // Буфер для чтения строки
    f    : text;    // Файловая переменная
    n    : integer; // Счетчик строк
begin
  write('Введите имя файла: '); readln(name); // Ввод имени файла
  assign(f,name); reset(f); // Открытие файла
  n:=0;                     // Обнуление счетчика
  while not eof(f) do       // Чтение «пока не конец файла»
  begin
    readln(f,s);           // Чтение одной строки из файла f
    writeln(n:4,': ',s);   // Вывод номера и строки на экран
    inc(n);                // inc(n) – операция увеличения на единицу
    if n mod 25 = 0 then readln; // Пауза после каждой 25-й строки
   end;
   close(f);
 end.
```

Задание 15

1. С помощью функций ReadKey и Ord напишите программу, которая выводила бы сразу символ и его код по нажатию одной клавиши. (2 *балла*)
2. Напишите целочисленную функцию с двумя параметрами, подсчитывающую сколько раз символ, заданный вторым параметром, встречается в строке, заданной первым параметром, и проверьте её работу. (2 *балла*)
3. Напишите программу, запрашивающую имена файлов, и копирующую один файл в другой, заодно вычисляя число строк (или символов) в файле (3 *балла*)

Для работы с текстовыми строками в языке Pascal существует набор функций, который немного расширен в реализации ABC Pascal. Приведем справочник по этим функциям.

Стандартные процедуры и функции для работы со строками

Имя и параметры	Типы параметров	Тип возвращаемого значения	Действие
`Length(s)`	`s` - **`string`**	`integer`	возвращает длину строки `s`
`Copy(s,index,count)`	`s` - **`string`**, `index` и `count` - `integer`	**`string`**	возвращает подстроку строки `s` длины `count`, начиная с позиции `index`
`Delete(s,index,count)`	`s` - **`string`**, `index` и `count` - `integer`		удаляет в строке `s` `count` символов начиная с позиции `index`
`Insert(subs,s,index)`	`s`, `subs` - **`string`**, `index` - `integer`		вставляет подстроку `subs` в строку `s` с позиции `index`
`Pos(subs,s)`	`s`, `subs` - **`string`**	`integer`	возвращает позицию первой подстроки `subs` в строке `s` (или 0 если подстрока не найдена)
`SetLength(s,n)`	`s` - **`string`**, `n` - `integer`		устанавливает длину строки `s` равной `n`

`Str(x,s)`
`Str(x:n,s)`
`Str(x:n:m,s)`
	`s` - **`string`**, `x` - `integer`, `real` и `n`, `m` - `integer`		преобразует `x` к строковому представлению (во втором и третьем случаях согласно формату вывода, устанавливаемому `n` и `m`) и записывает результат в строку `s`

`Val(s,v,code)`	`s` - **`string`**, `v` - `integer`, `real`, и `code` - `integer`		преобразует строку `s` к числовому представлению и записывает результат в переменную `v`. Если преобразование возможно, то в переменной `code` возвращается 0, если невозможно, то в `code` возвращается ненулевое значение
`Concat(s1,...,sn)`	`s1`,..., `sn` - **`string`**	**`string`**	возвращает строку, являющуюся результатом слияния строк `s1`,..., `sn`. Результат тот же, что у выражения `s1+s2+...+sn`
`UpCase(c)`	`c` - **`char`**	**`char`**	возвращает символ `c`, преобразованный к верхнему регистру
`LowCase(c)`	`c` - **`char`**	**`char`**	возвращает символ `c`, преобразованный к нижнему регистру
`UpperCase(s)`	`s` - **`string`**	**`string`**	возвращает строку `s`, преобразованную к верхнему регистру
`LowerCase(s)`	`s` - **`string`**	**`string`**	возвращает строку `s`, преобразованную к нижнему регистру
`Trim(s)`	`s` - **`string`**	**`string`**	возвращает копию строки `s` с удаленными лидирующими и заключительными пробелами

Занятие №16

МНОЖЕСТВА

В языке Pascal есть очень интересный тип данных *множество*. Множество представляет собой набор элементов одного порядкового типа. Элементы множества считаются неупорядоченными; каждый элемент может входить во множество не более одного раза. Тип множества описывается следующим образом:

```
set of базовый тип;
```

В качестве базового может быть любой порядковый тип с элементами, для которых функция **Ord** возвращает значения в диапазоне от 0 до 255. К таким типам, из изученных нами, относятся тип **char** и **byte** (а также их подмножества).[16] Аналогично массивам можно определить название нового типа в секции **type**, например:

```
type
  ByteSet = set of byte;
  CharSet = set of char;
  Digits  = set of '0'..'9'; // подмножества типа char
```

Сами множества задаются в виде перечисления их элементов (возможно с использованием диапазонов), заключенные в квадратные скобки:

```
var Vowels : CharSet;  // можно было написать vowels : set of char;
    Good   : Digits;   // можно было написать good : set of '0'..'9';

…
Vowels:=['A', 'E', 'O', 'I', 'U']; // Элементы явно перечислены
Good:=['3'.. '5', '10'];           // Диапазон и отдельный элемент
```

Для проверки принадлежности элемента множеству существует операция in:

```
var c : char;
…
if c in Vowels then inc(n); // Если c – гласная, то увеличить n на 1
```

На обороте страницы приведена программа, которая вводит текстовую строку с экрана и выводит ее на экран азбукой Морзе, заодно проигрывая ее. В этой программе используется много нового: работа с модулем Sound, массив из строк, индексация массива не целыми числами, а типом char, функция задержки выполнения Sleep, функция перевода буквы в верхний регистр Upcase, работа с множеством.

Задание 16

1. Внимательно изучите текст программы ABCMorse, задайте преподавателю вопросы.
2. Напишите программу, которая вводит текстовую строку, подсчитывает, сколько в ней гласных букв, согласных букв, знаков препинания. (3 *балла*)
3. Напишите программу, которая вводит из файла 10 текстовых строк и выводит их в другой файл, отсортированными в алфавитном порядке. Используйте алгоритм сортировки занятия №13, и операцию сравнения строк < или >. (5 *баллов*)
4. Основываясь на программе ABCMorse придумайте интересную задачу на использование текстовых строк и множеств. (*доп. баллы*)

[16] Другие возможные базовые типы – перечисления (об этом позже).

```
Program ABCMorze;
Uses Sounds,CRT;

var A : array ['A'..'Я'] of string[5];
    c : char;
    f : text;
    s : string;
    i,j : integer;
    dash, dot : integer;
    morze : string[5];

begin

// Читаем файл и заполняем массив строк A['А'], A['Б'] ,... A['Я'] азбукой Морзе
 assign(f,'morze.txt'); reset(f);
 for c:='A' to 'Я' do readln(f,A[c]);
 close(f);

 // Выводим таблицу на экран
 for c:='A' to 'Я' do write(c,' ',A[c]:5,' ');

 // Загружаем звуки
 dash:=LoadSound('dash.wav'); dot:=LoadSound('dot.wav');

 // Вводим текстовую строку
 writeln; write('Введите сообщение '); readln(s);

 for i:=1 to length(s) do // Разбираем ее по символам
  if Upcase(s[i]) in ['A'..'Я'] then // Проверка принадлежности
  begin // Если это русская буква (перевод в заглавные – Upcase)

   morze:=A[Upcase(s[i])]; // Читаем s[i] элемент массива A в строку morze

   for j:=1 to Length(morze) do // Разбираем строку Morze и проигрываем её
   begin

     write(morze[j]); // Вывод на экран буквы кодом Морзе

     if morze[j]='-' then
     begin
        PlaySound(dash);  Sleep(SoundLength(dash)); // играем dash
     end
     else
       begin
          PlaySound(dot); Sleep(SoundLength(dot));  // играем dot
       end;
     Sleep(50);
   end;

   write(' '); Sleep(500); // Пауза между буквами 500 мс

  end; // end к внешнему оператору if

end.
```

Файл morse.txt

```
.-
-...
.--
--.
-..
.
...-
--..
..
.---
-.-
.-..
--
-.
---
.--.
.-.
...
-
..-
..-.
....
-.-.
---.
----
--.-
--.--
-.--
-..-
..-..
..--
.-.-
```

Занятие №17

Вычислительные программы – квадратное уравнение

Решение квадратного уравнения выходит за рамки курса математики 7 класса, но мы сочли возможным внести эту классическую задачу программирования в дополнение.

Квадратным уравнением называ$d = 0$ется уравнение вида

$$ax^2 + bx + c = 0$$

На первом этапе его решения определяется так называемый дискриминант:

$$d = b^2 - 4ac$$

Далее рассматривают три случая

1. $d < 0$ – в этом случае вещественных решений нет[17];
2. $d = 0$ – один корень, определяемый формулой $\frac{-b}{2a}$;
3. $d > 0$ – два корня, вычисляемых по формуле $x_{1,2} = \frac{-b \pm \sqrt{d}}{2a}$, где знак «+» соответствует первому корню, а знак «–» – второму корню.

Составим программу для решения этой задачи:

```
Program SquareEquation;
var a, b, c : real; // Коэффициенты уравнения
          d : real; // Дискриминант
     x1, x2 : real; // Корни уравнения

begin
 write('Введите a, b, c '); readln(a,b,c); // Ввод исходных данных
 d:=b*b-4*a*c; // Вычисление дискриминанта

 if d<0 then writeln('Решений нет')
        else if d=0 then begin
                          x1:=-b/(2*a);
                          write('x=',x1)
                         end
                    else begin
                          x1:=(-b+sqrt(d))/(2*a);
                          x2:=(-b-sqrt(d))/(2*a);
                          write('x1=',x1,'  x2=',x2)
                         end
end.
```

Последнюю ветвь else можно оптимизировать, убрав повторяющиеся вычисления (особенно вычисление квадратного корня)

```
                    else begin
                          d:=sqrt(d); a:=2*a;
                          x1:=(-b+d)/a;
                          x2:=(-b-d)/a;
                          write('x1=',x1,'  x2=',x2)
                         end
```

[17] Решение есть только в комплексных числах