Согласовано Утверждаю:

Методическим советом директор МКОУ ОСОШ

№ \_\_\_ от \_\_.\_\_.\_\_\_\_\_\_г. \_\_\_\_\_\_\_\_\_ И.Н. Помпа

**ЭЛЕКТИВНЫЙ КУРС**

**ПО ИНФОРМАТИКЕ И ИКТ**

**«ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ TPASCAL»**

9 класс

Учитель:

Сивец Екатерина Михайловна

МКОУ ОСОШ

2012-2013 учебный год

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

Общие сведения:

Образовательная область: информатика.

Профиль: информационно-технологический.

Возрастная группа: 9 класс.

Количество часов: 34.

Элективный курс «Программирование на языке TPascal» предназначен для обучающихся 9 классов основной школы и рассчитан на 34 часов.

В меняющемся курсе информатики все меньше места отводится разделу «Алгоритмизация и программирование» Но для подготовки учащихся к обучению в ВУЗах учитель должен заложить основы знания одного из алгоритмических языков. Данный элективный курс позволит дать и закрепить на практике знания учащихся по основным конструкциям языка программирования Pascal. Его особенность в том, что важнейшие основные понятия программирования: типы данных, структуры данных, операторы, алгоритмы, рассматриваются на конкретных примерах, что делает их более понятными для начинающих программистов.

Программы на языке Pascal отличаются строгой структурой. Программирование на нем приучает к аккуратности, продуманности. Кажущаяся ненужной строгость в описании типов данных, процедур и функции оборачивается изящными и понятными конструкциями, а также высокой производительностью программ и экономным использованием памяти.

Элективное изучение программирования возможно лишь тогда, когда курс ориентирован на практическую деятельность. Поэтому за теоретическими выкладками по каждой теме следуют блоки задач. Задачи разбиты на группы и располагаются в порядке усложнения.

По окончании изучения тематического блока учащимся предлагается выполнить межпредметный проект (математика - информатика). Для его выполнения необходимо провести исследование в математической предметной области и проанализировать:

* Существующие численные методы вычисления площадей криволинейных фигур, заданных графиками (проект №1);
* Способы экспериментального нахождения sin (x), cos(x),log(x); (проект №2)
* Методы решения уравнений (проект №3).

**Содержательные линии курса:**

* Программирование циклических алгоритмов.
* Программирование задач по теории чисел.
* Создание программ для вычисления интегралов различными методами.
* Создание программ для вычисления тригонометрических функций, логарифмов, корней методом суммирования элементов ряда.
* Создание программ для нахождения корней уравнений различными методами.
* Программирование алгоритмических структур не входит с минимум содержания по информатике. Поэтому все линии составляют инвариант содержания.

**Цель курса:**

* формирование готовности использовать компьютер в качестве инструмента для обработки сложных математических объектов.

**Задачи курса:**

* Формирование способности использовать математический аппарат компьютера для нужд исследователя.
* Формирование способности создавать интерактивное программное обеспечение с дружественным интерфейсом для выполнения научных исследований.

Основная форма мониторинга достижений учащихся: на каждом занятии оценка за решение задач после прохождения очередной темы.

Итоговая оценка: защита проекта. Форма итоговой оценки: зачет.

**Основные образовательные результаты:**

Учащиеся должны:

* Изучить и проанализировать материал по каждой подтеме численных методов;
* Разработать и реализовать интерфейс будущей программы;
* Написать программы, реализующие алгоритмы вычислений интегралов, тригонометрических функций, логарифмов, корней чисел4, корней уравнений различными методами;
* Проанализировать эффективность каждого из алгоритмов, провести сравнительный анализ;
* Представить проект, подготовить пояснительную записку, доклад.

**УЧЕБНО-ТЕМАТИЧЕСКИЙ ПЛАН**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | Название темы | Количество часов | | | Форма проведения | Образов. продукт | Дата |
| Всего | Лекции | Практика |
| 1 | Введение в программирование. Знакомство с программным приложением. Основные команды для работы. | 1 | 0,5 | 0,5 | Демонстрационная лекция | Конспект, шаблон программы |  |
| 2 | Описание переменных. Решение простейших задач. | 1 | 0,5 | 0,5 | Практикум  «Решение задач» | Конспект |  |
| 3 | Условный оператор If | 1 | 0.5 | 0.5 | Практикум «Решение уравнений с меняющимися переменными» | Конспект |  |
| 4 | Оператор Case | 1 | 0.5 | 0.5 | Практикум «Создание меню». | Конспект |  |
| 5 | Циклический оператор For | 1 | 0,5 | 0,5 | Практикум «Таблица умножения» | Конспект |  |
| 6 | Самостоятельная работа на закрепление | 1 |  | 1 |  | Сам. работа |  |
| 7 | Переход в графический режим. Графические примитивы. | 1 |  | 1 | Практикум «Рисуем в Паскале» |  |  |
| 8 | Выбор цвета и заливки для отдельных объектов | 2 | 0,5 | 1,5 | Практикум «Рисуем в Паскале» |  |  |
| 9 | Программирование движения фигур в автоматическом режиме | 2 |  | 2 | Практикум «Паровозик» | Сам. работа |  |
| 10 | Программирование движения фигур по нажатию кнопок на клавиатуре | 2 |  | 2 | Практикум «Марионетка» | Сам. работа |  |
| 11 | Работа над проектом | 2 |  | 2 |  | Сам. работа |  |
| 12 | Зачет | 1 |  | 1 |  |  |  |

Текст пособия

Введение. Основные термины и понятия

Данный курс рассчитан на школьников, уже начавших изучать язык программирования Pascal. Курс содержит основные элементы структурного, процедурного и модульного программирования и их практическое применение в Pascal’e.

Прежде, чем непосредственно приступить к изучению курса напомним себе ключевые операторы и структурные элементы языка программирования Pascal:

Идентификатор – имя объекта, устанавливающее соответствие объекта некоторому набору символов.

Например: f gh k1 (1k) d\_r (d r d,r )

Константы – элементы данных, значения которых установлены в описательной части программы и в процессе выполнения программы не изменяются. Описание констант осуществляется в разделе const.

Const k=8; m=19;

Переменные – величины, которые могут менять свои значения в процессе выполнения программы. Описание переменных осуществляется в разделе var.

Var g: integer;

st: string;

Тип данных – множество величин, объединенных определенной совокупностью допустимых операций.

Например: 5.9 и -7.456 относятся к вещественному типу (real), их можно складывать, вычитать, умножать, делить и т.д.

Типы данных делятся на стандартные и пользовательские. Стандартные типы предложены разработчиками системы программирования Turbo Pascal, например, Boolean, real, byte, string, char.

Пользовательские типы данных разрабатывается пользователями системы, для обеспечения семантического контроля вводимых данных, значительного улучшения наглядности программы, более легкого поиска ошибок и экономии памяти. К пользовательским относят перечисляемый и интервальный типы.

Type color=(red, blue, white, black, brown);

month=1..12;

Выражение – конструкция языка, задающая порядок выполнения действий над элементами данных.

Операнд – элемент данных, над которыми производиться операция.

Операция – действия, которые необходимо выполнить над операндом.

( 42 – А / 2 ) + 8

операции

операнды

выражение

Оператор – предложение языка Pascal, задающее полное описание некоторого действия, которое необходимо выполнить. Операторы в Pascal разделяются точкой с запятой (;).

Оператор присваивания (:=) А:=78;

Оператор вызова процедуры ClrScr; {вызов стандартной процедуры очистки экрана}

Операторы ввода-вывода

Read обеспечивает ввод данных в память ЭВМ, при этом курсор остается на месте. Readln обеспечивает ввод данных в память ЭВМ, при этом курсор осуществляется перевод курсора на новую строку.

Write предназначена для вывода значений на экран. Writeln выводит значения данных на экран и переводит курсор в начало следующей строки.

Операторы ветвления

Оператор условия If <условие> then <оператор - да> else <оператор - нет>; предназначен для реализации простого ветвления алгоритма.

Пример: Проверим квадратное уравнение на наличие корней. для чего вычислим дискриминант и проверим его знак.

program diskriminant;

var a, b, c, d: real;

begin

writeln(‘Введите коэффициенты квадратного уравнения А, В и С’);

readln(a, b, c);

d:=sqr(b)-4\*a\*c;

if d>0 then writeln(‘Вещественные корни существуют’) else writeln(‘Нет вещественных корней’);

end.

Оператор выбора Case <селектор> of

<значение1>:<оператор1>;

<значение2>:<оператор2>;

………………………………;

<значениеN>:<операторN>

Else <оператор - нет>;

Оператор Case используется при множественном разветвлении алгоритма, например, если необходимо по числовому значению месяца указать время года (1, 2, 12 – зима; 3, 4, 5 – весна и т.д.).

program sezon;

var n: 1..12; {перечисляемый тип, здесь допускает только значения от 1 до 12 }

begin

writeln(‘Введите числовое значение месяца’);

readln(n);

case n of

1, 2, 12: writeln(‘зима’);

3..5: writeln(‘весна’);

6..8: writeln(‘лето’)

else writeln(‘осень’);

end.

Циклы

Циклы (операторы повтора) используются в программе при необходимости неоднократного повторения какого-либо оператора или группы операторов. В Pascal различают три вида циклов: цикл с параметром For, цикл с предусловием While, цикл с постусловием Repeat.

Оператор повтора For применяется, когда заранее известно число повторений. Переменная, хранящая количество повторений называется параметром цикла. Существует две структуры данного оператора.

Счет по возрастанию: For <параметр цикла>:=<целочисленное выражение1 > to < целочисленное выражение2> do <оператор цикла>;

Счет по убыванию: For <параметр цикла>:=<целочисленное выражение2 > downto < целочисленное выражение1> do <оператор цикла>;

Пример: Вывести на экран результат умножения 8 на числа от 4 до 15.

program ymnoj\_8;

var i: byte; {i- параметр цикла}

r: integer;

begin

writeln (‘Таблица умножения для 8’);

for i:=4 to 15 do begin {при выполнении в цикле более одного оператора ставятся операторные скобки Begin end;}

r:=8\*I;

write(r:5); {вывод результата в строку с промежутком в 5 позиций}

end;

end.

Формат цикла с предусловием While <условие> do <тело цикла>; Проверка условия осуществляется перед циклом, цикл выполняется, если условие принимает значение True (истина).

Пример: Посчитать сумму 10 произвольно введенных чисел.

program Summa;

const limit=10; {константа ограничения ввода чисел}

var i: integer;

sum, ch: real;

begin

i:=0; {счетчик чисел}

sum:=0; {переменная накапливающая сумму чисел}

while i< limit do

begin

i:=i+1;

write(‘введите ’,i,’-е число: ’);

readln(ch);

sum:=sum+ch;

end; {конец цикла}

writeln(‘Сумма равна ’,sum:5:2); {переменную sum вещественного типа выводим в 5 позиций с 2мя знаками в дробной части}

end.

Цикл с постусловием Repeat

<тело цикла>

Until <условие>;

Первоначально условие принимает значение False (ложь). Цикл выполняется до тех пор, пока условие не станет True (истина).

Пример: Подсчитать количество чисел, введенных до первого отрицательного.

program Kolichestvo;

var ch: real;

n: byte;

begin

writeln (‘Подсчитать количество чисел, введенных до первого отрицательного’);

writeln; {оформление подсказки}

repeat

write (‘Введите число ’);

readln(ch);

n:=n+1;

until ch<0;

writeln(‘Количество не отрицательных чисел равно ‘,n);

end.

Массивы

Массив – структурированный тип данных, состоящий из фиксированного числа элементов, имеющих один и тот же тип. Элементы, образующие массив, упорядочены таким образом, что каждому элементу соответствует номер (индекс), определяющий его местонахождение в общей последовательности.

Формат записи массива через раздел описания типов имеет вид:

Type <имя типа>=array [тип индекса] of <тип компонента>;

Var <идентификатор,…>: <имя типа>;

Формат записи массива через раздал описания переменных:

Var <идентификатор,…>: array [тип индекса] of <тип компонента>;

Массивы различают по количеству индексов: одномерные (1 индекс), двумерные и N-мерные (N индексов).

Пример: Type Vector = array [1..25] of real; {одномерный массив}

Matrix = array [1..20, 1..30] of byte; {двумерный массив}

Рассмотрим работу с массивами на примере следующей задачи: Даны два массива целых чисел. Вычислить произведение максимального элемента первого массива на минимальный элемент второго массива. Удалить максимальный элемент из первого массива и добавить его во второй массив после минимального.

program mass;

type massiv=array [1..40] of integer;

var a, b: massiv;

i, n, k, p, j, min, Imin, max, Imax: integer;

begin

{ввод массива с клавиатуры}

write('Введите размерность массива А n=');

readln(n);

writeln('Введите элементы массива А');

for i:=1 to n do begin

write('a[',i,']=');

readln(a[i]);

end;

{ввод массива случайным образом}

write('Введите размерность массива В k=');

readln(k);

randomize; {подключение генератора случайный чисел}

for i:=1 to k do

b[i]:=random(16); {заполнение массива случайными числами от 0 до 15}

{Вывод массива на экран}

Writeln('Массив В');

for i:=1 to k do write(b[i],' ');

writeln; {пустой оператор вывода}

{Поиск максимального элемента}

max:=a[1];

Imax:=1;

for i:=2 to n do

if a[i]>max then begin

max:=a[i]; {максимальный элемент}

Imax:=i; {индекс максимального элемента}

end;

{Поиск минимального элемента}

min:=b[1];

Imin:=1;

for i:=2 to n do

if b[i]<min then begin

min:=b[i]; {минимальный элемент}

Imin:=i; {индекс минимального элемента}

end;

{вычисление произведения и его вывод на экран}

p:=min\*max;

writeln('Произведение max и min равно ',p);

{Удаление элемента из массива с позиции Imax}

for i:=Imax to n-1 do

a[i]:=a[i+1];

n:=n-1; {Уменьшение количества элементов массива на 1}

{Вставка элемента в массив после элемента равного min}

i:=1;

while i<k do

begin

if b[i]=min then begin

for j:=k+1 downto i+1 do b[j]:=b[j-1]; {смещение элементов на один вправо, начиная с последнего}

k:=k+1; {Увеличение количества элементов на один}

b[i+1]:=max; {Вставка элемента на позицию i+1}

end;

i:=i+1; {Увеличение счетчика итераций}

end;

{Вывод массивов А и В на экран в строку}

writeln(‘Массив А: ‘);

for i:=1 to n do write(a[i],' ');

writeln;

writeln(‘Массив В: ‘);

for i:=1 to k do write(b[i],' ');

end.

Записи

Запись представляет собой наиболее общий и гибкий структурированный тип данных, так как она может быть организована из не однотипных компонентов и в ней явным образом выражена связь между элементами данных, характеризующими реальный объект. Широко используется при программировании информационно-поисковых систем.

Формат записи:

Type <имя\_типа>=record

<имя\_поля1>: тип;

<имя\_поля2>: тип;

…………………;

<имя\_поляN>: тип

end;

Элементы записи называются полями, а обращение к ним производится через использование их имен – идентификаторов полей. Отличие от обычной переменной записи в том, что имена полей должны предваряться ссылкой на идентификатор записи и отделяться от него точкой. Такая запись называется уточняющий идентификатор: <имя\_записи>.<имя\_поля>

Пример: Описание личных данных учащегося.

Type svedenia=record

f: string[40];

dat\_r: string[8];

pol: char;

klass: string[4]

end;

Задача. Пусть нам необходимо иметь сведения о всех учащихся нашего плавания. А затем из общего списка вывести фамилии учеников, которые учатся в 9 классе.

Для этого целесообразно организовать массив записей, потом отобрать только тех, у которых в поле klass есть цифра 9.

program anketa;

type svedenia=record

f: string[40];

dat\_r: string[8];

pol: char;

klass: string[4]

end;

var ychen: array [1..100] of svedenia;

i, j: integer;

begin

{Последовательно вводим каждую запись}

for i:=1 to 100 do begin

writeln(‘введите сведения о’, i, ‘-м ученике’);

write (‘введите фамилию и имя ’);

readln (ychen [i].f);

write(‘введите дату рождения’);

readln (ychen [i].dat\_r);

write(‘введите класс’);

readln(ychen[i].klass);

end;

writeln(‘ввод закончен’);

writeln;

{Просматриваем массив записей и выбираем только учеников 9-го класса}

for i:=1 to 100 do begin

for j:=1 to Length(ychen [i]. klass) do

if ychen [i]. klass[j]=9 then

writeln(‘фамилия ученика: ’, ychen[i].fio);

end.

Множества

Множество – это структурированный тип данных, представляющий собой набор взаимосвязанных по какому-либо признаку объектов. В отличие от массивов порядок расположения элементов во множестве не важен.

В выражениях на языке Pascal значения элементов множества указываются в квадратных скобках: [1, 6, 3, 7, 2, 4], [‘a’..’z’]. Множество, не имеющее элементов, называется пустым и обозначается [].

Формат записи множественного типа:

Type <имя типа>=set of <элемент1,…, элементN>;

Var <идентификатор>:<имя типа>;

Допустимыми операциями над множествами являются: «=», «<>», «>=», «<=», объединения (+), пересечения (\*), разности (-) множеств и операция включения (in). Операция in позволяет проверить принадлежность значения множеству.

Задача: Посчитать количество гласных и согласных букв в предложении.

program Glasn\_Sogl;

Type mnoj= set of 'A'..'я'; {Задаем множество букв русского алфавита}

var glasn, sogl: mnoj;

sr: string; {строковая переменная sr хранит вводимый текст}

i: byte; {параметр цикла}

g, s: byte; {переменные накапливающие количество гласных и согласных букв соответственно}

begin

{Задаем множества гласных букв перечислением, а множество согласных вычитанием из всего алфавита гласных букв, мягкого и твердого знаков}

glasn:=['A','a','O','o','E','e', 'И','и','Ё','ё','У','у', 'Ы','ы','Э','э','Я','я','Ю','ю'];

sogl:=['A'..'я'] – glasn - 'Ъ' - 'ъ' - 'Ь' - 'ь';

write('Введите предложение: ');

readln(sr);

{Обнуляем счетчики количества}

g:=0;

s:=0;

{Просматриваем все элементы предложения и смотрим, содержаться ли они во множествах glasn и sogl}

For i:=1 to Length(sr) do begin

if sr[i] in glasn then g:=g+1;

if sr[i] in sogl then s:=s+1;

end;

writeln('В данном предложении ',g,' гласных и ',s,' согласных букв');

end.

Процедуры и функции в Pascal

В процедурном программировании основное внимание уделяется алгоритму, то есть некоторой заданной последовательности действий, выполнение которых приводит к получению результата вычислений. Структуризация алгоритмов привела к выделению отдельных блоков (подпрограмм).

Подпрограммы значительно облегчают программирование, за счет (а) избавления от необходимости многократно повторять в тексте программы аналогичные фрагменты; (б) улучшения структуры программы, облегчая ее понимание при разборе; (в) повышения устойчивости программы к ошибкам программирования и непредвиденным последствиям при модификациях.

Различают два вида подпрограмм - процедуры и функции.

Процедура – это независимая именованная часть программы, которую можно вызвать по имени для выполнения определенных действий. Процедура не может выступать как операнд в выражении. Упоминание в тексте имени процедуры приводит к её активизации и называется вызовом.

Функция – аналогично процедуре, с двумя отличиями: (1) функция передает в точку вызова скалярное значение, (2) имя функции может использоваться в выражении как операнд.

Подпрограммы в Pascal могут обращаться сами к себе. Такое обращение называется рекурсией.

Структура процедур и функций полностью повторяет структуру самой программы, кроме заголовка. Заголовки подпрограмм оформляются следующим образом:

Procedure <имя\_процедуры> (<список формальных параметров>);

Function <имя\_функции> (<список формальных параметров>): <тип результата>;

Все подпрограммы располагаются выше начала основной части программы (перед первым begin).

Рассмотрим примеры подпрограмм:

Задача: Организация ввода координат вектора. Вычислить длину вектора. Для начала определим список формальных параметров: входные и выходные данные. Нам потребуется Размерность векторного пространства (k: byte) и переменная х пользовательского типа vector=array[1..100] of real.

Для ввода координат будем использовать procedure, так как выходных данных будет много (его координаты), а так как длина вектора это число, поэтому используем function.

program pro;

type vector=array [1..100] of real;

var k: byte;

dl:real;

x: vector;

procedure vvod (var y:vector); {Процедура ввода вектора}

var i:byte;

begin

writeln('Введите координаты вектора');

for i:=1 to k do

readln(y[i]);

end;

function dlvec(y:vector):real; {Функция вычисления длины вектора}

var i:byte;

s:real;

begin

for i:=1 to k do

s:= s+ sqr(y[i]);

dlvec:= sqrt(s); {Обязательно в конце нужно имени функции присвоить вычисленное значение}

end;

begin

write(' Введите размерность векторного пространства k=');

readln(k);

vvod(x); {вызов процедуры}

dl:=dlvec(x); {вызов функции}

writeln('Длина вектора Х равна', dl: 8: 2);

end.

Модульное программирование

Модульное программирование – это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определенным правилам.

Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок. Аппаратно-зависимые подзадачи могут быть строго отделены от других подзадач, что улучшает мобильность создаваемых программ.

Модуль – это автономно компилируемая программная единица, включающая в себя различные компоненты раздела описаний (типы, константы, переменные, процедуры и функции) и, возможно, некоторые исполняемые операторы инициирующей части.

Модуль начинается с зарезервированного слова Unit. Модуль не может быть запущен на выполнение!!! Лучше компиляцию модуля в среде Turbo Pascal осуществлять через режим Make (F9).

Любой модуль имеет следующую структуру:

Unit <имя\_модуля>;

interface

<интерфейсная часть>;

implementation

<исполняемая часть>;

begin

<инициирующая часть>;

end.

В разделе interface описывается взаимодействие данного модуля с другими пользовательскими и стандартными модулями, а также с главной программой. В разделе реализации implementation указывается исполняемая (внутренняя) часть описаний данного модуля, которая недоступна для других модулей.

Модули делят на стандартные и пользовательские. Пользовательские модули создаются пользователем. В среде Turbo Pascal 7.0 имеется 8 стандартных модулей, в которых содержится множество различных типов, констант, процедур и функций. Этими модулями являются system, dos, crt, graph, overlay, turbo3, graph3. Лишь один модуль system подключается к любой программе автоматически, все остальные становятся доступны только после указания их имен в списке подключаемых модулей Uses.

Рассмотрим пример реализации модуля с операциями над векторами. Вектор опишем как массив вещественных чисел (его координаты). Реализуем в модуле операции над векторами: ввод вектора, вывод вектора, сумма векторов, разность векторов, умножение вектора на число, скалярное произведение векторов, вычисление длины вектора.

unit vectory; {Заголовок модуля, имя модуля должно совпадать с именем файла, в котором храниться модуль }

{Интерфейсная часть модуля}

interface

const k=3;

type vector=array [1..100] of real; {Описание типа Vector}

{Описание процедур и функций}

procedure vvod (var x: vector);

procedure add (x, y: vector; var z: vector);

procedure minis (x, y: vector; var z: vector);

procedure ymnchislo(x: vector; t: real; var z: vector);

function scalayr (x, y: vector):real;

function dlvec(x: vector):real;

procedure vivod(x: vector);

{Раздел реализации}

implementation

procedure vvod; {Процедура ввода вектора}

var i:byte;

begin

writeln('Введите координаты вектора ');

for i:=1 to k do

readln(x[i]);

end;

procedure add; {Процедура суммы двух векторов}

var i: byte;

begin

for i:=1 to k do

z[i]:=x[i]+y[i];

end;

procedure minis; {Процедура разности двух векторов}

var i: byte;

begin

for i:=1 to k do

z[i]:=x[i]-y[i];

end;

procedure ymnchislo; {Процедура умножения вектора на число}

var i: byte;

begin

for i:=1 to k do

z[i]:=t\*x[i];

end;

function scalayr; {Функция скалярного произведения двух ветров}

var i: byte;

sc: real;

begin

sc:=0;

for i:=1 to k do

sc:=sc + x[i]\*y[i];

scalayr:=sc;

end;

function dlvec; {Функция вычисления длины вектора}

var i: byte;

s: real;

begin

for i:=1 to k do

s:= s+ sqrt(x[i]);

dlvec:= sqrt(s);

end;

procedure vivod; {Процедура вывода вектора на экран}

var i: byte;

begin

write(‘Координаты вектора: ');

for i:=1 to k do

write(x[i]:8:2);

writeln;

end;

end.

Файлы

В языке программирования Pascal предусмотрены специальные объекты (файлы), которые позволяют организовывать хранение информации на внешних запоминающих устройствах и доступ к этой информации.

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

Целесообразность применения файлов диктуются следующими причинами (а) ввод больших объемов данных, подлежащих обработке, утомителен и требует большого времени. Гораздо удобнее создать определенный файл данных, который может быть подготовлен заранее и, самое главное, применяться неоднократно. (б) Файл данных может быть подготовлен другой программой, становясь, таким образом, связующим звеном между двумя разными задачами, а также средством связи программы с внешней средой. (в) Программа, использующая данные из файла, не требует присутствия пользователя в момент фактического исполнения.

Формат записи файла:

Type <имя типа>=<тип компонентов>;

var <F> : File of <имя типа>; {F – файловая переменная, представитель файла в паскале}

<R> : <имя типа>; {переменная доступа к полям записи}

Средства обработки файлов:

Процедура Assign(<Файловая переменная>; <полный путь к файлу >) – связывает файловую переменную с конкретным файлом на внешнем устройстве.

Процедура Reset(<файловая переменная>) – открывает уже существующий файл.

Процедура Rewrite(<файловая переменная>) – создает и открывает новый файл.

Процедура Close(<файловая переменная >) – закрывает открытый файл.

Процедура Rename(<файловая переменная >;<новое имя файла>) – переименование любого неоткрытого файла.

Процедура Erase (<файловая переменная >) – удаление неоткрытого файла.

Условно файлы можно разделить на текстовые, типизированные и нетипизированные.

Текстовый файл – последовательность символов, разбитая на строки длиной от 0 до 256 символов. Для описания используется стандартный тип Text:

var F: text;

К типизированным файлам относят файлы строго определенного типа. Чаще всего это файлы, состоящие из записи.

Type FR= record

………

end;

var F: file of FR;

Нетипизированные файлы рассматриваются в Pascal как совокупность символов или байтов. для определения в программе нетипизированного файла служит зарезервированное слово File:

var F: file;

Пример: Прочитать последовательность 6 символов из первой строки текстового файла Input.txt, записать их в обратном порядке в файл Output.txt. Файл Input.txt создан на c:\temp\

program text\_file;

var F, R: text;

st, ts: string[6];

i: byte;

begin

assign(F, 'c:\temp\Input.txt'); {связывает переменную F с файлом Input.txt}

reset(F); {открывает F для чтения}

assign(R, 'c:\temp\Output.txt'); {связывает переменную R с файлом Output.txt}

rewrite(R); {создает и открывает R для записи}

while not Eoln(F) do read(F, st); {Проверка конца файла, чтение из F в переменную st}

for i:= 6 downto 1 do ts:=ts+st[i]; {Создание строки перевертыша}

writeln(R, ts); {Запись в R значения переменной ts}

close(F); {закрывает F}

close(R); {закрывает R}

end.