Московский государственный технический университет
имени Н. Э. Баумана
Факультет Информатика и системы управления
Кафедра Компьютерные системы и сети

«УТВЕРЖДАЮ»
Заведующий кафедрой ИУ-6
_____ Сюзев В.В.

Г. С. Иванова, Т.Н. Ничушкина

Основы алгоритмизации

Методические указания по выполнению лабораторных работ № 2 - 9
по дисциплине Основы программирования

Москва 2013

Содержание

Лабораторная работа № 2

Программирование с использованием ветвлений

Цель работы: изучение операторов организации ветвления, приемов создания программ, обеспечивающих выполнение альтернативных действий

Объем работы: 2 часа

Теоретическая часть

Ветвлением в программировании называют конструкцию обеспечивающую выполнение различных действий в зависимости от результатов проверки некоторого условия.

Для программирования ветвлений, т. е. ситуаций, когда возникает необходимость при выполнении условия реализовывать одни действия, а при нарушении – другие используют оператор условной передачи управления (рис. 1) Условие записывают в виде логического выражения, в зависимости от результата которого осуществляется выбор одной из ветвей: если результат выражения true, то выполняется оператор, следующий за служебным словом then, иначе – оператор, следующий за служебным словом else.

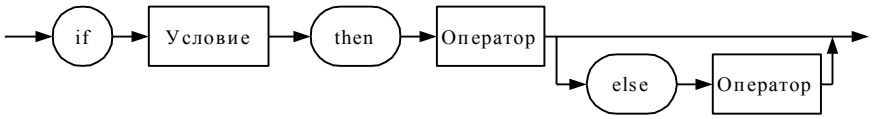


Рис. 1. Синтаксическая диаграмма <Оператор условной передачи управления>

В каждой ветви допускается запись одного оператора, в том числе и другого условного оператора. Если по алгоритму решения задачи необходимо в ветвях размещать более одного оператора, используют составные операторы.

Составным оператором в Delphi Pascal называют последовательность операторов, заключенную в операторные скобки begin…end. Операторы последовательности отделяют друг от друга точкой с запятой «;». Перед end точку с запятой можно не ставить. Перед else точка с запятой в операторе if не ставится никогда, поскольку в этом случае запись условного оператора продолжается.

В соответствии с синтаксической диаграммой допускается использовать оператор условной передачи управления с пропущенной (пустой) ветвью else.

Пример. Разработать программу, которая вычисляет значение функции, заданной следующим образом:

$$y = \begin{cases} |x|, & \text{при } |x| \le 1; \\ x^2, & \text{при } 1 < |x| \le 2; \\ 4, & \text{иначе.} \end{cases}$$

Программа должна начинаться с ввода значения аргумента. Затем проверяем введенное значение, и в зависимости от того, в какой интервал оно попадает, вычисляем значение функции по одному из заданных выражений. Для того чтобы проверить попадание в три указанных диапазона необходимо выполнить две проверки.

Алгоритм решения данной задачи представлен на рис. 2.

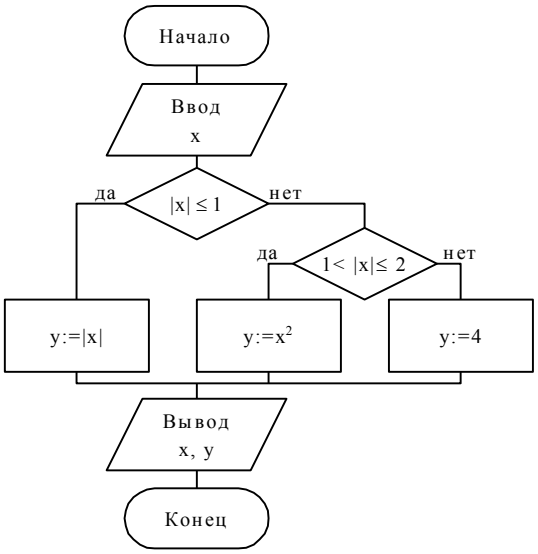


Рис. 2. Схема алгоритма программы вычисления функции, заданной на отрезках, в заданной точке

Текст программы имеет следующий вид:

```
Program ex;
 {$APPTYPE CONSOLE}
 uses SysUtils;
 Var x,y:real;
 Begin
     Writeln('Enter x:');
     readln(x);
  if abs(x)<=1 then y:=abs(x)    {первый отрезок}
     else
         if abs(x)<=2 then
                 y:=sqr(x)        {второй отрезок}
         else      y:=4;           {третий отрезок}
     WriteLn('For x=',x:8:5,' y=',y:8:5);
     ReadLn;
 End.
```

Пример. Разработать программу, которая строит список по типу стека из целых чисел, вводимых с клавиатуры. Количество чисел не известно, но отлично от нуля. Конец ввода – по комбинации клавиш CTRL-Z (конец файла на устройстве ввода).

Обычно построение списка по типу стека выполняется в два этапа: в список заносится первый элемент, а затем организуется цикл добавления элементов перед первым:

```
ReadLn(a);
new(first);        {запрашиваем память под элемент}
first^.num:=a;  {заносим число в информационное поле}
```

```
first^.p:=nil; {записываем nil в поле «адрес следующего»}
while not EOF do
   begin
     ReadLn(a);
     new(q);       {запрашиваем память под элемент}
     q^.num:=a;  {заносим число в информационное поле}
     q^.p:=first; {в поле «адрес следующего»
                        переписываем адрес первого элемента}
     first:=q;   {в указатель списка заносим адрес нового
                                                  элемента}
   end; ...
```

Другие примеры программ, содержащих ветвления, приведены в [1].

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Разработать схему алгоритма решения задачи.
3. Написать программу.
4. Вызвать среду программирования Turbo Delphi, создать новый проект консольного приложения и ввести текст программы в среду программирования.
5. Подобрать тестовые данные (не менее 3-х вариантов).
6. Отладить программу на выбранных тестовых данных.
7. Продемонстрировать работу программы преподавателю.
8. Составить отчет по лабораторной работе.
9. Защитить лабораторную работу преподавателю.

Требования к отчету

Отчет должен быть выполнен на бумаге формата А4 или А5 в том числе в тетрадях или на тетрадных листах. Если отчет выполняется на отдельных тетрадных листах, то они должны быть аккуратно обрезаны по линии подшивки и скреплены. Неаккуратно выполненные, оборванные или грязные отчеты не принимаются.

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема лабораторной работы;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента;

е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать**:**

1) схему алгоритма, выполненную вручную или в соответствующем пакете;
2) текст программы;
3) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

4) выводы.

Контрольные вопросы

1. Что такое «ветвление»? В каких случаях используется эта конструкция?
2. Как показать ветвление в схеме алгоритма?
3. Какой оператор реализует ветвление в программе?
4. Какой синтаксис имеет этот оператор?
5. Объясните, почему в вашей программе следует использовать ветвление?
6. Как подбирают тесты для отладки программ, содержащих ветвления?

Лабораторная работа № 3

Программирование с использованием циклов

Цель работы: изучение операторов организации циклов, приемов создания программ, обеспечивающих выполнение циклических процессов

Объем работы: 4 часа

Теоретическая часть

Циклическая структура процесса вычислений определяет, что для получения результата некоторые действия необходимо выполнить несколько раз.

Для реализации циклических процессов используют операторы циклов. В теории программирования выделяют несколько основных видов циклических конструкций:

- счетный цикл (рис. 3,*а*);
- цикл-пока (рис. 3, *б*);
- цикл-до (рис. 3, *в*).

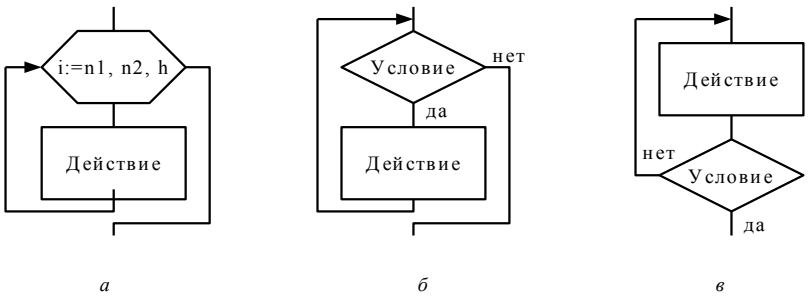


Рис. 3 Структура циклов, реализованных в Delphi Pascal:

счетный цикл (***а***), цикл-пока (***б***) и цикл-до (***в***)

Счетный цикл. Цикл выполняется, пока переменная – параметр цикла принимает значения в заданном диапазоне с единичным шагом. Синтаксическая диаграмма оператора приведена на рис. 4.

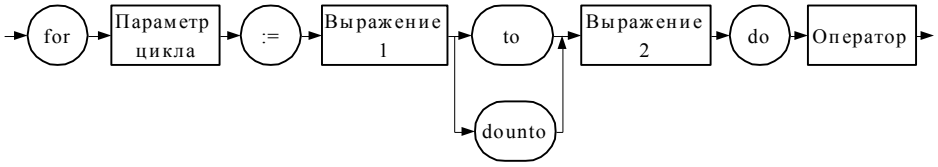


Рис. 4. Синтаксическая диаграмма <Цикл с заданным количеством повторений>

Переменная цикла должна иметь порядковый тип, т.е. она может быть целого, логического или символьного типов. Выражение 1 определяет начальное значение этой переменной, выражение 2 – ее конечное значение. Соответственно начальное и конечное значения должны принадлежать к тому же типу, что и переменная цикла. Если

используется служебное слово to, то при каждом выполнении цикла переменной цикла присваивается следующее значение порядкового типа переменной. Если используется служебное слово downto, то при каждом выполнении цикла переменной цикла присваивается предыдущее значение порядкового типа переменной. Если диапазон значений переменной цикла пуст, то цикл не выполняется.

Пример. Разработать программу вычисления суммы n первых натуральных чисел.

Сумма определяется *методом накопления*. Количество суммируемых чисел известно, поэтому используем цикл с заданным количеством повторений. При каждом проходе к сумме будем добавлять переменную цикла, которая будет изменяться от 1 до n. Перед циклом переменную суммы необходимо обнулить. На рис. 5 приведена схема алгоритма программы.

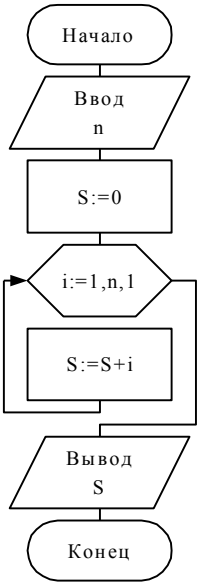


Рис. 5. Алгоритм вычисления суммы n натуральных чисел

Текст программы приведен ниже.

```
Program ex;
  {$APPTYPE CONSOLE}
   Uses   SysUtils;
   Var  i, n, S:integer;
   Begin
        Writeln('Enter n');
        ReadLn(n);
        S:=0;
        for i:=1 to n do S:=S+i;
        Writeln('The sum is equal ', S);
        ReadLn;
   End.
```

Цикл-пока. Синтаксическая диаграмма оператора, реализующего цикл-пока приведена на рис. 6. Условие записывают в виде логического выражения. Оператор тела цикла повторяется, пока условие истинно. Проверка осуществляется на входе в цикл и при каждом повторении. Если при входе в цикл условие не выполняется, то оператор тела цикла игнорируется.

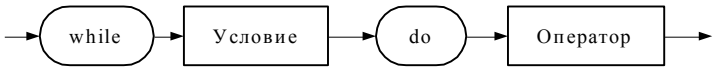


Рис. 6. Синтаксическая диаграмма оператора <Цикл-пока>

Если в тело цикла необходимо поместить несколько операторов, то используют составной оператор.

Цикл-до. Операторы тела цикла повторяются до выполнения условия, условие проверяется на выходе каждый раз после выполнения тела цикла. Синтаксическая диаграмма оператора «цикл-до» приведена на рис. 7. В тело цикла можно поместить несколько операторов, разделив их точкой с запятой «;». В отличие от цикла-пока в цикле-до операторы тела цикла всегда выполняются хотя бы один раз. Кроме того, конструкция цикла-до позволяет записывать в теле цикла произвольное количество разделенных точкой с запятой «;» операторов, не используя составных операторов.

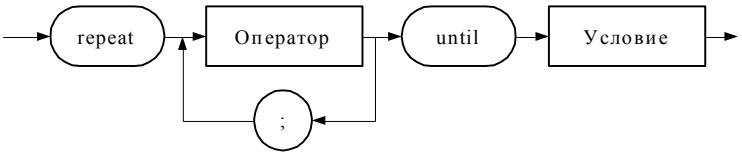


Рис. 7. Синтаксическая диаграмма оператора <Цикл-до>

Пример. Разработать программу, определяющую сумму ряда $1 - 1/x + 1/x^2 - 1/x^3 + \ldots$ с заданной точностью ε.

На рис. 8 представлен алгоритм программы. Анализ алгоритма показывает, что он *неструктурный*, так как в нем присутствует цикл суммирования, который не является ни циклом-пока, ни циклом-до, ни циклом с заданным количеством повторений. Для реализации данного алгоритма на языке необходимо его преобразовать в структурный, чтобы можно было использовать один из имеющихся операторов цикла.

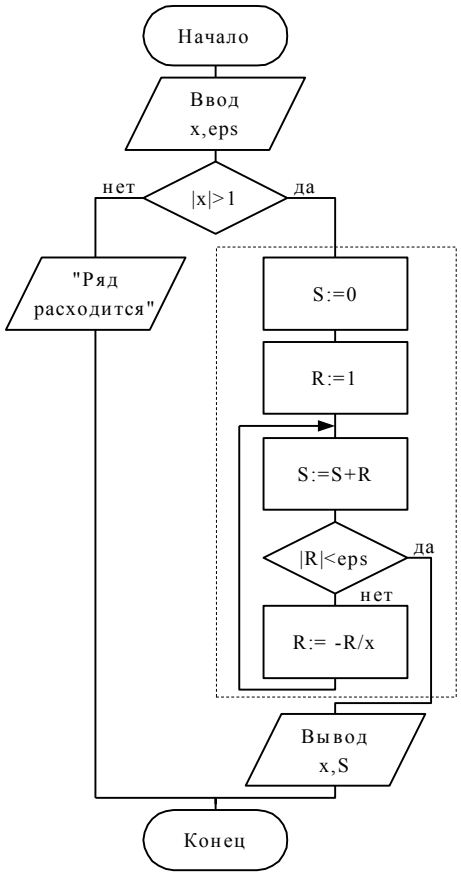


Рис. 8. Разработка алгоритмов определения суммы ряда с заданной точностью

Поскольку количество повторений цикла определить не удается, попробуем преобразовать неструктурный цикл к циклу-пока. Для этого необходимо операцию S=S+R продублировать: одну копию поместить до цикла, а вторую – в конце цикла.

Операторы S=0 и S=S+R, записанные до цикла, заменим оператором S=1, так как в этот момент R=1. Условие выхода из цикла также необходимо заменить на противоположное. Окончательный вариант фрагмента алгоритма с ц и к л о м - п о к а показан на рис. 9, *а*.

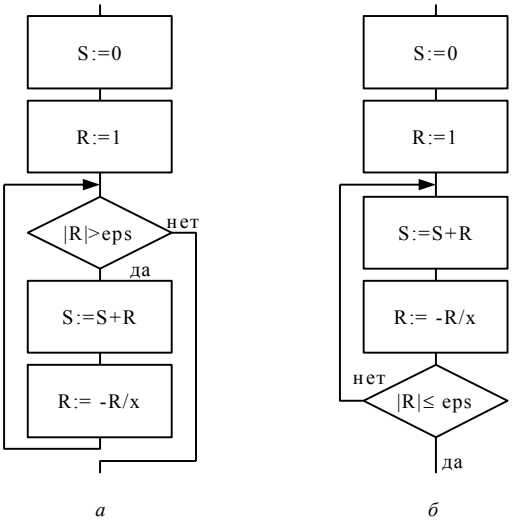


Рис. 9. Структурные варианты алгоритма:

с использованием цикла-пока (*а*) и с использованием цикла-до (*б*)

Его реализация представлена ниже:

```
Program ex;
 {$APPTYPE CONSOLE}
 Uses   SysUtils;
 Var S,R,X,eps:real;
 Begin
    WriteLn('Enter x and epsilon:');
    ReadLn(X, eps);
    if abs(x)>1 then  {если x > 1, то ищем сумму ряда}
      begin
        S:=1;  R:=1;
        while abs(R)>eps do
           begin
             R:=-R/X;
             S:=S+R;
           end;
        WriteLn('If x=',x:6:2,' then S=',S:8:2,
                                ' and R=', R:8:6)
      end
    else Writeln('A series misses.');
                                          {Ряд расходится}
 End.
```

Тот же алгоритм можно преобразовать так, чтобы цикл можно было реализовать с использованием ц и к л а - д о.

.

```
Program ex;
 {$APPTYPE CONSOLE}
 Uses   SysUtils;
 Var S,R,X,eps:real;
 Begin
   Writeln('Enter x and epsilon:');
   ReadLn(X,eps);
   if x>1 then
     begin
```

```
      S:=0;   R:=1;
      repeat
        S:=S+R;
        R:=-R/X
      until abs(R)<=eps;
      WriteLn('If x=',x:6:2,' S=',S:8:2,
                              ' and R=',R:8:6)
    end
  else Writeln('A series misses.');
                                    {Ряд расходится}
  ReadLn;
End.
```

Другие примеры программ, содержащих циклы, приведены в [1].

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Разработать схему алгоритма решения задачи.
3. Написать программу.
4. Вызвать среду программирования Turbo Delphi, создать новый проект консольного приложения и ввести текст программы в среду программирования.
5. Подобрать тестовые данные (не менее 3-х вариантов).
6. Отладить программу на выбранных тестовых данных.
7. Продемонстрировать работу программы преподавателю.
8. Составить отчет по лабораторной работе.
9. Защитить лабораторную работу преподавателю.

Требования к отчету

Отчет должен быть выполнен на бумаге формата А4 или А5 в том числе в тетрадях или на тетрадных листах. Если отчет выполняется на отдельных тетрадных листах, то они должны быть аккуратно обрезаны по линии подшивки и скреплены. Неаккуратно выполненные, оборванные или грязные отчеты не принимаются.

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема лабораторной работы;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента;

е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать**:**

1) схему алгоритма, выполненную вручную или в соответствующем пакете, для решения вашей задачи с одним из трех циклов;
2) текст программы для лучшего с вашей точки зрения варианта;
3) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

4) выводы.

Контрольные вопросы

1. Что такое «цикл»? В каких случаях используется эта конструкция?
2. Как показать циклический процесс в схеме алгоритма?
3. Какие операторы реализуют ветвление в программе?
4. Какой синтаксис имеют эти операторы?
5. Объясните, почему в вашей программе следует использовать цикл?
6. Почему вы выбрали конкретный тип цикла?

Лабораторная работа № 4

Обработка одномерных массивов

Цель работы: изучение приемов обработки одномерных массивов

Объем работы: 2 часов

Теоретическая часть

Массив – это упорядоченная совокупность однотипных данных. Этот тип используют для представления табличных данных. Каждому элементу массива соответствует один или несколько *индексов*, определяющих положение элемента в массиве. Синтаксическая диаграмма объявления массива представлена на рис. 4.1.

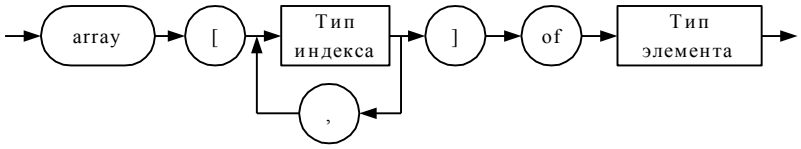


Рис. 1. Синтаксическая диаграмма <Объявление массива>

Тип индекса определяет его допустимые значения. В качестве типа индекса может быть указан любой *порядковый* тип (Boolean, Char, Integer, Word, Byte, перечисляемый тип и т. п., а также диапазоны этих типов), кроме типа Longint и его производных.

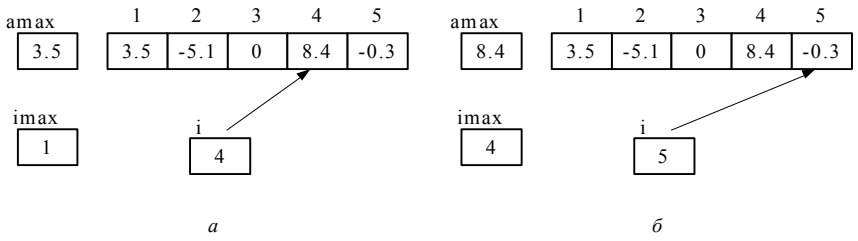
Ограничения на количество индексов в Delphi Pascal нет. Однако суммарная длина массива не должна превышать 2 Гбайт.

Пример. Разработать программу определения максимального элемента массива A(5) и его номера.

Вначале элементы массива необходимо ввести. Для выполнения этой операции используем цикл с заданным числом повторений.

Поиск максимального элемента требует дополнительных переменных: amax – для хранения текущего и найденного максимального значения и imax – для хранения номера текущего и найденного максимального значения. Саму операцию поиска выполним следующим образом.

Запомним в качестве текущего максимального, т. е. запишем в amax первый элемент и зафиксируем в imax его номер. Затем будем последовательно просматривать элементы массива, сравнивая их со значением, хранящимся в amax. Если очередной элемент больше значения в amax, то сохраняем его в качестве максимального в amax и запоминаем его номер в imax (рис. 2).



а *б*

Рис. 2. Поиск максимального элемента массива: состояние на момент проверки четвертого элемента массива (*а*), изменение текущего значения максимального элемента и его номера по результатам проверки четвертого элемента массива (*б*)

Для организации последовательного просмотра используем цикл с заданным числом повторений, изменяя переменную цикла от 2 до 5, так как первый элемент мы уже учли. Просмотрев все элементы массива, найдем максимальный элемент и его номер. После этого необходимо вывести на экран исходный массив, максимальный элемент и его номер.

На рис. 3 представлена схема алгоритма программы. Поскольку операции ввода-вывода массивов выполняют однотипно, на схеме алгоритма соответствующих циклов, так же как и запросов на ввод данных, обычно не показывают. Вместо этого в схему вставляют блок операции ввода/вывода, в котором указано имя массива и количество элементов, участвующих в операции.

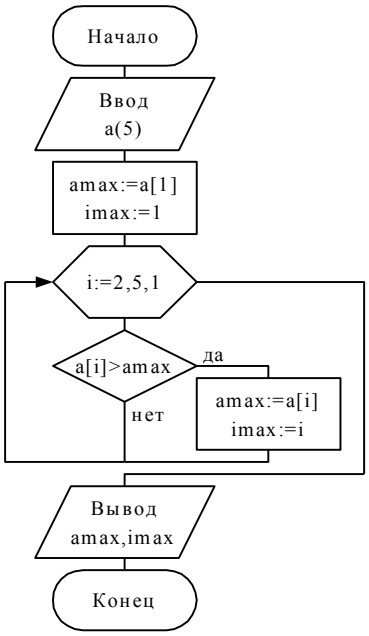


Рис. 3. Схема алгоритма поиска максимального элемента массива

Ниже приведен текст программы.

```
Program ex;
 {$APPTYPE CONSOLE}
 Uses   SysUtils;
 Var a:array[1..5] of real;
```

```
    amax:real;
    i, imax:byte;
  Begin
    {запрос на ввод массива}
    WriteLn('Enter 5 numbers:');
     {ввод элементов массива}
    for i:=1 to 5 do Read(a[i]);
    ReadLn;
     {поиск максимального элемента}
    amax:=a[1];
    imax:=1;
    for i:=2 to 5 do
       if a[i]>amax then
           begin
              amax:=a[i];
              imax:=i;
           end;
     {вывод массива}
    WriteLn('Data: ');
    for i:=1 to 5 do Write(a[i]:5:2);
    WriteLn;
    {вывод результата}
    WriteLn('Max value ',amax:5:2,
                  ' and it''s number ', imax);
    ReadLn;
 End.
```

Другие примеры программ, содержащих обработку массивов, приведены в [1].

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Разработать схему алгоритма решения задачи.
3. Написать программу.

4. Вызвать среду программирования Turbo Delphi, создать новый проект консольного приложения и ввести текст программы в среду программирования.

5. Подобрать тестовые данные (не менее 3-х вариантов).

6. Отладить программу на выбранных тестовых данных.

7. Продемонстрировать работу программы преподавателю.

8. Составить отчет по лабораторной.

9. Защитить лабораторную работу преподавателю.

Требования к отчету

Отчет должен быть выполнен на бумаге формата А4 или А5 в том числе в тетрадях или на тетрадных листах. Если отчет выполняется на отдельных тетрадных листах, то они должны быть аккуратно обрезаны по линии подшивки и скреплены. Неаккуратно выполненные, оборванные или грязные отчеты не принимаются.

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема лабораторной работы;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента;

е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать**:**

1) схему алгоритма, выполненную вручную или в соответствующем пакете;
2) текст программы;
3) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

4) выводы.

Контрольные вопросы

1. Что такое «массив»? В каких случаях используется эта структура данных?
2. Как показать операции с массивом на схеме алгоритма?
3. Какой объявить массив в программе?
4. Как осуществляется ввод и вывод элементов массива?
5. Объясните, почему в вашей программе следует использовать массив?
6. Как подбирают тесты для отладки программ, содержащих массивы? Какие недостатки и достоинства имеет использование для заполнения массивов датчиков случайных чисел?

Лабораторная работа № 5

Обработка матриц

Цель работы: изучение приемов обработки матриц в программах

Объем работы: 4 часов

Теоретическая часть

Матрицей называют двумерный массив, т.е. массив, для которого объявлены два индекса.

Пример. Разработать программу вычисления сумм элементов строк матрицы A(4,5). Полученные суммы записать в новый массив B.

Итак, задана матрица, имеющая 4 строки и 5 столбцов (рис. 1, *а*). Требуется сформировать одномерный массив B из четырех элементов, который будет содержать суммы элементов строк (рис. 1, *б*). Распечатать результат лучше так, чтобы суммы были выведены после соответствующей строки матрицы, как на рис. 1.

A

	1	2	3	4	5
1	3.1	5.7	8.1	-0.7	3.6
2	4.3	6.8	-0.3	5.7	9.2
3	6.4	2.7	5.5	-5.3	2.7
4	5.1	-2.7	7.7	1.7	5.1

а

B

1	19.8
2	25.7
3	12.0
4	16.9

б

Рис. 1. Исходные данные (а) и результат (б) примера

Программа должна начинаться со ввода матрицы. Основной цикл программы – цикл по строкам. Переменная цикла i в этом цикле будет изменяться от 1 до 4. Для каждой i-й строки в этом цикле должно выполняться суммирование элементов. Суммирование будем осуществлять методом накопления, для чего перед суммированием обнулим соответствующий i-й элемент массива B, а затем в цикле выполним добавление элементов строки. После завершения цикла суммирования эту строку и ее сумму можно сразу выводить. На рис. 2 представлена схема алгоритма программы (пунктиром выделено суммирование элементов i-й строки).

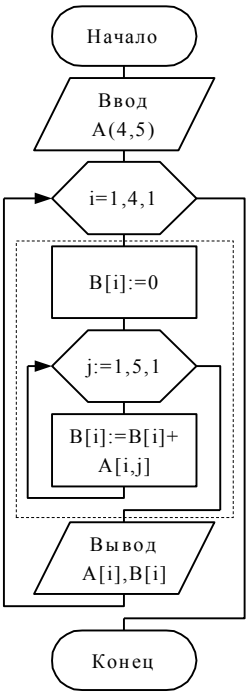


Рис. 2. Схема алгоритма программы нахождения сумм элементов строк (пунктиром выделено суммирование элементов i-ой строки)

Ниже приведен текст программы.

```
Program ex;
  {$APPTYPE CONSOLE}
  Uses   SysUtils;
 Var A:array[1..4,1..5] of real;
     B:array[1..4] of real;
     i,j:byte;
 Begin
    WriteLn('Enter a matrix in the lines:');
    for i:=1 to 4 do    {вводим матрицу}
          begin
             for j:=1 to 5 do Read(A[i,j]);
             ReadLn;
          end;
     WriteLn('Results:');
     for i:=1 to 4 do   {для каждой строки}
       begin
         B[i]:=0;   {обнуляем накапливаемую сумму}
         for j:=1 to 5 do B[i]:=B[i]+A[i,j];
                          {суммируем элементы строки}
```

```
        for j:=1 to 5 do Write(A[i,j]:7:2);
                                        {выводим строку}
        WriteLn('The sum is equal ',B[i]:7:2);
                                        {выводим сумму}
      end;
    ReadLn;
  End.
```

Другие примеры программ, содержащих обработку матриц, приведены в [1].

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Разработать схему алгоритма решения задачи.
3. Написать программу.
4. Вызвать среду программирования Turbo Delphi, создать новый проект консольного приложения и ввести текст программы в среду программирования.
5. Подобрать тестовые данные (не менее 3-х вариантов).
6. Отладить программу на выбранных тестовых данных.
7. Продемонстрировать работу программы преподавателю.
8. Составить отчет по лабораторной работе.
9. Защитить лабораторную работу преподавателю.

Требования к отчету

Отчет должен быть выполнен на бумаге формата А4 или А5 в том числе в тетрадях или на тетрадных листах. Если отчет выполняется на отдельных тетрадных листах, то они должны быть аккуратно обрезаны по линии подшивки и скреплены. Неаккуратно выполненные, оборванные или грязные отчеты не принимаются.

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема лабораторной работы;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента;

е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать**:**

1) схему алгоритма, выполненную вручную или в соответствующем пакете;
2) текст программы;
3) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

4) выводы.

Контрольные вопросы

1. Что такое «матрица»? В каких случаях используется эта структура данных?
2. Как показать обработку матриц в схеме алгоритма?
3. Какие приемы обработки матриц вы знаете?
4. Какой синтаксис имеет описание матриц?
5. Объясните, почему в вашей программе следует использовать матрицу?
6. Как подбирают тесты для отладки программ, содержащих обработку матриц?

Лабораторная работа № 6

Обработка строк

Цель работы: изучение приемов обработки текстовой информации

Объем работы: 2 часов

Теоретическая часть

Для упрощения работы с текстами в Delphi Pascal существует специальный тип данных – *строка*, который приспособлен для обработки символьной информации.

Синтаксическая диаграмма объявления строкового типа данных представлена на рис. 1.

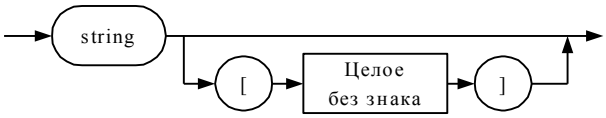


Рис. 1. Синтаксическая диаграмма <Объявление строкового типа>

Целое без знака – это максимальная длина строки, которая не должна превышать 255 байт. Если длина не указана, то по умолчанию принимается максимальное значение – 255 символов.

Внутреннее представление строки показано на рис. 2, откуда видно, что строка представляет собой одномерный символьный массив, индексы которого изменяются от 0 до максимального значения, указанного при объявлении строкового типа. Следовательно, физическая длина строки на единицу превышает максимальную.

Рис. 2. Внутреннее представление строки

Ввод-вывод переменных строкового типа осуществляется одной операцией Read (ReadLn) или Write (WriteLn), например:

```
ReadLn(S1);
WriteLn(S1);
```

При вводе за строку принимается последовательность символов до кода клавиши ENTER. Если длина введенной строки больше указанной максимальной длины, то лишние

символы отбрасываются, а в нулевой байт записывается значение максимальной длины. В противном случае в нулевой байт записывается количество введенных символов. Поскольку строкой считаются все символы до кода клавиши ENTER, ввести в одной строке строковое значение, а затем, например, число нельзя.

Если при вводе строки просто нажать клавишу Enter, не вводя никаких символов, то считается, что введена пустая строка.

Все основные действия над строками и символами реализуют с помощью стандартных процедур и функций.

Пример. Дана строка не более 40 символов, состоящая из слов, разделенных пробелами. Разработать программу удаления «лишних» пробелов. Лишними считать пробелы в начале строки до первого символа, второй и более пробелы между словами и пробелы в конце строки.

При решении данной задачи с использованием строкового типа отпадает необходимость посимвольного анализа строки. Функция Pos, которой в качестве подстроки заданы два пробела подряд, позволит определить все места в строке, где записаны несколько пробелов подряд. Поочередно удалив лишние пробелы, получим строку, в которой останется только проверить и при необходимости удалить пробел в начале и пробел в конце (рис. 3).

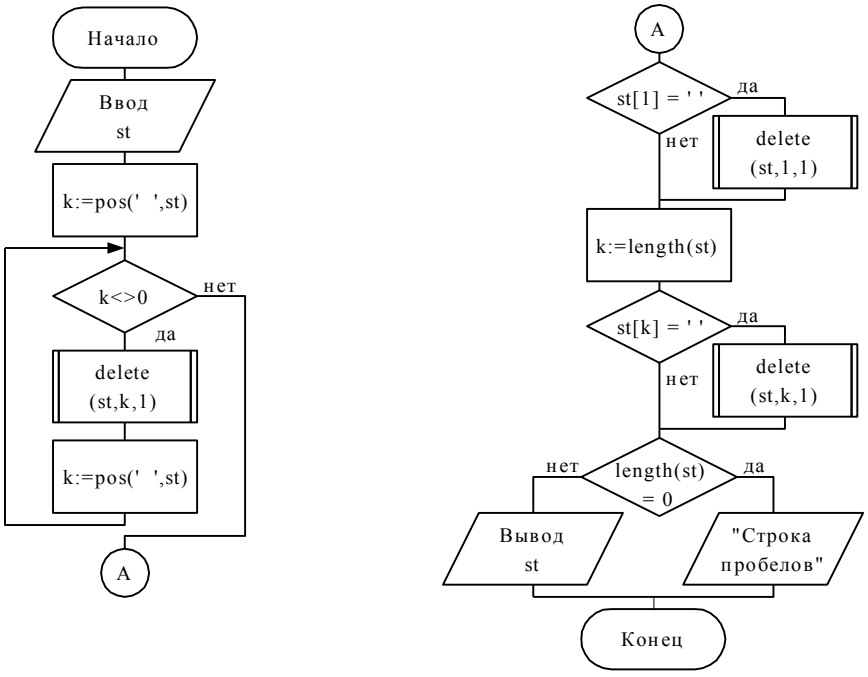


Рис. 3. Схема алгоритма программы удаления «лишних» пробелов

Ниже приведен текст программы.

```
Program ex;
  {$APPTYPE CONSOLE}
```

```
  Uses   SysUtils;
Var  st:string[40];
     k:byte;
Begin
   WriteLn('Enter string <= 40 символов');
   ReadLn(st);
   Write('String:');
   WriteLn(st);
   k:=Pos('  ',st);   {ищем сдвоенные пробелы}
   while k<>0 do  {пока есть сдвоенные пробелы}
     begin
       Delete(st,k,1);    {удаляем первый пробел}
       k:=pos('  ',st);   {ищем сдвоенные пробелы}
     end;
   if st[1]=' '  then
          Delete(st,1,1);   {удалили пробел в начале}
   k:=Length(st);
   if st[k]=' ' then
          Delete(st,k,1); {удалили пробел в конце}
   WriteLn('Result:');
   if length(st)<>0 then writeln(st)
   else WriteLn('Only spaces are in string.');
   ReadLn;
 end.
```

Другие примеры программ, содержащих обработку строк приведены в [1].

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.

2. Разработать схему алгоритма решения задачи.

3. Написать программу.

4. Вызвать среду программирования Turbo Delphi, создать новый проект консольного приложения и ввести текст программы в среду программирования.

5. Подобрать тестовые данные (не менее 3-х вариантов).
6. Отладить программу на выбранных тестовых данных.
7. Продемонстрировать работу программы преподавателю.
8. Составить отчет по лабораторной.
9. Защитить лабораторную работу преподавателю.

Требования к отчету

Отчет должен быть выполнен на бумаге формата А4 или А5 в том числе в тетрадях или на тетрадных листах. Если отчет выполняется на отдельных тетрадных листах, то они должны быть аккуратно обрезаны по линии подшивки и скреплены. Неаккуратно выполненные, оборванные или грязные отчеты не принимаются.

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема лабораторной работы;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента;

е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать**:**

1) схему алгоритма, выполненную вручную или в соответствующем пакете;
2) текст программы;
3) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

4) выводы.

Контрольные вопросы

1. Что такое «строка»? В каких случаях используется эта конструкция?
2. Как реализована строка Паскаля?
3. Как показать ввод-вывод и обработку строк в программе?
4. Какой синтаксис имеет описание строки?
5. Объясните, почему в вашей программе следует использовать строки. Какие приемы со строками вы применили?
6. Как подбирают тесты для отладки программ, содержащих строки?

Лабораторная работа № 7

Программирование с использованием подпрограмм

Цель работы: изучение соглашений о передаче параметров и средств организации передачи управления и данных

Объем работы: 4 часа

Теоретическая часть

Процедуры и функции представляют собой *относительно самостоятельные фрагменты программы, соответствующим образом оформленные и снабженные именем* (программные блоки).

Каждый блок имеет такую же структуру, как основная программа, т.е. включает заголовок, раздел описаний и раздел операторов, но заканчивается не точкой, а точкой с запятой (рис. 1).

Рис. 1. Синтаксическая диаграмма конструкции <Программный блок>

В отличие от процедуры *функция всегда возвращает в точку вызова скалярное значение, адрес или строку*. Тип возвращаемого результата описывается в заголовке функции (рис. 2).

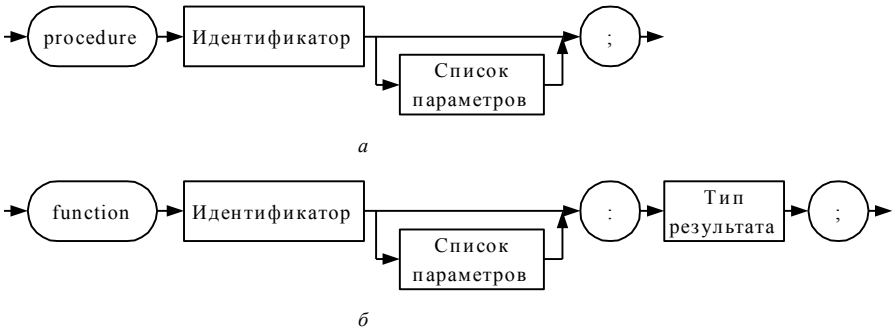


Рис. 2. Синтаксические диаграммы конструкций <Заголовок процедуры> (*а*) и <Заголовок функции> (*б*)

Данные для обработки процедуры и функции получают из вызвавшей их основной программы или подпрограммы. Для размещения рабочих полей подпрограммы могут объявлять новые типы и переменные в собственном разделе описаний. Результаты же они обычно должны возвращать вызвавшей программе или подпрограмме.

Данные в подпрограммы следует передавать через параметры. Список параметров описывается в заголовке подпрограммы (рис. 3). Параметры, перечисленные в этом

списке, получили название *формальных*, так как для их размещения не отводится память. При обращении к подпрограмме для каждого параметра должно быть указано фактическое значение (*фактический параметр*, *аргумент*) – литерал, константа или переменная того же типа, что и формальный параметр.

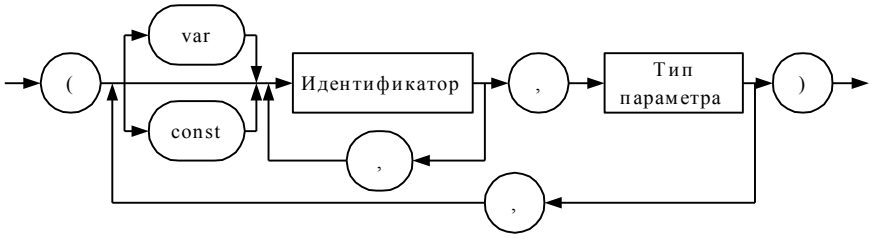


Рис. 2. Синтаксическая диаграмма конструкции <Список параметров>

В Delphi Pascal параметры в подпрограмму могут передаваться тремя способами:

- как значения – в подпрограмму передаются *копии* значений параметров, и никакие изменения этих копий не возвращаются в вызывающую программу;
- как переменные – в подпрограмму передаются *адреса* фактических параметров, соответственно все изменения этих параметров в подпрограмме на самом деле происходят с переменными, переданными в качестве фактических параметров; такие параметры при описании помечаются служебным словом var; в качестве фактических значений параметров-переменных нельзя использовать литералы;
- как неизменяемые переменные (именованные константы) – в подпрограмму, так же как и в предыдущем случае, передаются адреса фактических параметров, но при попытке изменить значение параметра компилятор выдает сообщение об ошибке; такие параметры при описании помечаются служебным словом const.

Пример. Разработать программу, которая определяет площадь четырехугольника по заданным длинам сторон и диагонали.

Будем считать площадь четырехугольника как сумму площадей двух треугольников, определенных по формуле Герона. Вычисление площади треугольника оформим как подпрограмму. Исходные данные такой подпрограммы – длины сторон треугольника. Подпрограмма не должна менять значения параметров, поэтому их можно передать как параметры-значения или параметры-константы. Результат работы этой подпрограммы – скалярное значение, значит, она может быть реализована как функция. Однако ее также можно реализовать как процедуру, которая возвращает результат через параметр-переменную. Схемы алгоритма данной программы с использованием подпрограмм обоих типов приведены на рис. 4.

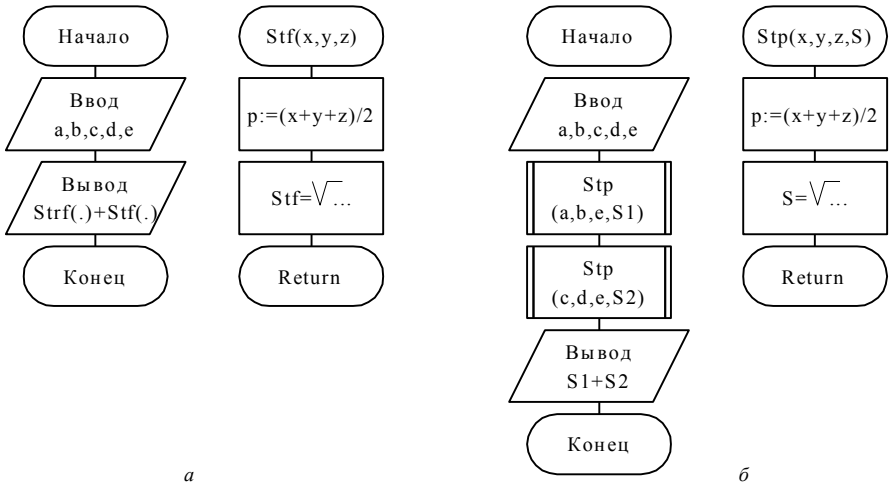


Рис. 3. Схемы алгоритмов программы определения площади четырехугольника с использованием функции (*а*) и процедуры (*б*)

Ниже приведены тексты соответствующих программ.

Вариант с использованием функции:

```
Program ex;
{$APPTYPE CONSOLE}
Uses SysUtils;
Var A,B,C,D,E:real;   {глобальные переменные}
{описание функции}
Function Stf(const X,Y,Z:real):real;
 Var p:real;          {локальная переменная}
    begin             {раздел операторов функции}
      p:=(X+Y+Z)/2;
      Result:=sqrt(p*(p-X)*(p-Y)*(p-Z));
    end;
{раздел операторов основной программы}
 begin
   WriteLn('Enter a,b,c,d,e:');
   ReadLn(A,B,C,D,E);
   WriteLn('S =', Stf(A,B,E)+Stf(C,D,E):7:3);
   ReadLn;
 End.
```

Вариант с использованием процедуры:

```
Program ex;
{$APPTYPE CONSOLE}
Uses SysUtils;
Var A,B,C,D,E,S1,S2:real; {глобальные переменные}
{описание процедуры}
Procedure Stp(const X,Y,Z:real;var S:real);
  Var p:real;          {локальная переменная}
    begin              {раздел операторов процедуры }
         p:=(X+Y+Z)/2;
         S:=sqrt(p*(p-X)*(p-Y)*(p-Z));
    end;
{раздел операторов основной программы}
begin
   WriteLn('Enter a,b,c,d,e:');
   ReadLn(A,B,C,D,E);
   Stp(A,B,E,S1);   {вызов процедуры}
   Stp(C,D,E,S2);   {вызов процедуры}
   WriteLn('S = ',S1+S2:7:3);
   ReadLn;
End.
```

Другие примеры программ, содержащих ветвления приведены в [1].

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Разработать схему алгоритма решения задачи.
3. Написать программу.
4. Вызвать среду программирования Turbo Delphi, создать новый проект консольного приложения и ввести текст программы в среду программирования.
5. Подобрать тестовые данные (не менее 3-х вариантов).
6. Отладить программу на выбранных тестовых данных.
7. Продемонстрировать работу программы преподавателю.
8. Составить отчет по лабораторной.
9. Защитить лабораторную работу преподавателю.

Требования к отчету

Отчет должен быть выполнен на бумаге формата А4 или А5 в том числе в тетрадях или на тетрадных листах. Если отчет выполняется на отдельных тетрадных листах, то они должны быть аккуратно обрезаны по линии подшивки и скреплены. Неаккуратно выполненные, оборванные или грязные отчеты не принимаются.

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема лабораторной работы;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента;

е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать**:**

1) схемы алгоритма основной программы и подпрограмм, выполненные вручную или в соответствующем пакете;
2) текст программы;
3) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

4) выводы.

Контрольные вопросы

1. Что такое «подпрограмма»? Чем различаются подпрограммы процедуры и функции?
2. Как показать подпрограммы и их вызовы в схеме алгоритма?
3. Какие способы передачи параметров в подпрограмму существуют?
4. Какой синтаксис используется для описания формальных параметров? Чем формальные параметры отличаются от фактических?
5. Какие особенности существуют при описании параметров структурных типов?
6. Как выполняется отладка программ с подпрограммами?

Лабораторная работа № 8

Программирование с использованием файлов

Цель работы: изучение приемов и средств, обеспечивающих хранение данных на дисках и их обработку

Объем работы: 2 часа

Теоретическая часть

Различают файлы трех типов: текстовые, типизированные и нетипизированные.

Текстовый файл – это файл, компонентами которого являются символьные строки переменной длины, заканчивающиеся специальным маркером конца строки (рис. 1).

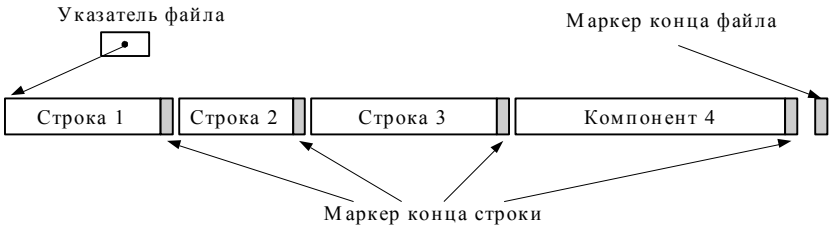


Рис. 1. Структура текстового файла

Типизированный файл – это файл, все компоненты которого одного типа, заданного при объявлении файловой переменной (рис. 2). Компоненты файла хранятся на диске во внутреннем (двоичном) формате и нумеруются с 0. Если посмотреть такой файл любым текстовым редактором, то можно распознать только символьную информацию, на месте же чисел в файле будут располагаться пробелы или символы псевдографики.

Рис. 2. Типизированный файл

Текстовые файлы используют для хранения и обработки текстовой информации: символов, строк, символьных массивов. Логические и числовые данные при записи в текстовые файлы должны преобразовываться в символьные строки. При выполнении операций ввода-вывода с текстовыми файлами используют процедуры:

Процедура **`Read[Ln]([Var f:text;]v1,v2,...vn)`** – обеспечивает ввод из файла символов, строк и чисел. Список ввода представляет собой последовательность из одной или более переменных типа CHAR, STRING, а также любого целого и вещественного типа. при отсутствии файловой переменной ввод осуществляется из

стандартного файла INPUT. Процедура **ReadLn** – также обеспечивает ввод символов, строк и чисел, но после чтения последней переменной оставшаяся часть строки до маркера конца строки пропускается, так что следующее обращение к ReadLn или Read начнется с первого символа новой строки. Процедура может быть вызвана без указания списка ввода, что приведет к пропуску всех символов текущей строки до маркера конца строки.

Процедура **Write[Ln]([Var f;]v1,v2,...,vn)** – обеспечивает вывод данных в текстовый файл или передачу их на логическое устройство. Список вывода – последовательность из одного или более выражений типа CHAR, STRING, BOOLEAN, а также целого или вещественного типов. При выводе числовых значений последние преобразуются в символьное представление. Процедура **WriteLn** при вызове обеспечивает запись маркера «конец строки».

Пример 1. Разработать программу, которая формирует текстовый файл из 26 строк, содержащих случайное количество соответствующих прописных букв латинского алфавита, например:

AAAAA

BBBBB

C

DDDDDDDDDDDDDDDDDDDD

EEEEEEEEEEEEE и т.д.

```
Program form_text_file;
{$APPTYPE CONSOLE}
Uses SysUtils;
Var f:text; {файловая переменная для текстового файла}
    a:char;  n,i:integer;  fname,st:string[30];
Begin
   WriteLn('Введите имя файла');
   ReadLn(fname);
   AssignFile(f,fname); {инициализируем файловую
                                          переменную}
   ReWrite(f);        {открываем файл для записи}
   Randomize;    {инициализируем датчик случайных чисел}
   for a:='A' to 'Z' do  {формируем строки}
      begin
        st:='';
        n:=random(30)+1;
```

```
        for i:=1 to n do    st:=st+a;
        WriteLn(f,st);  {записываем строку в текстовый
                                                файл}
        WriteLn(st); {а также – выводим ее на экран}
      end;
   CloseFile(f);    {закрываем файл}
  End.
```

Для работы с типизированными файлами используют специальные процедуры и функции:

1. Процедура **Read(Var f;c1,c2,...,cn)** – осуществляет чтение очередных компонентов типизированного файла. Список переменных ввода содержит одну или несколько переменных того же типа, что и компоненты файла, разделенных запятыми. Если файл исчерпан, обращение к процедуре вызывает ошибку ввода-вывода.

2. Процедура **Write(Var f;c1,c2,...,cn)** – осуществляет запись данных в типизированный файл. Список вывода содержит одно или более выражений того же типа, что и компоненты файла, разделенных запятыми.

Пример 2. Разработать программу, которая создает типизированный файл, содержащий список фамилий и даты рождения. Осуществить поиск в этом файле даты рождения по заданной фамилии.

```
Program ex;
{$APPTYPE CONSOLE}
Uses SysUtils;
Type fam=record    {тип запись «сведения о сотрудниках»}
        ff:string[20]; {фамилия}
        year:word;     {год рождения}
        month:1..12;    {месяц рождения}
        day:1..31       {день рождения}
     end;
  Var f:file of fam; {файловая переменная «файл
                                        сотрудников»}
      fb:fam;
      n,i:integer;
      fff:string;
      key:boolean;
```

```
Begin
  AssignFile(f,'a.dat'); {связываем файловую
                          переменную с файлом}
  ReWrite(f);        {открываем файл для записи}
  WriteLn('Enter data or CTRL-Z');
  while not EOF do {цикл, пока не введено CTRL-Z}
    begin
      ReadLn(fb.ff,fb.year,fb.month, fb.day);
      {вводим данные по полям, фамилию вводим в
      отдельной строке, так как ввод строки завершается
      нажатием клавиши Enter}
      Write(f,fb); {заносим запись в файл как один
                    компонент}
    end;
  CloseFile(f);        {закрываем файл}
  WriteLn('Enter family');
  Readln(fff);
  key:=false; {устанавливаем признак «запись не
               найдена»}
  ReSet(f);        {открываем файл для чтения}
  while (not EOF(f)) and (not key) do {пока не
            обнаружен конец файла и не найдена запись}
   begin
     Read(f,fb);    {читаем запись из файла}
     if fb.ff=fff then {если фамилии совпадают, то}
      begin  {выводим данные}
        WriteLn('Date:',fb.year,fb.month:3,
                                  fb.day:3);
        key:=true;  {устанавливаем признак «запись
                     найдена»}
      end;
   end;
  if not key then     {если признак не установлен}
     WriteLn('No data.');{то выводим «нет данных»}
```

```
    CloseFile(f); {закрываем файл}
end.
```

Другие примеры программ, содержащих обращение к дисковой памяти, приведены в [1].

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Разработать схему алгоритма решения задачи.
3. Написать программу.
4. Вызвать среду программирования Turbo Delphi, создать новый проект консольного приложения и ввести текст программы в среду программирования.
5. Подобрать тестовые данные (не менее 3-х вариантов).
6. Отладить программу на выбранных тестовых данных.
7. Продемонстрировать работу программы преподавателю.
8. Составить отчет по лабораторной.
9. Защитить лабораторную работу преподавателю.

Требования к отчету

Отчет должен быть выполнен на бумаге формата А4 или А5 в том числе в тетрадях или на тетрадных листах. Если отчет выполняется на отдельных тетрадных листах, то они должны быть аккуратно обрезаны по линии подшивки и скреплены. Неаккуратно выполненные, оборванные или грязные отчеты не принимаются.

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема лабораторной работы;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента;

е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать**:**

1) схему алгоритма, выполненную вручную или в соответствующем пакете;
2) текст программы;
3) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

4) выводы.

Контрольные вопросы

1. Что такое «файл»? В каких случаях используют хранение данных в файлах?
2. Какие типы файла вы знаете? Чем различаются текстовые и типизированные файлы?
3. Какие процедуры и функции используются для работы с текстовыми и типизированными файлами?
4. Какова последовательность операций при работе с файлами?
5. Зачем выполняют открытие и закрытие файлов? Инициализацию файловой переменной?

Лабораторная работа № 9

Программирование с использованием динамической памяти

Цель работы: изучение средств языка и приемов работы со списками, размещенными в динамической памяти

Объем работы: 6 часов

Теоретическая часть

Линейные односвязные списки используют чаще других списковых структур, так как они сравнительно просты, но одновременно в отличие от одномерных массивов позволяют:

- работать с произвольным количеством элементов, добавляя и удаляя их по мере необходимости;
- осуществлять вставку и удаление записей, не перемещая остальных элементов последовательности.

Недостатком этой структуры является то, что при поиске элемента по номеру приходится просматривать все ранее расположенные элементы, в то время как в одномерном массиве возможен прямой доступ к элементу по индексу. К тому же реализация линейного односвязного списка требует дополнительной памяти для хранения адресной части элементов.

Рассмотрим более подробно, как выполняются основные операции с линейными односвязными списками.

Исходные установки. В начале программы необходимо описать элемент и его тип:

```
Type tpel=^element;   {тип «указатель на элемент»}
     element=record
         num:integer;  {число}
         p:tpel;    {указатель на следующий элемент}
     end;
```

В статической памяти описываем переменную-указатель списка и несколько переменных-указателей, используемых при выполнении операций со списком:

```
Var first, {указатель списка – адрес первого элемента}
```

```
n,f,q:tpel;  {вспомогательные указатели}
```

Исходное состояние «список пуст» (в Delphi Pascal выполняется по умолчанию при инициализации переменных нулями):

```
first:=nil;
```

Обработку списков рассмотрим на примере добавления нового элемента к списку.

Добавление элемента к списку включает запрос памяти для размещения элемента и заполнение его информационной части. Построенный таким образом элемент добавляется к уже существующей части списка.

В общем случае при добавлении элемента к списку возможны следующие варианты:

- список пуст, добавляемый элемент станет единственным элементом списка;
- элемент необходимо вставить перед первым элементом списка;
- элемент необходимо вставить перед заданным (не первым) элементом списка;
- элемент необходимо дописать в конец списка.

Добавление элемента к пустому списку состоит из записи адреса элемента в указатель списка, причем в поле «адрес следующего» добавляемого элемента необходимо поместить nil:

```
new(first);        {запрашиваем память под элемент}
first^.num:=5;  {заносим число в информационное поле}
first^.p:=nil;{записываем nil в поле «адрес следующего»}
```

На рис. 1 показана последовательность операций при добавлении элемента к пустому списку.

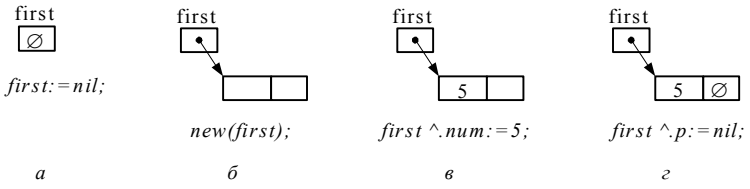


Рис. 1. Последовательность операций при добавлении элемента к пустому списку: исходное состояние (*а*), запрос памяти под элемент (*б*), заполнение элемента (*в*), занесение nil в поле адреса следующего элемента (*г*)

Добавление элемента перед первым элементом списка. При выполнении этой операции необходимо в поле «адрес следующего» переписать адрес первого элемента списка, а в указатель списка занести адрес добавляемого элемента (рис. 2):

```
new(q);        {запрашиваем память под элемент}
```

q^.num:=4; {заносим число в информационное поле}

q^.p:=first; {в поле «адрес следующего» переписываем адрес первого элемента}

first:=q; ... {в указатель списка заносим адрес нового элемента}

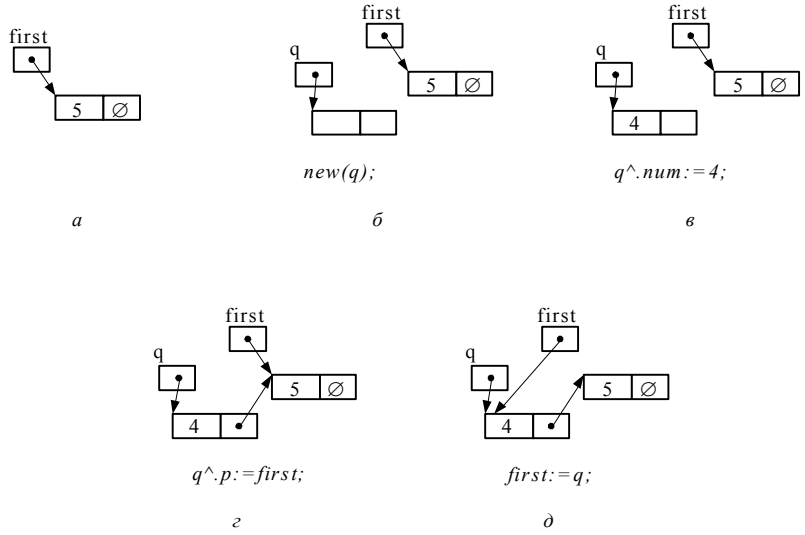


Рис. 2. Последовательность операций при добавлении элемента перед первым: исходное состояние (*а*), запрос памяти под элемент (*б*), заполнение элемента (*в*), шаги включения элемента в список (*г*, *д*)

Добавление элемента перед заданным (не первым). Для выполнения операции необходимо знать адреса элементов, между которыми вставляется элемент, так как адресные части этих элементов при выполнении операции будут корректироваться (рис. 3). Пусть f – адрес предыдущего элемента, а n – адрес следующего элемента, тогда:

new(q); {запрашиваем память под элемент}

q^.num:=3; {заносим число в информационное поле}

q^.p:=n; {в поле «адрес следующего» нового элемента переписываем адрес следующего элемента}

f^.p:=q; ... {в поле «адрес следующего» предыдущего элемента заносим адрес нового элемента}

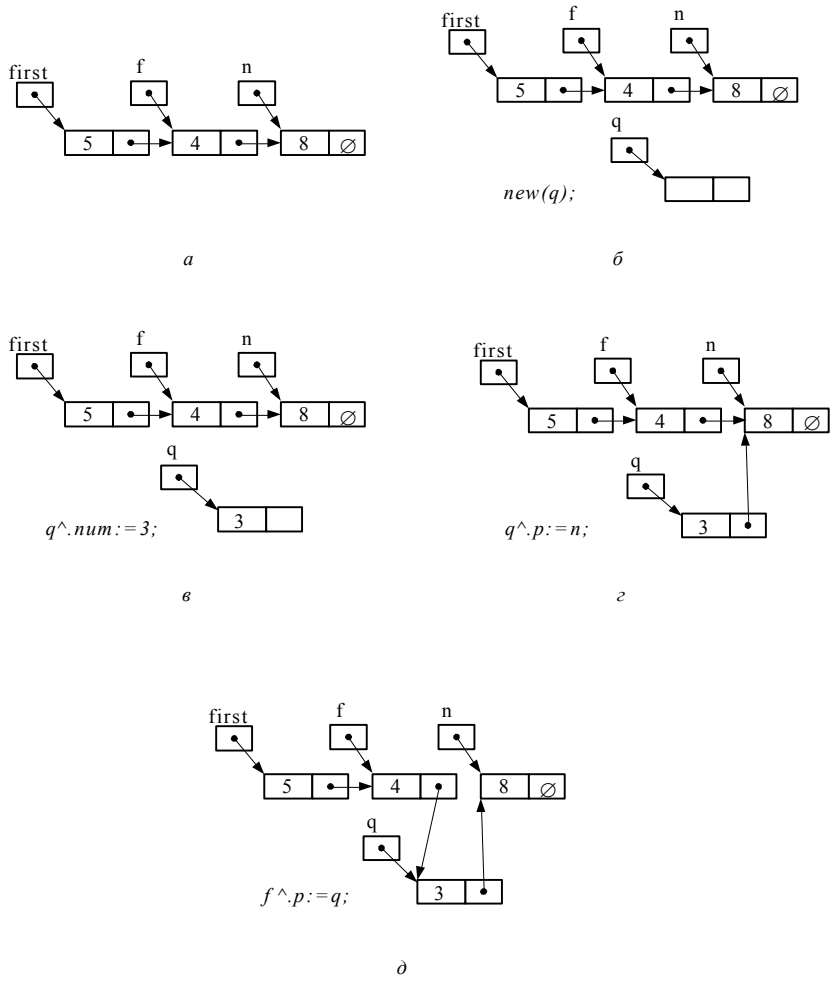


Рис. 3. Добавление элемента перед заданным (не первым): исходное состояние (*а*), запрос памяти под элемент (*б*), заполнение элемента (*в*), шаги включение элемента в список (*г*, *д*)

Добавление элемента в конец списка. В этом случае должен быть известен адрес элемента, после которого добавляется новый элемент (рис. 4):

```
new(q);          {запрашиваем память под элемент}
q^.num:=7;  {заносим число в информационное поле}
q^.p:=nil; {в поле «адрес следующего» элемента записываем nil}
f ^.p:=q; ...   {в поле «адрес следующего» предыдущего
                                элемента заносим адрес нового элемента}
```

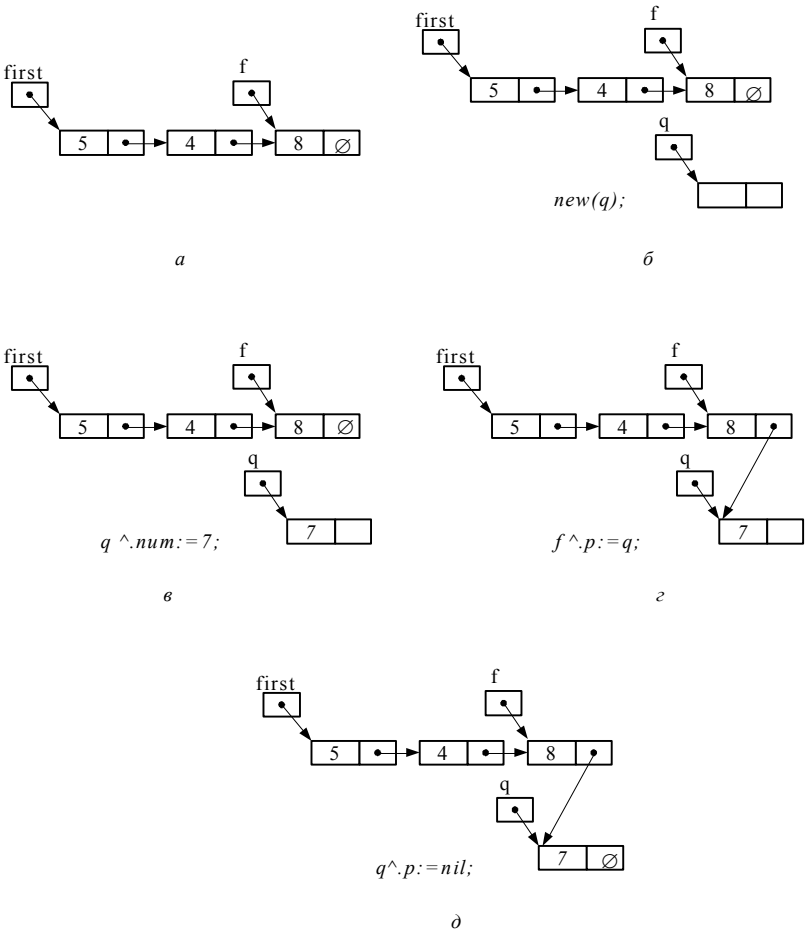


Рис. 4. Добавление элемента в конец списка: исходное состояние (*а*), запрос памяти под элемент (*б*), заполнение элемента (*в*), включение элемента в список (*г, д*)

Более подробно особенности выполнения операций со списками рассмотрены в [1].

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Разработать схему алгоритма решения задачи.
3. Написать программу.
4. Вызвать среду программирования Turbo Delphi, создать новый проект консольного приложения и ввести текст программы в среду программирования.
5. Подобрать тестовые данные (не менее 3-х вариантов).
6. Отладить программу на выбранных тестовых данных.
7. Продемонстрировать работу программы преподавателю.
8. Составить отчет по лабораторной работе.
9. Защитить лабораторную работу преподавателю.

Требования к отчету

Отчет должен быть выполнен на бумаге формата А4 или А5 в том числе в тетрадях или на тетрадных листах. Если отчет выполняется на отдельных тетрадных листах, то они должны быть аккуратно обрезаны по линии подшивки и скреплены. Неаккуратно выполненные, оборванные или грязные отчеты не принимаются.

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема лабораторной работы;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента;

е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать**:**

1) схему алгоритма, выполненную вручную или в соответствующем пакете;
2) текст программы;
3) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

4) выводы.

Контрольные вопросы

1. Что такое «список» в программировании? В каких случаях используется эта конструкция?
2. Какие типы списковых структур вы знаете?
3. Что такое указатели и как опи объявляются?
4. Как описывается элемент списка?
5. Какие варианты возможны для добавления элемента к списку?
6. Как отлаживают программы, содержащие обработку списков?

Литература

1. Г.С. Иванова. Программирование. Учебник для вузов. – М.: Издательство «Кнорус» 2013.