Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Воронежский государственный технический университет»

Кафедра конструирования и производства радиоаппаратуры

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ № 1-2 по дисциплине «Информационные технологии» для студентов направления 11.03.03 «Конструирование и технология электронных средств» (профиль «Проектирование и технология радиоэлектронных средств»)
всех форм обучения

Воронеж 2021

УДК 004.432.2
ББК 32.97

Составители:
д-р. техн. наук М.А. Ромащенко,
канд. техн. наук А.А. Пирогов,
И.В. Свиридова

Методические указания к выполнению лабораторных работ № 1-2 по дисциплине «Информационные технологии» для студентов направления 11.03.03 «Конструирование и технология электронных средств» (профиль «Проектирование и технология радиоэлектронных средств») всех форм обучения / ФГБОУ ВО «Воронежский государственный технический университет»; сост. М.А. Ромащенко, А.А. Пирогов, И.В. Свиридова. Воронеж: Изд-во ВГТУ, 2021, 27 с.

Методические указания предназначены для выполнения лабораторных работ № 1-2 по дисциплине «Информационные технологии» студентами очной и заочной форм обучения.

Предназначены для студентов первого курса обучения.

Методические указания подготовлены в электронном виде и содержатся в файле ЛР1-2.pdf.

Ил. 2. Библиогр.: 4 назв.

УДК 004.432.2
ББК 32.97

Рецензент - О. Ю. Макаров, д-р техн. наук, проф.
кафедры конструирования и производства
радиоаппаратуры ВГТУ

Издается по решению редакционно-издательского совета Воронежского государственного технического университета

1. ЛАБОРАТОРНАЯ РАБОТА №1

ПОДПРОГРАММЫ. ПРОЦЕДУРЫ И ФУНКЦИИ. РЕКУРСИЯ.

Цель работы: изучить принципы работы с подпрограммами, ознакомиться с локализацией имен в процедурах и функциях, ознакомиться с формальными и фактическими параметрами, уяснить назначение параметров-значений, параметров-переменных и параметров-констант. Освоить методы работы с рекурсией.

Время работы: 8 часов.

1.1. Домашние задания и методические указания по их выполнению

Задание 1 – получить представление о подпрограммах. Уяснить различие между процедурой и функцией.

Процедуры и функции представляют собой относительно самостоятельные фрагменты программы, оформленные особым образом и снабженные именем. Упоминание этого имени в тексте программы называется ***вызовом процедуры (функции).*** Отличие процедуры от функции в том, что результатом выполнения функции является одно число, поэтому вызов функции можно производить непосредственно в выражениях с константами и переменными. Процедуры и функции обобщают понятием ***подпрограмма***.

Подпрограммы представляют собой инструмент, с помощью которого любая программа может быть разбита на ряд независимых друг от друга частей. Такое разбиение необходимо, потому что:

- это средство экономии памяти, каждая подпрограмма существует в программе в единственном экземпляре, в то время как обращаться к ней можно многократно из разных точек программы. При вызове подпрограммы активизируется последовательность

образующих ее операторов, а с помощью передаваемых в подпрограмму параметров нужным образом модифицируется реализуемый в ней алгоритм.

- возможна реализация методики нисходящего программирования. В этом случае алгоритм представляется в виде последовательности относительно крупных подпрограмм, реализующих более или менее самостоятельные смысловые части алгоритма. Подпрограммы в свою очередь могут разбиваться на менее крупные подпрограммы нижнего уровня и т.д. Последовательное структурирование программы продолжают до тех пор, пока реализуемые подпрограммами алгоритмы не станут достаточно простыми, чтобы их можно было легко запрограммировать.

Задание 2 – ознакомиться с локализацией имен в программе и подпрограммах.

Любое имя в программе должно быть обязательно описано перед тем, как оно появится среди исполняемых операторов. Точно также обстоят дела и с подпрограммами – все используемые процедуры и функции необходимо описать в разделе описания.

Описать подпрограмму это, значит, указать ее заголовок и тело. В заголовке объявляются имя подпрограммы и формальные параметры, если они есть. Для функции, кроме того, указывается тип возвращаемого ею результата. За заголовком следует тело подпрограммы, которое, подобно программе состоит из своего раздела описания и раздела исполняемых операторов. В разделе описаний подпрограммы могут встретиться описания подпрограмм низшего уровня, в тех – описания других и т.д.

Пример структуры

```
Program Test;
    Procedure A;
        Procedure A1;
```

```
        {Раздел описаний ПРОЦЕДУРЫ A1}
    begin
        {Раздел исполняемых операторов
    ПРОЦЕДУРЫ A1}
    end;
    Procedure A2;
        {Раздел описаний ПРОЦЕДУРЫ A2}
    begin
        {Раздел исполняемых операторов
    ПРОЦЕДУРЫ A2}
    end;
begin
    {Раздел исполняемых операторов ПРОЦЕДУРЫ A}
end;
Procedure B;
    {Раздел описаний ПРОЦЕДУРЫ B}
begin
    {Раздел исполняемых операторов ПРОЦЕДУРЫ B}
end;
begin
    {Раздел исполняемых операторов ПРОГРАММЫ}
end.
```

Подпрограмма любого уровня обычно имеет множество имен констант, переменных, типов и вложенных в нее подпрограмм низшего уровня. Считается, что все имена, описанные внутри подпрограммы, локализуются в ней, т.е. они как бы «невидимы» за пределами подпрограммы. Т.о. со стороны операторов обращающихся к подпрограмме, они трактуются как «черный ящик», в котором реализуется тот или иной алгоритм. Все детали этой реализации скрыты от пользователя подпрограммы и поэтому недоступны ему. В структуре приведенной на рис. 1 из основной программы можно обратиться к процедурам A и B, но нельзя вызвать вложенные процедуры A1 и A2. Это относится не только к именам подпрограмм, но и к константам, переменным и меткам.

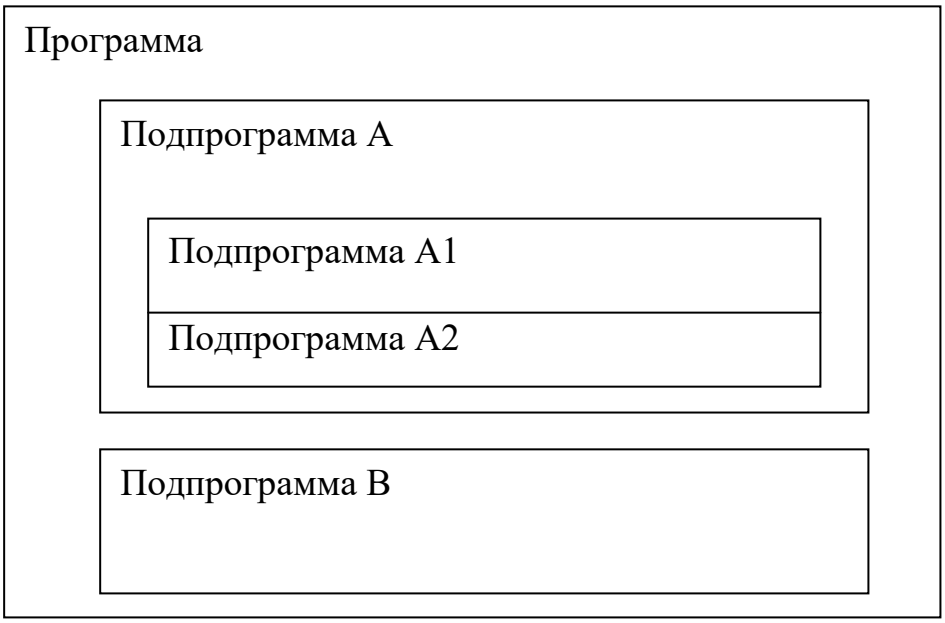

Рис. 1. Локализация имен в Паскале

При входе в подпрограмму низшего уровня становятся доступными объявленные в ней имена, но при этом сохраняется доступ ко всем именам верхнего уровня. Можно представить, что любая программа окружена тонированным стеклом – снаружи мы не видим ее внутренности, но, попав в подпрограмму, можем наблюдать все, что делается снаружи. Так например, из подпрограммы А2 мы можем вызвать подпрограмму А и А1, использовать переменные объявленные в основной программе и подпрограммах А и А1 и обращаться к ним.

При взаимодействии подпрограмм одного уровня вступает в силу основное правило ТурбоПаскаля: любая подпрограмма перед ее использованием должна быть описана. Поэтому из подпрограммы В можно вызвать подпрограмму А, но из А вызвать В невозможно. Продолжая образное сравнение, подпрограмму можно уподобить ящику с непрозрачными стенками и дном, но с тонированной крышкой – из подпрограммы можно смотреть «вверх» и нельзя «вниз». Т.е. подпрограмме доступны только те объекты верхнего уровня, которые описаны до описания данной подпрограммы. Эти объекты называются ***глобальными*** по отношению к подпрограмме.

Имена, локализованные в подпрограмме, могут совпадать с ранее объявленными глобальными именами. В этом случае считается, что локальное имя «закрывает» глобальное и делает его недоступным.

Задание 3 – Изучить описание подпрограмм. Закрепить знания о формальных и фактических параметрах.

Описание подпрограммы состоит из заголовка и тела подпрограммы. Заголовок процедуры имеет вид

procedure *<имя> (список формальных параметров);*

Заголовок функции имеет вид

function *<имя> (список формальных параметров): <тип>;*

Сразу за заголовком подпрограммы может следовать одна из стандартных директив ***assembler, external, far, forward, inline, interrupt, near***. Эти директивы уточняют действия компилятора и распространяются на всю подпрограмму, и только на нее.

Список формальных параметров не обязателен и может отсутствовать. Если же он есть, то в нем должны быть перечислены имена формальных параметров и их типы, например:

procedure *sb(a, x, y: real; b: integer; c: char);*

function *summa(x, y: real; rx, glob: integer): real;*

Операторы тела подпрограммы рассматривают список формальных параметров как своеобразное расширение раздела описаний: все переменные из этого списка могут использоваться в любых выражениях внутри подпрограммы. Таким способом осуществляется механизм замены формальных параметров фактическими, позволяющий нужным образом настроить алгоритм, реализованный в подпрограмме. Турбо Паскаль следит за тем, чтобы количество и типы формальных параметров строго соответствовали количеству и типам фактических параметров в момент обращения к подпрограмме. Смысл используемых фактических параметров зависит от того, в каком порядке они перечислены при вызове подпрограммы. Программист должен сам следить за правильным порядком перечисления фактических параметров при обращении к

подпрограмме.

Любой из формальных параметров подпрограммы может быть параметром-значением, параметром-переменной или параметром-константой. Выше были даны примеры параметров-значений. Если параметры определяются как параметры-переменные, перед ними ставится зарезервированное слово ***var***, а если это параметры-константы, то слово ***const***, например:

```
procedure demo(var a: real; b: real; const c: string);
```

Если параметр определен как параметр-значение, то перед вызовом подпрограммы его значение вычисляется, полученный результат копируется во временную память и передается подпрограмме. Даже если в качестве фактического параметра указано простейшее выражение в виде переменной или константы, все равно подпрограмме будет передана лишь копия. Любые изменения в подпрограмме параметра-значения никак не воспринимаются вызывающей программой, т.к. в этом случае изменяется копия фактического параметра.

Если параметр определен как параметр-переменная, то при вызове подпрограммы передается сама переменная, а не ее копия. Изменение параметра-переменной приводит к изменению самого фактического параметра в вызывающей программе.

В случае параметра-константы в подпрограмму также передается сама переменная или вычисленное значение, однако компилятор блокирует любые присваивания параметру-константе нового значения в теле подпрограммы.

```
program demo;
var
    a, b: integer;
procedure Inc2 (var c: integer; b: integer);
begin
    c:=c+c;
    b:=b+b;
    writeln(`удвоенные =`, c:5, b:5);
end;
```

```
begin
    a:=5; b:=7;
    writeln(`исходные =`, a:5, b:5);
    Inc2(a, b);
    writeln(`результат =`, a:5, b:5);
end.
```

В результате прогона на экране будет выведено:

```
исходные =      5     7
удвоенные =     10    14
результат =     10    7
```

С одной стороны параметры-переменные используются как средство связи алгоритма, реализованного в подпрограмме, с внешним миром: с помощью этих параметров подпрограмма может передавать результаты своей работы вызывающей подпрограмме. Имеется и другой способ передачи результатов – через глобальные переменные. Однако злоупотребление глобальными связями делает программу, как правило, запутанной, трудной в понимании, сложной в отладке. Требования хорошего стиля программирования подразумевают, по возможности, передавать результаты через фактические параметры-переменные.

С другой стороны, описание всех формальных параметров как переменных нежелательно по двум причинам. Во-первых, это исключает возможность вызова подпрограммы с фактическими параметрами в виде выражений. Во-вторых, в подпрограмме возможно случайное использование формального параметра, например, для временного хранения промежуточного результата, т.е. всегда существует опасность непреднамеренно испортить фактическую переменную. Параметрами-переменными следует объявлять только те из параметров, через которые подпрограмма в действительности передает результаты вызывающей программе. Чем меньше параметров будет объявлено параметрами-переменными и чем меньше в подпрограмме используется глобальных переменных,

тем меньше опасность получения непредусмотренных программистом побочных эффектов, связанных с вызовом подпрограммы, тем проще подпрограмма в понимании и отладке.

Задание 4 – ознакомиться с понятием рекурсия.

Рекурсия – это такой способ организации вычислительного процесса, при котором подпрограмма в ходе выполнения составляющих ее операторов обращается сама к себе. Например, вычисление факториала при помощи рекурсии представлен на рис. 2.

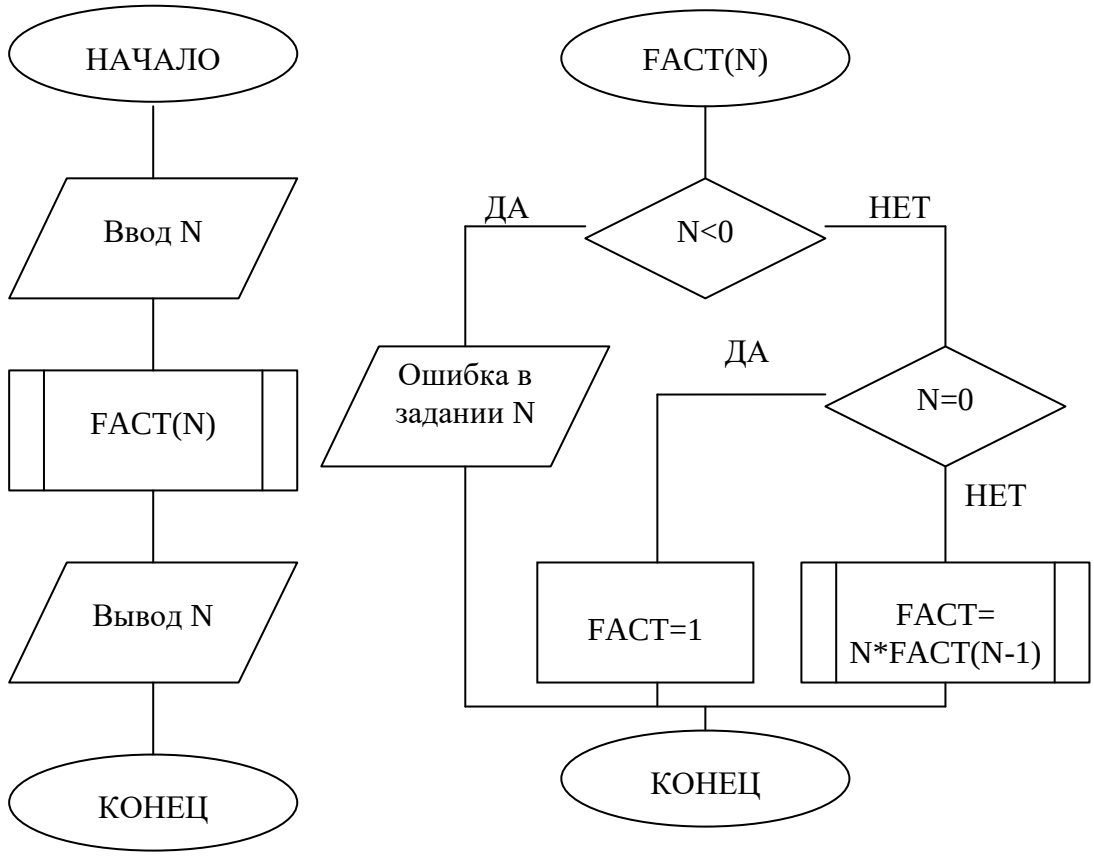

Рис. 2. Алгоритм рекурсивного вычисления факториала

```
program Factorial;
var
    n: integer;
function Fact (n: integer): Real; {рекурсивная функция вычисляющая n!}
```

```
begin
    if n<0 then
        writeln (`Ошибка в задании N`);
    else
        if n=0 then
            Fact:=1;
        else
            Fact:=n*Fact(n-1);
end;
begin
    writeln(`введите N`);
    readln(n);
    writeln(`N! = `, Fact(n));
end.
```

Рекурсивная форма организации алгоритма обычно выглядит изящнее итерационной и дает более компактный код, но выполняется такая программа, как правило, медленнее.

2.2. Лабораторные задания

Задание 1 – Составьте блок-схему алгоритма и напишите программу на языке Паскаль для работы с одномерными массивами в соответствии со своим вариантом. Массивы должны заполняться пользователем вручную.

1. даны четыре одномерных массива A, B, C, D, сформировать новый массив из сумм элементов каждого массива
2. даны четыре одномерных массива A, B, C, D, сформировать новый массив из произведений элементов каждого массива
3. даны четыре одномерных массива A, B, C, D, сформировать новый массив из минимальных значений элементов каждого массива
4. даны четыре одномерных массива A, B, C, D, сформировать новый массив из максимальных значений элементов каждого массива

5. даны четыре одномерных массива A, B, C, D, сформировать новый массив из средних арифметических элементов каждого массива

6. даны четыре одномерных массива A, B, C, D, вычислить их скалярные произведения A и B, C и D

7. даны четыре одномерных массива A, B, C, D, определить количество отрицательных элементов в каждом массиве

8. даны четыре одномерных массива A, B, C, D, определить количество положительных элементов в каждом массиве

9. даны четыре одномерных массива A, B, C, D, упорядочить элементы в каждом из них в порядке возрастания

10. даны четыре одномерных массива A, B, C, D, упорядочить элементы в каждом из них в порядке убывания

Задание 2 – Составьте блок-схему алгоритма и напишите программу на языке Паскаль для работы с двухмерными массивами в соответствии со своим вариантом. Массивы должны заполняться автоматически случайными числами.

1. даны четыре двумерных массива A, B, C, D, преобразовать каждый из них в одномерный массив

2. даны четыре двумерных массива A, B, C, D, для каждого из них вычислить суммы элементов на главных диагоналях

3. даны четыре двумерных массива A, B, C, D, для каждого из них вычислить суммы элементов на побочных диагоналях

4. даны четыре двумерных массива A, B, C, D, выполнить транспонирование (заменить строки на столбцы) каждого массива

5. даны четыре двумерных массива A, B, C, D, для каждого из них определить количество отрицательных элементов

6. даны четыре двумерных массива A, B, C, D, для каждого из них определить количество положительных элементов

7. даны четыре двумерных массива A, B, C, D, для каждого из них найти среднее арифметическое

8. даны четыре двумерных массива A, B, C, D, для каждого из

них найти индекс минимального элемента

9. даны четыре двумерных массива A, B, C, D, для каждого из них найти индекс максимального элемента

10. даны четыре двумерных массива A, B, C, D, для каждого из них найти максимальное и минимальное значение

2.3. Контрольные вопросы для отчета работы

1. Какие преимущества дает использование подпрограмм?
2. В чем отличие процедуры от функции?
3. В чем заключается описание подпрограммы? Приведите примеры описания процедуры и функции.
4. Приведите пример структуры подпрограммы.
5. Как происходит локализация имен в Паскале?
6. Что такое глобальные объекты?
7. Что произойдет, если в подпрограмме описать идентификатор, который ранее был использован для глобального объекта?
8. Что такое формальные и фактические параметры?
9. Чем отличаются параметр-значение, параметр-переменная и параметр-константа.
10. Что такое рекурсия? Приведите пример вычисления факториала.

2. ЛАБОРАТОРНАЯ РАБОТА №2

РАБОТА С ФАЙЛАМИ СРЕДСТВАМИ ЯЗЫКА ПАСКАЛЬ.

Цель работы: закрепить знания об использовании файлов в языке Паскаль. Изучить процедуры и функции для работы с файлами разных типов (текстовый, типизированный, нетипизированный). Получить навык работы с файлами средствами языка Паскаль.

Время работы: 8 часов.

2.1. Домашние задания и методические указания по их выполнению

Задание 1 – расширить и закрепить знания о файлах.

Под ***файлом*** понимается либо именованная область внешней памяти компьютера, либо логическое устройство – потенциальный источник или приемник информации. Любой файл имеет три характерные особенности. 1 – у него есть имя, что дает возможность программе работать одновременно с несколькими файлами. 2 – он содержит компоненты одного типа (любой тип Паскаля, кроме файлов). 3 – размер вновь создаваемого файла никак не оговаривается при его объявлении и ограничивается только емкостью устройств внешней памяти.

Переменную файлового типа можно задать следующим образом:

*<имя>=**file of** <тип>;*

<имя>=text;

*<имя>=**file**;*

Такое объявление файла соответствует трем видам файлов: типизированные; текстовые; нетипизированные.

Любые файлы становятся доступны программе только после

выполнения процедуры открытия файла, которая заключается в связывании ранее объявленной файловой переменной с именем существующего или вновь создаваемого файла, а также направлении обмена информацией: чтение или запись из файла. Используется стандартная процедура

assign*(файловая переменная, `имя файла`);*

Имя файла это любое выражение строкового типа, которое строится по правилам определения имен в MS-DOS.

Затем необходимо инициировать файл, т.е. указать направление передачи данных. Можно производить только чтение, только запись или чтение и запись одновременно.

Для чтения файл инициируется с помощью стандартной процедуры

reset*(файловая переменная);*

В результате специальная переменная указатель, связанная с этим файлом, будет указывать на начало файла, т.е. компонент с порядковым номером 0.

Для записи информации в файл используется стандартная процедура

rewrite*(файловая переменная);*

Этой процедурой нельзя инициировать запись в ранее существоваший файл, он будет полностью удален. Используется только для записи в новый файл.

Для записи в ранее существовавший файл используется стандартная процедура

append*(файловая переменная);*

Данная процедура применима только к текстовым файлам (тип файловой переменной *text*).

Задание 2 – закрепить знания о процедурах и функциях для работы с файлами на языке Паскаль.

Процедура ***close*** закрывает файл, однако связь файловой переменной с именем файла установленной ранее сохраняется

close*(файловая переменная);*

Процедура автоматически выполняется по отношению ко всем открытым файлам при нормальном завершении программы.

Процедура *rename* переименовывает файл

rename*(файловая переменная, `новое имя`);*

Перед выполнением процедуры необходимо закрыть файл.

Процедура *erase* удаляет файл

erase*(файловая переменная);*

Перед выполнением процедуры необходимо закрыть файл.

Процедура *flush* очищает внутренний буфер файла и, т.о. гарантирует сохранность всех последних изменений файла на диске

flush*(файловая переменная);*

Любое обращение к файлу осуществляется через некоторый буфер, что необходимо для согласования с MS-DOS. В ходе выполнения процедуры *flush* все новые записи будут действительно записаны на диск.

Функция *EOF* тестирует конец файла и возвращает значение типа *Boolean*

EOF*(файловая переменная);*

Функция возвращает True, если файловый указатель расположен в конце файла. При записи это означает, что очередной компонент будет добавлен в конец файла, при чтении, что файл исчерпан.

Процедура *ChDir* обеспечивает изменение текущего каталога

ChDir*(`путь`);*

Процедура *GetDir* позволяет определить имя текущего каталога

GetDir*(<устройство>, <каталог>);*

<Устройство> - выражение типа ***word***, содержащее номер устройства (1 – диск A, 2 – диск B и т.д.); *<каталог>* - переменная типа ***String*** в которой возвращается путь к текущему каталогу на указанном диске.

Процедура *MkDir* создает новый каталог на указанном диске

MkDir*(<каталог>);*

Процедура *RmDir* удаляет каталог

RmDir*(<каталог>);*

Удаляемый каталог должен быть пустым.

Функция ***IOResult*** возвращает условный признак последней операции ввода-вывода в виде значений типа ***word***. Если операция завершилась успешно функция возвращает 0, иначе код ошибки. Функция становится доступной только при отключенном автоконтроле ошибок ввода-вывода. Директива компилятора *{$I-}* отключает, а директива *{$I+}* включает автоконтроль. Если операция привела к возникновению ошибки, устанавливается флаг ошибки, и все последующие попытки ввода-вывода блокируются, пока не будет вызвана функция *IOResult*.

Задание 3 – изучить особенности текстовых файлов языка Паскаль.

Предназначены для хранения текстовой информации. Компоненты текстового файла могут иметь переменную длину, что влияет на характер работы с ними. Текстовый файл трактуется как совокупность строк переменной длины. Доступ к каждой строке возможен лишь последовательно, начиная с первой. В конце каждой строки ставится признак *EOLN*, в конце всего файла *EOF*. Эти признаки можно протестировать одноименными функциями.

Для доступа к строкам применяются процедуры *Read, ReadLn* – (чтение из файла в переменную), *Write, WriteLn*.

Read*(файловая переменная, <список ввода>);*

Пример программы ввода массива вещественных чисел из файла

```
Program Files;
const
    N=1000; {максимальная длина ввода}
var
    F: text;
    M: array [1..N] of real;
    I: integer;
begin
    assign(F, `prog.dat`);
```

```
        reset(F);
        I:=1;
        while not EOF(F) and (I<=N) do
            begin
                read(F, M[I]);
                I:=I+1;
            end;
        close(F);
end.
```

Отличие процедуры *ReadLn* в том, что после считывания последней переменной оставшаяся часть строки до маркера EOLN пропускается, поэтому следующее обращение к *Read* или *ReadLn* начинается с первого символа новой строки. Для пропуска текущей строки можно вызвать процедуру *ReadLn* без списка ввода.

Процедура Write обеспечивает ввод информации в текстовый файл.

Write*(файловая переменная, <список вывода>);*

Отличие процедуры *WriteLn* в том, что после ее выполнения курсор переходит на новую строку. Использование процедуры *WriteLn* без списка вывода в файл будет записана пустая строка.

Пример программы определения объема рукописи

```
Program Rukopis;
var
        F: text;
        S: String;
        Sum: LohgInt;{счетчик количества символов}
begin
        writeln(`Имя файла `);
        readln(S);
        assign(F, S);
        reset(F);
        while not EOF(F) do
            begin
```

```
            readln(F, S);
            Sum:=Sum+Length(s);
        end;
    close(F);
    writeln(`Объем =`, Sum/4000:6:2, ` уч.изд.л.`);
end.
```

Пример программы наполнения и чтения текстового файла

```
Program Rabota_s_textom;
var
    F: text;
    S: String;
begin
    assign(F, `Text.txt`);
    rewrite(F);
    while not EOF(F) do
        begin
            readln(S);
            writeln(F, S);
        end;
    close(F);
    reset(F);
    while not EOF(F) do
        begin
            readln(F, S);
            writeln(S);
        end;
    close(F);
end.
```

Задание 4 – изучить особенности типизированных файлов языка Паскаль.

Длина любого компонента типизированного файла строго постоянна, что дает возможность организовать прямой доступ к

каждому из них по порядковому номеру.

Перед первым обращением к процедурам ввода-вывода указатель файла стоит в его начале и указывает на первый компонент с номером 0. После каждого чтения или записи указатель сдвигается к следующему компоненту файла. Переменные в списках ввода-вывода должны иметь тот же тип, что и компоненты файла.

Процедура Read обеспечивает чтение очередных компонентов типизированного файла – ***Read****(файловая переменная, <список ввода>).* Список ввода должен содержать одну или более переменных такого же типа, что и компоненты файла. Файловая переменная должна быть объявлена предложением ***file of…*** и связана с именем файла процедурой ***assign***. Файл необходимо открыть процедурой ***reset.***

Процедура *Write* используется для записи данных в типизированный файл.

Write*(файловая переменная, <список вывода>);*

Процедура Seek смещает указатель файла к требуемому компоненту.

Seek*(файловая переменная, <N компонента>);*

N компонента – выражение типа *LongInt* указывающее номер компонента файла. Первый компонент файла имеет номер 0.

Функция *FileSize* возвращает значение типа *LongInt* которое содержит количество компонентов файла.

FileSize*(файловая переменная);*

Функция *FilePos* возвращает значение типа *LongInt* содержащее порядковый номер компонента файла, который будет обрабатываться следующей операцией ввода-вывода.

FilePos*(файловая переменная);*

Пример программы, которая создает новый файл для размещения целых чисел, наполняет его случайным числами и затем выводит его содержимое на экран.

```
Program Files;
const
    N=20;
 var
```

```
    F: file of Integer;
    I, K: integer;
begin
    assign(F, `data.dat`);
    rewrite(F);
    for K:=1 to N do
        begin
            I:=random(100);
            write(F, I);
        end;
    close(F);
    reset(F);
    while not EOF(F) do
        begin
            read(F, I);
            writeln(I);
        end;
    close(F);
end.
```

Задание 5 – изучить особенности нетипизированных файлов языка Паскаль.

Объявляются как файловые переменные типа file и отличаются тем, что для них не указан тип компонентов. Отсутствие типа делает эти файлы, с одной стороны, совместимыми с любыми другими файлами, а с другой позволяет организовать высокоскоростной обмен данными между диском и памятью.

При инициализации нетипизированного файла с помощью процедуры reset или rewrite можно указать длину записи нетипизированного файла в байтах. Пример

```
F: file;
…….
assign(F, `myfile.dat`);
rewrite(F, 512);
```

Если длина записи не указана она принимается равной 128 байт.

При работе с нетипизированными файлами могут применяться все процедуры и функции, доступные типизированным файлам, за исключением Read и Write, которые заменяются соответственно высокоскоростными процедурами ***BlockRead*** и ***BlockWrite***.

BlockRead(файловая переменная, буфер, N, [NN]);

BlockWrite (файловая переменная, буфер, N, [NN]);

Буфер это имя переменной, которая будет участвовать в обмене данными с файлом; N – количество записей, которые должны быть прочитаны или записаны за одно обращение к файлу; NN – необязательный параметр, содержащий при выходе из процедуры количество фактически обработанных записей.

За одно обращение к процедурам может быть передано до N*RECS байтов, где RECS – длина записи нетипизированного файла. Передача идет начиная с первого байта буфера. При завершении процедуры указатель смещается на NN записей. Процедурами Seek, FilePos и FileSize можно обеспечить доступ к любой части нетипизированного файла.

Пример программы, которая готовит массив buf из N случайных целых чисел и записывает его в нетипизированный файл. Затем массив buf очищается, в него читается содержимое файла и массив выводится на экран строками по 10 чисел.

```
Program Files;
const
      N=50;
 var
      F: file;
      I, K: integer;
      BUF: array [1..N] of Integer;
begin
      for K:=1 to N do
            BUF[K]:=random(100);
      assign(F, `untyped.dat`);
```

```
        rewrite(F, 1);
        blockwrite(F, BUF, SizeOf(BUF));
        close(F);
        for K:=1 to N do
                BUF[K]:=-1;
        reset(F);
        blockread(F, BUF, SizeOf(BUF), K);
        close(F);
        for K:=1 to N do
                begin
                        write(BUF[K]);
                        if K mod 10=0 then
                                writeln;
                end;
end.
```

2.2. Лабораторные задания

Задание 1 – Составьте блок-схему алгоритма и напишите программу на языке Паскаль выполняющую следующее действие – одномерный массив считывается из файла исходных данных *in.txt*, а затем *без изменений* записывается в файл результатов *out.txt*.

Задание 2 – Модифицируйте программу, составленную в задании 1 так, чтобы в результате ее выполнения в файл результатов записывался массив с которым произведены вычисления в соответствии с выданным вариантом задания.

1. найти в массиве максимальный элемент и записать его в первую строку файла, во вторую строку записать сам массив
2. найти в массиве минимальный элемент и записать его в первую строку файла, во вторую строку записать сам массив

3. найти в массиве порядковый номер максимального элемента и записать его в первую строку файла, во вторую строку записать сам массив

4. найти в массиве порядковый номер минимального элемента и записать его в первую строку файла, во вторую строку записать сам массив

5. сдвинуть циклически элементы массива влево на три позиции и записать получившийся массив в файл

6. сдвинуть циклически элементы массива вправо на две позиции и записать получившийся массив в файл

7. заменить в исходном массиве каждый элемент на сумму стоящих после него элементов и записать получившийся массив в файл

8. заменить в исходном массиве элементы, имеющие дробную часть на нули и записать получившийся массив в файл

9. упорядочить элементы в исходном массиве по возрастанию и записать получившийся массив в файл

10. упорядочить элементы в исходном массиве по убыванию и записать получившийся массив в файл

2.3 Контрольные вопросы для отчета работы

1. Что такое файл? Какие характерные особенности он имеет?
2. Какие виды файлов в Паскале Вы знаете? Как объявить файловую переменную для каждого из этих типов?
3. Каково назначение и синтаксис стандартной процедуры ***assign***?
4. Что такое инициирование файла? Какими стандартными процедурами это производится? Каков синтаксис этих процедур?
5. Каково назначение и синтаксис стандартных процедур ***close, rename, erase***?
6. Каково назначение и синтаксис стандартных процедур ***flush, chdir, getdir***?

7. Каково назначение и синтаксис стандартной процедуры ***EOF***?

8. Как происходит чтение и запись в текстовый файл при помощи процедур языка Паскаль? Каковы особенности этого типа файлов по сравнению с другими?

9. Как происходит чтение и запись в типизированный файл при помощи процедур языка Паскаль? Каковы особенности этого типа файлов по сравнению с другими?

10. Как происходит чтение и запись в нетипизированный файл при помощи процедур языка Паскаль? Каковы особенности этого типа файлов по сравнению с другими?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Фаронов В.В. Турбо Паскаль 7.0: Начальный курс: учебное пособие / В.В. Фаронов. – 7-е изд., перераб. – М.: Нолидж, 2002. – 576 с.: ил.

2. Фаронов В.В. Турбо Паскаль: учебное пособие / В.В. Фаронов. – СПб.: Питер, 2007. – 367 с.: ил.

3. Информатика: Базовый курс: учеб. пособие для втузов / под ред. С.В. Симоновича. СПб.: Питер, 2003. – 640 с.: ил.

4. Архангельский А.Я. Программирование в Delphi: учебник по классическим версиям Delphi / А.Я. Архангельский. – М.: Бином, 2006. – 1152 с.: ил.

СОДЕРЖАНИЕ

1. Лабораторная работа №5 1
2. Лабораторная работа №6 12
Библиографический список 24

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к выполнению лабораторных работ № 1-2 по дисциплине «Информационные технологии» для студентов направления 11.03.03 «Конструирование и технология электронных средств» (профиль «Проектирование и технология радиоэлектронных средств»)
всех форм обучения

Составители:
д-р. техн. наук М.А. Ромащенко,
канд. техн. наук А.А. Пирогов,
И.В. Свиридова

Компьютерный набор М.А. Ромащенко

Подписано к изданию _____
Уч.-изд. л. _____

ФГБОУ ВО «Воронежский государственный технический университет»
394026 Воронеж, Московский проспект, 14