МИНИСТЕРСТВО ОБЩЕГО И ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ СВЕРДЛОВСКОЙ ОБЛАСТИ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ СВЕРДЛОВСКОЙ ОБЛАСТИ «КАМЫШЛОВСКИЙ ТЕХНИКУМ ПРОМЫШЛЕННОСТИ И ТРАНСПОРТА»

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО
ЛАБОРАТОРНЫМ И ПРАКТИЧЕСКИМ РАБОТАМ
учебной дисциплины
Основы программирования
для студентов профессии
09.01.03 «Мастер по обработке цифровой информации»

Составила:
Ю. Л. Бейтельмахер
преподаватель спецдисциплин, ВКК

Камышлов
2016

АННОТАЦИЯ

Данное учебно-методическое пособие предназначено для использования в рамках программы среднего профессионального образования по профессии "Мастер по обработке цифровой информации", учебной дисциплины «Основы программирования». Практикум предназначен для закрепления студентами начальных сведений и получения первых навыков программирования, освоения базовых конструкций языков программирования.

Практикум состоит из серии лабораторных работ, каждая из которых включает краткий теоретический и справочный материал по рассматриваемой теме, примеры решения задач.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Программирование сегодня - это бурно развивающаяся отрасль производства программных продуктов. В конце прошлого века общаться с компьютерами можно было лишь с помощью программирования, и поэтому программирование было включено в программы фактически всех учебных заведений (в том числе и школ). Времена изменились, общаться с компьютерами можно уже с помощью готовых компьютерных программ, и нужда в массовом обучении программированию вроде бы отпала. Однако все оказалось не так просто. В современные прикладные пакеты включаются, как правило, дополнительные средства программирования, обеспечивающие расширение возможностей этих пакетов. Например, практически в любом пакете MS Office есть среда программирования VBA (Visual Basic for Applications), обеспечивающая расширение возможностей этого пакета; профессиональная работа с системой «1С Предприятие» требует постоянного программирования для настройки на потребности каждой фирмы. Речь уже ведется о новом подходе, в рамках которого программирование - это обязательная компонента подготовки специалистов, планирующих профессионально работать в определенной сфере, предполагающей использование IT-технологий.Программирование сегодня - это не только и не столько знание языка программирования. Прежде всего, это знание *технологии* программирования, умение *проектировать* и *разрабатывать* программы и программные комплексы на основе этой технологии, умение *строить модели,* ставить задачи и иметь представление о коллективной разработке программных продуктов. Все это принято называть *«культурой программирования».*

В данных рекомендациях получите практические советы по изучению языка программирования Паскаль (Pascal), с элементами *структурного программирования* (т.е. программирования без использования оператора безусловного перехода), с *технологией проектирования «сверху — вниз»* и с *модульным программированием* (т.е. разбиением программы на подпрограммы для удобства отладки и коллективной реализации).

В каждой теме определены цели и задачи выполнения лабораторных работ, описание каждой работы включает в себя необходимые для выполнения работы теоретические сведения, экспериментальную часть, указания по обработке результатов и их представлению в отчете. В приложении дан минимальный справочный материал.

Особенностью практикума является во многом самостоятельная деятельность студента при наличии необходимого контроля со стороны преподавателя и дифференцированный подход к распределению задач. Результатом прохождения практикума предполагается формирование у обучающихся ряда профессиональных компетенций, в том числе способность применять в профессиональной деятельности языки и технологии программирования, разработку алгоритмических и программных решений профессиональных задач. Практикум предназначен для закрепления студентами начальных сведений и получения первых навыков программирования, освоения базовых конструкций языков программирования.

1. ОСНОВНАЯ ЧАСТЬ

Лабораторная работа 1. Составляющие языка, основные знаки и символы

1. **Как начать работу со средой Pascal ABC (**Сначала подготовимся к работе).

Задание 1. Создайте папку на диске «**Рабочая**». В папке **Рабочая** создайте папку **Pascal,** а в ней - папку с Вашей фамилией. Далее эту папку будем называть «Вашей папкой». Из папки **Учебная\Examp1es** в свою папку скопируйте все имеющиеся там файлы.

1.1. Алфавит языка Pascal

Алфавит языка Pascal состоит из:

- 26 букв английского алфавита и знака подчеркивания - "_";
- 32 букв русского алфавита (только для комментариев и текстовых сообщений);
- 10 арабских цифр (0 - 9);
- знаков математических операций +, -, *, /, mod, div;
- знаков операций отношений <, >, <=, >=, =, <>;
- разделителей - ".", ",", ";", ":", (), [], {}, ' ;
- спецсимволов - #, $, ^, &, @, := .

Прописные и строчные буквы Pascal'ем не различаются. Знаки «возведение в степень» и двойная кавычка - " - в языке отсутствуют.

1.2. Запуск среды

Задание 2. Запустите среду программирования PascalABC **(Пуск → Все программы → PascaABC → PascalABC).** Если среда запустилась, перед Вами появится окно (Рис. 2).

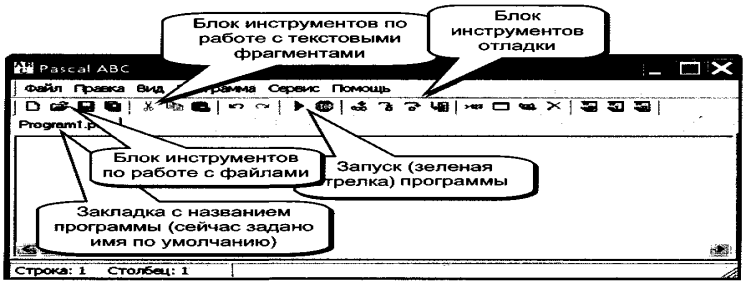


Рис. 2. Структура окна среды PascalABC

Задание 3. Изучите комментарии, приведенные на Рис. 2.

Задание 4. Выберите пункт меню **Помощь,** там - подпункт **Содержание.** Щелкните по закладке **Содержание,** в разделе **Общие сведения** прочтите пункты **О системе PascalABC** и **Типы приложений.** Выясните, что такое:

а) *консольные приложения* и где при выполнении этих приложений располагается окна вывода и ввода;

б) *графические приложения* и чем они отличаются от консольных.

1.3. Работа со страницами. Запуск программ

Лист, на котором будем размещать программу, будем называть *страницей.*

1.3.1. Как открыть программу и запустить ее на выполнение

Задание 5. Откройте программы **demoO.pas, demo 1.pas, demo2.pas, demo3.pas, colors.pas** из вашей папки. Программы (точнее - тексты программ) открываются обычным образом: **Файл → Открыть,** или кнопка Открыть на стандартной панели инструментов. Обратите внимание, что имена открытых файлов появляются на закладках в верхней части окна. Переход от программы к программе реализуется с помощью щелчка мыши на нужную закладку.

Задание 6. Запустите программу **demoO.pas.** Для запуска программы следует:

а) перейти на страницу с текстом данной программы. Для этого достаточно щелкнуть по закладке с названием **demoO.pas** в верхней части рабочего окна;

б) и щелкнуть по зеленой стрелке в панели инструментов;

в) в случае если программу следует прервать, нужно щелкнуть по красному значку в этой же панели (Рис. 3).

Убедитесь, что перед вами - консольное приложение. Определите для этого приложения расположение окна вывода.

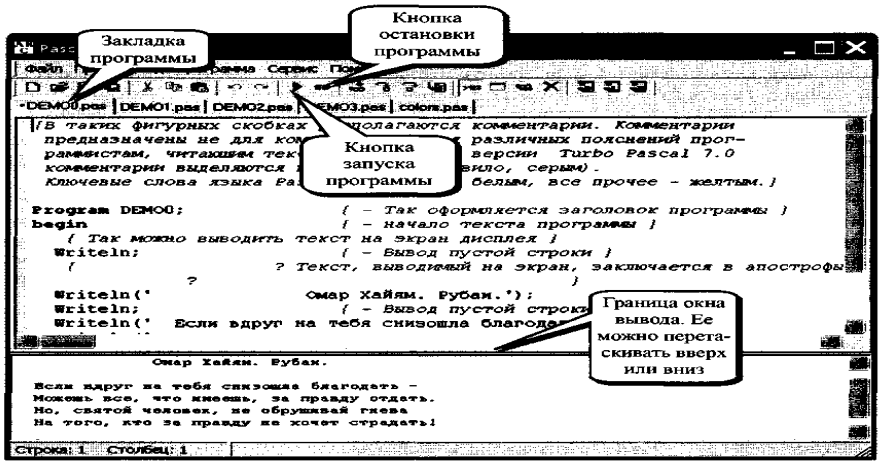


Рис. 3. Основные компоненты, используемые при запуске и остановке программы

1.3.2. Как убрать окна ввода и вывода

Задание 7. Скройте окно вывода. Для этого можно либо щелкнуть по кнопке **Окно вывода** (на ней надпись «>123»), расположенной в панели инструментов, либо просто нажать клавишу F5.

Задание 8. Запустите программу **demo2.pas.** Ответьте на все вопросы программы. Убедитесь, что перед вами - консольное приложение. Определите для этого приложения расположение окон вывода и ввода. Если программа работает правильно (а она должна угадать любое задуманное вами число), то также закройте окно вывода.

☞*Чтобы было удобней читать, расширьте окно вывода.*

Задание 9. Запустите программу **demo3.pas.** Ответьте на все вопросы программы. (Убедитесь, что перед вами - графическое приложение). Чем данная программа отличается от программы **demo2.pas?**

Задание 10. Запустите программу **Colors.pas.** Попытайтесь понять, как именно работает эта программа. (Убедитесь, что перед вами - также графическое приложение).

1.3.3. Как убрать имеющуюся страницу

Если изменений в текстах Вы не производили, а текст больше не нужен для текущей работы, страницу можно убрать с рабочего стола. Для этого достаточно щелкнуть правой клавишей мыши по закладке и в контекстом меню выбрать либо пункт **Закрыть** (т.е. закрыть текущее окно), либо **Закрыть неактивные,** т.е. закрыть все окна, кроме текущего.

Задание 11. Закройте все неактивные страницы. Выясните, можно ли закрыть оставшуюся страницу.

1.3.4. Как создать страницу

Фактически новая страница - это новый документ. Для создания нового документа используется пункт меню **Файл → Новый.** В этом случае появится новая страница, а информация с этой страницы будет записываться в файл с именем **ProgramLPAS.** В случае, если в новом листе началась разработка новой программы (т.е. просто начали набирать некий текст), то по попытке создать новую страницу будет создан файл с именем **Program2.PAS** и т.д.

Задание 12. Создайте новую страницу. После этого наберите текст следующей программы:

```
Program exO;
uses Crt;            {Присоединяем библиотеку с именем Crt}
Begin
TextColor(red);      {цвет текста - красный}
TextSize(14);        {размер текста -14 пунктов}
TextBold;            {текст должен быть жирным}
Write('Mofl первая программа')
end.
```

Запустите программу. Выясните, это приложение консольное или графическое.

☞*Приложение становится графическим, если в нем подключается какая-либо библиотека.*

☝***Пригласите преподавателя и продемонстрируйте ему все выполненные Вами задания. После этого сохраните все программы и выйдите из среды PascalABC.***

Лабораторная работа 2 Зарезервированные слова, использование зарезервированных слов в программе

1. ***Структура программы на языке Паскаль***

Программа на языке Паскаль имеет следующую структуру:

program <имя>; <блок описания данных> **Begin** <оператор 1>; <оператор 2>; <оператор N> **end.**

- заголовок программы;

- начало исполняемой части программы;
- текст («тело») программы;
- конец программы.

Обратите внимание:

(1) операторы (команды) отделяются друг от друга ТОЧКОЙ С ЗАПЯТОЙ;

(2) *перед* словом **END** и *после* слова **BEGIN** точка с запятой не ставится;

(3) *в конце* программы после слова **END** обязательно ставится точка.

2.1. Резервирование памяти для работы или описание переменных

Самое главное действие, которое выполняет компьютер, - это запись информации в память. Суть действия проста: необходимо указать компьютеру, КУДА записывать и ЧТО записывать.

Для начала Вы можете воспользоваться следующими стандартными (т.е. «понимаемыми» компилятором) типами для описания переменных:

Integer - для целых чисел в диапазоне от -2147483648 до 2147483647;
byte - для целых чисел в диапазоне от 0 до 255;
word - для целых чисел в диапазоне от 0 до 65535;
real - для вещественных чисел;
Char - для хранения *ОДНОГО* символа;
String - для строк текста, содержащих не более 255 символов.

var num : **integer;**	Описание блока памяти с именем **num.** в котором можно хранить *ЦЕЛОЕ* число (от -2147483648 до 2147483647.). Слово **«integer»** (целое) после двоеточия указывает *ТИП* переменной (блока памяти) и означает, что требуется зарезервировать блок памяти для хранения именно целого числа. В таком случае говорится, что «переменная num является переменной целого типа» или «переменной типа integer (читается как «интедже»)». Имя блока памяти - вещь обязательная. Ведь в программе Вам придется выполнять различные действия с *СОДЕРЖИМЫМ* этого блока. А чтобы указать компьютеру, с каким именно блоком Вы хотите работать, требуется *ИМЯ.*

2.2. Запись данных в память, или оператор присваивания

В предыдущем пункте Вам фактически предложено работать с данными трех типов: целыми, вещественными и строковыми. Следует запомнить:

(1) целые числа в программах записываются так же, как принято в математике. Например: 345 , -12222 ;

(2) вещественные числа могут записываться двумя способами: первый - аналогичный математической записи: -123.234, 92929.3456 (обратите внимание, что здесь используется десятичная ТОЧКА, а не запятая), второй - в так называемом «плавающем» виде (правильное название: «десятичное число с плавающей точкой»). Например, число 1,23»10^{12} может быть записано следующим образом: **<u>1.23E+12</u>** Здесь «E+12» читается как *«умножить на десять в двенадцатой степени»;*

(3) строки представляют собой произвольный набор символов, заключенный в АПОСТРОФЫ:

'Какой чудесный был пирог. Я от него ... 2+2=5'

✍*Оператор присваивания* предназначен для указания компьютеру *ЗАПИСИ ДАННЫХ В КОНКРЕТНЫЙ БЛОК ПАМЯТИ.* Общий вид оператора:<имя блока памяти>:=<значение выражения>

☞*Обратите внимание на знак присваивания - он состоит из двух значков: двоеточия и равно (:=), которые записываются друг за другом без пробелов!*

Пример 1.

```
program examplel; {заголовок программы}
var num : integer;  {резервирование блока памяти для хранения целого числа}
day : byte;         {резервирование блока памяти для хранения целого числа от 0 до 255}
name : String;      {резервирование блока памяти для хранения строки до 255 символов}
begin
num:=355;           {запись в блок num числа 355}
day :=31;           {запись в блок day числа 31}
name:='Скорпион'    {запись в блок пате слова «Скорпион»}
end.
```

Как следует из определения оператора присваивания, в правой части оператора присваивания может быть и *ВЫРАЖЕНИЕ.* В этом случае в блок памяти *(переменную)* записывается *ЗНА ЧЕНИЕ* выражения.

2.3. Вывод данных на экран дисплея

Примеры программ, приведенных выше, иллюстрируют команду присваивания, но для Вас их выполнение на компьютере бессмысленно. Ведь программа записывает данные в память компьютера и заканчивает работу. Вы ничего не увидите. Чтобы увидеть, в языке есть процедура *вывода* **WriteLn.**

Задание 1. Считайте с диска файл **lab2_lB.pas.** Разберитесь в тексте программы. Выполните предложенные там задания. *В Помощи выясните возможности редактирования текста в данном редакторе (раздел «Описание интегрированной среды»).*

Задание 2. Считайте с диска файл **lab2_lc.pas.** Разберитесь в тексте программы. Выполните предложенные там задания. Запишите в тетрадь команду подключения стандартной библиотеки **CRT,** название и назначение процедуры **CLRSCR.**

Задание 3. Считайте с диска файл **lab2_ld.pas.** Выполните предложенные там задания.

Задание 4. Считайте с диска файл **lab2_le.pas.** Разберитесь в тексте программы и выполните предложенные там задания. Выпишите в тетрадь различие между командами WRITE и WRITELN. Запишите в тетрадь назначение процедуры TextColor.

2.4. Форматированный вывод информации

В окне можно поместить строго определенное количество символов. Обычно это 25 строк по 80 символов в строке. Место для одного символа принято называть «позицией». Таким образом, в окне имеется 25x80=2000 позиций для размещения текста.

При использовании процедур вывода **WriteLn (Write)** Вам предоставляется возможность самим определять количество позиций, которое отводится для данного вида информации. Вывод данных с указанием числа позиций принято называть *«форматированный вывод».*

Задание 5. Откройте Помощь, найдите там раздел «Справочник по языку», там - подраздел «Ввод и вывод. Форматы вывода». Прочтите статью. Выясните особенности форматного вывода символов, целых и вещественных чисел.

Задание 6. Считайте с диска файл **lab2_lf.pas.** Разберитесь в тексте программы. Уточните способы форматированного вывода целых чисел. Выведите целое число так, чтобы перед ним было соответственно 20, 30 и 40 пробелов.

Задание 7. Считайте с диска файл **lab2_lg.pas.** Разберитесь в тексте программы. Уточните способы форматированного вывода строк. Измените программу так, чтобы выводились еще две строки: первая - чтобы было всего 2 пробела перед текстом, вторая - в правой части окна.

Задание 8. Считайте с диска файл **lab2_lh.pas.** Разберитесь в тексте программы. Отметьте в тетради способы форматированного вывода вещественных чисел. Измените программу так, чтобы выводились еще два вещественных числа: первое - 133.567, чтобы было 20 пробелов перед числом и 5 знаков после запятой; второе - 79954.23451 так, чтобы точка числа была под точкой предыдущего числ, а три знака после запятой.

Задание 9. Считайте с диска файл **lab2_li.pas** и выполните предложенное там задание.

Задание 10. Считайте с диска файл **lab2_lj.pas** и выполните предложенное там задание.

Задание 11. В Помощи (Стандартные модули → Модуль СП) выясните, как можно задать (и можно ли) вид шрифта и его размер. Напишите программу вывода вашей фамилии, а также имени и отчества жирным шрифтом, 24 кеглем и разными цветами. Расположите их в центре окна.

☝Пригласите преподавателя и продемонстрируйте ему все выполненные Вами задания. После этого сохраните все программы и выйдите из среды PascalABC.

Лабораторная работа 3 Операторы языка Pascal
Алгоритмические конструкции: условный оператор

Вы познакомились с командой присваивания, командами ввода и вывода информации. Это так называемые простые операторы. Для того, чтобы было удобно записывать алгоритмы, используются алгоритмические конструкции. К таким алгоритмическим конструкциям относятся:

- *составной оператор;*
- *условные операторы;*
- *циклы.*

1. Составной оператор

Составной оператор представляет собой совокупность последовательно выполняемых операторов, заключенных в так называемые операторные скобки **BEGIN** и **END. BEGIN**

<оператор 1>;
<оператор 2>;
<оператор N> END;

Составной оператор используется в тех случаях, когда в соответствии с правилами построения конструкций языка можно использовать один оператор, а выполнить требуется несколько действий.

Отдельные операторы внутри составного оператора отделяются друг от друга точкой с запятой. Однако перед ключевым словом END точку с запятой можно не ставить. Использование составного

оператора будет продемонстрировано ниже.

2. Условные операторы

2.1. Команда ветвления

Очень часто возникают ситуации, при которых необходимо сначала проверить некоторое условие, а затем, в зависимости от результатов проверки, выполнить какие-то действия. Такой алгоритм в общем виде можно записать так:

Если <условие> то <действие>; (пример **1)**

или

Если <условие> то <действие 1>

иначе <действие 2>; (пример 2)

Такая алгоритмическая конструкция называется развилкой (ветвлением, условным оператором) и в языке Паскаль записывается следующим образом: **IF <условие> THEN**

<оператор 1> ELSE

<оператор 2> На месте операторов могут быть записаны простые операторы или составные. Условия записываются, как правило, в виде логических выражений, построенных с помощью следующих отношений:

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

В качестве примера рассмотрим программу, которая определяет максимальное из двух чисел:

```
PROGRAM MAXIMUM;
VAR X,Y: INTEGER;
BEGIN
WRITE('Введите два целых числа:');
READLN(X,Y);
WRITE('Максимальное из двух чисел равно ');
IF X>Y THEN WRITE(X)
ELSE WRITE(Y)
END.
```

Обратите внимание, что ключевые слова THEN и ELSE отделяются от следующих за ними операторов пробелами (если вы поставите точку с запятой, то это будет ошибкой!).

Если в команде ветвления после слов THEN или ELSE требуется выполнить не одну, а несколько команд, то они заключаются в операторные скобки BEGIN - END. Например:

```
IF D>=0 THEN
BEGIN
X1:=((-B+sqrt(D))/(2*A);
X2:=((-B-sqrt(D))/(2*A);
END
ELSE WRITELN('flencTBHTenbHbix корней нет');
```

Допустима и сокращенная форма команды ветвления:

IF <условие> THEN <оператор>;

Например: **IF D<0 THEN WRITE(**Делить на 0 нельзя');

3. Сложные условия

3.1. Что такое True и False

Все приведенные выше примеры содержат *простые* условия. Под *условием* мы будем понимать какое-либо выражение, о котором (после определения всех значений входящих в него переменных) можно сказать, что оно либо ИСТИННО (по-английски TRUE - читается как ТРУ-У-У), либо ЛОЖНО (по-английски FALSE - читается как ФЭ-Э-ЛС). Например: If a>b then ... - здесь условие a>b является True только тогда, когда значение переменной a больше значения переменной Ь. Это же условие можно записать по другому: a-b>0 или b<a или b-a<0. И во всех приведенных случаях выражение является истинным (True) тогда и только тогда, когда значение переменной a больше значения переменной b.

Выражения, значения которых могут равняться True или False, принято называть ЛОГИЧЕСКИМИ. Как правило, в логических выражениях всегда присутствует знаки сравнения: <, <=, =, >=, >, о.

Попробуйте сами определить, когда приведенные ниже значения выражений являются True, а когда - False:

a=b a>=b a+1=b-1 s*s<=121 abs(x-x0)<e

3.2. Логический тип данных

Учитывая, что в языке Паскаль имеются выражение, принимающие конкретные значения (True и False), был введен специальный тип данных. Этот тип данных принято называть ЛОГИЧЕСКИМ типом данных. Переменные данного типа описываются в блоке описания VAR с помощью ключевого слова BOOLEAN. Например:

```
var   X : integer;
      F : boolean;
```

Значением переменной f может быть либо True, либо False. True и False известны компилятору языка и Вы можете использовать их явно в программе. Например:

```
F:=True;
if F then WriteLn('А значение-то ИСТИННО!!!');
```

Как еще можно использовать переменные логического типа? Учитывая, что они принимают точно такие же значения, как и ЛОГИЧЕСКИЕ выражения, их можно использовать ВМЕСТО этих выражений (как предложено в вышеприведенном примере). Возможные варианты:

```
Write('Сколько Вам лет');
ReadLn(Year);
Write('А сколько лет Вашему другу');
ReadLn(Year_Friend);
F:=(Year+5<Year_Friend);
if F then WriteLn(' Ого, Ваш друг старше Вас более чем на 5 лет ?!');
```

Достаточно часто встречаются ситуации, когда одно и тоже условие используется в программе более одного раза. В этом случае приведенный способ использования ЛОГИЧЕСКОЙ переменной позволит Вам, во-первых, сократить текст программы, а, во-вторых, облегчит ее модификацию. Представляете, если одно и то же условие (логическое выражение) встречается в Вашей программе 10 раз, а потом вдруг выяснилось, что там ошибка...

3.3. Сложные условия

Давайте решим простую задачку:

Составить программу, которая определит возраст человека.

Решение

```
Program YEARS;
Uses Crt;
Const year1 =2007;     {Текущий год}
var Old,year:integer;
Begin
Write('В каком году Вы родились:');
ReadLn(year);
Old:=year1-year;   {Вычислили возраст}
If (Old<0) or (Old>150) then WriteLn('Не врите, столько не живут')
else WriteLn('ВаМ полных ',Old,' лет'); end.
```

Обратите внимание на СЛОЖНОЕ условие в операторе If. Слово OR (читается как О-О-Р) переводится как ИЛИ и называется ЛОГИЧЕСКОЙ ОПЕРАЦИЕЙ «ИЛИ». Кроме этой логической операции имеется еще ряд операций, которые приведены в таблице 1. В данной таблице в качестве F1 и F2 выступают логические выражения.

Например:

If (a>3) and (a<5) then ... - условие истинно тогда и только тогда, когда истины оба выражения: a>3 И a<5. Во всех других ситуациях - False.

If (b+1>=4) or (c='ку-ку') then ... - условие True тогда и только тогда, когда True одно из логических выражений, соединенных OR.

☞*Обратите внимание, что при записи сложного логического выражения простые выражения заключаются в скобки!*

Таблица 1 **Логические операции**

Логические операции	Название операции	Запись	Результат операции
NOT	Логическое	NOT F1	Логическое значение, противоположное F1
AND	Логическое «И»	F1 AND F2	Логическое значение True, если F1 и F2 равны Тгае, и False во всех других случаях
OR	Логическое «ИЛИ»	F1 0R F2	Логическое значение Тгае, если хртябы одно из значений F1 или F2 равно Тгае, и False, если F1 и F2 равны False
XOR	Логическое исключающее «ИЛИ»	F1 XOR F2	Логическое значение Тгае, если F1 и F2 различны, и False, если они равны.

4. Оператор выбора CASE

С помощью этого оператора можно выбрать вариант из любого количества вариантов. Структура этого оператора в языке Паскаль:

case S of
c1: instM;
c2: instr2; …
cN: instrN; else instr end;

В этой структуре:

S - некоторое выражение одного из перечислимых типов (т.е. принимает целое или символьное значение);

instrl, instr2,... instrN - операторы, из которых выполняется тот, с константой которого (c1.xN) совпадает значение выражение S;

instr - оператор, который выполняется, если значение выражения S не совпадает ни с одной из констант cl,... ,cN.

Ветвь оператора else является необязательной.

☞*Обратите внимание, что конструкция CASE завершается словом END!!!*

5. Вопросы и задания для самоконтроля

(1) Что такое «операторные скобки»?
(2) Когда используется составной оператор?
(3) Нужно ли ставить точку с запятой перед End?
(4) Найдите ошибки в записи следующих операторов ветвления:

а) if (a+b) then a:=2*a;
б) if a+b=0 and a-b=-3 then a:=5;
в) if a>0 then else a:=a*a;
г) if a<=0 else a:=a-2;
д) If (a=5) and (a-5)=3 then WriteLn(a);
е) if a>0 then begin a:=a*a; Writeln(a);
ж) if (a<0 and a>l) then a:=a+999;
з) if (a<0) or (a>l) then else WriteLn('ошибки');

(5)В программе описаны две переменные целого типа а и Ь; им заданы соответственно значения 5 и 7. Определите значения следующих логических выражений:

а) (a+b>12) or (Ь-a<2);
б) (a>=5) and (b>=7);
в) (a+1=6) xor (b=7);
г) not (a*b>30);
д) (a=5) and (b-5>3);
е) (a*b>35) or (b/a<l);
ж) (a>=5) or (b>7);
з) (a+b>20) and (a*b<30);

(6) Можно ли в операторе выбора в качестве константы использовать вещественное число?
(7) Обязательно ли в операторе выбора использовать конструкцию Case?
(8) Даны несколько строк из различных конструкций Case. Какие из этих строк недопустимы и почему:

'а'..'я': writeln('русская буква');
А..Я: writeln('большая русская буква');
'0'..'7': writeln('число от 0 до 7');

✌Пригласите преподавателя, продемонстрируйте ему выполненные задания и ответьте на его вопросы.

Лабораторная работа 4. Синтаксис операторов: присваивания, ввода – вывода

1. *Оператор присваивания*

Наиболее простым и часто используемым оператором языка является ***оператор присваивания:*** <переменная> **: =** <выражение>**;**

Выражение – это формула для вычисления значения. Она образуется из операндов, соединенных знаками операций и круглыми скобками. В качестве операндов могут выступать переменные, константы, указатели функций.

Тип переменной в левой части оператора присваивания обычно должен совпадать с типом значения выражения в правой части. Возможны случаи несовпадения типов, например, когда слева переменная вещественного типа, а справа выражение целого типа.

Выражения являются составной частью операторов.

В Паскале приоритеты выполнения операций следующие (в порядке убывания):

- одноместный минус;
- операция **NOT**;
- операции типа умножения ;
- операции типа сложения;
- операции сравнения (отношения).

Одноместный минус применим к операндам арифметического типа. Операция **NOT** – к операндам логических и целых типов. Если в одном выражении несколько операций одного приоритета, то они выполняются, начиная слева. Приоритеты можно изменить, поставив скобки. В логических выражениях необходимы скобки во избежание конфликта типа по приоритету.

Например, если в выражении **… (X > 5) AND (Y > 10) …** не поставить скобки, то будет синтаксическая ошибка, так как приоритет операции **AND** выше приоритета операций сравнения >.

<операции типа умножения> **:: = * | / | div | mod | and**

<операции типа сложения> **:: = + | – | or | xor**

<операции сравнения> **:: = = | <> | < | > | <= | >= | in**

Операции сравнения применимы для всех стандартных простых типов. Причем в одном выражении возможно использование операндов различных типов. Результат сравнения всегда имеет логический тип.

Например, **(5 + 6) < (5 – 6) = TRUE** в результате даст **FALSE**, а **NOT(8.5 < 4)** будет равно **TRUE**.

Сравнение строк символов выполняется слева направо посимвольно. Более короткие строки дополняются пробелами справа.

2. *Синтаксис операторов ввода-вывода.*

Решим задачу, прокомментировав каждое свое действие в фигурных скобках. Напомним, что комментарий не воспринимается компьютером, а нам он нужен для того, чтобы лучше понять как работает программа.

<u>Задача</u>. Напишите программу, которая бы очищала экран и вычисляла произведение двух чисел, вводимых пользователем.

```
Program Proizv2;
Uses
  Crt;{Подключаем модуль Crt}
Var
  number1, {переменная, в которой будет содержаться первое число}
  number2, {переменная, в которой будет содержаться второе число}
  rezult {переменная, в которой будет содержаться результат}
  : integer;
Begin
  ClrScr;{Используем процедуру очистки экрана из модуля Crt}
  Write ('Введите первое число ');
  {Выводим на экран символы, записанные между апострофами}
  Readln (number1);
  {Введенное пользователем число считываем в переменную number1}
```

```
  Write ('Введите второе число ');
  {Выводим на экран символы, записанные между апострофами}
  Readln (number2);
  {Введенное пользователем число считываем в переменную number2}
  rezult := number1 * number2;
  {Находим произведение введенных чисел и присваиваем переменной rezult}
  Write ('Произведение чисел ', number1, ' и ', number2, ' равно ', rezult);
  {Выводим на экран строчку, содержащую ответ задачи}
  Readln;{Процедура задержки экрана}
End.
```

Ответьте на вопросы:

1. Почему программу назвали Proizv2?
2. Зачем в раздел Uses поместили модуль Crt?
3. Какое назначение переменных number1, number2, rezult?
4. Какой тип у этих переменных? что это значит?
5. Если присвоить переменным number1 и number2 соответственно значение 5 и 7, то какую строчку выдаст компьютер при исполнении последней процедуры Write? Запишите ее в тетрадь.
6. В каких строчках у пользователя запрашиваются значения переменных?
7. В какой строчке происходит умножение чисел?
8. Что делает оператор присваивания в этой программе?

Задание. Измените программу так, чтобы она запрашивала у пользователя еще одну переменную и выводила результат произведения трех чисел.

Лабораторная работа 5. Синтаксис безусловного и условного переходов

До сих пор Вы использовали линейные алгоритмы, т.е. алгоритмы, в которых все этапы решения задачи выполняются строго последовательно. Сегодня Вы познакомитесь с разветвляющимися алгоритмами.

Определение. Разветвляющимся называется такой алгоритм, в котором выбирается один из нескольких возможных вариантов вычислительного процесса. Каждый подобный путь называется ветвью алгоритма.

Признаком разветвляющегося алгоритма является наличие операций проверки условия. Различают два вида условий - простые и составные.

Простым условием (отношением) называется выражение, составленное из двух арифметических выражений или двух текстовых величин (иначе их еще называют операндами), связанных одним из знаков:

< - меньше, чем...
> - больше, чем...
<= - меньше, чем... или равно
>= - больше, чем... или равно
<> - не равно
= - равно

Например, простыми отношениями являются следующие:

x-y>10; k<=sqr(c)+abs(a+b); 9<>11; ‘мама’<>‘папа’.

В приведенных примерах первые два отношения включают в себя переменные, поэтому о верности этих отношений можно судить только при подстановке некоторых значений:

- если x=25, y=3, то отношение x-y>10 будет верным, т.к. 25-3>10
- если x=5, y=30, то отношение x-y>10 будет неверным, т.к. 5-30<10

Проверьте верность второго отношения при подстановке следующих значений:

1. k=5, a=1, b=-3, c=-8
2. k=65, a=10, b=-3, c=2

Определение. Выражения, при подстановке в которые некоторых значений переменных, о нем можно сказать истинно (верно) оно или ложно (неверно) называются булевыми (логическими) выражениями.

Определение. Переменная, которая может принимать одно из двух значений: True (правда) или False (ложь), называется булевой (логической) переменной. Например,

```
    K:=True;
Flag:=False;
Second:=a+sqr(x)>t
```

Задание. Наберите текст программы. Протестируйте программу со следующими значениями переменных и сделайте вывод.

1. x=23, y=5;
2. x=-5, y=15;
3. x=8, y=8.

Каждая программа, насколько это возможно, должна осуществлять контроль за допустимостью величин, участвующих в вычислениях. Здесь мы сталкиваемся с разветвлением нашего алгоритма в зависимости от условия. Для реализации таких условных переходов в языке Паскаль используют операторы If и Else, а также оператор безусловного перехода Goto.

Рассмотрим оператор If.

Для нашей задачи нужно выполнить следующий алгоритм:

Если x>=y, **то** вычислить значение квадратного корня, **иначе** выдать на экран сообщение об ошибочном введении данных.

Запишем его с помощью оператора If. Это будет выглядеть так.

```
    if x>=y
 then
   Koren:=Sqr(x-y)
 else
```

```
write (‘Введены недопустимые значения переменных‘);
```

Теперь в зависимости от введенных значений переменных x и y, условия могут выполняться или не выполняться.

Условный оператор работает по следующему алгоритму.

Сначала вычисляется значение логического выражения, расположенного за служебным словом IF. Если его результат ***истина***, выполняется <оператор 1>, расположенный после слова THEN, а действия после ELSE пропускаются; если результат***ложь,***то, наоборот, действия после слова THEN пропускаются, а после ELSE выполняется <оператор 2>.

Рассмотренный выше оператор if осуществляет переход к выполнению соответствующего оператора в зависимости от выполнения условия или предложенного выбора. Однако в практике программирования задач возникает необходимость безусловного перехода для выполнения нужной последовательности операторов. Об использовании оператора безусловного перехода читайте в параграфе «Оператор GOTO».

1. Составной оператор.

Управляющая структура if может показаться негибкой, так как выполняемые действия могут быть описаны только одним оператором. Иногда может потребоваться выполнение последовательности операторов. В этом случае хотелось бы заключить всю последовательность в воображаемые скобки. В Паскале предусмотрен этот случай.

Если в качестве оператора должна выполниться серия операторов, то они заключаются в операторные скобки begin-end. Конструкция Begin ... End называется составным оператором.

Определение. Составной оператор - объединение нескольких операторов в одну группу. Группа операторов внутри составного оператора заключается в операторные скобки (begin-end).

Символ “;” в данном случае разделяет оператор присваивания S:=0 и пустой оператор.

Пустой оператор не влечет никаких действий и в записи программы никак не обозначается.

Например, составной оператор

```
begin
end.
```

включает лишь один пустой оператор.

Если Вы обратили внимание, программа на языке Паскаль всегда содержит один составной оператор - раздел операторов программы.

Внимание! Перед служебным словом Else разделитель (точка с запятой) не ставится.

Отметим, что большинство операторов в программах на языке Паскаль заканчиваются точкой с запятой, но после некоторых операторов точка с запятой не ставится. Сформулируем общие правила употребления точки с запятой:

1. Каждое описание переменной и определение константы заканчиваются точкой с запятой.
2. Каждый оператор в теле программы завершается точкой с запятой, если сразу за ним не следуют зарезервированные слова End, Else, Until.
3. После определенных зарезервированных слов, таких, как Then, Else, Var, Const, Begin, никогда не ставится точка с запятой.

Задача. Составить программу, которая, если введенное число отрицательное меняет его на противоположное.

```
    Program Chisla;
Var
 x : integer; {вводимое число}
Begin
 writeln(‘Введите число ‘); {вводим целое число}
 readln(x);
 if x<0
   then
     x:=-x;
     writeln (x);
     readln;
End.
```

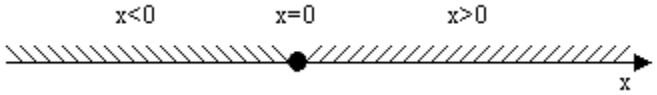
2. Вложенные условные операторы.

Вложенные условные операторы. Решение задач. При решении задач часто приходится рассматривать не два, а большее количество вариантов. Это можно реализовать, используя несколько условных операторов. В этом случае после служебных слов Then и Else записывается новый условный оператор. Рассмотрим пример.

Задача. Вычислить значение функции:

$$y=\begin{cases} x-12, x>0 \\ 5, x=0 \\ x^2, x<0 \end{cases}$$

Для решения этой задачи рассмотрим координатную прямую, на которой отметим промежутки, на которые разбиваются все значения переменной x.



Начнем записывать условный оператор:

если x>0

то

вычислить y по формуле y=x-12

иначе

Что же должно выполниться в случае иначе? На эту ветку оператора попадают все не положительные значения x. Если бы для этих чисел нужно было бы выполнить один и тот же оператор (или группу операторов), то проблемы бы не стояло. Но нам нужно этот промежуток разделить еще на две части (отрицательные и ноль), и для части выполнить свой оператор. Поэтому ветка Иначе будет содержать еще один условный оператор и наш вложенный условный оператор будет иметь вид.

Фрагмент программы для решения этой задачи будет выглядеть так:

```
    if x>0
 then
  y := x-12
 else
  if x=0
   then
    y := 5
   else
    y := sqr(x);
```

Итак, когда оператор if появляется внутри другого оператора if, они считаются вложенными. Такое вложение используется для уменьшения числа необходимых проверок. Этот метод часто обеспечивает большую эффективность, однако одновременно он уменьшает наглядность программы. Не рекомендуется использовать более одного-двух уровней вложения if. За вторым уровнем вложения становится трудно восстановить последовательность проверки условий каждым условным оператором.

Если часть else используется во вложенных if, то каждое else соответствует тому if, которое ему непосредственно предшествует. Таким образом, при определении последовательности выполнения фрагментов нет двусмысленности.

Рассмотрите еще один пример.

Задача. Даны целые числа a, b, c. Если a ≤ b ≤ c, то все числа заменить их квадратами, усли a>b>c, то каждое число заменить наибольшим из них, в противном случае сменить знак каждого числа.

Для решения этой задачи перепишем условие задачи следующим образом:

a:=a2, b:=b2, c:=c2, если a ≤ b ≤ c

b:=a, c:=a, если a>b>c

a:=-a, b:=-b, c:=-c, в остальных случаях

Программа для решения этой задачи представлена ниже.

```
    Program Example3;
Var
 a, b, c : integer;
```

```
Begin
  writeln(‘Введите числа a, b, c’);
  readln(a,b,c);
  if (a<=b) and (b<=c)
    then
      begin
        a:=sqr(a);
        b:=sqr(b);
        c:=sqr(c);
      end
    else
      if (a>b) and (b>c)
        then
          begin
            b:=a;
             c:=a;
          end
        else
          begin
            a:=-a;
            b:=-b;
            c:=-c;
          end
  writeln(a,b,c);
  readln;
End.
```

Задание. Найдите в этой программе (если есть) условный оператор, вложенный условный оператор, составной оператор, булево условие.

Внимание! Подчеркнем еще раз один тонкий момент: поскольку каждый из операторов может быть оператором любого типа (в том числе и условным), и в то же время не каждый из "вложенных" условных операторов может иметь часть else, то возникает неоднозначность трактовки условий. Turbo-Pascal решает эту проблему таким радикальным способом: любая встретившаяся часть else соответствует ближайшей к ней части then.

Лабораторная работа 6. Составной оператор

Составной оператор – это последовательность произвольных операторов программы, заключенная в операторные скобки – зарезервированные слова begin ... end. Операторы разделяются точкой с запятой (;).

Исполняемая часть программы является составным оператором такого рода.

Составной оператор служит, в первую очередь, для того, чтобы несколько операторов синтаксически объединить в один. Это часто требуется там, где нужно выполнить несколько операторов, когда допустим лишь один. Понятие составного оператора позволяет с помощью команд begin и end объединить несколько операторов и рассматривать их с точки зрения синтаксиса как один оператор. На характер операторов, входящих в составной оператор, не накладывается никаких ограничений. Среди них могут быть и другие составные операторы.

При исполнении операторов точка с запятой служит разделителем для двух операторов. Точкой с запятой перед заключительным end можно пренебречь.

```
begin begin
read(i); read(i);
write(i); write(i)
end. end.
```

Обе записи верны, поскольку можно считать, что между write(i); и end находится пустой оператор. **Пустой оператор**– оператор, который не выполняет никаких операций и ничего не изменяет в данных и в программе. Пустому оператору соответствует отсутствие записи на том месте, где по правилам должен быть какой-нибудь оператор. После него можно ставить символ точки с запятой, например:

```
A := B;
R := 2;
K := 7.2;
```

У начинающих программистов часто возникает вопрос: где правильно поставить знак точки с запятой? Чтобы на него ответить, обратимся к обычному естественному языку. В любом перечне элементов между ними ставится запятая, например:

А, В, С, Д.

Если эти элементы объединить в одну группу, заключив их в круглые скобки (А, В, С, Д), то запятая ставится опять-таки между элементами: после открывающей и перед закрывающей скобками запятая не указывается. Если эта группа элементов входит в состав другой группы, то запятая ставится и между группами, например:

((А, В, С, Д), (К, М), Е, (Х, У))

Подобная система введена и в языке Паскаль, только в нем роль круглых скобок выполняют операторные скобки BEGIN – END, вместо запятой ставится точка с запятой, а вместо элементов – операторы.

Контрольное задание

Вариант 1. Составьте программу, которая умеет решать квадратные уравнения (по заданным коэффициентам уравнения А, В, С находит корни уравнения. Рассматриваются различные случаи: нет корней, есть один корень, два различных корня).

Вариант 2. По заданной стороне[4] квадрата и радиусу круга выяснить, поместится ли:

а)круг в квадрат; б) квадрат в круг?

Результаты вывести на экран.

Лабораторная работа 7. Цикл со счетчиком

for <переменная-счетчик> := <начальное значение> ***to*** *<конечное значение> do*

где <***переменная-счетчик***> — переменная целоисчисленного типа (byte, integer);

<***начальное значение***> — целое число, которое будет начальным значением переменной-счетчика;

<***конечное значение***> — целое число, которое должно быть больше <начального значения>.

В данном цикле переменная счетчик будет увеличиваться на единицу каждый раз при выполнении тела цикла, пока не достигнет конечного значения включительно. Тело цикла – оператор после служебного слова do. Если необходимо выполнить несколько операторов, то их замыкают между begin и end; (с точкой с запятой).

Данный цикл выведет 10 раз (первоначальное i равно 1, конечное равно 10) на экран слово «Привет!»:

for i := 1 to 10 do writeln('Привет!');

Следующий цикл выведет 10 раз слово «Привет!» и посчитает сумму чисел от 1 до 10:

```
for i := 1 to 10 do
begin
writeln('Привет!');
sum := sum + i;
end;
```

Два выполняемых в теле цикла оператора (writeln и операция накопления суммы) заключены между begin и end;.

При необходимости можно воспользоваться следующей конструкцией:

for <переменная-счетчик> := <начальное значение> ***downto*** *<конечное значение> do*

Действие этого цикла равнозначно предыдущему за одним исключением: параметр **downto** дает команду процессору уменьшать значение переменной-счетчика на единицу при каждом проходе тела цикла (а не увеличивать его, как в случае с параметром **to**). То есть начальное значение всегда должно быть выше конечного значения.

```
for i := 10 downto 1 do
begin
writeln('Привет!');
sum := sum + i;
end;
```

Результат этой конструкции будет аналогичен предыдущему, пользователь не заметит никаких различий. Но математика алгоритма немного другая. Если при параметре **to** в переменную sum поступает цепочка: 1 + 2 + 3 + 4 + … + 9 + 10, то при **downto** это будет: 10 + 9 + 8 + 7 + … + 3 + 2 + 1.

Контрольное задание

Задание 1. Считайте с диска файл с именем Iab7_1.pas. Содержащаяся в нем программа 15 раз печатает на экране слово «Халва...» (убедитесь в этом, запустив программу). Модифицируйте программу так, чтобы:

а) слово печаталось не 15, а 10 раз;
б) слова печатались в одну строку;
в) перед первым словом печаталось слово «Начало», а после последнего – слово «Конец»;
г) каждое слово печаталось с новой строки, и между ними была пустая строка;
д) перед каждым словом «Халва...» печатался его порядковый номер (значение переменной i).

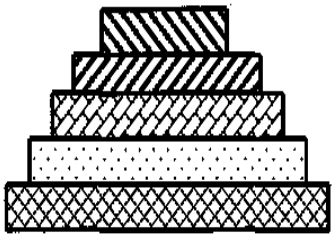
Задание 2 . В файле Iab7_2.pas находится программа печати таблицы значений функции f(x) = sin(x) при x = 0, 0.1, 0.2,..., 1. Модифицируйте программу так, чтобы:

а) печатался заголовок;
б) печатались еще и значения аргумента x;
в) печатались значения при x = 0, 0.2, 0.4,..., 4;
г) то же самое печаталось в обратном порядке.

Задание 3. Смоделируйте на экране равномерное прямолинейное движение «материальной точки» (небольшой окружности):

а) по горизонтали;
б) по вертикали;
в) «с горки».

Задание 4. Используя конструкцию цикла со счетчиком, самостоятельно



составьте программу рисования детской игрушечной пирамидки (вид сбоку).

☝*Пригласите преподавателя и продемонстрируйте ему все выполненные Вами задания. После этого сохраните все программы и выйдите из среды PascalABC. Будьте готовы ответить на вопросы преподавателя.*

Лабораторная работа 8. Цикл с предусловием

В случае оператора цикла Паскаля с постусловием входящая в него последовательность операторов заведомо будет выполняться хотя бы один раз. Между тем довольно часто встречаются такие циклические процессы, когда число повторений цикла тоже неизвестно заранее, но при некоторых значениях исходных данных предусмотренные в цикле действия вообще не должны выполняться, и даже однократное выполнение этих действий может привести к неверным или неопределенным результатам.

Пусть, например, дано вещественное число M. Требуется найти наименьшее целое неотрицательное число k, при котором 3 k> M. Эту задачу можно решить по следующему алгоритму: предварительно положить y=1 и k=0; затем в цикле домножать значение y на 3 и увеличивать значение k на 1 до тех пор, пока текущее значение y впервые окажется больше значения M. На первый взгляд, здесь можно воспользоваться оператором цикла с постусловием:

Пример кода оператора цикла Паскаля с постусловием

```
    y:=1; k:=0;
Repeat
  y:=y*3;
  k:=k+1;
Until y> M;
```

Однако нетрудно убедиться в том, что при M<1 будет получен неправильный результат k=1, тогда как должно быть получено k=0: в этом случае предварительно сформированное значение k=0 является окончательным результатом и действия, предусмотренные в цикле, выполняться не должны.

Для задания подобного рода вычислительных процессов, когда число повторений цикла заранее неизвестно и действия, предусмотренные в цикле, могут вообще не выполняться, и служит оператор цикла с предусловием. Этот оператор цикла имеет в Паскале следующий вид:

```
    While B do S,
```

где **while** (пока), **do** (делать, выполнять) – служебные слова, **B** – логическое выражение, **S** – оператор. Здесь оператор S выполняется ноль или более раз, но перед каждым очередным его выполнением вычисляется значение выражения B, и оператор S выполняется только в том случае, когда значение выражения B true. Выполнение оператора цикла завершается, когда выражение B впервые принимает значение false. Если это значение выражение B принимает при первом же его вычислении, то оператор S не выполнится ни разу.

В рассматриваемой нами задаче правильное значение k при любом значении M может быть получено следующим образом:

Пример кода оператора цикла Паскаля с предусловием

```
    y:=1; k:=0;
While y<=M do
Begin
  y:=y*3;
  k:=k+1;
End;
```

Оператор цикла Паскаля с предусловием можно считать наиболее универсальным – с использованием таких операторов можно задать и циклические процессы, определяемые операторами цикла с параметром и постусловием.

Отметим **отличия и особенности хорошего стиля работы** с рассмотренными циклическими операторами.

Цикл с предусловием While (пока условие истинно)	**Цикл с постусловием Repeat (до истинности условия)**
1. До начала цикла должны быть сделаны начальные установки переменных, управляющих условием цикла, для корректного входа в цикл	
2. В теле цикла должны присутствовать операторы, изменяющие переменные условия так, чтобы цикл через некоторое число итераций завершился	

3. Цикл работает пока условие истинно (пока True)	3. Цикл работает пока условие ложно (пока False)
4. Цикл завершается, когда условие становится ложным (до False)	4. Цикл завершается, когда условие становится истинным (до True)
5. Цикл может не выполниться ни разу, если исходное значение условия при входе в цикл False	5. Цикл обязательно выполнится как минимум один раз
6. Если в теле цикла требуется выполнить более одного оператора, то необходимо использовать составной оператор	6. Независимо от количества операторов в теле цикла, использование составного оператора не требуется
Цикл со счетчиком (с параметром) For	
• Начальная установка переменной счетчика цикла до заголовка не требуется	
• Изменение в теле цикла значений переменных, стоящих в заголовке не допускается	
• Количество итераций цикла неизменно и точно определяется значениями нижней и верхней границ и шага приращения	
• Нормальный ход работы цикла может быть нарушен оператором goto или процедурами Break и Continue	
• Цикл может не выполниться ни разу, если шаг цикла будет изменять значение счетчика от нижней границы в направлении, противоположном верхней границе	

Оператор, который выполняется в цикле, сам может быть циклом. Это относится ко всем видам циклов. В результате мы получаем **вложенные циклы**. Механизм работы вложенных циклов удобнее всего рассмотреть на примере вложенных циклов с параметром. Пусть нам нужно описать работу электронных часов, начиная с момента времени 0 часов, 0 минут, 0 секунд. Значение минут станет равным 1 только после того, как секунды «пробегут» все последовательные значения от 0 до 59. Часы изменят свое значение на 1 только после того, как минуты «пробегут» все последовательные значения от 0 до 59. Таким образом, вывод всех значений времени от начала суток до конца суток может быть представлен следующим фрагментом программы:

```
For h:=0 to 23 do
For m:=0 to 59 do
For s:=0 to 59 do
Writeln(h,':',m,':',s);
```

Для удобства реализации циклических структур на Паскале в последних версиях языка введены операторы **break** и **continue**, применяемые внутри циклов. Они расширяют возможности использования циклов и улучшают структуру программы.

В процессе выполнения тела цикла до полного завершения цикла могут возникнуть дополнительные условия, требующие завершения цикла. В этом случае цикл может быть прекращен оператором break.

В ходе выполнения цикла может возникнуть условие, при котором необходимо пропустить все или некоторые действия, предусмотренные в цикле, не прекращая работу цикла совсем. Для этого используется оператор continue, который передает управление в ту точку программы, где проверяется условие продолжения или прекращения цикла.

Контрольное задание

Задание 1. (посвящено циклу WHILE) Выполните следующие действия:

а) загрузите файл **laM 1_3.pas;**

б) прочитайте текст программы и проанализируйте его;

в) запустите программу на выполнение (рекомендуем задать в качестве значения числа M значения 100, 200, 0, -5). Оцените, как программа ведет себя в различных ситуациях;

г) измените программу так, чтобы она проверяла, является ли введенное число суммой кубов первых N натуральных чисел. Если является, следует указать это N. Для проверки используйте число 2025, здесь N=9.

Задание 2. В каких случаях целесообразно использовать цикл FOR? WHILE? REPEAT?

Задание 3. С помощью какого из циклов можно смоделировать два других?

Задание 4. Какое минимальное количество раз выполняется тело цикла REPEAT? WHILE?

Задание 5. При каком значении логического выражения (TRUE или FALSE) завершается работа цикла WHILE? REPEAT?

Задание 6. Можно ли составить цикл, выполняющийся бесконечное количество раз с помощью конструкции WHILE? REPEAT?

FOR?

☝ *Пригласите преподавателя и продемонстрируйте ему все выполненные Вами задачи и задания. После этого сохраните все программы в своей папке и выйдите из среды PascalABC. Будьте готовы ответить на вопросы преподавателя*

Лабораторная работа 9. Цикл с постусловием

Рассмотрим теперь математическую задачу. Пусть нам необходимо вычислить сумму первых членов гармонического ряда, удовлетворяющих условию 1/i>= e, где 0< e<1, а i=1,2,3…...Эту задачу можно решить по следующему алгоритму: положить предварительно y=0 и i=0, а затем в цикле увеличивать i на 1, к значению y добавлять очередное слагаемое 1/ i до тех пор, пока текущее значение 1/ i впервые окажется больше заданного значения 0< e<1.

Очевидно, что число повторений этого цикла заранее не известно. В подобного рода случаях мы можем лишь сформулировать условие, при выполнении которого процесс добавления к сумме очередного слагаемого должен завершиться.

Для задания таких вычислительных процессов и служит оператор цикла Паскаля с постусловием. Этот оператор имеет вид:

Repeat S1; S2;…; Si until B,

где **repeat** (повторять) и **until** (до) – служебные слова, через **Si** обозначен любой оператор Паскаля, а через **B** – логическое выражение.

При выполнении этого оператора цикла последовательность операторов, находящихся между словами repeat и until, выполнится один или более раз. Этот процесс завершается, когда после очередного выполнения заданной последовательности операторов логическое выражение B примет (впервые) значение true. Таким образом, с помощью логического выражения B задается условие завершения выполнения оператора цикла. Поскольку в данном случае проверка условия производится после выполнения последовательности операторов (тела цикла), этот оператор цикла и называется оператором цикла с постусловием.

С использованием этого вида оператора цикла Паскаля задача о суммировании первых членов гармонического ряда, удовлетворяющих заданному условию, может быть реализована следующим образом:

Пример кода оператора цикла Паскаля с постусловием

```
readln(e);
i:=0;
y:=0;
Repeat
   i:=i+1;
   y:=y+1/i;
Until 1/i<e;
```

Заметим, что оператор цикла с постусловием является более общим, чем оператор цикла с параметром — любой циклический процесс, задаваемый с помощью цикла с параметром можно представить в виде цикла с постусловием. Обратное утверждение неверно. Например, задача о суммировании первых n членов гармонического ряда, рассмотренная ранее, с оператором цикла с постусловием будет выглядеть так:

Пример кода оператора цикла Паскаля с постусловием

```
Readln(n);
i:=0;
y:=0;
Repeat
   i:=i+1;
   y:=y+1/i;
Until i>n;
```

Цикл с постусловием (REPEAT)

Этот цикл используется также в ситуации, когда количество повторений неизвестно, а известно УСЛОВИЕ завершения работы цикла. При этом само условие должно проверяться ПОСЛЕ ВЫПОЛНЕНИЯ действий в цикле. Общая структура цикла: REPEAT

<оператор 1>; <оператор 2>;

<оператор N> UNTIL **<условие>** Цикл выполняется до тех пор, пока <условно> ЛОЖНО.

Практическая работа

Задание 1. (посвящено циклу REPEAT) Выполните следующие действия:

а) загрузите файл **laM 1_4.pas;**

б) прочитайте текст программы и проанализируйте его;

в) запустите программу на выполнение;

г) измените программу так, чтобы она завершала работу при A=999.

Задание 2. (также посвящено циклу REPEAT) Выполните следующие действия:

а) загрузите файл **laM 1_5.pas;**

б) прочитайте текст программы и проанализируйте его;

в) запустите программу на выполнение;

г) обратите внимание, как программа спрашивает, нужно ли продолжать и что требуется вводить в качестве ответа;

д) измените программу так, чтобы количество цветов стало хотя бы 13.

Самостоятельная работа

Задание 1. В каких случаях целесообразно использовать цикл FOR? WHILE? REPEAT?

Задание 2. Какое минимальное количество раз выполняется тело цикла REPEAT? WHILE?

Задание 3. При каком значении логического выражения (TRUE или FALSE) завершается работа цикла WHILE? REPEAT?

Задание 4. Можно ли составить цикл, выполняющийся бесконечное количество раз с помощью конструкции WHILE? REPEAT? FOR?

☝ Пригласите преподавателя и продемонстрируйте ему все выполненные Вами задачи и задания. После этого сохраните все программы в своей папке и выйдите из среды PascalABC. Будьте готовы ответить на вопросы преподавателя.

Лабораторная работа 10. Массивы как структурированный тип данных

1.Теория

1. Достаточно часто возникает задача многократной обработки какой-то последовательности данных. Для того чтобы облегчить решение такого рода задач, существует тип данных, называемый *массивом.* В общем случае

Массив - это *упорядоченная* последовательность данных *одного* типа.

2. В памяти компьютера массив можно представить в виде последовательности блоков памяти, имеющих одно имя и отличающихся каким-то признаком (назовем этот признак индексом или порядковым номером). Например, массив с именем DAY, состоящий из 16 элементов, представляется следующим образом:

DAY - имя массива

Доступ к каждому элементу массива осуществляется по его индексу (порядковому номеру).

Например, 5-й элемент массива DAY в программе записывается как DAY[5], 7-й – как DAY[7] и т.д.

Обратите внимание на то, что индексы указываются после имени массива в *квадратных* скобках.

3. Прежде чем работать с массивом, его нужно описать, т.е. включить в блок Var такую запись:

<Имя> : array [<элем1>..<элем1N>] of <Тип>

Например, строка

var Mas : array[1..17] of Real;

описывает массив из 17 чисел типа **Real,** имеющих порядковые номера (индексы) с 1,2,..., 17, а строка

var Mas1 :array [0..15] of Integer; описывает массив из 16 целых чисел с номерами 0, 1,..., 15.

4. В общем случае в языке Паскаль в качестве индексов может выступать последовательность элементов любого перечислимого типа (т.е. такого типа данных, для каждого элемента которого известен предыдущий и/или последующий. Под такой тип явно не подходит тип REAL (какое число предшествует числу 1.2?) и тип String. Конечно, чаще всего используются числовые индексы. Рассмотрим несколько примеров описания массивов, имеющих индексы, отличные от числовых:

Type
mans=(boy,girl,man,woman,grandfather,grandmather);
var sign : **array** ['a'..'z'] **of byte;**
sportsman : **array** [boy..woman] **of real;**
words:array ['а'..'я'] of char;

5. Как уже отмечалось, обращение к отдельному элементу массива происходит так же, как и к элементу строки: после имени массива в квадратных скобках указывается номер нужного элемента. Например, запись **a:=Mas[11]** означает, что переменной a будет присвоено значение, взятое из 11-го элемента массива **Mas,** запись **a** := **Mas[n]** означает, что следует взять элемент массива Mas с номером, хранящимся в переменной n, и занести значение этого элемента в переменную a. И наконец, Mas[3]:=5; означает, что 3-му элементу массива присвоено значение 5. Обращение к элементу массива, имеющему нечисловой индекс, осуществляется аналогично:

m:=sign['f'];
sportsman[girl]:=12.5;
words[^']:='3';

1. Пусть дан массив:

Var m: **array [1 ..15] of integer; …**

Типовые фрагменты работы с этим массивом выглядят следующим образом: …

```
// ввод массива
For i:=1 to 15 do
begin
Write('Число,'i:2,' = ');
ReadLn(m[i]); end;…
// вывод массива
For i:=1 to 15 do WriteLn(m[i]);…
// четным элементам массива присваивается 0, нечетным - 1.
```

```
For i:=1 to 15 do
if I mod 2 =0 then m[i]:=0 else m[i]:=1
```

2. Практика

Задание 1. Запустите среду PascalABC. Составить программу, в которой:

а) всем элементам массива присваивается заданное целое значение (количество элементов массива - не более 50);

б) всем элементам массива присваивается вводимый текст;

в) каждому элементу массива с нечетным индексом присваивается его номер, а элементу с нечетным индексом - его номер с противоположным знаком;

г) элементам массива, стоящим на четных местах, присваивается символ " @" , на нечетных местах - "#".

д) каждому элементу, стоящему на 1-м, 4-м, 7-м и т.д. местах, присваиваться число 0, остальным - число 3, и все элементы массива выводятся на экран дисплея.

Задание 2. Загрузите в новое окно программу из файла **array_1.pas.** Эта программа позволяет ввести последовательность из 15 чисел и вывести их в обратном порядке. Разберитесь в том, как работает эта программа. Выполните задание, написанное в конце программы (после END.)

Задание 3. Модифицируйте программу из файла **array_1.pas** так, чтобы сначала печатались в строчку все положительные числа, а в следующей строке - все отрицательные.

Задание 4. Загрузите файл **array_2.pas.** Модифицируйте программу, размещенную в этом файле, так, чтобы она запрашивала не только массы, но и названия компонентов и печатала их в рецепте. Имейте в виду, что элементами массива могут быть переменные любого типа, в том числе и строковые. Для этой программы Вам потребуется два массива - один, как и прежде, числовой (Real) для хранения масс, а другой строковый (String[...]) - для хранения названий.

Задание 5. Напишите программу для вычисления дисперсии (и обязательно сохраните - она нам еще понадобится!).

****Задание 6.*** Напишите программу, которая вводит фамилию и возраст посетителей поликлиники, а по окончании ввода печатает отдельные списки больных до 18 лет, от 18 до 50 лет и старше 50 лет. (*Подсказка* По-видимому, в этой задаче Вам понадобится один массив для хранения возраста, а еще один - для хранения фамилий...)

Вопросы и задания для самоконтроля

1. Что такое массив? В каких случаях необходимо использовать массивы?
2. Что такое размерность массива?
3. Что такое элемент массива? Индекс массива?
4. Какие типы данных могут использоваться в качестве индексов для массивов?
5. Как ввести массив чисел?
6. Постройте правильные объявления на Паскале для девяти массивов (если это можно) по их словесным описаниям:
   а) массив, содержащий десять строк с максимальной длиной 15, пронумерованных числами от 1 до 10;
   б) массив из 19 вещественных чисел;
   в) ряд целых чисел, пронумерованных от 1950 до 1992;
   г) ряд целых чисел, пронумерованных от -6 до 4;
   д) целочисленный ряд, индексами которого служат буквы от а до f.
7. Дан массив, описанный следующим образом:

```
var   a  :  array[1..6] of integer;
i, j,q :                integer;
```

В массив записаны следующие данные:

Значение
7
-1
0
4
15
3

Укажите, какими станут эти значения после выполнения каждого из приведенных ниже фрагментов программ:

а) q:=a[4]+a[l];
a[5]:=q;

б)a[4]:=a[2]+a[2+1];
a[l]:=a[7-l]+a[7-2];

в)j:=4;
a[3]:=a[j]+a[j-1];
a[2]:=a[j+l];

г)i:=2;
j:=i+3; a[i]:=a[j]+a|j+l];

д)for i:=to5do
a[i]:=a[i+l];

e)j:=l;
a[5]:=a[j]+l+a[j+l].

8. Определить значение массива b после выполнения каждого из следующих фрагментов:

Type week=(Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday, Sum);
var b : array [week] of integer;
day : week;

а) for day:= Monday to do Sunday b[day]:=0;
б) b[Monday]:=l; for day:= Monday to Friday do b[succ(day)]:=b[day]*2;
в) b[Sum]:=0;
for day:= Monday **to do** Sunday **do**
begin
b[day]:=ord(day);
b[Sum]:=b[Sum]+b[day] end;
г) b[сумма]:=946; for day:= Friday downto Monday do b[day]:=b[Sum]-10.

9. Приведенная ниже программа WINDING, обрабатывая показанные здесь данные, завершилась аварийно. Объясните, почему. Объясните, как можно избежать ошибки:

Данные: 5 111 32 27 87

```
Program WINDING;
var numb : array[1..4] of integer;
i,n      : integer;
begin
Readln(n);
for i:=1 to n do readln(numb[i]);
end.
```

10. Дан массив a, описанный следующим образом:

```
var   a  :   array ['a'..'d'] of real;
i   :   char;
```

Выберите, какой из предложенных фрагментов обеспечит ввод данных в массив:

а) read(a);
б) for i:='a' to *d' do readln(a[i]);
в) readln(a['a'..'d']);
г) for i:='a' to 'd' do readln(a[i]);

11. Какая максимальная размерность (т.е. максимальное количество индексов) массива допустима в языке Паскаль?
12. Каково максимально допустимое количество элементов в массиве? Чем оно определяется?

☝ *Пригласите преподавателя и продемонстрируйте ему сделанные Вами программы. Будьте готовы ответить на вопросы преподавателя.*

Лабораторная работа 11. Объявление множества. Операции над множествами

Множество — это структурированный тип данных, представляющий собой набор взаимосвязанных по какому-либо признаку или группе признаков объектов, которые можно рассматривать как единое целое. Каждый объект в множестве называется элементом множества.

Все элементы множества должны принадлежать одному из порядковых типов, содержащему не более 256 значений. Этот тип называется базовым типом множества. Базовый тип задается диапазоном или перечислением.

Область значений типа множество — набор всевозможных подмножеств, составленных из элементов базового типа. В выражениях на языке Паскаль значения элементов множества указываются в квадратных скобках: [1,2,3,4], ['a',‘b','c'], ['a'..'z'].

Если множество не имеет элементов, оно называется пустым и обозначается как []. Количество элементов множества называется его мощностью.

Множество может принимать все значения базового типа. Базовый тип не должен превышать 256 возможных значений. Поэтому базовым типом множества могут быть byte, char, boolean и производные от них типы.

Множество в памяти хранится как массив битов, в котором каждый бит указывает является ли элемент принадлежащим объявленному множеству или нет. Максимальное число элементов множества 256, а данные типа множество могут занимать не более 32 байт.

Число байтов, выделяемых для данных типа множество, вычисляется по формуле:

ByteSize = (max div 8) - (min div 8) + 1,

где max и min — верхняя и нижняя границы базового типа данного множества.

Номер байта для конкретного элемента E вычисляется по формуле:

ByteNumber = (E div 8) - (min div 8),

номер бита внутри этого байта по формуле:

BitNumber = E mod 8

Не имеет значения порядок записи элементов множества внутри конструктора. Например, [1, 2, 3] и [3, 2, 1] — это эквивалентные множества.

Каждый элемент в множестве учитывается только один раз. Поэтому множество [1, 2, 3, 4, 2, 3, 4, 5] эквивалентно [1..5].

Переменные множественного типа описываются так:

Var <идентификатор> : set of <базовый тип>;

Например:

```
Var A, D : Set Of Byte;
    B : Set Of 'a'..'z';
    C : Set Of Boolean;
```

Нельзя вводить значения во множественную переменную процедурой ввода и выводить процедурой вывода.

Множественная переменная может получить конкретное значение только в результате выполнения оператора присваивания:

<множественная переменная> := <множественное выражение>;

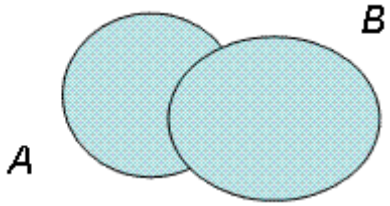
Например:

```
A : = [50, 100, 150, 200];
B : = ['m', 'n', 'k']; C : = [True, False];
D : = A;
```

Кроме того, выражения могут включать в себя операции над множествами.

Операции над множествами

Объединением двух множеств A и B называется множество, состоящее из элементов, входящих хотя бы в одно из множеств A или B. Знак операции объединения в Паскале «+».

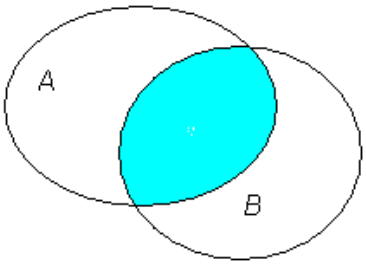


Примеры:

1) [1, 2, 3, 4] + [3, 4, 5, 6] => [1, 2, 3, 4, 5, 6]
2) []+[‘a’..’z’]+[‘A’..’E’, ‘k’] => [‘A’..’E’, ‘a’..’z’]
3) [5<4, true and false] + [true] => [false, true]

Пересечением двух множеств A и B называется множество, состоящее из элементов, одновременно входящих во множество A и во множество B.

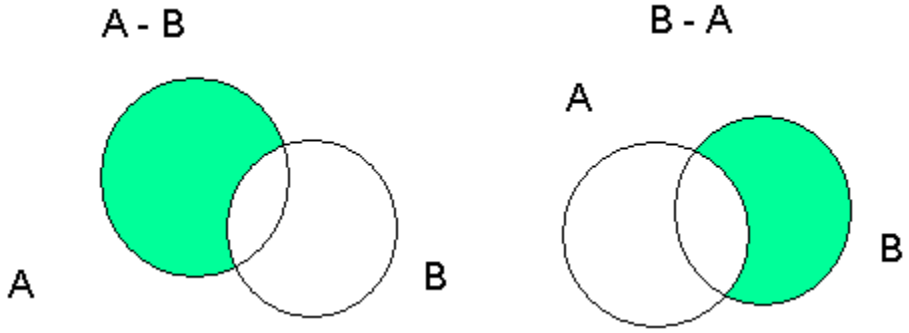
Знак операции пересечения в Паскале «*»



Примеры:

1) [1, 2, 3, 4] * [3, 4, 5, 6] => [3, 4]
2) [‘a’..’z’]*[‘A’..’E’, ‘k’] => [‘k’]
3) [5<4, true and false] * [true] => []

Разностью двух множеств A и B называется множество, состоящее из элементов множества A, не входящих во множество B.



Примеры:

1a) [1, 2, 3, 4] - [3, 4, 5, 6] => [1, 2]
1b) [3, 4, 5, 6] - [1, 2, 3, 4] => [5, 6]
2a) [‘a’..’z’]-[‘A’..’E’, ‘k’] => [‘a’..’j’, ‘i’..’z’]
2b) [‘A’..’E’, ‘k’] - [‘a’..’z’] => [‘A’..’E’]
3a) [5<4, true and false] - [true] => [false]
3b) [true] - [5<4, true and false] => [true]

Операция вхождения. Это операция, устанавливающая связь между множеством и скалярной величиной, тип которой совпадает с базовым типом множества. Если x — такая скалярная величина, а M — множество, то операция вхождения записывается так: **x in M.**

Результат — логическая величина true, если значение x входит в множество M, и false — в противном случае.

Например, 4 in [3, 4, 7, 9] — true, 5 in [3, 4, 7, 9] — false.

Используя данную операцию, можно не только работать с элементами множества, но и, даже если в решении задачи явно не используются множества, некоторые логические выражения можно записать более лаконично.

1) Натуральное число n является двухзначным. Вместо выражения **(n >= 10) and (n <=99)** можно записать **n in [10..99]**.

2) Символ c является русской буквой. Вместо выражения **(c >= ‘А’) and (c <= ‘Я’) or (c>=‘a’) and (c<=‘п’) or (c>=‘р’) and (c<=‘я’)** пишем **c in [‘А’.. ‘Я’, ‘а’.. ‘п’, ‘р’.. ‘я’]** и т.д.

Добавить новый элемент в множество можно с использованием операции объединения. Например, a:= a+[5] Для этих же целей в Turbo Pascal 7.0 предназначена процедура Include: **include (M, A)** M – множество, A – переменная того же типа, что и элементы множества M. Тот же пример можно записать так: Include (a, 5)

Исключить элемент из множества можно с помощью операции «разность множеств». Например, a:= a-[5] Для этих же целей в Turbo Pascal 7.0 предназначена процедура Exclude: **exclude (M, A)** M – множество, A – переменная того же типа, что и элементы множества M. Тот же пример можно записать так: Exclude (a, 5).

Рассмотрим несколько примеров использования множеств при решении задач.

Задача 1. В городе имеется n высших учебных заведений, которые производят закупку компьютерной техники. Есть шесть компьютерных фирм: «Диалог», «Avicom», «Нэта», «Сервер», «Декада», «Dega.ru». Ответить на следующие вопросы:
1) в каких фирмах закупка производилась каждым из вузов?
2) в каких фирмах закупка производилась хотя бы одним из вузов?
3) в каких фирмах ни один из вузов не закупал компьютеры?

Решим задачу с использованием множеств. Для удобства дальнейших манипуляций в порядке следования занумеруем компьютерные фирмы, начиная с единицы. Занесём информации о месте закупок компьютеров каждым из вузов в отдельное множество.

Ответ на первый вопрос можно получить, выполнив пересечение всех таких множеств.

Ответ на второй вопрос – результат объединения множеств.

И, наконец, на последний – разность множества всех фирм и множества фирм, где хотя бы один вуз делал покупки.

```
program ex_set_1;
type firma = set of 1..6;
    v = array[0..20] of firma;
const f: array [1..6] of string[10] =  ('Диалог', 'Avicom', 'Нэта', 'Сервер', 'Декада', 'Dega.ru');

{процедура ввода информации о закупке компьютеров в очередной фирме}
procedure vvod(var a: firma);
var i: byte; ans: 0..1;
begin
   a:= [];
   for i := 1 to 6 do
   begin
      Write('Вуз покупал компьютеры в фирме ', f[i], ' (1 - да, 0 - нет)? ');  ReadLn(ans);
      if ans = 1 then a:=a+[i]
   end;
end;

{процедура вывода элементов массива, номера которых содержатся в множестве}
procedure Print(a : firma);
var i: byte;
begin
   for i := 1 to 6 do    if i in a then write(f[i]:10);
   writeln
end;

{Процедура, дающая ответ на первый вопрос}
procedure Rez1(a: v; n : byte; var b : firma);
var i : byte;
begin
   b := [1..6];
   for i := 0 to n-1 do      b := b * a[i];
end;

{Процедура, дающая ответ на второй вопрос}
procedure Rez2(a: v; n : byte; var b : firma);
var i : byte;
```

```
begin
   b := [];
   for i := 0 to n-1 do      b := b + a[i];
end;

var a: v; n, i : byte; c : firma;
begin
   write('Сколько вузов делали закупку? ');    readln(n);
   for i := 0 to n-1 do vvod(a[i]);
   Rez1(a, n, c);
   writeln('Каждый из вузов закупил компьютеры в фирмах: ');    Print(c);
   Rez2(a, n, c);
   writeln('Хотя бы один из вузов закупил компьютеры в фирмах: ');    Print(c);
   writeln('Ни один из вузов не закупил компьютеры в фирмах: ');    Print([1..6]-c);
end.
```

Задача 2. Сгенерировать *n* множеств (нумерацию начать с 1). Вывести элементы, которые входят во все множества с номерами, кратными трём, но не входят в первое множество.

```
program ex_set_2;
type mn = set of byte;
    v = array[1..30] of mn;
{процедура ввода информации в очередное множество}
procedure vvod(var a: mn);
var i, n, vsp: byte;
begin
   a:= [];    n := 1 +random(200);
   for i := 1 to n do
   begin
     vsp:= random(256);
     a:=a+[vsp]
   end;
end;
{процедура вывода элементов множества}
procedure Print(a : mn);
var i: byte;
begin
   for i := 0 to 255 do    if i in a then write(i:4);
   writeln
end;

{Процедура, дающая ответ на вопрос}
procedure Rez(a: v; n : byte; var b : mn);
var i : byte;
begin
   b := [0..255];
   i:= 3;
   while i <= n do
   begin
     b := b * a[i];
     i := i + 3
   end;
   b := b - a[1]
end;

var a: v; n, i : byte; c : mn;
```

```
begin
   randomize;
   write('Сколько множеств? ');    readln(n);
   for i := 1 to n do
   begin vvod(a[i]);
            print (a[i])
   end;
   Rez(a, n, c);
   Print(c);
end.
```

Задача 3. Дана строка. Сохранить в ней только первые вхождения символов, удалив все остальные.

```
program ex_set_3;
var m : set of char;
   s : string; i : byte;
begin
   write('Введите строку: ');
   readln(s);
   m :=[];
   i := 1;
   while i <= length(s) do
   if s[i] in m then delete(s, i, 1)
                 else begin m:=m+[s[i]]; i := i + 1 end;
   writeln(s)
end.
```

Контрольные вопросы и задания

1. Что такое множество?
2. Почему множество является структурированным типом данных?
3. Как хранится множество в памяти ЭВМ? Какой максимальный объем оперативной памяти может быть отведен под хранение одного множества?
4. Какие операции можно выполнять над множествами?
5. Как добавить элемент в множество?
6. Как исключить элемент из множества?
7. Как вывести элементы множества? Как подсчитать количество элементов в множестве?
8. Как может быть использована операция вхождения?

Лабораторная работа 12. Объявления строковых типов данных

Цель: *изучение принципов работы с символами и строками. Закрепление навыков использования управляющих структур программирования, ввода-вывода данных и по отладке и тестированию программ.*

Оборудование и программное обеспечение: компьютер, TurboPascal.

Теоретические сведения:

Символьный тип

Переменные, предназначенные для хранения одиночных символов, называются символьными переменными. Символьный тип позволяет проводить сравнения символов между собой. В переменную этого типа может быть помещён любой из 256 символов расширенного кода ASCII. Объявление символьных переменных осуществляется в разделе объявления переменных с помощью служебного слова char. Обычно значения для переменных типа char задаются в апострофах. Например, если в программе есть описания

u, v: char

то возможны операторы присваивания

u:='a'; v:=u; v:='*' и т.д.

Штрих ' - принятая в Паскале форма кавычки - употребляется всякий раз, когда значение типа char явно указывается в программе. Выполнение операторов

u:='b'; write(u)

приводит к высвечиванию на экране символа b.

Кроме того, имеется возможность задавать значения указанием непосредственно числового значения ASCII-кода. В этом случае Вы должны перед числом, обозначающим код символа ASCII, поставить знак "#"

Функций для работы с символьными переменными.

chr(x: byte): char;

Преобразует целое число, имеющее тип byte, в один символ ASCII-кода.

ord(c: char): byte;

Функция ord выполняет действие, обратное функцииchr, т.е. возвращает порядковый номер символа параметра в таблице ASCII.

upcase(c: char): char;

Осуществляет преобразование символов английского алфавита из строчных символов в прописные. Все остальные символы при применении этой функции остаются непреобразованными.

pred(ch)

Возвращает предыдущее значение того же типа что переменной ch, которая имеет перечислимый тип. Например,pred(‘b’)=’a’; pred(8)=9.

succ(ch)

Возвращает последующее значение того же типа что переменной ch, которая имеет перечислимый тип. Например,succ(‘a’)=’b’;succ(9)=8.

Строковый тип.

Строка - это последовательность символов. При использовании в выражении строка обязательно заключается в апострофы. Длина строки не должна превышать 255-ти символов. Формат

<идентификатор, ...> : string [максимальная длина строки];

Над строковыми данными допустимы операции сцепления и операции отношения. Операция сцепления (+) применяется для сцепления нескольких строк в одну результирующую строку.

Операции отношения (=, <>, >, <, >=, <=) используются для сравнения двух строк. При сравнении строк используется понятие лексикографического порядка. Результат выполнения операций отношения над строками имеет логический тип, т.е. принимает значения True или False.

Стандартные процедуры

delete(St, Poz, N) - удаление N символов строки St начиная с позиции Poz.

insert(Str1, Str2, Poz) - вставка строки Str1 в строку Str2 начиная с позиции Poz.

str(I, St) - преобразование числового значения величины I и помещение результата в строке St.

val(St, I, Cod) - преобразование значения строки St в величину целого или вещественного типа. Результат помещается в переменную I. Cod - переменная целочисленного типа, признак завершения. В случае успешного завершения преобразования переменная Cod должна быть равна 0.

Стандартные функции

copy(St, Poz, N) - выдает подстроку, выделенную из строки St, длиной N символов начиная с позиции Poz.

length(St) - возвращает размер в символах строки St.

pos(Str1, Str2) - обнаруживает первое появление в строке Str2 подстроки Str1. Результат - номер позиции или 0.

concat(S1 [,S2, ..., Sn]) - объединяет несколько строк в одну.

Пример 2: Дан текст, слова в котором, могут разделяться пробелами, запятыми, точками и т.д. Требуется напечатать все слова с удвоенной буквой "н".

Этапы решения задачи: а) Формирование тела программы, объявление переменных;

б) Ввод текста;

в) Очистка текста от "ненужных" символов до первого слова;

г) Вычисление длины первого слова;

д) Поиск в слове буквы "н";

е) Подсчет стоящих рядом букв "н";

ж) Печать найденного слова;

з) Удаление первого слова;

и) Если текст не закончился возвращение к пункту (в).

```
а) program example1;
var st, st1:string;
i,j,k,n:integer;
flag:boolean;
const
znak=[' ','.',',',':',';','!','?'];
begin
end.
```

Назначение переменных:

t- содержит введенный текст

st1 - хранит первое слово текста

i,j,k,n - вспомогательные переменные

flag - указывает, что данное слово искомое

```
б) writeln('Введите текст'); readln(st);
в) repeat
whilest[1] in znak do delete(st,1,1);
г) i:=1; while(not (st[i] in znak)) and (i<=length(st)) doinc(i);
st1:=copy(st,1,i-1);
flag:= false;
д) while (pos('н',st1)>0) and (not flag) do
begin
е) j:=pos('н',st1); n:=j; k:=0;
whilest1[n]='н' do begin inc(n);inc(k); end;
if k=2 then flag:= true;
delete(st1,j,k)
end;
ж) if flag then writeln(copy(st,1,i-1));
з) delete(st,1,i);
и) until st='';
```

Порядок выполнения работы:

Задание: Создать и отладить программу для решения следующую задачу (см. Приложение), используя

1. тип char (символьный);
2. тип string(строковый).

Содержание отчета по каждому заданию:

- исходные данные (условие задачи);
- алгоритм (блок-схема) решения задачи;

- текст программы (или основной фрагмент программы);
- результаты выполнения программы
- тестовый вариант в форме с фиксированной точкой.

Контрольные вопросы:

1. Какие переменные называют символьными переменными?
2. Опишите тип данных char.
3. Какие преобразования выполняет функция ord?
4. Какие преобразования выполняет функция chr?
5. Какие преобразования выполняет функция upcase?
6. Что такое строка?
7. Какова максимальная длина строки?
8. Опишите функции для работы со строками.
9. Опишите процедуры для работы со строками.
10. Как обратиться к отдельному элементу строки?

Лабораторная работа 13. Процедуры и функции

Цель работы

Приобретение навыков разработки программ с использованием процедур и функций пользователя на языке Турбо-Паскаль.

Задание на лабораторную работу

1 Разработать программу для вычисления значений функции в соответствии с вариантом задания (табл. 5.1) при значении аргументов x и y.

2 Программа должна выводить сообщения - подсказки перед вводом данных сообщение о выводе результатов.

3 Программа должна состоять из подпрограмм и основной программы.

4 Ввод значении параметров x,yреализовать при помощи процедуры.

5 Вычисление значений функции при конкретном значении аргумента реализовать при помощи функции.

6 Основная программа должна осуществлять:

- ввод начальных и конечных значений аргумента;
- вычисление заданной функции;
- вывод результата на экран.

7 Ответить на контрольные вопросы.

8 Оформить отчет.

Справки по структуре и операторам Паскаль-программы, использующей пользовательские процедуры и функции

Процедуры и функции.

Процедуры и функции имеют общее название - подпрограммы. Применение подпрограмм дает возможность уменьшать число повторений одной и той же последовательности операторов, а также конструировать программу как набор отдельных подпрограмм.

В программе описание процедур и функций должно располагаться между разделами переменных и операторов. Каждая процедура или функция определяется только один раз, но может использоваться многократно.

Правило расположения описаний процедур и функций относительно друг друга: описание вызывающей подпрограммы должно быть ниже описания вызываемой подпрограммы, иначе следует использовать директиву forward(п. 5.3.2).

Структура Паскаль-программы с процедурами и/или функциями:

```
ProgramName_Prog;
{———Разделы основной (головной) программы—}
uses...
const...
type...
var...
{————————Описание процедуры 1————————}
Procedure Proc1 (...);
Const...;
Type...;
Var...;
Begin
End;
{————————Описание процедуры 2————————————}
Procedure Proc2 (...);
Const...;
Type ...;
Var...;
Begin
End;
{————————Описание функции 1————————————}
```

Function Func1 (...):Real;
Const ...;
Type...;
Var...;
Begin
End;
{————Начало блока головной программы————}
Begin
..... {Блок содержит вызовы процедур и функций)
End.

Структура, процедур и функций аналогична структуре полной программы на языке Паскаль. В процедурах и функциях могут быть описаны собственные метки, константы, типы, а также собственные процедуры и функции. Внутренние описания должны следовать в том же порядке, что и разделы основной программы.

Передача данных из главной программы в подпрограмму и возврат результата выполнения функции осуществляется с помощью параметров, указываемых в заголовке подпрограммы.

Описание процедуры. Оператор процедуры.

Процедура представляет собой программу, которая может вызываться другой программой, и служит для выполнения любого рода действий (т.е. диапазон возможностей процедуры - такой же, как у любой программы).

Описание каждой процедуры начинается с заголовка, в котором задаются имя процедуры и список формальных параметров с указанием их типов. Процедура может быть и без параметров, тогда в ее заголовке указывается только ее имя. С помощью параметров осуществляется передача исходных данных в процедуру, а также передача результатов работы обратно в вызывающую ее программу.

Структура описания процедуры:

Procedure<имя> (<список формальных параметров>) <директива>;
Const...;
Type...;
Var...;
{————————————Блок процедуры————————————}
Begin<операторы>
End;

<имя> - любой допустимый идентификатор, напр., Proc1.

Список формальных параметров - последовательность идентификаторов (имен) формальных параметров и их типов, напр., *Step:real*,
*Mas: Type_mas,...,*разделенных запятой.

Список формальных параметров может включать в себя параметры-значения, параметры-переменные (перед ними должно стоять ключевое слово Var), параметры процедуры (перед ними должно стоять ключевое слово Procedure) и параметры-функции (перед ними должно стоять ключевое словоFunction), нетипизированные параметры, перед которыми должно стоять служебное слово Var и отсутствует указание типа.

<директива> - одна из директив: Interrupt, External, Assanbler, Inline, Forward.

При выполнении лабораторной работы может быть использована директива опережающего описания Forward. Опережающее описание заключается в том, что объявляется лишь заголовок процедуры, ее тело заменяется зарезервированным словомForward, а само тело процедуры записывается в другом месте той же программы.

Вызов и выполнение процедуры осуществляется при помощи оператора процедуры:

<имя процедуры>{<список фактических параметров>);

Между формальными и фактическими параметрами должно быть полное соответствие, т.е. формальных и фактических параметров должно быть одинаковое количество; порядок следования фактических и формальных параметров должен быть один и тот же; тип каждого фактического параметра должен совпадать с типом соответствующего ему формального параметра.

При вызове процедуры значения фактических параметров присваиваются формальным параметрам, а результаты работы присваиваются параметрам-переменным.

Процедуры могут возвращать результат в основную программу не только при помощи параметров, во и непосредственно изменяя глобальные переменные. Переменные, описанные в основной программе, являются глобальными по отношению к внутренним процедурам и функциям.
Переменные, описанные внутри процедур и функций, называются локальными. Они порождаются при каждом входе в процедуру и уничтожаются при выходе из этой процедуры, т.е. локальные переменные существуют только при выполнении процедуры и недоступны в основной программе.

Вызов процедуры осуществляется отдельным оператором.

На блок-схемах вызов процедуры обозначается прямоугольником с двойными вертикальными стенками. Блок-схема же самой процедуры отличается от обычной программы тем, что вместо слова "Начало" в стартовый овал вписывается список входных параметров, а вместо слова "Конец" - список выходных параметров процедуры.

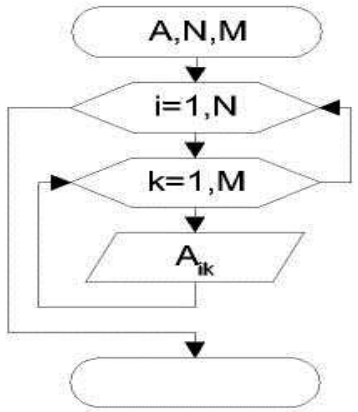


Рис. 5.1. Алгоритм процедуры вывода

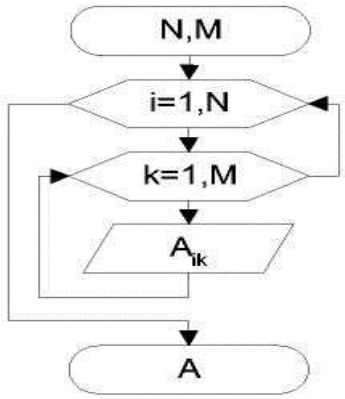


Рис. 5.2. Блок-схема процедуры ввода двумерного массива.

Например, если программа оперирует с несколькими двумерными массивами, то для вывода этих массивов на экран требуется каждый раз записывать двойной цикл, что достаточно громоздко. В этом случае удобно создать подпрограмму вывода двумерного массива на экран. Исходными данными для подпрограммы являются сам массив, а также число строк и столбцов в нем. Блок-схема представлена на рис. 5.1.

```
procedure Outp(a: massiv; N,M: integer);
var i,j:integer;
begin
for i:=1 to N do
begin
for j :=1 to M do
write (a[i,j]);
writeln;
end;
end;
```

Как видно, у этой процедуры выходных параметров вообще нет. При написании текста процедуры предполагалось, что в основной программе, из которой будет вызываться процедура, описан тип-массив massiv. Теперь для того, чтобы вывести на экран массив C (5x3) нужно использовать операторOutp(C,5,3).
Если процедура должна вводить данные в массив, то ей нужно сообщить его размеры, а она запросит ввод с клавиатуры нужного количества данных, запишет их в массив, и вернет этот массив в основную программу. Таким образом, массив для этой процедуры является выходным параметром, а значит, должен быть описан как параметр-переменная. Блок-схема данной процедуры отличается от предыдущей только изменением роли параметра-массива (см. рис. 5.2).

```
procedure Inp(var A:massiv; N,M:integer);
var i,j:integer;
begin
```

```
for i:=1 to N do begin
for j:=1 to M do read(a[i,j]); readln;
end; end;
```

Пример 1.Ввод исходных значений с использованием процедуры. В данном случае t – это текст, выводимый на экран, а x – вводимая переменная.

```
{Процедура ввода исходных значений}
Program primer1;
var
y: real;
procedure Vvod(t:string; var x:real);
begin
Writeln(t);
Readln(x);
end;
Begin
Vvod(‘Введите A‘,A); {Вызов процедуры Vvod}
Vvod(‘Введите B‘,B); {Вызов процедуры Vvod}
y:=a*b;
Writeln(‘y=‘, y:5:3);
End.
```

Описание функции. Указатель функции.

Функция представляет собой подпрограмму специального вида, основной задачей которой является определение какой-либоодной величины простого типа. Таким образом, имя функции всегда связано с некоторой величиной простого типа, вычисление которой этой функцией производится.

Функция, определенная пользователем, состоит из заголовка и тела функции. Заголовок содержит зарезервированное слово **function**, идентификатор (имя) функции, необязательный список формальных параметров, заключенный в круглые скобки, и тип возвращаемого функцией значения. Тело функции представляет собой локальный блок, по структуре аналогичный программе:

Function<имя>[(<список формальных параметров>)]:<тип результата>
Const...;
Type...;
Var...;
Begin
<операторы>
End;

<имя> - любой допустимый идентификатор, напр., Fun1;

<список формальных параметров> - последовательность идентификаторов (имен) формальных параметров и их типов, напр., Step:real,Mas:Type_mas, ... разделенных запятой;

<тип результата > - тип возвращаемого функцией результата, напр. Integer,Real, и др.

Среди входящих в функцию операторов должен обязательно присутствовать оператор присваивания, в левой части которого стоит имя данной функции. В точку вызова возвращается результат последнего присваивания.

Обращение к функции осуществляется по имени с указанием списка аргументов. Каждый аргумент должен соответствовать формальным параметрам, указанным в заголовке и иметь тот же тип.

Локализация имен в Турбо-Паскале.

В Турбо-Паскале допускается любой уровень вложенности процедур и функций. Для сложных программ имеются правила локализации имея, определяющие область действия для любого имени.

Любое имя (константы, типа, переменной, процедуры или функции) определено только в пределах той процедуры или функции, в которой оно описано. Область действия распространяется на все внутренние процедуры или функции.

Одно и то же имя может быть определено в каждой отдельной процедуре, функции или программе. При этом областью действия этого имени является процедура, функция или вся программа, в которой описан объект с данным именем, за исключением внутренних процедур, содержащих описание объекта с тем же именем.

Методические рекомендации

Пример 2. Вычислить значение выражения:

$$y=\frac{\lg(tg^2x+\cos^2x)}{\left(tg(x+\lg^2x)+\sin^2x\right)^2}$$

РЕШЕНИЕ.

В выражении несколько раз встречаются функции тангенса и десятичного логарифма, которых в Паскале нет. Несомненно, удобно создать такие функции и использовать их в программе. Блок-схема программы и подпрограмм-функций показана на рис. 5.6.

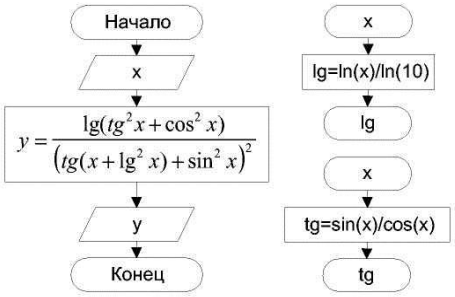


Рис. 5.6. Блок-схемы программ и подпрограмм к примеру 2.

```
program Expression;
var x,y:real;
procedure Vvod(t:string; var x:real);
begin
Writeln(t);
Readln(x);
end;
function lg(x:real) :real; {Функция вычисления десятичного логарифма}
begin
lg:=ln(x)/ln(10)
end;
function tg (x: real) : real; {Функция вычисления тангенса}
begin
tg:=sin(x)/cos(x)
end;
BEGIN
vvod('Введите x',x);
y:=lg(sqr(tg(x))+sqr(cos(x)))/sqr(tg(x+sqr(lg(x)))+sqr(sin(x));
writeln('y=',y:10:4);
END.
```

Контрольные вопросы

1. Что называется подпрограммой?
2. Какова структура программы с подпрограммами?
3. Какова структура процедур?
4. Какова структура подпрограммы-функции?
5. В чем состоит различие и сходство подпрограмм-процедур и подпрограмм-функций в Турбо аскале?
6. Как обратиться к процедурам и функциям?
7. Что называется параметром и каково его назначение?
8. Назначение формальных и фактических параметров и их взаимосвязь.
9. Опишите последовательность событий при вызове процедур или функций.
10. Для чего используется пошаговый режим с заходом в процедуры и как его осуществить?
11. Каковы отличия параметров-значений от параметров-переменных, особенности их описания и применения?
12 Чем отличаются локальные и глобальные параметры? Какова область их действия?

Лабораторная работа 14. Организация ввода-вывода данных. Работа с файлами

Программа при вводе данных и выводе результатов взаимодействует с внешними устройствами. Совокупность стандартных устройств ввода (клавиатура) и вывода (экран) называется консолью. В языке C# нет операторов ввода и вывода. Вместо них для обмена данными с внешними устройствами используются специальные объекты. В частности, для работы с консолью используется стандартный *класс* Console, определенный в пространстве имен System.

Вывод данных

В приведенных выше примерах мы уже рассматривали метод WriteLine, реализованный в классе Console, который позволяет организовывать вывод данных на экран. Однако существует несколько способов применения данного метода:

1. Console.WriteLine(x); //на экран выводится значение идентификатора x
2. Console.WriteLine("x=" + x +"y=" + y); /* на экран выводится строка, образованная последовательным слиянием строки "x=", значения x, строки "y=" и значения y */
3. Console.WriteLine("x={0} y={1}", x, y); /* на экран выводится строка, формат которой задан первым аргументом метода, при этом вместо параметра {0} выводится значение x, а вместо {1} – значение y*/

Замечание.

Рассмотрим следующий фрагмент программы:

int i=3, j=4; Console.WriteLine("{0} {1}", i, j);

При обращении к методу WriteLine через запятую перечисляются три аргумента: "{0} {1}", i, j. Первый аргумент определяет формат выходной строки. Следующие аргументы нумеруются с нуля, так переменная i имеет номер 0, j – номер 1. Значение переменной iбудет помещено в выходную строку на место {0}, а значение переменной j – на место {1}. В результате на экран будет выведена строка:3 4. Если мы обратимся к методу WriteLine следующим образом Console.WriteLine("{0} {1} {2)", j, i, j), то на экран будет выведена строка: 4 3 4.

Последний вариант использования метода WriteLine является наиболее универсальным, потому что он позволяет не только выводить данные на экран, но и управлять форматом их вывода. Рассмотрим несколько примеров:

1. *Использование управляющих последовательностей*:

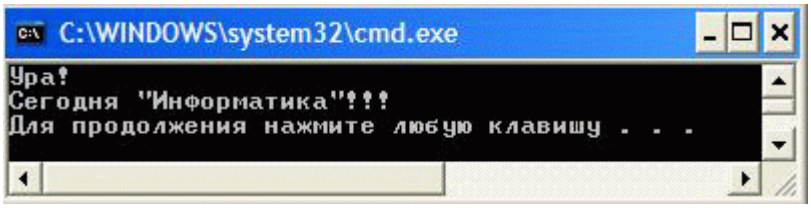
Управляющей последовательностью называют определенный символ, предваряемый обратной косой чертой. Данная совокупность символов интерпретируется как одиночный символ и используется для представления кодов символов, не имеющих графического обозначения (например, символа перевода курсора на новую строку) или символов, имеющих специальное

Вид	Наименование
\a	Звуковой сигнал
\b	Возврат на шаг назад
\f	Перевод страницы
\n	Перевод строки
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\\	Обратная косая черта
\'	Апостроф
\"	Кавычки

обозначение в символьных и строковых константах (например, апостроф). Рассмотрим управляющие символы:

Пример:

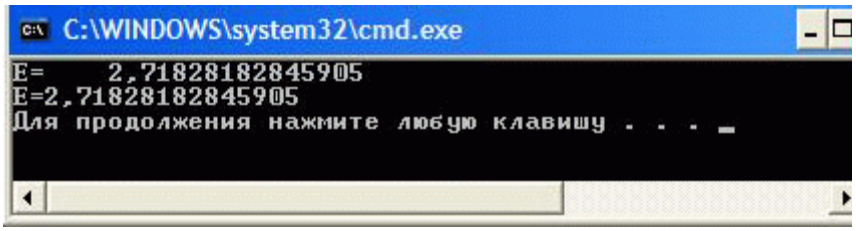
```
static void Main()
{
Console.WriteLine("Ура!\nСегодня \"Информатика\"!!!");
}
```



2. *Управление размером поля вывода*:

Первым аргументом WriteLine указывается строка вида {n, m} – где n определяет номер идентификатора из списка аргументов метода WriteLine, а m – количество позиций (размер поля вывода), отводимых под значение данного идентификатора. При этом значение идентификатора выравнивается по правому краю. Если выделенных позиций для размещения значения идентификатора окажется недостаточно, то автоматически добавится необходимое количество позиций. Пример:

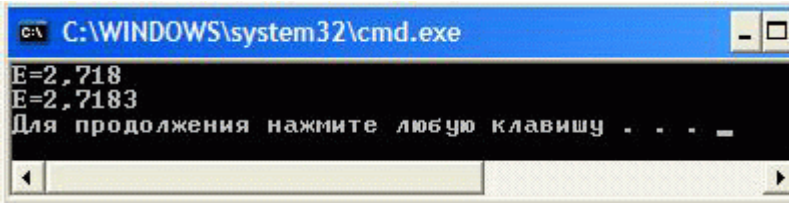
```
static void Main()
  {
   double x= Math.E;

Console.WriteLine("E={0,20}", x);

Console.WriteLine("E={0,10}", x);
   }
```



3. *Управление размещением вещественных данных*:

Первым аргументом WriteLine указывается строка вида {n: ##.###} – где n определяет номер идентификатора из списка аргументов метода WriteLine, а ##.### определяет *формат вывода* вещественного числа. В данном случае под целую часть числа отводится две позиции, под дробную – три. Если выделенных позиций для размещения целой части значения идентификатора окажется недостаточно, то автоматически добавиться необходимое количество позиций. Пример:

```
static void Main()
  {
   double x= Math.E;
Console.WriteLine("E={0:##.###}", x);
Console.WriteLine("E={0:.####}", x);
   }
```



Задание. Измените программу так, чтобы число e выводилось на экран с точностью до 6 знаков после запятой.

4. *Управление форматом числовых данных*:

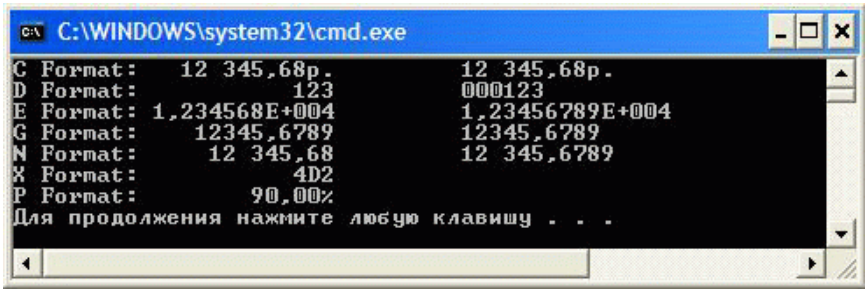
Первым аргументом WriteLine указывается строка вида {n: <спецификатор>m} – где n определяет номер идентификатора из списка аргументов метода WriteLine, <спецификатор> - определяет формат данных, а m – количество позиций для дробной части значения идентификатора. В качестве спецификаторов могут использоваться следующие значения:

Параметр	Формат	Значение
C или c	Денежный. По умолчанию ставит знак р. Изменить его можно с помощь объекта NumberFormatInfo	Задается количество десятичных разрядов.
D или d	Целочисленный (используется только с целыми числами)	Задается минимальное количество цифр. При необходимости результат дополняется начальными нулями
E или e	Экспоненциальное представление чисел	Задается количество символов после запятой. По умолчанию используется 6
F или f	Представление чисел с фиксированной точкой	Задается количество символов после запятой
G или g	Общий формат (или	Задается количество символов

	экспоненциальный, или с фиксированной точкой)	после запятой. По умолчанию выводится целая часть
N или n	Стандартное форматирование с использованием запятых и пробелов в качестве разделителей между разрядами	Задается количество символов после запятой. По умолчанию – 2, если число целое, то ставятся нули
X или x	Шестнадцатеричный формат	
P или p	Процентный	

Пример:

```
static void Main()
 {Console.WriteLine("C Format:{0,14:C} \t{0:C2}", 12345.678);
  Console.WriteLine("D Format:{0,14:D} \t{0:D6}", 123);
  Console.WriteLine("E Format:{0,14:E} \t{0:E8}", 12345.6789);
  Console.WriteLine("G Format:{0,14:G} \t{0:G10}", 12345.6789);
  Console.WriteLine("N Format:{0,14:N} \t{0:N4}", 12345.6789);
  Console.WriteLine("X Format:{0,14:X} ", 1234);
  Console.WriteLine("P Format:{0,14:P} ", 0.9);
 }
```



Ввод данных

Для ввода данных обычно используется метод ReadLine, реализованный в классе Console. Особенностью данного метода является то, что в качестве результата он возвращает строку (string). Пример:

```
static void Main()
 {string s = Console.ReadLine();
  Console.WriteLine(s); }
```

Для того чтобы получить числовое значение необходимо воспользоваться преобразованием данных. Пример:

```
static void Main()
 { string s = Console.ReadLine();
  int x = int.Parse(s); //преобразование строки в число
  Console.WriteLine(x);}
```

Или сокращенный вариант:

```
static void Main()
 { //преобразование введенной строки в число
  int x = int.Parse(Console.ReadLine());
  Console.WriteLine(x);}
```

Для преобразования строкового представления целого числа в тип int мы используем метод int.Parse(), который реализован для всех числовых типов данных. Таким образом, если нам потребуется преобразовать строковое представление в вещественное, мы можем воспользоваться методом float.Parse() или double.Parse(). В случае, если соответствующее преобразование выполнить невозможно, то выполнение программы прерывается и генерируется исключение System.FormatExeption (*входная строка* имела неверный формат).

Задание. Измените предыдущий фрагмент программы так, чтобы с клавиатуры вводилось вещественное число, а на экран это число выводилось с точностью до 3 знаков после запятой.

Лабораторная работа 15. Файлы произвольного доступа. Порядок работы с файлами произвольного доступа

Файлы произвольного доступа создаются для решения задач, требующих оперативного доступа к хранимой информации или при наличии зависимости значения поля компонента от порядкового номера компонента в файле.

Последовательный доступ в этих случаях неэффективен.

Работа с файлом произвольного доступа предполагает его организацию, обработку и корректировку. Организация файла произвольного доступа.

Организовать файл произвольного доступа можно двумя способами:

1) создать последовательный файл и обращаться к компонентам по их порядковому номеру, трактуя последовательный файл как произвольный;

2) создать файл фиктивных записей, затем загрузить его по ключу фактическими данными; обращение к компонентам по ключу предполагает использование процедуры Seek. Первый способ аналогичен созданию последовательного файла, поэтому рассмотрим второй способ. Форматизация заключается во внесении в файл компонентов, значение полей которых носит фиктивный характер, например, нули или пробелы. Доступ к компонентам файла произвольного доступа Доступ к компонентам файла произвольного доступа может быть как последовательный, так и произвольный. Последовательный доступ не отличается от описанного в

Задании № 1. Для организации произвольного доступа используется процедура Seek. Произвольный доступ организовывается по ключу, значение которого равно порядковому номеру в файле нужного компонента. Ключ должен иметь целочисленное значение, не превышающее количество компонентов в файле. Порядок действий при обработке файлов произвольного доступа следующий: присвоить файлу имя (процедура AssignFile); открыть файл (процедура Reset) и запросить ключ; подвести указатель по ключу к нужному компоненту (процедура Seek); считать нужный компонент (оператор Read); выполнить обработку считанной информации; закрыть файл (процедура CloseFile). Корректировка файла произвольного доступа Корректировка файла произвольного доступа заключается в изменении значений полей компонентов в целом или частично или расширении файла. Для корректировки полей компонентов файла необходимо: присвоить имя файлу (процедура AssignFile); открыть корректируемый файл (процедура Reset); подвести указатель файла к корректируемому компоненту (Seek); считать корректируемый компонент (Read); откорректировать нужные поля; повторить инструкцию подвода указателя (процедура Seek); записать откорректированный компонент (процедура Write); закрыть файл (процедура CloseFile). Шестой пункт необходим, так как после считывания записи указатель файла переместится к следующей записи, поэтому, чтобы откорректированную запись сохранить на старом месте, указатель надо переместить на один компонент вверх. Процедура корректировки полной записи аналогична описанной выше, только новая запись вносится на место уже существующей фактической записи, а не фиктивной. Возможен и другой вид корректировки файла произвольного доступа – добавление компонентов.

Файл состоит из записей одинакового размера и напоминает базу данных, что обеспечивает быстрый поиск информации. При создании файла указывается максимальная длина записи. Главным отличием от рассмотренных ранее данных является то, что такая структура данных может содержать компоненты (отдельные данные) разного типа.

Рассматриваемая структура данных часто называется записью - ***Record,*** компоненты этой структуры - полями записи -***Fields.*** Если запись состоит из ***K*** - полей, то каждое поле записи будет иметь порядковый номер от 1 до ***K.*** Запись, как и база данных, содержит множество наборов единообразных записей, т. е. записей, состоящих из одинакового количества полей, и***i***-е поля всех записей имеют одинаковый тип и размер.

Набор записей можно представить в виде строк таблицы, у которой ***n*** столбцов, а ***i***-й столбец таблицы соответствует ***i***-у полю любой записи.

Пользовательский тип данных

Пользователь может объявить собственный тип данных - пользовательский, который представляет логически объединенный набор данных, состоящий из нескольких переменных стандартного типа, выступающих под одним именем. Объявить эти данные можно следующим образом:

Private / Public Type <имя типа данных>

Локальная / Глобальная

<элемент 1> As <тип элемента 1>
. . . . . .
<элемент N> As <тип элемента N>
End Type

<Имя типа данных> - имя, которое присваивается определяемому типу данных, <элемент 1> ... <элемент N> - название полей записи, для которой создается пользовательский тип данных. <Тип элемента 1> . . . <тип элемента N> - стандартные типы данных, причем данные типа ***String*** должны иметь фиксированную длину. Например:

Type Mens
Name As String*15 'имя'
Famile As String*20 'фамилия'
DateR As Date ' дата рождения'
Prim As String*30 'примечание'
End Type

Произвольный доступ к файлу предполагает, что файл состоит из записей, каждая из которых содержит одно или несколько полей. Запись с одним полем соответствует любому стандартному типу. Запись с несколькими полями соответствует типу, определяемому пользователем. При открытии файла обязательно указывается длина записи, что делает возможным считывать определенное количество байт и выбрать нужную запись или вывести данные в указанную запись. Синтаксис оператора открытия файла следующий:

Open <имя файла> for Random As #<Дескриптор> [Len = <длина записи>]

Если указанная длина записи меньше, чем дейтвительная, то воспроизводится ошибка. Для работы с файлом произвольного доступа следует определить тип записи, объявив пользовательский тип в программном модуле. В программном коде необходимо объявить переменную пользовательского типа, переменную для хранения длины записи, а также переменную для поиска следующей записи:

Dim <имя переменной> As <имя типа данных:>

Например,

Dim mst As Mens

В отличие от текстовых файлов при произвольном доступе к данным нет различия между файлами для записи и для чтения: и при записи и при чтении файлы открываются в режиме, определенном словом ***Random.***

Длина записи ***Len*** - целое число, равно длине переменной пользовательского типа, которая применяется для хранения одной записи файла.

Закрытие файлов произвольного доступа выполняется аналогично последовательному доступу - ***Close*** # <Дескриптор>.

Для чтения новой записи в файл - в режиме запись - применяется оператор, который в качестве обязательного параметра содержит также номер записи:

Put # <Дескриптор>, <номер записи>, <имя переменной>

Для чтения записей из файла - в режиме чтение - применяется оператор, который должен содержать номер записи:

Get # <Дескриптор>, <номер записи>, <имя переменной>

Пример

Dim mst As Mens
Dim Nts As Integer, Lts As Integer
Lts=Len(mst)
Open "D:\ UTF\ tt3.txt" for Random As #1 Len = Lts
Get #1, Nts, mst
Text2.Text =Trim(mst.Prim)
Put #1, Nts, mst

В данном примере величины ***Nts*** и ***Lts*** определяют соответственно номер текущей записи и длину текущей записи. Функция***Len*** (значение) возвращает номер текущей записи, этот номер используется при открытии файла произволь¬ного доступа. Далее запись с номером ***Nts*** читается в переменную ***mst.*** Функция ***Trim*** (значение) отсекает пробелы справа при выводе данных - в данном случае в текстовое окно. И наконец, оператор добавляет новую запись или изменяет существующую

(если запись с таким номером имеется). Если количество записей в файле - переменная ***ks,*** то, для того чтобы добавить запись в конец файла, необходимо

ks = ks + 1

Put #<Дескриптор>, ks> mst

Чтобы удалить запись из файла произвольного доступа, следует создать новый файл и скопировать в него все записи, за исключением той, которую необходимо удалить. Исходный файл можно удалить совсем с помощью оператора ***Kill,*** а новый переименовать с помощью оператора ***Name.*** Например:

Kill "<полное имя удаляемого файла>"

Name <старое имя> As <новое имя>

Двоичный доступ представляет собой частный случай произвольного доступа. В этом режиме размер записи всегда равен 1 байту, и достаточно лишь указать позицию произвольного байта или его порядковый номер в файле. Прежде чем открыть файл в режиме двоичного доступа, следует получить свободный номер. Синтаксис оператора следующий:

Open <имя файла> for Binary As # <Дескриптор>

После того как файл открыт, операторы позволяют поместить позицию ввода или вывода на произвольный байт и прочитать или записать произвольное число байт:

Put # <Дескриптор>, <номер байта>, <имя строки для вывода>
Get # <Дескриптор>, <номер байта>, <имя строки для ввода>

Закрытие файлов двоичного доступа выполняется аналогично тому, как это было описано выше:

Close # <Дескриптор>.

В режиме двоичного доступа можно открыть как последовательный файл, так и файл произвольного доступа. Можно одновременно и читать, и записывать в файл.

ЗАКЛЮЧЕНИЕ.

Материал необходимо изучать последовательно. Предложено два вида материалов: теоретические разделы, каждый из которых следует внимательно изучить и ответить на приведенные в конце блока вопросы, и практические работы, которые следует выполнять за компьютером. Для выполнения практических работ необходима среда программирования PascalABC любой модификации, а также электронная поддержка практических работ, размещенная в папке EXAMPLES. Можно, конечно, использовать для обучения и другие среды (Turbo Pascal, Borland Pascal, Free Pascal, TNT Pascal и т.п.), однако материал подготовлен именно для названной среды. Среда бесплатная и ее можно скачать с http://sunschool.math.rsu.ru/pabc/.

- традиционные Турбо-среды (тот же Borland Pascal) являются DOS-средами, т.е. для работы в них нужно использовать стандарты операционной системы MS DOS, что во времена Windows является, по меньшей мере, анахронизмом, поскольку данные навыки вряд ли кому-то пригодятся в будущем;
- возможности среды PascalABC, в отличие от других сред, максимально приближены к Delphi, что позволяет более естестественно перейти от программирования учебного к программированию объектно-ориентированному;
- графика PascalABC, хотя и включает графику Borland Pascal, скорее приближена к графике Delhi;
- среда PascalABC содержит прекрасную и понятную справку на русском языке с множеством примеров;
- кроме того, в среду включен компьютерный задачник с возможностью автоматической проверки решений.

СПИСОК ЛИТЕРАТУРЫ И ИСТОЧНИКОВ

1. Анисимов А.Е. Сборник задач по началам программирования. – Ижевск, Из-во УдГУ, 2014. – 149 с.
2. Анисимов А.Е., Пупышев В. В.. Сборник заданий по основаниям программирования. – М., ИНТУИТ, 2012. – 352 с.
3. Павловская Т.А. Паскаль. Программирование на языке высокого уровня: Учебник для вузов.– Спб.: Питер, 2013. – 400с .

Приложение 1

Тематика задач (для самостоятельной работы студентам).

1. Дана последовательность из n символов. Подсчитать сколько раз среди данных символов встречается символ + и сколько раз - символ *.
2. Дана последовательность из n символов. Подсчитать общее число вхождений символов +, -, * в данную последовательность.
3. Вывести на экран изображение всех возможных символов вместе с их номерами.
4. Дана последовательность из n символов. Выяснить, имеются ли в последовательности два подряд идущих символа ",-".
5. Подсчитать, сколько раз среди последовательности символов встречается символ, задаваемый вводом.
6. Напечатайте заданную последовательность символов, заменяя каждую точку многоточием.
7. Дана последовательность из n символов. Известно, что в последовательности имеется хотя бы одна запятая. Найти номер первой по порядку запятой.
8. Дана последовательность из n символов. Известно, что в последовательности имеется хотя бы одна запятая. Найти номер последней по порядку запятой.
9. Дана последовательность из n символов. Выяснить, каких символов в последовательности больше, запятых или точек с запятой. (Не исключается случай равенства).
10. Дана последовательность символов, заканчивающаяся символом '+'. Определить и вывести на экран номера символов, имеющих наибольший и наименьший номер.
11. Дана последовательность из n символов. Выяснить, в какой половине последовательности, в первой или во второй, больше вопросительных знаков. (Не исключается случай равенства).
12. Дана последовательность из n символов. Выяснить встречается ли в данной последовательности группа из трёх стоящих рядом точек.
13. Дана последовательность из n символов. Подсчитать наибольшее число букв "а" идущих подряд в данной последовательности.
14. Дана последовательность символов, заканчивающаяся символом '?' Подсчитать наибольшее количество символов "пробел" идущих в строке подряд.
15. Дана последовательность из n символов. Подсчитать сколько раз среди данных символов встречается символы ‘r’, ‘k’.
16. Во введённой строке каждую цифру заменить на следующую по порядку цифру. Цифру 9 заменить на цифру 0.
17. Преобразуйте строку, заменив в ней каждую точку многоточием.
18. Во введённой строке замените все маленькие латинские буквы на большие.
19. Во введённой строке удалите все символы, стоящие на нечётных местах.
20. Дана строка символов, содержащая хотя бы два двоеточия. Необходимо вывести на экран все символы, расположенные между первым и вторым двоеточием.
21. Определить и вывести на экран длину самого большого слова во введённой строке.
22. Определить и вывести на экран количество слов во введённой строке.
23. Выясните, какая из букв, первая или последняя, встречается во введённой строке чаще?
24. В строке, введённой с клавиатуры, удалите все лишние пробелы.
25. Во введённой строке подсчитайте количество символов '*' и символов '!'.
26. Во введённой строке подсчитайте общее число вхождений символов '+', '-', '*'.
27. Определить в строке введённой с клавиатуры количество гласных (латинских) букв.
28. В строке введённой с клавиатуры заменить все X на Y.
29. Введённую с клавиатуры строку А записать в обратном порядке в строку Б. Строку Б вывести на экран.
30. Во введённой строке все '123' заменить на '45'.

$$y=\frac{\arcsin(x+x^3)}{x^5*\arcsin(x)}$$

$$y=\frac{\sqrt[3]{x+6}\cdot tg(x)}{tg\left(x+\sqrt[3]{x}+7\right)^3}$$

$$y=\arccos\left(x^a\right)\cdot\arccos\left(x^{a+7}\right)$$

$$y=\frac{\sqrt{x+5}ctg(x)}{\left(ctg\left(x+\sqrt{x+5}\right)\right)^2}$$

$$y=\frac{n!+(x+3)!}{\left(n^x+3\right)}+x^{a+\sqrt[4]{3}}$$

$$y=\frac{n!+(n+1)!}{\left(n+x^a\right)!}+\left(x^{a+\sqrt[3]{5}}\right)^2$$

$$y=\frac{(x-1)!+(x+4)!}{\left(x^2-1\right)!}+\left(x^{3+\sqrt[3]{7}}\right)^2$$

$$y=\frac{\lg\left(ctg^3(x)+\sin^2(x)\right)}{\left(ctg(x+\lg^2(x)+\cos^2(x)\right)^2}$$

$$y=\frac{x^a+(x+2)!}{\left(x^2+x^{a+1}\right)!}$$

$$y=\frac{\log_a\left(n!-n^5\right)}{\left(n\cdot\left(\log_a n!\right)\right)}$$

$$y=\frac{\lg\left(tg^3(x+1)\right)+\cos^2(x)}{tg\left(x+\lg^3(x)+\sin^2(x)\right)}$$

$$y=\frac{x^a+n!}{\left(n!+x^{a/n}\right)}$$

$$y=\frac{\log_a\left(2n!+n^3-1\right)}{2\cdot n!\cdot\log_a n^2}$$

$$y=\frac{\log_a\left(ctg\left(x^2\right)+\cos\left(x^3\right)\right)}{\log_a\left(\sqrt[a]{x}\right)+\sin(x)-ctg\left(x^a\right)}$$

$$y=\frac{\log_a\left(n!+n^4-2n\right)}{\left(n\cdot\log_a(n+3)\right)!}$$

$$y=\frac{\log_7\left(tg\left(\sqrt[3]{x^4}\right)+\cos\left(x^3-7\right)\right)}{\log_7\left(\left(tg\left(x^3-2\right)\right)+\sin\left(\sqrt[3]{x^5}+2\right)\right)}$$

$$y=\frac{\sqrt[a]{x}+x!}{n+x^{\sqrt[a]{n}}}$$

$$y=\frac{\arccos\left(x^4-x+2\right)}{x^6\cdot\arccos\left(x^3-2\right)}$$

$$y=\arcsin\left(x^a+4\right)\cdot\arcsin\left(x^{a+9}\right)$$