Министерство образования Республики Беларусь
БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра «Гидропневмоавтоматика и гидропневмопривод»

М.И. Жилевич
Л.Г. Филипова

ИНФОРМАТИКА

Учебно-методическое пособие
к лабораторным работам для студентов специальности 1-36 01 07
«Гидропневмосистемы мобильных и технологических машин»

Минск 2009

УДК 621.113:681.3(075.8)
ББК 33я7
Ж 72

Рецензенты:
В.В. Равино, А.С. Поварехо

Жилевич, М.И.

Ж 72 Информатика: учебно-методическое пособие к лабораторным работам для студентов специальности 1-36 01 07 «Гидропневмосистемы мобильных и технологических машин» / М.И. Жилевич, Л.Г. Филипова. – Минск: БНТУ, 2009. – 40 с.

ISBN 978-985-525-012-9.

Данное пособие предназначено для выполнения лабораторных работ по дисциплине «Информатика» и содержит необходимые теоретические сведения и практические рекомендации, позволяющие программировать на ЭВМ различные вычислительные процессы на языке Паскаль.

Пособие может быть полезно студентам для выполнения расчетов в ходе курсового и дипломного проектирования.

УДК 621.113:681.3(075.8)
ББК 33я7

ISBN 978-985-525-012-9

ПРАВИЛА ТЕХНИКИ БЕЗОПАСНОСТИ

К работе с персональным компьютером (далее ПК) допускаются студенты, прошедшие инструктаж по охране труда с обязательной записью в журнале.

Студенты, не прошедшие инструктаж по охране труда, к выполнению работ на ПК не допускаются.

Проведение работ с ПК осуществляется под руководством и наблюдением преподавателя.

При работе в лаборатории студенты обязаны:

- соблюдать правила внутреннего распорядка, дисциплину;
- выполнять требования охраны труда, соблюдать правила личной гигиены;
- выполнять только порученную лабораторную работу;
- знать правила работы с ПК.

Перед выполнением лабораторной работы необходимо:

изучить инструкцию по выполнению данной лабораторной работы;

подготовить свое рабочее место, убрав лишние предметы;

включение оборудования ПК в электрическую сеть производит преподаватель или учебно-вспомогательный персонал;

работу следует начинать только по разрешению преподавателя или учебно-вспомогательного персонала лаборатории.

ЗАПРЕЩАЕТСЯ приступать к работе на ПК:

при выраженном дрожании изображения на мониторе;

обнаружении неисправности оборудования;

наличии поврежденных кабелей или проводов, разъемов штепсельных соединений;

отсутствии или неисправности защитного заземления (зануления).

Введение

Настоящее учебно-методическое пособие является дополнением к лабораторному практикуму, изданному в 2003 году (Информатика: учебно-методическое пособие для студентов специальности 1-36 01 07 «Гидропневмосистемы мобильных и технологических машин» / М.И. Жилевич, Л.Г. Филипова. Минск: БНТУ, 2003. 77 с.), и учитывает практический опыт проведения лабораторных занятий со студентами по дисциплине «Информатика».

В пособие включены достаточно простые лабораторные работы для изучения основных операторов языка *Turbo-Paskal* и типовых алгоритмов, используемых в программировании. Преподаватель по своему усмотрению может, например, проводить лабораторную работу № 1 данного пособия в продолжение (дополнение) лабораторной работы № 2 предыдущего издания. Лабораторная работа № 2 может быть предложена к выполнению после работы № 7 предыдущего издания, лабораторная работа № 3 – перед работой №10 предыдущего издания, работа № 4 – после (вместо) работы № 10, работа № 5 – после работы № 12.

Примерное *содержание отчета* по лабораторной работе:

- номер и название лабораторной работы;
- цель работы;
- задание и исходные данные в соответствии с вариантом;
- схема алгоритма решения задачи (головной программы и подпрограмм);
- таблица идентификаторов;
- распечатки головной программы, подпрограмм и результатов расчета;
- выводы (анализ результатов).

Лабораторная работа № 1

ОБЩИЙ ПОРЯДОК РАБОТЫ С ПАСКАЛЬ-ПРОГРАММОЙ

Цель работы

1. Приобретение навыков работы в среде программирования Турбо-Паскаль.
2. Изучение структуры программы на языке Паскаль, подготовка и отладка простейшей программы.
3. Ознакомление с организацией диалога «оператор–ЭВМ».

Теоретические сведения

Программа на языке *Pascal* состоит из заголовка и собственно программы, называемой блоком.

Заголовок состоит из слова *PROGRAM*, имени программы и точки с запятой. Например, *PROGRAM LAB1*;

Блок состоит из двух частей: описательной и исполнительной.

В *описательной части* должны быть перечислены и описаны все данные (переменные, константы, массивы и др.), используемые в программе.

Раздел операторов (исполнительная часть) заключается в операторные скобки *BEGIN ... END*. В этом разделе указывается последовательность действий, которые должна выполнить ЭВМ для решения поставленной задачи.

Разделителем между разделами и операторами является точка с запятой. В конце программы должна стоять точка.

В любом месте программы можно расположить *комментарий*, в котором содержится текстовая информация о назначении программы, разработчике и т.д. Комментарий заключается в фигурные скобки или символы (*...*) и является невыполняемой частью программы.

Общая *структура программы:*

PROGRAM имя; – заголовок;

USES ...; – раздел подключаемых библиотечных модулей;
LABEL ...; – раздел меток;
CONST ...; – раздел констант;
TYPE ...; – раздел определения типов данных;
VAR ...; – раздел описания переменных;
PROCEDURE имя; – раздел описания процедур;
FUNCTION имя; – раздел описания функций;

} Описательная часть

BEGIN
(операторы) – исполнительная часть;
END.

Основную программу называют *глобальным* блоком. Глобальный блок может содержать другие блоки. Их называют *локальными*. Локальные блоки – это процедуры и функции. Любой из разделов, кроме раздела операторов основной программы, может отсутствовать.

Набор текста программ, их отладка и выполнение осуществляются в *среде программирования Turbo Pascal*. Для разработки программы пользователь последовательно должен выполнить следующие действия:

1) загрузить *Turbo Pascal*;
2) перейти в персональный каталог;
3) ввести программу (создать файл и набрать текст);
4) сохранить набранный текст;
5) отредактировать программу (устранить синтаксические ошибки);
6) запустить программу на выполнение;
7) просмотреть и проанализировать результат;
8) получить распечатки текста программы и результатов расчета.

Загрузка *Turbo Pascal* осуществляется при помощи запуска файла *Turbo.exe*, который находится в директории пакета *Turbo Pascal* (*TP*). Это можно сделать с помощью:

1) щелчка по соответствующему ярлыку на рабочем столе *Windows*;

2) программ «Мой компьютер» или «Проводник»;

3) кнопки «Пуск» → «Программы»;

4) оболочки типа *Total Commander* или *Norton Commander*.

После выполнения загрузки на экране дисплея появляется основной экран интегрированной среды, состоящий из строки главного меню, поля экрана (для набора текста программы), строки состояния. Первая строка содержит все команды главного меню. В последней строке экрана отображаются доступные на каждый текущий момент функциональные клавиши с указанием их назначения. Содержимое строки состояния меняется при изменении режима работы среды.

Переход в персональный каталог необходим для рациональной организации хранения файлов и ускорения доступа к ним при работе на компьютере большого количества пользователей. Для этого необходимо войти в меню *File* и выполнить команду *Change dir*. В раскрывающемся окне по дереву каталогов выбирают диск (*Drives*) и персональную папку (двойной щелчок левой кнопки мышки или кнопка *Chdir*). Если в верхней строчке окна отображается требуемый маршрут, нажимают кнопку *OK*. После выполнения этой команды по умолчанию файлы будут записываться в персональный каталог и считываться из него.

Для ***создания*** файла необходимо выполнить команду *File* → → *New* (то же можно сделать нажатием комбинации функциональных клавиш *ALT* + *F* + *N*). На экране появляется окно редактирования с неопределенным именем *Noname00. pas*. Можно набирать текст, но лучше заранее присвоить файлу имя и определить место его хранения.

Открыть существующий файл можно с помощью команды *File → Open* или клавиши *F3*. На экране появляется диалоговое окно "*Open a file*". Необходимо выбрать требуемый файл и для подтверждения нажать клавишу ввода или кнопку *Open* в окне.

Сохранение нового файла (или переименование уже существующего) выполняется командой *File → Save as* . На экране появляется диалоговое окно "*Save file as*". Необходимо набрать имя файла и нажать кнопку *OK*. В случае необходимости можно изменить место хранения файла, указав соответствующий диск и каталог.

При наборе текста новой программы и редактировании текста ранее созданных программ необходимо *периодически* выполнять команду сохранения, чтобы свести к минимуму потери при различных сбоях оборудования или зависании программ. Для этого предназначена команда *File → Save*, однако удобнее пользоваться клавишей *F2*. Также необходимо сохранить файл после завершения работы с текстом.

Редактирование (компиляция) программы – войти в меню *Compile* и выполнить команду *Compile* (или комбинация клавиш *ALT+F9*). Компиляция позволяет обнаружить синтаксические ошибки (формальную структуру программы, правильность написания операторов), но не ошибки программирования.

Если обнаружена ошибка, в верхней строке окна редактирования высвечивается ее код, а курсор указывает предполагаемое место. Необходимо исправить ошибку, сохранить программу с изменениями и повторно произвести компиляцию.

Если ошибки отсутствуют, выводится сообщение "*Press any key*". Для возврата в окно редактирования надо нажать любую клавишу.

Запуск программы на выполнение – войти в меню *Run* и выполнить команду *Run* (или комбинация клавиш *CTRL+F9*). При загрузке программы автоматически выполняется компиляция, и если ошибки не были устранены, выводится их код, программа не запускается.

Просмотреть результаты работы программы (протокол, команды операционной системы) можно через окно просмотра. Для этого необходимо войти в меню *Debug* и выполнить команду *Outptut* или *User screen* (комбинация клавиш *CTRL+F5)*. Можно также организовать задержку выполнения программы, например, перед последним *END* записать оператор *READLN*;, тогда во время задержки на экране будут отображаться результаты, а для возврата в окно редактирования достаточно нажать клавишу ввода.

Для ***распечатки*** текстов программ предусмотрена команда *File → Print*, однако предпочтительнее пользоваться другими способами:

а) через программу «Блокнот»;

б) путем вставки файла в текстовый редактор *WORD* (с опцией «текст *DOS*»);

в) через оболочки типа *Total Commander* (найти в каталогах требуемый файл, выделить его, выполнить команду *Copy* (*F5*), в раскрывающемся окне с запросом о месте копирования набрать *PRN* (принтер) и нажать клавишу ввода).

Выход из среды программирования *Turbo Pascal - File → → Exit* (*ALT+X*).

Следует отметить, что при выполнении данной лабораторной работы нет необходимости уделять много внимания изучению структуры и принципа действия используемых в программе операторов. Достаточно уяснить лишь их общее назначение и место в программе. Детальное изучение операторов Паскаля предусмотрено в последующих лабораторных работах.

Задание

Набрать и отладить программу для вычисления выражения $y = a + b$.

Порядок выполнения работы

1. Произвести подготовительные операции. Для этого:

1.1) создать персональный каталог (папку). Обычно это делается на диске *D* в каталоге *USERS*;

1.2) загрузить Паскаля;

1.3) выполненить команды по изменению каталога (все «свои» программы следует хранить у себя в «сейфе», а не разбрасывать по компьютеру);

1.4) создать новый файл, сохранить его путем присвоения имени, например, *PROSTO.PAS*.

2. Начинаем с самой простой программы. В Паскаль-программе может отсутствовать все, кроме исполнительной (выполняемой) части, и даже в выполняемой части можно ничего не выполнять:

2.1) набрать текст программы:

BEGIN END; {в конце – **точка с запятой**}.

2.2) сохранить набранный текст;

2.3) выполнить компиляцию (поиск синтаксических ошибок). Даже в такой «сложной» программе не исключена ошибка – в верхней части экрана появляется строка красного цвета с указанием кода и краткой расшифровки ошибки. Головная программа (глобальный блок) должна заканчиваться точкой, а не точкой с запятой;

2.4) внести изменения в текст, сохранить его, повторно откомпилировать. Ошибок нет, выводится сообщение "*Press any key*". Нажать любую клавишу. Выполняется возврат в окно редактирования, программа готова к загрузке;

2.5) запустить программу на выполнение. Ничего заметного не происходит, ведь наша программа ничего не делает. Но она работает!

3. Усовершенствование **программы**:

3.1) дополнить текст:

PROGRAM PROSTO; {заголовок}
BEGIN
A:=4; {оператор присваивания – соответствует (но не совсем) знаку равно}
B:=6.5; {дробная часть отделяется точкой}
Y:=A+B;
END.

3.2) сохранить программу под тем же именем, откомпилировать;

3.3) опять ошибка! ЭВМ не знает, что такое «*A*» (аналогичная ситуация будет с «*B*» и «*Y*»). Все данные, используемые в программе (константы, переменные и др.), должны быть описаны в разделе описаний, располагаемом после заголовка;

3.4) так как значения *Y*, *A*, *B* могут изменяться в ходе выполнения программы, необходимо отнести их к разряду переменных (они описываются ключевым словом *VAR*). Но переменные тоже могут быть разными, например, целыми или с дробной частью. Последние относят к вещественному типу данных, для их описания используют ключевое слово *REAL*. В инженерных расчетах следует отдавать предпочтение вещественным переменным (за исключением особых случаев, предусмотренных, например, структурой некоторых операторов Паскаль).

После заголовка программы вставить в текст строку

VAR A, B, Y: REAL;

и выполнить компиляцию;

3.5) запустить программу на выполнение;

3.6) открыть окно просмотра, чтобы увидеть результат. Искомой цифры там нет. ЭВМ рассчитала результат и хранила его в памяти, но не было дано команды на его вывод;

3.7) для отображения результатов работы программы на мониторе используют оператор *WRITE* или *WRITELN*:

```
PROGRAM PROSTO;
VAR A, B, Y: REAL;
BEGIN
A:=4;
B:=6.5;
Y:=A+B;
WRITELN(Y);
END.
```

3.8) сохранить изменения, запустить программу, открыть окно просмотра. Там отображается ожидаемое число 10,5 (но форма представления может удивить начинающего программиста).

4. В разработанной программе ввод исходных данных осуществляется в ее тексте с помощью оператора присваивания. Это удобно при отладке программ, но нерационально, так как при изменении набора данных приходится открывать текст программы.

4.1) для ввода данных используют операторы *READ* или *READLN*:

```
PROGRAM PROSTO;
VAR A, B, Y: REAL;
BEGIN
READLN(A); {прочитать значение введенного A и передать его в оперативную память}
READLN(B);
Y:=A+B;
WRITELN(Y);
END.
```

4.2) после запуска программы ее выполнение прерывается: ПЭВМ ждет ввода с клавиатуры значения *A*. Набрать 6.5 (через точку) и нажать клавишу ввода;

4.3) далее ПЭВМ должна прочитать значение *B*. Следует набрать 4 и нажать клавишу ввода;

4.4) в окне просмотра будут отображаться не только результат, но и введенные исходные данные;

4.5) необходимо помнить о сохранении программы после внесения изменений.

5. Чтобы не работать «вслепую» (например, при вводе исходных данных), рекомендуется организовывать общение с ПЭВМ в форме диалога с помощью пояснительных надписей на экране. Например, «ВВЕДИ А», «РЕЗУЛЬТАТ РАВЕН». В протокол работы программы можно включить номер и наименование лабораторной работы, фамилию исполнителя и др. Такие надписи, заключенные в апострофы, выводятся с помощью оператора *WRITELN*. Кроме того, в Паскале предусмотрен форматный вывод чисел, позволяющий задавать их структуру при отображении, что облегчает восприятие информации.

5.1) дополнить текст:

```
PROGRAM PROSTO;
VAR A, B, Y: REAL;
BEGIN
WRITELN('ВВЕДИ A='); READLN(A);
WRITE('ВВЕДИ B='); READLN(B);
Y:=A+B;
WRITELN('РЕЗУЛЬТАТ Y=',Y:5:2);
END.
```

5.2) запустить программу на выполнение. Обратить внимание на отличия при вводе *A* и *B*;

5.3) исправить: *WRITE('ВВЕДИ A=')* и запустить программу повторно.

6. Внести последние **штрихи**:

```
PROGRAM PROSTO;
USES CRT; {подключение модуля}
VAR A, B, Y: REAL;
BEGIN
CLRSCR; {очистка экрана}
WRITELN('ЛАБОРАТОРНАЯ РАБОТА №1');
WRITELN('СТРУКТУРА ПАСКАЛЬ-ПРОГРАММЫ');
WRITE ('ВВЕДИ A='); READLN(A);
WRITE('ВВЕДИ B='); READLN(B);
Y:=A+B;
WRITELN('РЕЗУЛЬТАТ Y=',Y:5:2);
WRITELN('ВЫПОЛНИЛ Ст.Гр. 101719 ИВАНОВ П.П.');
READLN; {задержка выполнения}
END.
```

Оператор очистки экрана стирает протоколы предыдущих сеансов работы программы. Оператор *READLN* в конце программы позволяет организовать задержку ее выполнения и увидеть результаты без входа в окно просмотра. Для возврата в окно редактирования необходимо нажать клавишу ввода.

Лабораторная работа № 2

ТИПОВЫЕ АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

Цель работы

1. Изучить типовые алгоритмы обработки данных.
2. Закрепить знания по применению условного оператора и оператора цикла для обработки данных без использования массивов.
3. Разработать алгоритм и программу в соответствии с вариантом задания.

Теоретические сведения

Для составления программ самого разнообразного назначения используют ряд типовых алгоритмов, к числу простейших из них можно отнести определение суммы и произведения элементов, нахождение значения минимального и максимального элемента, определение количества и номеров элементов, удовлетворяющих заданному критерию, алгоритмы замены и сортировки элементов.

На рисунке 2.1, *а* представлен фрагмент алгоритма для нахождения минимального элемента. На первом шаге задается предполагаемое значение минимального элемента. Если элементы известны заранее, можно предположить, что минимальным является первый. Если элементы не известны, в качестве первоначального предполагаемого минимального элемента задают бесконечно большое число. Далее в цикле по очереди все элементы сравнивают с предполагаемым минимальным. Если проверяемый элемент меньше, чем минимальный на текущем шаге, минимальным становится проверяемый. Кроме того, запоминается его номер, как номер минимального. Если проверяемый элемент больше, чем минимальный на текущем шаге, никакие действия не выполняются, текущий элемент игнорируется, осуществляется переход к проверке следующего. Следует отметить, что если элементы заранее не известны (нет массива данных), необходимо организовать чтение данных внутри цикла (рисунок 2.1, *б*). Поиск максимального элемента выполняется по алгоритму, аналогичному показанному на рисунке 2.1, *а*.

На рисунке 2.1, *б* представлен фрагмент алгоритма суммирования положительных элементов. Первоначально сумме присваивается нулевое значение. Затем в цикле вводится значение очередного элемента и сравнивается с нулем. Если элемент положительный, его значение добавляется к текущей сумме, если нет – элемент игнорируется. Произведение рассчитывают по аналогичному алгоритму, но первоначальному значению присваивают значение единицы.

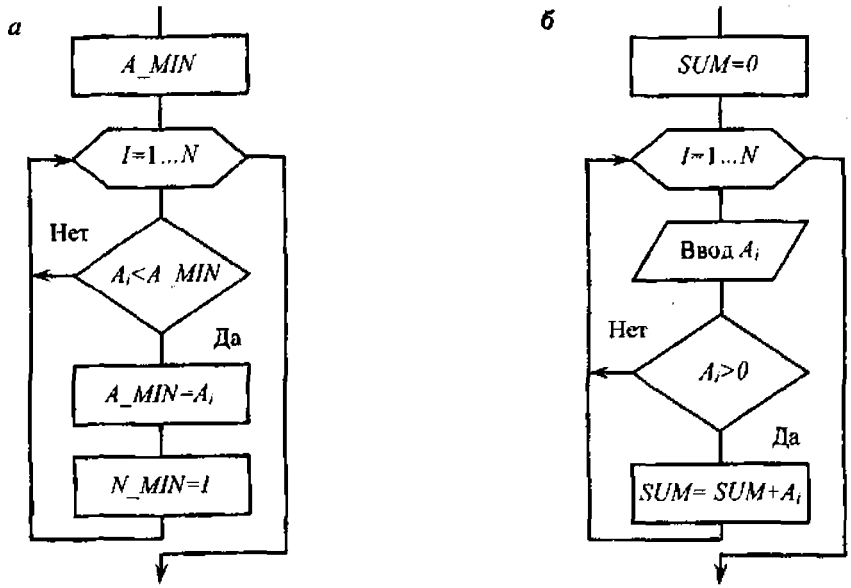

Рисунок 2.1 – Фрагменты типовых алгоритмов:
а – определение значения и номера минимального элемента;
б – расчет суммы положительных элементов

Задание

Организовать ввод с клавиатуры семи вещественных чисел *без использования массивов*. Обеспечить их форматный вывод на дисплей, сопроводив надписью «Исходные данные» и подписав наименование каждого элемента, например: A[1]= ; A[2]= ; и т.д. Способ вывода – в соответствии с вариантом (таблица 2.1). Выполнить обработку данных в соответствии с вариантом задания и вывести на экран результаты расчетов, сопроводив их комментариями.

Таблица 2.1 – Варианты заданий

№ варианта	Задание	Способ вывода
1	Определить значение минимального элемента и его номер	Строка
2	Вычислить произведение положительных элементов и их количество	Столбец
3	Вычислить сумму положительных элементов и их количество	Строка
4	Определить значение максимального элемента и его номер	Столбец
5	Вычислить произведение отрицательных элементов и их количество	Строка
6	Вычислить сумму отрицательных элементов и их количество	Столбец
7	Определить значение максимального по модулю элемента и его номер	Строка
8	Вычислить произведение ненулевых элементов и их количество	Столбец
9	Вычислить сумму ненулевых элементов и их количество	Строка
10	Определить значение минимального по модулю элемента и его номер	Столбец
11	Вычислить произведение квадратов отрицательных элементов и их количество	Строка
12	Вычислить сумму модулей отрицательных элементов и их количество	Столбец
13	Вычислить произведение модулей отрицательных элементов и их количество	Строка
14	Вычислить сумму квадратов отрицательных элементов и их количество	Столбец

Лабораторная работа № 3

ОДНОМЕРНЫЕ МАССИВЫ

Цель работы

1. Изучить порядок описания и ввода одномерных массивов.
2. Изучить алгоритм нахождения максимального (минимального) элемента массива и его номера.
3. Приобрести навыки программирования циклических процессов с использованием массивов данных.

Теоретические сведения

Массивом называется упорядоченная совокупность конечного числа данных одного типа. Массив может быть *одномерным* и *многомерным*. Например, одномерный массив – последовательность чисел.

При *обозначении* массива все его элементы имеют *одинаковое* имя, но каждый элемент имеет *свой индекс*, определяющий место элемента массива в упорядоченной последовательности данных (чисел). Имя массива составляется так же, как имя переменной (алфавитно-цифровыми символами). Размерность массива задается при его описании в начале программы. Для того чтобы определить *место* элемента в массиве, к имени массива дописываются *индексы в квадратных скобках*.

Массив описывается в разделе описаний программы. Сделать это можно *двумя способами*. Для одномерного массива:

1) ***TYPE*** имя типа = ***ARRAY***[*t1*] ***OF*** *type1*;
   VAR имя массива: имя типа;
2) ***VAR*** имя массива: ***ARRAY***[*t1*] ***OF*** *type1*;

где *t1* – тип *индексов* массива;
type1 – тип *элементов*, из которых состоит **массив.**

Значение индекса может быть любым данным скалярного типа, кроме *REAL*. Для вычислительных процессов наиболее часто употребляются ограниченные типы индексов и целые значения индексов (например, 1..10). Конкретное значение индекса определяет *место элемента в массиве*. Для того чтобы указать произвольный элемент массива, необходимо записать имя массива и индекс элемента, например, *MAS* [6] – шестой элемент массива с именем *MAS*.

Пример описания массива по первому способу:

```
TYPE C=ARRAY[1 .. 10] OF INTEGER;
VAR MAS1:C;
```

В программе создается некоторый новый тип данных *C*, представляющий собой одномерный массив, состоящий из десяти элементов целого типа. Все переменные, которые будут относиться к типу *C*, представляют собой массив указанной структуры. Далее следует запись, указывающая, что переменная *MAS*1 относится к типу *C*.

В соответствии со второй формой описания

```
VAR MAS1:ARRAY[1 .. 10] OF INTEGER;
```

Ввод-вывод одномерных массивов осуществляется с помощью *операторов цикла*. Например,

```
FOR I:=1 TO Imax DO READ(MAS1[I]);
```

где *Imax* – размер массива (количество элементов).

В лабораторной работе требуется найти максимальный или минимальный элемент массива и поменять его местами с указанным в соответствии с вариантом задания.

Общий алгоритм решения задачи заключается в следующем:

1) в разделе описаний описываются все типы данных, в том числе массивы;

2) с помощью оператора цикла вводятся элементы массива;

3) далее необходимо организовать вывод исходного массива;

4) затем в цикле с помощью условного оператора *IF* определяются *максимальный (минимальный) элемент массива и его номер* (индекс). Например, для нахождения минимального элемента сначала предполагается, что таковым является первый. В цикле все элементы массива поочередно сравниваются с минимальным. Если в массиве находится элемент, меньший, чем предполагаемый минимальный, то этот элемент становится минимальным, причем необходимо запомнить его номер, присвоив соответствующей переменной текущее значение параметра цикла. Если в дальнейшем появится еще меньший элемент, произойдет переприсваивание минимального значения и номера элемента (фрагмент алгоритма представлен на рисунке 2.1,*а* лабораторной работы № 2);

5) в конце программы следует вывести искомый элемент с указанием его индекса.

Задание

Разработать программу для определения номера и значения максимального (минимального) элемента одномерного массива. Варианты исходных данных представлены в таблице 3.1.

Ввод исходных данных осуществить с клавиатуры в диалоговом режиме. Обеспечить их форматный вывод на дисплей, снабдив надписью «Исходные данные» и подписав наименование каждого элемента, например: A[1]= ; A[2]= ; и т.д. Предусмотреть форматный вывод результатов, очистку экрана и задержку выполнения программы в конце расчета.

Таблица 3.1 – Варианты заданий

Номер варианта	Количество элементов	Тип элементов	Искомый элемент массива
1	5	Целый	Минимальный
2	6	Целый	Максимальный
3	8	Вещественный	Минимальный
4	7	Вещественный	Максимальный
5	6	Целый	Минимальный
6	8	Целый	Максимальный
7	7	Вещественный	Минимальный
8	5	Вещественный	Максимальный
9	7	Целый	Минимальный
10	9	Целый	Максимальный
11	7	Вещественный	Минимальный
12	6	Вещественный	Максимальный
13	8	Целый	Минимальный
14	7	Целый	Максимальный
15	6	Вещественный	Минимальный
16	5	Вещественный	Максимальный
17	8	Целый	Минимальный
18	7	Целый	Максимальный
19	6	Вещественный	Минимальный
20	5	Вещественный	Максимальный

Лабораторная работа № 4

ВНЕШНИЕ ФАЙЛЫ

Цель работы

1. Ознакомиться со стандартными процедурами для работы с внешними файлами.

2. Изучить порядок чтения и записи данных через внешний файл.

3. Разработать программу в соответствии с вариантом задания.

Теоретические сведения

При решении инженерных задач для создания баз данных и формирования технических отчетов необходимо хранить результаты выполнения программ на внешних носителях, например, на жестких дисках. Для этого используют *внешние файлы* – структурированные типы данных, содержащие последовательность компонентов одного типа и хранящиеся на внешних носителях. Они могут существовать отдельно от программы.

Число элементов в файле (его длина) не фиксировано. Элементы файла записываются последовательно с помощью некоторого устройства, называемого *указателем файла*. При чтении (записи) указатель перемещается к следующему элементу и делает его (и только его) доступным для обработки.

Внешний файл, из которого считываются данные, называется *входным*. Файл, в который записываются данные по результатам работы программы, называется *выходным*. В ходе выполнения программы файл с одним и тем же именем может быть как входным, так и выходным.

Для краткого обозначения файла используют файловую переменную, которая должна быть описана в разделе описаний одним из двух способов:

1) ***TYPE*** имя типа = ***FILE OF*** базовый тип;
   VAR имя файла: имя типа;
2) ***VAR*** имя файла: ***FILE OF*** базовый тип;

где базовый тип – тип данных, входящих в файл.
Например, для файла из целых чисел

1) *TYPE FD=FILE OF INTEGER; VAR F1:FD;*
2) *VAR F1:FILE OF INTEGER*;

Для работы с внешними файлами предусмотрен ряд стандартных процедур. Самые распространенные операции при работе с внешними файлами – чтение и запись данных.

Чтение файла – ввод данных из внешнего файла в оперативную память ЭВМ для выполнения программы. Для чтения данных из внешнего файла необходимо выполнить следующие действия.

1. Описать файловую переменную.
2. Присоединить файл.

Для этого используется стандартная процедура *ASSIGN*, связывающая файловую переменную с именем внешнего файла и его адресом. Эта команда должна быть записана в программе в разделе операторов до начала работы с внешним файлом:

ASSIGN (*имя*, '*адрес*'); ,

где *имя* – имя файловой переменной;

адрес – имя накопителя, где хранится файл (указывается в апострофах).

3. Открыть файл для чтения.

Используется стандартная процедура *RESET*, подготавливающая файл для считывания из него данных, причем указатель файла устанавливается на первый элемент. Записывается перед первым оператором чтения данных:

RESET (*имя*); ,

где *имя* – имя файловой переменной.

4. Прочитать данные из файла.

Используется команда *READ*:

READ (*имя*, *список*); ,

где *имя* – имя файловой переменной;

список – список считываемых переменных, разделенных запятыми.

5. Закрыть файл.

Используется процедура *CLOSE*:

***CLOSE** (имя);* ,

где *имя* – имя файловой переменной.

Обычно ставится в конце программы или после последнего *READ*.

Например, файл данных для лабораторной работы с именем *lab_FD.pas*, хранящийся в текущем каталоге, можно обозначить файловой переменной *FD* (она предварительно описывается в разделе описаний *VAR FD:FILE OF REAL*;), а операции по считыванию переменных *A*, *B* будут выглядеть следующим образом (значения переменных заранее должны быть записаны в файл):

```
ASSIGN(FD, 'lab_FD.pas')
RESET(FD);
READ(FD,A,B);
CLOSE(FD).
```

Запись в файл – это вывод данных из оперативной памяти ЭВМ на внешний носитель в ходе выполнения программы. Для записи данных во внешний файл необходимо выполнить следующие действия:

1. Описать файловую переменную.
2. Присоединить файл (аналогично чтению).
3. Открыть файл для записи.

Используется стандартная процедура *REWRITE*, подготавливающая файл к приему некоторой информации, которая получается в результате выполнения программы и должна сохраниться на длительный срок, причем вся предыдущая информация из файла удаляется. Записывается перед первой командой записи данных в файл:

***REWRITE** (имя);* ,

где *имя* – имя файловой переменной (для входных и выходных файлов имена файловых переменных могут быть разными).

4. Записать данные в файл.

Используется команда *WRITE*:

***WRITE** (имя, список);* ,

где *имя* – имя файловой переменной;

список – список записываемых переменных, разделенных запятыми.

5. Закрыть файл.

Используется процедура *CLOSE*, которая обычно ставится в конце программы или после последнего *WRITE*.

Например, результирующий (выходной) файл для лабораторной работы с именем *lab_FR.pas*, хранящийся в текущем каталоге, можно обозначить файловой переменной *FR* (описывается *VAR FR:FILE OF REAL;*), а операции по записи переменных *AA*, *BB* будут выглядеть следующим образом:

```
ASSIGN(FR, 'lab_FR.pas')
REWRITE(FR);
WRITE(FR,AA,BB);
CLOSE(FR).
```

Предусмотрены и некоторые другие операции с внешними файлами.

Для добавления данных к файлу выполнить следующие действия:

1. Открыть существующий файл для чтения (*RESET*).

2. Установить указатель файла за последним его компонентом.

Для этого используется процедура *SEEK*, позволяющая осуществить прямой доступ к элементам файла (перемещает указатель, но не читает или записывает):

***SEEK** (имя, N);* ,

где *имя* – имя файловой переменной;

N – целая константа, соответствующая порядковому номеру элемента в файле, причем первый элемент имеет номер *N* = 0, второй – *N* = 1 и т.д.

Если воспользоваться стандартной функцией *FileSize* (имя), позволяющей определить число элементов файла, записав ее на место *N*, то указатель переместится в конец файла.

3. Записать дополнительные данные (*WRITE*).

4. Закрыть файл (*CLOSE*).

Для корректировки отдельных элементов файла необходимо выполнить следующие действия:

1) открыть корректируемый файл для чтения (*RESET*);

2) подвести указатель к корректируемому элементу (*SEEK*);

3) прочитать корректируемый элемент (*READ*);

4) изменить значение элемента;

5) повторить процедуру подвода указателя (он сместился при чтении);

6) записать откорректированный элемент (*WRITE*);

7) закрыть файл (*CLOSE*).

Для организации циклов при работе с файлами удобно пользоваться стандартной функцией *EOF*(имя), которая принимает значение *True*, когда достигнут конец файла, в остальных случаях ее значение *False*.

Задание

Организовать ввод с клавиатуры *N* чисел заданного типа и записать их во внешний файл. Прочитать их из файла. Ввести еще один элемент заданного типа. Записать его в файл на место элемента с номером *K*, а элемент, стоявший ранее на этом месте, дописать в конец файла. Вывести на экран значения элементов файла до и после преобразований, снабдив соответствующими надписями. Варианты заданий представлены в таблице.

Варианты заданий

№ варианта	Количество элементов N	Номер преобразуемого элемента K	Тип элементов файла	Способ вывода на экран
1	5	3	Вещественный	Строка
2	7	4	Целый	Столбец
3	4	2	Вещественный	Строка
4	5	4	Целый	Столбец
5	7	5	Вещественный	Строка
6	4	3	Целый	Столбец
7	5	2	Вещественный	Строка
8	7	3	Целый	Столбец
9	4	3	Вещественный	Строка
10	5	2	Целый	Столбец
11	7	2	Вещественный	Строка
12	4	1	Целый	Столбец
13	5	5	Вещественный	Строка
14	7	7	Целый	Столбец

Лабораторная работа № 5

ТЕКСТОВЫЕ ВНЕШНИЕ ФАЙЛЫ

Цель работы

1. Изучить дополнительные стандартные процедуры для работы с текстовыми внешними файлами.

2. Изучить порядок описания и ввода-вывода двумерных массивов.

3. Приобрести навыки по оформлению результатов выполнения лабораторной работы в виде текстового файла.

Теоретические сведения

Для оформления технических отчетов по результатам выполнения расчетных задач удобно пользоваться *текстовыми файлами.* Такой файл состоит из обычных символов (букв, цифр) и разбивается на строки различной длины, как в текстовом документе. Каждая строка заканчивается символом конца строки. Общие принципы работы с текстовым файлом такие же, как и с обычным внешним файлом (см. лабораторную работу № 4).

Некоторую особенность имеет описание текстового файла:

VAR имя: TEXT; ,

где *имя* – имя файловой переменной.

Например, *VAR F1:TEXT*; .

К ним применимы дополнительные стандартные процедуры:

WRITELN (*имя*) – конец текущей строки при записи, переход на новую строку;

READLN (*имя*) – переход к началу следующей строки при вводе;

WRITELN (*имя, список*) – вывод (печать) *списка* переменных, разделенных запятыми, с завершением текущей строки и переходом на следующую;

READLN (*имя, список)* – чтение *списка* переменных, разделенных запятыми, с переходом к новой строке;

APPEND (*имя*) – открытие уже существующего текстового файла для добавления данных в конец.

Операции *SEEK, FileSize* для текстовых файлов применять нельзя.

Определить конец строки в текстовом файле можно с помощью функции *EoLn* (имя). Она принимает значение *True*, если достигнут конец строки, и *False* в противном случае.

Все файлы, кроме текстовых (например, *of real*), представляют собой набор чисел. Символьных или текстовых записей

они не воспринимают. В текстовом же файле можно работать с различными типами данных по аналогии с выводом на дисплей или «ручным» оформлением документов.

Для *чтения данных из текстового* внешнего файла (цифровые значения отделяются пробелом) необходимо выполнить следующие действия:

1) описать файловую переменную (*VAR*);
2) присоединить файл (*ASSIGN*);
3) открыть файл для чтения (*RESET*);
4) прочитать данные из файла (*READ, READLN*);
5) закрыть файл (*CLOSE*).

Для *записи данных в текстовый* внешний файл:

1) описать файловую переменную (*VAR*);
2) присоединить файл (*ASSIGN*);
3) открыть файл для записи (*REWRITE*);
4) записать данные в файл (*WRITE, WRITELN*);
5) закрыть файл (*CLOSE*).

Массив – упорядоченная совокупность конечного числа данных одного типа. Пример *двумерного массива* – матрица. Используют два способа описания двумерных массивов в разделе описаний программы:

1) *TYPE имя типа=ARRAY*[*t*1,*t*2] *OF type*1;
   VAR имя массива: *имя типа*;
2) *VAR имя массива*: *ARRAY*[*t*1,*t*2] *OF type*1; ,

где *t*1, *t*2 – тип индексов двумерного массива по каждому из измерений;

*type*1 – тип элементов, из которых состоит массив.

Например, массив *MAS*2 должен быть описан как *MAS*2 [1..2,1..5], т.е. $n = 2$ (двумерный), в котором тип индексов *t*1 и *t*2 является ограниченным, а значения индексов – целые числа от 1 до 2 и от 1 до 5 по соответствующим измерениям массива. Для того чтобы указать произвольный элемент двумерного

массива, необходимо записать имя массива и индексы элемента *MAS2* [*I,J*], причем *I, J* должны быть описаны как данные целого типа.

Описание массива *MAS* 2 по *первому способу*:

```
TYPE D=ARRAY[1..2,1..5] OF INTEGER;
VAR MAS2:D; .
```

В программе создается тип данных *D*, представляющий собой двумерный массив размерностью 2 х 5, состоящий из десяти элементов целого типа. Следующая запись указывает, что переменная *MAS2* относится к типу *D*.

В соответствии со *второй* формой описания:

```
VAR MAS2:ARRAY[1..2,1..5] OF INTEGER; .
```

Для *ввода-вывода* двумерного массива используются вложенные циклы, причем сначала выполняется внутренний цикл, а затем – внешний. Например, для ввода элементов двумерного массива *MAS2*:

```
FOR I:=1 TO Imax DO
FOR J:=1 TO Jmax DO
READ (MAS2 [I, J]);
```

где *Imax*, *Jmax* – данные, **определяющие размер массива по** каждому из двух измерений.

Задание

Подготовить текстовый файл со значениями элементов двумерного массива размерностью *N* строк и *M* столбцов (создать в среде Турбо-Паскаль и записать туда числовые данные). Прочитать данные из файла, вывести их на экран в виде матрицы, снабдив надписью вида «*A*[*I,J*]=…», указав конкретные

значения индексов элементов массива. Организовать внешний файл для записи отчета о лабораторной работе. В результирующий файл поместить:

1) номер лабораторной работы;

2) название работы;

3) значения считанных элементов массива в соответствии с заданным форматом в виде матрицы с подписью каждого элемента вида «$A[I,J]$=...» и указанием конкретных значений индексов элементов массива;

4) сведения об исполнителях.

Варианты заданий представлены в таблице.

Варианты заданий

№ варианта	Количество строк N	Количество столбцов M	Тип элементов массива	Формат вывода
1	5	3	Вещественный	Фиксированный, 10
2	4	3	Целый	1
3	3	4	Вещественный	Плавающий, 5:2
4	4	5	Целый	2
5	5	3	Вещественный	Фиксированный, 12
6	4	3	Целый	3
7	5	3	Вещественный	Плавающий, 4:1
8	5	4	Целый	1
9	4	3	Вещественный	Фиксированный, 8
10	2	5	Целый	2
11	5	2	Вещественный	Плавающий, 4:2
12	4	2	Целый	3
13	5	4	Вещественный	Плавающий, 6:3
14	4	5	Целый	2

Лабораторная работа № 6

ПАРАМЕТРЫ ПРОЦЕДУРНОГО ТИПА В ПОДПРОГРАММАХ

Цель работы

1. Приобрести навыки использования подпрограмм.
2. Изучить механизм передачи данных в подпрограммах.
3. Изучить порядок описания и применения процедурных переменных.

Теоретические сведения

В Паскале используют два вида подпрограмм: *процедуры и функции*. Их структура аналогична структуре головной программы и включает заголовок, раздел описаний и раздел операторов. Текст подпрограммы в основной программе располагается в разделе описаний, перед разделом операторов, например:

PROGRAM LAB;
VAR ...;
текст подпрограммы
BEGIN ... END.

Подпрограмма может быть записана в *отдельный файл* (в конце после end ставят точку с запятой, а не точку, как в головной программе). Для того чтобы включить эту подпрограмму в текст головной программы, используют так называемые директивы компилятора (дополнительные указания), которые записывают перед разделом операторов:

{$I имя и маршрут доступа к файлу}.

Например, если подпрограмма оформлена в виде отдельного файла *PP.PAS*, запись будет выглядеть следующим образом:

PROGRAM LAB;
VAR...;
{$I PP.PAS} – т.е. включить в программу ***LAB*** текст файла *PP.PAS.*
Begin ... end.

Структура подпрограммы-функции имеет следующий вид:

FUNCTION *имя (формальные параметры):* ***тип результата;***
раздел описаний;
BEGIN раздел операторов ***END***;

Структура процедуры:

PROCEDURE *имя (формальные параметры);*
раздел описаний;
BEGIN раздел операторов ***END***; .

Здесь *имя* – название процедуры или функции. *Формальные параметры* – список переменных с указанием их типа. Переменные одного типа разделяются запятой, перед указанием типа ставится двоеточие, списки переменных разных типов разделяются точкой с запятой. Те переменные, которые описываются как формальные параметры (в скобках), в разделе описания подпрограммы не описываются.

Формальные параметры – переменные, с использованием которых написана универсальная последовательность действий (для всех пользователей).

Параметры могут быть и фактическими. *Фактические* – это те данные, с которыми работает программист в конкретной программе. Формальные и фактические параметры могут совпадать.

Параметры могут быть *трех типов:* входные (заданные значения); выходные (вычисляемые переменные); параметры процедурного типа.

Пример *заголовка* подпрограммы-функции:

FUNCTION F(X,Y:REAL): REAL;

Результат выполнения функции – есть *имя* функции, т.е. результат присваивается идентификатору, обозначающему название подпрограммы-функции. Результатом выполнения функции (выходным параметром) может быть только одно значение, совпадающее с именем этой функции, поэтому в заголовке указывается тип имени, т.е. *тип результата*. В разделе операторов функции должен обязательно присутствовать оператор присваивания какого-либо значения переменной, обозначающей имя функции.

Пример *заголовка* процедуры:

PROCEDURE PP(X,Y:REAL; Z:Integer; VAR A:REAL);

При описании выходных параметров процедуры перед ними ставится ключевое слово ***VAR***. В процедуре может быть несколько выходных параметров. Если они относятся к разным типам, *VAR* записывают перед списком переменных каждого типа.

Вызов подпрограммы-функции осуществляется непосредственно в арифметическом выражении указанием ее имени и перечислением фактических параметров (подпрограмма-функция – часть некоторого оператора):

имя (фактические параметры);

Вызов процедуры осуществляется указанием в программе ее имени и перечислением фактических параметров (самостоятельный оператор):

имя (фактические параметры);

Список фактических параметров подпрограмм должен соответствовать списку формальных параметров по их количеству, типу и последовательности перечисления.

Если в качестве параметров процедуры используются *данные сложных типов* (например, массивы), необходимо в головной программе описать имя типа этих данных (с помощью *TYPE*), а затем имя типа указывается в списке формальных параметров.

Например, если в качестве входного параметра используется массив *A* из пяти вещественных элементов, его описывают в головной программе:

```
TYPE MAS1=ARRAY[1..5] OF REAL;
VAR A:MAS1;  ,
```

а в процедуре указывают, что переменная *A* относится к типу *MAS*1:

```
PROCEDURE B(A:MAS1;VAR S: REAL);   .
```

В ряде случаев необходимо, чтобы в списке параметров процедур и функций находились имена других процедур или функций. Имена таких подпрограмм называют *процедурными параметрами.* Переменные процедурного типа должны быть специально описаны:

```
TYPE имя типа 1=PROCEDURE(формальные параметры);
        имя типа 2=FUNCTION(формальные параметры):тип результата;
VAR  переменная 1: имя типа 1; переменная 2: имя типа 2;
```

Рекомендуется все процедуры и функции, имена которых присваиваются процедурным переменным, транслировать в специальном режиме: перед заголовком такой процедуры или функции ставится директива компилятора {*$F+*}, а после нее –

{*$F-*}, т.е. компилятору дается указание, что имя процедуры в этой части программы должно восприниматься как возможный процедурный параметр. Например:

PROGRAM LAB;
TYPE FUNK=FUNCTION (X:REAL):REAL;
VAR ...;
{$F+}
FUNCTION A (X:REAL): REAL;
раздел описаний подпрограммы- функции;
BEGIN раздел операторов *END*;
{$F-}
PROCEDURE PP (A:FUNK; Z:Integer; VAR B:REAL);
раздел описаний процедуры;
BEGIN раздел операторов *END*;
BEGIN
текст головной программы
END.

Задание

Разработать процедуру формирования массива из N элементов, полученных по результатам расчета некоторой функции $F(x)$ на интервале $[a, b]$. Используя процедуру, просуммировать два массива, сформированных в соответствии с вариантом задания (таблица 6.1). Функция, по которой рассчитываются значения элементов массива, должна быть оформлена в виде подпрограммы. Вывести во внешний текстовый файл сформированные и результирующий массивы.

Таблица 6.1 – Варианты заданий

№ варианта	Количество элементов	Интервал для первого массива	Интервал для второго массива	Функция для первого массива	Функция для второго массива
1	6	[0; π/2]	[1; 10]	x+sin(x)	ctg(x/3) + sin(x)
2	7	[1; π]	[0; π/2]	sin(1/x)	x-sin(x)
3	8	[1; π/3]	[π/2; π]	cos(1/x)	sin(x)
4	6	[0; π]	[0,π]	x-sin(x)	cos(x)
5	5	[0; 1]	[0; π/3]	sin(x)	tg(x)
6	7	[0; π]	[1; 6]	cos(x)	ctg(x)
7	6	[1; π]	[0; π/2]	cos(x) + ctg(x)	x + sin(x)
8	8	[0; 7]	[1; 5]	tg(x)	sin(1/x)
9	7	[1;8]	[π/2; π]	ctg(x)	cos(1/x)
10	5	[0;1]	[1; 4]	arcsin(x)	$\sin(x^2)$
11	6	[π/2; π]	[1; 5]	ctg(x/3) + sin(x)	$\cos(x^2)$
12	7	[1; 8]	[0; π/3]	$\sin(x^2)$	cos(x) + ctg(x)
13	8	[0; 7]	[0,5; 1]	$\cos(x^2)$	arcsin(x)
14	5	[0;π/3]	[0; 0,5]	tg(x/2)	arccos(x)

Рекомендация к выполнению работы: формальными параметрами процедуры должны быть количество элементов, границы интервала, имя функции, выходной массив.

Список использованных источников

1. Офицеров, Д.В. Программирование в интегрированной среде Турбо-Паскаль: справочное пособие / Д.В. Офицеров, В.А. Старых. – Минск: Беларусь. , 1992. – 240 с.

2. Офицеров, Д.В. Программирование на персональных ЭВМ: Практикум: учебное пособие / Д.В. Офицеров, А.Б. Долгий, В.А. Старых. – Минск: Вышэйшая школа, 1993. – 256 с.

3. Жилевич, М.И. Информатика: учебно-методическое пособие для студентов специальности 1-36 01 07 «Гидропневмосистемы мобильных и технологических машин» / М.И. Жилевич, Л.Г. Филипова. – Минск: БНТУ, 2003. – 77 с.

Содержание

Учебное издание

ЖИЛЕВИЧ Михаил Иванович
ФИЛИПОВА Людмила Геннадьевна

ИНФОРМАТИКА

Учебно-методическое пособие
к лабораторным работам для студентов специальности 1-36 01 07
«Гидропневмосистемы мобильных и технологических машин»

Редактор Т.Н. Микулик
Компьютерная верстка Д.К. Измайлович

Подписано в печать 29.04.2009.
Формат 60×84 1/16. Бумага офсетная.
Отпечатано на ризографе. Гарнитура Таймс.
Усл. печ. л. 2,32. Уч.-изд. л. 1,82. Тираж 100. Заказ 1100.

Издатель и полиграфическое исполнение:
Белорусский национальный технический университет.
ЛИ № 02330/0494349 от 16.03.2009.
Проспект Независимости, 65. 220013, Минск.