Федеральное агентство по образованию
ГОУ ВПО «Уральский государственный технический университет – УПИ
имени первого Президента России Б. Н. Ельцина»

В.Г. Томашевич

РЕШЕНИЕ ЗАДАЧ В ИНТЕГРИРОВАННОЙ СРЕДЕ ТУРБО ПАСКАЛЬ

Учебное пособие

Подготовлено кафедрой «Интеллектуальные информационные технологии»
Научный редактор: доц., канд. техн. наук И.Н. Обабков

Учебное пособие по дисциплине «Информатика» предназначено для студентов всех форм обучения всех специальностей

В учебном пособии рассматриваются основные конструкции алгоритмического языка Паскаль, приводятся варианты заданий для проведения практических занятий и лабораторных работ по основным разделам языка. Приведены справочные сведения, необходимые для правильного оформления блок-схем. Для контроля степени усвоения материала по каждой теме приводятся контрольные вопросы.

Екатеринбург
2009

Оглавление

Введение

Благодаря простоте и доступности алгоритмический язык Паскаль получил широкое распространение. Именно эти качества ставил во главу угла швейцарский профессор Никлаус Вирт, создавший в конце 60-х годов этот язык – язык для обучения начинающих программировать.

Успеху замысла способствовало и то, что по своей идеологии Паскаль близок к современной теории и технологии программирования, так как хорошо отражает идеи структурного программирования. Кроме того, он приспособлен для применения технологии разработки программ методом пошаговой детализации. И, наконец, язык предоставляет широкие возможности работы – как с числовой, так и с символьной информацией. Паскаль позволяет создавать не только простые программы, но и структурированные программы большой сложности.

Язык постоянно совершенствовался. Для персональных компьютеров появились передовые для своего времени компиляторы с языка Паскаль для IBM PC, представляющие диалоговые системы, называемые *интегрированными средами* (Turbo Pascal). К настоящему времени существует ряд версий интегрированных сред Turbo Pascal: 5.0, 5.5, 6.0, 7.0 и 8.0; причем соблюдается принцип их совместимости снизу вверх. В версию 5.0 был включен интегрированный Turbo-отладчик; версия же 5.5 позволила создавать объектно-ориентированные программы. В свою очередь версия 6.0 располагала тремя новыми возможностями: интегрированной средой для разработчика; дополнительными режимами транслятора и встроенным ассемблером; объектно-ориентированной оболочкой для создания прикладных программ – Turbo-Vision. Кроме того, версия 6.0 позволяет редактировать несколько файлов и дает новые возможности управления интегрированной средой при помощи манипулятора «мышь».

Задача данного пособия состоит в том, чтобы познакомить студентов с интегрированной средой Turbo Pascal и научить их решать простейшие задачи с использованием этой среды.

Основные понятия Турбо Паскаль
и структура программы

Даже самые мощные современные процессоры могут выполнять лишь определенный набор относительно простых действий, таких как арифметические и логические операции, пересылка данных из одного места в другое, сравнение данных, передача управления от одного участка программы в другой и т. п. Все интеллектуальные функции компьютера реализуются с помощью соответствующего программного обеспечения, которое по мере развития техники все более усложняется. Разработка новых программных средств в машинных кодах или на языке низкого уровня – ассемблере – слишком трудоемка, поэтому и используются языки высокого уровня (Паскаль, Си, Фортран, Бейсик и другие), которые легче воспринимаются человеком. В этих языках исполняемая программа в машинных кодах получается из исходного текста программы на соответствующем языке путем компиляции.

Программа на языке высокого уровня представляет собой текст, состоящий из имен, чисел, строк, зарезервированных ключевых слов, специальных символов, при необходимости отделяемых друг от друга разделителями – пробелами.

Любая программа на языке Паскаль должна включать в себя две части – раздел описаний всех используемых данных и раздел операторов, определяющих выполняемые действия над данными. В раздел описаний входят объявления констант, переменных, типов, процедур и функций. Раздел операторов представляет собой исполняемую часть программы – основной блок, располагающийся между ключевыми словами **begin** и **end.** Кроме того, в программу следует включать пояснения – комментарии, заключаемые в фигурные скобки. Комментарии необходимы для облегчения понимания программы, что особенно важно, если в дальнейшем возможна модификация программы.

Общую структуру программы на Turbo Pascal можно представить так.

Program	Заголовок программы;
uses	список подключаемых библиотек (модулей);
label	список используемых меток;
const	объявление констант;
type	объявление типов;
var	объявление переменных;
procedure	тексты процедур
function	и функций;
begin	основной блок программы;

ОПЕРАТОРЫ

end.

Заголовок программы состоит из зарезервированного слова **Program** и имени программы. В Турбо Паскале заголовок можно опустить. Раздел **uses** может быть использован в программе только один раз и должен предшествовать всем другим описаниям. Остальные разделы, кроме основного блока, могут располагаться в программе в любом порядке и встречаться несколько раз. Единственное ограничение состоит в том, что любое описание данных должно предшествовать их использованию. В принципе все части программы, кроме основного блока, могут отсутствовать. Все данные, используемые в программе, должны иметь имена – идентификаторы. Идентификатор представляет собой последовательность букв латинского (именно латинского, а не русского!) алфавита и цифр, начинающуюся с буквы. Внутри идентификатора допускается символ подчеркивания, но не пробел. Заглавные и прописные буквы не различаются, например, имена NAME, Name и name идентичны. В качестве идентификаторов нельзя использовать зарезервированные слова языка, такие, например, как **begin**, **end**, **program** и т. п. Длина идентификатора не ограничена, но значащими считаются первые 63 символа. Примеры правильных и неправильных идентификаторов приведены ниже.

Правильные	**Неправильные**
a12	4ax
First_program	First program
B456_x_ext	?ae
XYZ	русский_текст
integer_val	integer

Данные в программе делятся на константы и переменные. Константы не изменяют свои значения в процессе выполнения программы, их тип определяется способом записи. Например, 56 – целая константа, 3.14 – вещественная, ‘*c*’ – символьная. Описание раздела констант представляет собой следующую структуру:

const имя1 = значение1; имя2 = значение2; ... имя*n* = значение*n*;

Например,

const *N* = 10;

Stroka = ‘это текстовая константа’;

Переменные являются основным элементом программы, предназначенным для хранения, изменения и передачи данных. В отличие от констант значения переменных изменяются в ходе выполнения программы. Описать переменную – значит определить ее имя и тип. Объявление переменных начинается зарезервированным словом **var**, за которым следуют списки имен с указанием через двоеточие их типа. Списки разделяются символом «точка с запятой», а несколько переменных одного типа можно перечислять через запятую перед объявлением их типа. Например,

var

a, b, c: integer;{целые переменные с именами a, b, c}

x, y: real;{вещественные переменные x и y}

ch: char; {символьная переменная ch}

В качестве типа можно использовать стандартные (предопределенные) типы языка Паскаль или типы, ранее определенные программистом в разделе **type**.

Термины «процедура» и «функция» применяются в Паскале для выделения в рамках общего алгоритма специально оформленной последовательности действий, чаще всего представляющих из себя логически завершенный алгоритм. Процедуры и функции могут вызываться из любого места основного блока или другой процедуры, описание которой располагается ниже. Использование процедур и функций позволяет структурировать программу, представляя ее в виде совокупности более простых модулей, что делает программу проще в понимании и отладке.

Основной блок программы начинается зарезервированным словом **begin**, завершается **end.** (end с точкой) и состоит из операторов, разделяемых символом «точка с запятой». Работа программы состоит в последовательном выполнении всех операторов основного блока.

Все операторы языка Турбо Паскаль делятся на простые и структурные. Простые операторы не содержат в себе других операторов. К ним относятся: оператор присваивания (:=), оператор перехода (**goto**) и оператор обращения к процедуре. Структурными являются операторы, состоящие из других операторов. Это:

- составной оператор (блок);
- условный оператор **if**;
- оператор выбора **case**;
- операторы цикла **while**, **repeat**, **for**;
- оператор присоединения (для записей) **with**.

Составной оператор представляет собой последовательность операторов, заключенную в операторные скобки **begin** и **end**. Он позволяет рассматривать несколько последовательно выполняемых операторов как один. Перечисленные операторы будут подробно рассмотрены далее.

Отдельно стоят операторы ввода данных **read** (**readln**) и вывода данных **write** (**writeln**). Фактически они являются вызовами процедур и предназначены для ввода и вывода данных, перечисленных в круглых скобках.

Ниже приведен пример простой программы на Паскале, которая вводит с клавиатуры два числа, вычисляет их полусумму и выводит результат на экран.

```
Program Summa;
{Программа предназначена для вычисления полусуммы двух вещественных чисел}
Var
    a, b, sum: Real;
Begin
    WriteLn (‘Введите два числа’);
    ReadLn (a, b);
    sum:= (a+b)/2;
    WriteLn (‘полусумма (a+b)/2 = ’, sum);
    ReadLn;
end.
```

Если набрать текст этой программы в редакторе Турбо Паскаля и запустить ее, то сразу после запуска программа выведет на экран текст «Введите два числа», который является подсказкой для пользователя. В ответ нужно набрать на клавиатуре два числа через пробел (через несколько пробелов или клавишу «Enter») и нажать клавишу «Enter». После этого произойдет возврат обратно в редактор текстов программ, а на экране появится текст «полусумма (a+b)/2 = » и результирующее число в экспоненциальном формате, принятом по умолчанию для вещественных чисел. Наличие оператора ReadLn(readln) в предпоследней строке дает возможность посмотреть результат не прибегая к помощи комбинации клавиш <Alt>+<F5>.

Контрольные вопросы

1. Можно ли изменить значение константы в процессе выполнения программы?
2. Можно ли опустить в программе раздел описания переменных?
3. Может ли отсутствовать в программе основной блок?
4. Может ли ключевое слова **end.** (end с точкой) встретиться внутри основного блока программы?
5. Можно ли повторять в программе слово **uses**?
6. Может ли в программе быть несколько разделов **const**?
7. Может ли ключевое слово **begin** встретиться несколько раз в основном блоке программы?
8. Каким символом разделяются операторы в программе?
9. Является ли оператор присваивания структурным?
10. Какие из следующих примеров идентификаторов неправильные?
    - XY_del_2
    - XY/2
    - begin
    - x28_1
    - EndProgram
11. Какие из следующих описаний неверны?
    - **va** X: integer;
    - **const** c:real;
    - **var** N=10;
    - **const** d=100.0;
12. Исправить ошибку в следующем описании переменных *x* и *y* как целых:
    - **var** x; y: integer;

Общие требования
к выполнению лабораторных работ

Общими положениями при проведении лабораторных работ являются следующие:

1. К лабораторным работам на ЭВМ допускаются только подготовленные студенты, прошедшие инструктаж по технике безопасности с обязательной отметкой в специальном журнале.
2. Подгруппы разбиваются на бригады численностью не более 2 человек.
3. Состав бригады и вариант, указанный преподавателем, остаются неизменными в течение цикла выполнения лабораторных работ.
4. До работы студенты должны ознакомиться с общими сведениями о компьютере.
5. В лабораторных работах может быть два варианта, отличающихся степенью сложности.
6. Вариант сложности определяется преподавателем. При ограниченном объеме часов некоторые работы могут быть опущены.
7. К очередной лабораторной работе, указанной преподавателем в конце занятий, студенты должны подготовиться.
8. Каждый студент должен отчитаться за выполненную работу (подготовить отчет).

Содержание отчета

1. Задание.
2. Схема алгоритма.
3. Листинг программы, содержащий фамилию, номер группы и вариант задания, выведенный печатающим устройством.
4. Распечатка исходных данных.
5. Распечатка результатов расчета.

Лабораторная работа № 1

Освоение среды программирования Turbo Pascal

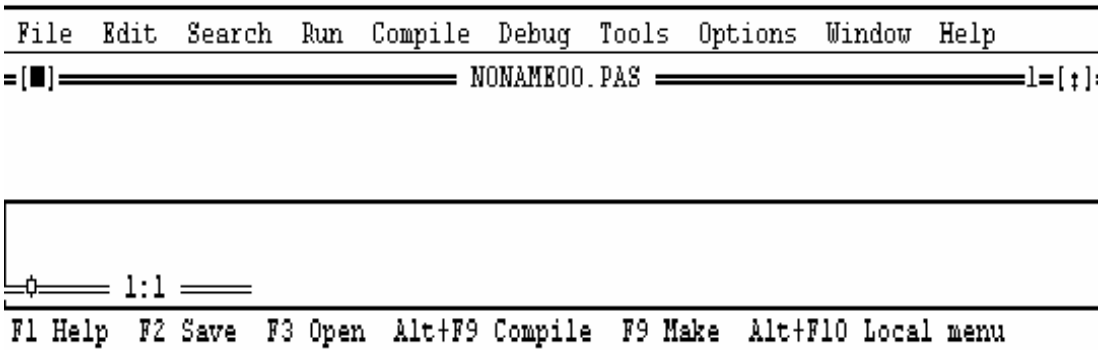
Цель работы: научиться использовать средства, имеющиеся в интегрированной среде (ИС) системы Turbo Pascal.

Теоретическая часть

Интегрированная среда Turbo Pascal

Экран и меню

После того, как загрузится Turbo Pascal, на экране появится главное меню и информация о данной версии. После нажатия любой клавиши эта информация исчезает.



Поле экрана разделено на части:

1. Главное меню с командами среды Turbo Pascal.
2. Окно редактора (с именем файла в заголовке) для ввода текста программы.
3. Указатель координат курсора (номер строки и позиция символа в строке).
4. Нижняя строка, показывающая, какие клавиши в данный момент активны, и их назначение.

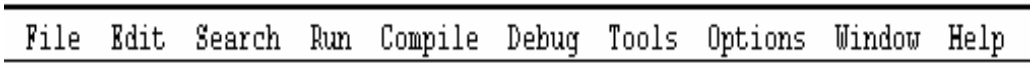
Для работы в интегрированной среде используют следующие средства:

1. Для выбора команд из меню пользуются мышью, выделенными заглавными буквами или клавишами со стрелками.
2. Для выхода из меню необходимо нажать <Esc>.

3. Для перемещений между меню и активным окном нажимают <F10>.

4. Если пользователь находится в главном или дополнительном меню, то для перемещения в окно, которое было активно ранее, нужно нажать <Esc>.

Команды главного меню



В текущем разделе будут рассмотрены основные команды главного меню и подменю, без которых трудно обойтись при выполнении даже достаточно простых программ.

File – команда взаимодействия с файлами.

New – открыть новое окно.

Open – загрузить существующий файл. Для определения списка файлов, один из которых необходимо загрузить, можно использовать собирательные символы * и ?, а также указывать диск, с которого должна производиться загрузка. При указании несуществующего диска или папки на экране появится сообщение об ошибке. Данная команда вызывается также клавишей <F3>.

Save – сохранить в текущей папке с прежним именем (если файл имеет имя NONAME01, то система предложит ввести уникальное имя).

Save as – сохранить файл под новым именем или изменить расположение. Файл, находящийся в редакторе, переписывается на диск под новым именем. Если файл с таким именем уже существует, то появится вопрос, надо ли его заменить.

Change dir – управление каталогами: просмотреть и (или) изменить.

Print – печать содержимого текущего окна.

DOS shell – временный выход в DOS. Для возвращения в Turbo Pascal необходимо набрать команду EXIT.

Exit – завершение работы Turbo Pascal.

Edit – позволяет создавать и редактировать исходные тексты программ с помощью встроенного текстового редактора, операции выполняются над выделенными фрагментами текста.

Пометить участок программы можно с помощью мыши или при одновременном использовании клавиши <Shift> и клавиш со стрелками. Выполнение этих действий вне текста программы снимает выделение.

Undo – редактор отменяет последнее действие.

Redo – редактор восстанавливает изменения, удаленные предыдущей командой.

Cut – удаляет выделенный блок текста в буфер.

Copy – копирует выделенный участок текста в промежуточный буфер.

Paste – помещает информацию из буфера в текущую позицию.

Clear – удаляет выделенный участок программы.

Show clipboard – открывает окно промежуточного буфера. Окно буфера Clipboard является разновидностью окна редактирования. Данные в это окно переносятся командами Cut и Copy. Текст в буфере можно редактировать, переносить и копировать, как в любом окне.

Search – меню поиска информации. Команда предназначена для поиска фрагментов текста и места обнаружения ошибок.

Find – найти фрагмент. Открывается окно диалога, где задается цепочка символов для поиска, указываются условия поиска: различать или нет прописные и строчные буквы, искать как слово или часть слова, по всему тексту или в выделенном фрагменте, вперед или назад от курсора и т. д.

Replace – заменить фрагмент. Отличается от предыдущей команды тем, что одновременно с поиском выполняется замена.

Search again – повторное выполнение команд Find или Replace при заданных условиях поиска.

Go to line number – осуществляет переход к строке с заданным номером.

Show last compile error – показывает последнюю ошибку, обнаруженную при компиляции, выдает сообщение о виде ошибки.

Run – меню позволяет запускать программу, а также выполнять отдельные части программы во время ее отладки.

Run – запускает программу, если ее исходный текст не был изменен. Если программа была изменена, то при очередном запуске программы происходит ее перекомпиляция (см. меню компиляции). Затем начинается выполнение программы. Для просмотра результатов программы используйте команду User Screen из меню Debug либо сочетание клавиш <Alt>+<F5>.

Для того чтобы прервать выполнение программы, например, в случае зависания, следует нажать комбинацию клавиш<Ctrl>+<Break>. Программа запускается либо до конца, либо до ближайшей из точек останова, если они были заданы.

Step over – трассировка без захода в подпрограммы. Выполняет очередные операторы программы, соответствующие одной строке текста, причем трассировка подпрограмм (выполнение по отдельным операторам) не производится – они рассматриваются как единые операторы.

Trace into – трассировка с заходом в подпрограммы.

Go to cursor – выполнить до строки, помеченной курсором. Выполняет программу до строки, на которой находится курсор. Если курсор находится на строке, не являющейся выполняемым оператором, будет выдано предупреждение об ошибке.

Program reset – прекращает сеанс отладки программы, закрывает все открытые в процессе работы программы файлы, программа становится готовой для повторного запуска с начала.

Compile – позволяет компилировать исходный текст, находить синтаксические ошибки, возникающие при вводе, и получать системную информацию.

Compile – компилирует файл, загружавшийся в редактор последним. Если ошибки не обнаружены, выдается сообщение об успешной компиляции. Во

время компиляции раскрывается окно, содержащее информацию о процессе: имя главного файла, компилируемый файл, компилируемая строка, имеющаяся память и успешно ли окончен процесс компиляции. При успешном окончании нажимают любую клавишу, чтобы убрать это окно.

В случае обнаружения синтаксической ошибки выдается сообщение об этой ошибке, а курсор помещается в место ее нахождения.

Destination – размещение файла. Определяет, где следует разместить исполняемый файл программы – в оперативной памяти (Memory) или на диске (Disk).

Information – выводит информацию о текущем файле и оперативной памяти.

Debug – меню отладки, позволяет задавать и просматривать параметры, необходимые при отладке программы.

Watch – активизирует окно наблюдаемых параметров. В окне наблюдения содержится список тех выражений из программы, за текущим значением которых необходимо наблюдать в режиме отладки. Наблюдаемые выражения вычисляются каждый раз заново, когда запускают программу на обычное или пошаговое выполнение.

Текущее выражение в окне наблюдения отмечается выделенной полосой, если это окно активно; если же окно неактивно, текущее выражение помечается точкой.

Output – активизирует окно выходных результатов.

User screen – открывает и активизирует окно пользователя. Появляется экран с изображением результатов программы. Тот же экран используется при выполнении и отладке программ, а также при временном выходе в DOS.

Add watch – добавляет в окно наблюдаемых параметров новый параметр (переменную или выражение).

Options – меню параметров среды.

Environment – условия работы. Подменю Colors/Syntax – цвета. Команда позволяет выбрать цвет символов (Foreground) и фона (Background) всех элементов интегрированной среды Turbo Pascal. Выделяются другим цветом:

Reserved words – ключевые слова.

Identifiers – идентификаторы.

Comments – комментарии.

Symbols – символы.

Strings – строковые постоянные.

Numbers – числа.

Window – меню окон, позволяет открывать, закрывать, активизировать окна, перемещать их в поле экрана.

Tile – размещение окон без перекрытия. Окна располагаются в поле экрана встык друг к другу.

Cascade – каскадное размещение окон. Окна перекрывают друг друга.

Close all – удалить все окна. Очищает поле экрана, закрывая все окна, и очищает все списки предыстории.

Refresh display – обновить экран. Обновляет экран среды, если программа пользователя изменила его содержимое.

Size/Move – позволяет изменить размеры окна, переместить его по полю экрана. Для изменения размеров экрана следует с клавишей <Shift> одновременно нажать одну из клавиш «стрелка вверх», «стрелка вниз», «стрелка влево» или «стрелка вправо». Для перемещения окна по полю экрана следует воспользоваться клавишами «стрелка вверх», «стрелка вниз», «стрелка влево» или «стрелка вправо». Завершить работу следует нажатием клавиши <Enter>. Эти же операции можно выполнить с помощью «мыши».

Zoom – позволяет увеличить размеры окна до всего поля экрана. Если окно уже раскрыто, оно уменьшается до первоначальных размеров.

Next – следующее окно.

Previous – предыдущее окно.

Close – закрывается активное окно.

List – открывает окно диалога со списком открытых окон, которые можно активизировать либо закрыть.

Help – меню информационной помощи. Меню позволяет получить имеющуюся в системе справочную информацию.

Contents – сведения о выводимой на экран информации: активном окне, выбранной команде меню, обнаруженной ошибке и т. д.

Index (ключевые слова) – выводится в алфавитном порядке список всех имеющихся в системе информационной помощи ключевых слов, по которым имеется справка. Для поиска нужного слова можно либо воспользоваться клавишами перемещения курсора, либо набрать на клавиатуре интересующее слово или его начало (достаточное, чтобы выделить его среди других слов). Если искомого слова нет, выбирается слово, у которого совпадает с требуемым максимальное число начальных символов.

Topic search (предметный поиск) – выводится информация о слове, на котором находится курсор. Если информации о таком слове нет, выводится список ключевых слов, содержащий выделенное слово, у которого совпадает с требуемым максимальное число начальных символов.

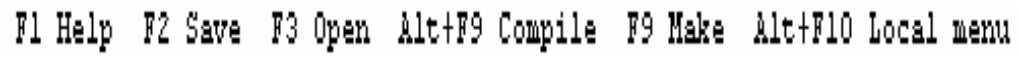
Previous topic (предыдущая тема) – выводятся сведения по теме, соответствующей предыдущему запросу. Система сохраняет до 20 предыдущих запросов.

Нижняя полоса на экране ИС

При открытии любого пункта меню полоса, расположенная внизу, позволит получить справочную информацию о функциональных клавишах.

Содержание нижней полосы меняется в зависимости от того, в каком из подменю находится пользователь.

Для главного меню или окна редактора содержимое нижней полосы следующее:



<FI> – Help. Открывает окно, содержащее информацию о командах редактора Turbo Pascal.

<F2> – Save. Сохраняет текущий файл.

<F3> – Open. Загружает ранее созданный файл.

<Alt>+<F9> – Compile. Компилирует исходный текст.

<F9> – Make. Вызывает встроенную в Turbo Pascal программу Make. В результате этого начинается компиляция последнего из загруженных в редактор файла. Turbo Pascal проверяет все файлы, от которых зависит компилируемый.

<Alt>+<F10> – Local menu. Позволяет активизировать локальное меню, которое существует помимо основного меню в версии 7.0.

Локальное меню можно вызвать щелчком правой кнопки мыши.

Local menu в основном повторяет наиболее часто используемые команды основного меню.

В режиме редактирования в меню входят следующие команды:

Cut – удаляет выделенный блок текста и помещает его в промежуточный буфер (Clipboard).

Copy – копирует выделенный блок текста в промежуточный буфер (Clipboard) без удаления его из текущего файла.

Paste – помещает информацию из промежуточного буфера (Clipboard) в текущий файл, начиная с позиции курсора.

Clear – удаляет выделенный блок текста.

Open file at cursor – открывает в новом окне диалога файл, на имя которого в данный момент указывает курсор.

Topic search (предметный поиск) – выводится информация о слове, на котором находится курсор. Если информации о таком слове нет, выводится список ключевых слов, содержащий выделенное слово, у которого совпадает с требуемым максимальное число начальных символов.

Toggle breakpoint – задает точку останова в строке, на которой находится курсор. Если на этой строке уже задана точка останова, то при использовании этой команды она будет удалена.

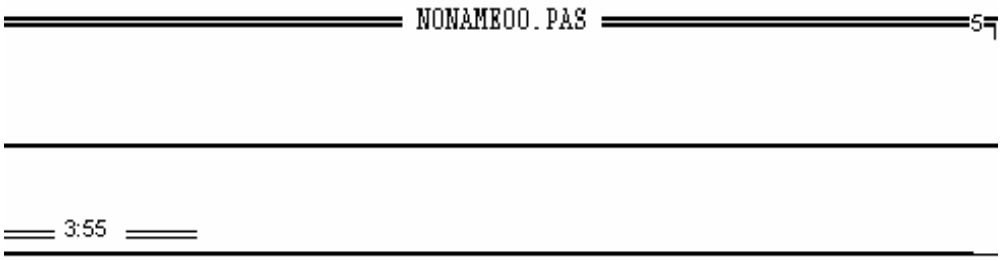
Go to cursor – выполняет программу до строки, на которой находится курсор. Если курсор находится на строке, не являющейся выполняемым оператором, будет выдано предупреждение об ошибке.

Add watch – добавляет в окно наблюдаемых параметров новый параметр (переменную или выражение).

Содержимое локальных меню для других режимов работы в среде отличается от указанного выше, но, как правило, в них размещаются команды, повторяющие команды основного меню.

Окно редактора

Экран редактора Turbo Pascal содержит, помимо основной рамки редактора, линию текущего состояния редактора. В этой линии указана информация о местоположении курсора (3:55, где 3 – номер строки, 55 – номер позиции). В верхнем правом углу указан номер текущего из открытых окон (5).



Если курсор на экране представлен как «_» (символ подчеркивания), это означает, что включен режим вставки текста; если курсор изображается как вертикально вытянутый прямоугольник – включен режим замены. Переключение осуществляется клавишей <Insert>.

При вводе текста программы для перехода на следующую строку надо нажимать <Enter>, максимальное число символов в строке – 248, при переполнении строки выдается звуковой сигнал. В окне редактора помещается 77 столбцов, при переходе за 77-й столбец текст сдвигается влево.

При вводе текста программы не существует жестких правил написания. Однако следует придерживаться общих рекомендаций, которые позволят легко ориентироваться в программах не только разработчику, но и другим пользователям.

Рекомендации позволяют быстрее понять суть работы программы, легко находить и исправлять ошибки.

1. При вводе текста программы целесообразно использовать систему отступов, когда операторы, вложенные в другие операторы или операторные скобки, пишутся на строке с отступом вправо по отношению к другим операторам (на несколько пробелов или позиций табуляции). Такое расположение операторов позволяет проще разобраться со структурой программы, быстрее найти некоторые ошибки, например несоответствие операторных скобок Begin и End.

2. Не следует в одной строке объединять несколько сложных или больших операторов, так как это осложняет поиск ошибок.

3. Рекомендуется последовательность операторов, выполняющих какое-то законченное действие, отделять от предыдущих и последующих операторов пустыми строчками.

4. Идентификаторам в программе следует давать имена, отражающие их суть, например Familia, Summa, Stroka и т. д. Можно использовать и длинные имена, так как редактор позволяет копировать их многократно.

5. Рекомендуется использовать комментарии для пояснения отдельных разделов программы или операторов. Комментарий открывается парой символов (*, а закрывается – *) или же его можно записать между фигурными скобками. Например:

(*Вывод результатов на экран*) или

{Группа СМ-16032, студент Иванов Б.Е., вариант 12}

6. Следует помнить, что все лишние с точки зрения языка Паскаль пробелы, пустые строчки, знаки табуляции, комментарии компилятором игнорируются. Они добавляются пользователем для удобочитаемости программы.

Задание 1

1. Войти в режим редактирования и набрать текст программы

```
Program Example;
{Лабораторная работа №1, студент, группа}
Var
    A, B, C, Y: Real;
Begin {Исходные данные}
    WriteLn ('Введите исходные данные A, B, C: ');
    ReadLn (A, B, C);
    WriteLn ('A=', A:4:1, 'B=', B:4:1, 'C=', C:4:1);
    Y:=A*SQR(B)+C*B;
    WriteLn ('Результат расчета Y= ',Y:5:2);
    ReadLn;
End.
```

2. Запустить программу на компиляцию командой Compile / Compile. При получении сообщений об ошибках отредактировать текст программы. Повторить компиляцию: <Ctrl>+<F9>.

3. Запустить программу на выполнение командой Run / Run. При получении новых сообщений об ошибках вновь внести изменение в текст программы.

4. В процессе выполнения программа требует ввода трех величин (*A*, *B*, *C*) и вычисляет одну величину *Y*.

5. Выполнить программу по шагам Run / Step over (затем <F8>).

6. Записать программу с именем START командой File / Save (при активном окне редактора).

7. Изменить цвет для любых элементов окна: Options / Environment / Colors.

8. Организовать поиск переменной A командами Search / Find с фиксацией Whole words only (искать как слово). Командами Search / Find / Search again продолжить поиск. Найти фрагмент текста A командами Search / Find, очистив поле Whole words only.

9. Завершить работу с Turbo Pascal (File / Exit). Загрузить Turbo Pascal, вызвать программу (File / Open / start.PAS).

Задание 2

Изучить команду главного меню **Run**. Пошагово выполнить данную программу. Посмотреть в **Watch**-окне значения исходных данных и результат расчета.

Задание 3

Изучить команду главного меню **Edit**. Открыть новое окно и переписать часть набранной программы с 1-й строки до зарезервированного слова **Begin** в новое окно.

Контрольные вопросы

1. Как загрузить Turbo Pascal?
2. Как завершить работу с Turbo Pascal?
3. Как выделить фрагмент программы?
4. Какие действия редактора предусмотрены для выделенного фрагмента?
5. Как сохранить текст программы на диске?
6. Как сохранить тот же текст программы с новым именем?
7. Как сохранить тот же текст программы на дискете?
8. Какие варианты просмотра результатов на экране предусмотрены в языке Turbo Pascal?
9. В чем назначение комментария?
10. Как запустить программу на решение?
11. Какие средства Turbo Pascal предназначены для отладки программ?
12. Объяснить назначение клавиш <FI>, <F2>, <F3> при работе в среде Turbo Pascal.
13. Перечислить этапы решения задачи с ввода программы до просмотра результатов.

Лабораторная работа №2

Программирование линейных алгоритмов

Цель работы: получение практических навыков при программировании линейных вычислительных процессов.

Теоретическая часть

Линейным называется такой вычислительный процесс, этапы которого выполняются однократно и последовательно один за другим. С помощью линейного вычислительного процесса осуществляется, например, вычисление значения функции по формуле.

Этапы линейного вычислительного процесса:

1. Ввод исходных данных, вычисление значений искомых переменных.
2. Вывод на экран результатов вычислений.

Этапы выполняются однократно и последовательно друг за другом (см. рис. 2.1).

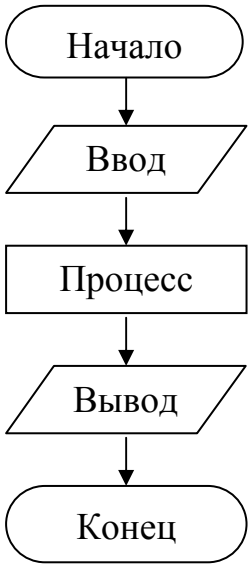


Рис. 2.1. Блок-схема линейного процесса

Для реализации линейного вычислительного процесса, как минимум, необходимы операторы присваивания, ввода и вывода.

В результате выполнения оператора присваивания переменная принимает значение некоторого выражения. Примеры операторов:

x1:= (–b+sqrt (sqr (b)–4*a*c)) / (2*a);

m:= 0.231;

Во всех случаях вначале вычисляется значение выражения, расположенного справа от комбинации символов :=, а затем вычисленное значение присваивается переменной, расположенной слева. Для того чтобы оператор присваивания мог быть выполнен, необходимо, чтобы все переменные, которые входят в выражение, имели на момент вычисления выражения некоторые значения.

Для ввода данных и вывода результатов используются **операторы ввода и вывода**.

Ввод может выглядеть, например, так:

ReadLn (a);(**readln**(a);)

Read (x1, x2, y);(**read**(a);)

Оператор ввода состоит из идентификатора read (readln) и следующего за ним в круглых скобках списка переменных. Число переменных в списке может быть любым; если переменных больше одной, то они разделяются запятыми.

При выполнении оператора ввода переменным присваиваются значения исходных данных. Числа, которые являются исходными данными, надо своевременно набрать на клавиатуре, разделяя их пробелом (пробелами или нажатием Enter). Пусть на клавиатуре набрано число 3.6, тогда в результате выполнения оператора readln (a) переменная a получит значение 3.6, после чего начнет выполняться следующий оператор программы.

При чтении числовых данных в использовании операторов read и readln есть небольшие различия. Оператор readln не только читает данные, но и делает перевод строки, что позволяет последующим операторам ввода осуществлять ввод с новой строки.

Оператор вывода состоит из идентификатора Write (write) и следующего за ним в круглых скобках списка переменных, выражений и констант:

write (x, 2*x–a, sqrt(x), ‘Результат=', y);

В Паскале имеется оператор вывода WriteLn (writeln), который выполняется так же, как write, с той разницей, что после его выполнения последующий вывод данных будет начинаться с новой строки экрана. Переход на новую строку без вывода значений, в соответствии со сказанным, может быть выполнен с помощью оператора writeln без списка данных.

Например, последовательность операторов вывода writeln: writeln (‘x1=’, (–b+d)/a); writeln (‘x2=’, (–b–d)/a); задает вывод результатов в следующем виде:

x1=…

x2=...

Пример 2.1. Программа korni вычисляет корни квадратного уравнения $ax^2+bx+c=0$, заданного коэффициентами *a*, *b*, *c* (предполагается, что *a* не равно 0 и что дискриминант уравнения неотрицателен):

Программа 2.1

```
Program korni;
    var a, b, c: real;
    Begin
        WriteLn (‘Введите значения a, b, c’);
        ReadLn (a, b, c);
        Write((–b+sqrt(sqr(b)–4*a*c))/(2*a):9:2,’   ‘,
            (–b–sqrt(sqr(b)–4*a*c))/(2*a):9:2);
        ReadLn;
    End.
```

При выполнении этой программы придется дважды вычислить значение sqrt(sqr(b)–4*a*c) и дважды – значение 2*a. Поэтому более разумным вариантом будет программа:

Программа 2.2

```
program korni;
    var a, b, c, d, e: real;
    begin
        writeln (‘Введите значения a, b, c’);
        read (a, b, c);
        d:=sqrt(sqr(b) –4*a*c));
        e:=2*a;
        writeln (‘корень 1 = ’, (–b+d)/e,’      корень 2 = ‘,(–b–d)/e);
        readln;
    end.
```

При выполнении оператора writeln пользователь увидит на экране результат в экспоненциальном виде, где под вещественное число отведено 17 позиций, например:

корень 1 = –7.2436795801E+01

Изменить стандартную форму вывода можно, используя систему форматов языка Паскаль.

Ниже рассмотрена **система форматов для вывода** информации.

В списке оператора write (writeln) можно указывать размер поля для каждой из выводимых величин, другими словами, количество позиций для данных.

Так, для вывода числа 999 будут требоваться три позиции. Для вывода числа –23.045 требуется семь позиций (с учетом знака и десятичной точки), из которых три позиции будет занимать дробная часть.

Если пользователь укажет больше позиций, чем необходимо, то левые от значения числа позиции заполнятся пробелами.

Если указанный размер поля меньше требуемого, то значение печатается без пробелов и учета указанного пользователем поля.

Для вывода вещественных данных указывается общая длина поля и количество позиций под дробную часть (в том числе).

Если требуется, то дробная часть числа округляется до указанного количества позиций.

Так, если значения переменных *d* и *s* равны «1234» и «–123.451» соответственно, то при выполнении оператора вывода с форматами результаты представлены в таблице 2.1.

Таблица 2.1

Форматы	**Представление результата**	**Примечание**
d:4	1234	
d:7	_ _ _1234	
d:12	_ _ _ _ _ _ _ _1234	
d:2	1234	вывод без учета указанного поля
s:9:3	_–123.451	
s:8:1	_ _–123.5	
s:12:2	_ _ _ _ _–123.45	
s:4:3	–123.451	вывод без учета указанного поля

При использовании формата оператор программы 2.2 writeln может быть, например, следующим:

writeln (‘корень 1= ’,(–b+d)/e:12:4,

‘корень 2 = ’,(–b–d)/e:12:4);.

Задание

Написать простейшую программу для расчета по формулам, приведенным в таблице 2.2. Программу составить с использованием комментариев. Получить листинг. Выполнить программу, задавшись значениями переменных из диапазона [0,25–6,50].

Таблица 2.2

№ п/п	Арифметические выражения	
1	$\frac{\sqrt[3]{\lvert x+\sin x^2\rvert}-2ab}{\log_2\lvert x\rvert}$	$\frac{\pi\cdot\lg\lvert x\rvert-a}{\cos^4 x}$
2	$\frac{\sqrt{c^2+a^2}+2c\cdot d\cdot tgx}{\ln\lvert x\rvert+e^{\sqrt{\lvert q-1\rvert}}}$	$\frac{1.8\cdot\sqrt[3]{\lvert\sin x\rvert}+e^{\sqrt{\lvert x-1\rvert}}}{\sqrt{\lvert c\rvert}+\sqrt[3]{\lvert a\rvert}}$
3	$\frac{a^2-4a\cdot b+b^2}{3x\cdot\lg b}$	$\frac{(a+b)\cdot e^x}{\lvert x+a\rvert}+\frac{x^2+w}{w\cdot\lvert\cos x\rvert}$
4	$\frac{\sqrt[3]{\lvert a\cdot x+r\rvert}}{\sqrt{\lvert tgx\rvert}}-e^a$	$\frac{\sin x^2+tga}{a+\lvert\sin a\rvert}$
5	$\frac{2\pi\cdot j+tgx^2}{2j\cdot\lvert\sin x\rvert}$	$\frac{a^6+\cos^4 x}{\sqrt[3]{\lvert j\rvert}}$
6	$\frac{(a\cdot x+2b\cdot x+x^2)\cdot\lvert\cos x^2\rvert}{\pi\cdot e^x}$	$\frac{e^{\sqrt{\lvert w+1\rvert}}\cdot tgx\cdot\sin w}{1.85w+\sqrt[3]{y^2}}$
7	$\frac{\sqrt{\lvert t\rvert}+a^3}{t\cdot\lvert\sin\ a\ /\ 2\rvert}$	$\frac{(\sin a+tg^2x)\cdot e^{\lvert a\rvert}}{\log_2 a}$
8	$\frac{\sqrt{\lvert(a+b)\rvert}/(c+d)}{3\pi\sqrt[3]{\lvert a-b\rvert}}$	$\frac{(\lvert\ln x^2\rvert+a\cdot c)\cdot e^{\lvert a\cdot b\rvert}}{b^2+c^2}$
9	$\frac{\sqrt[3]{\lvert x+\sin x^2\rvert}-2a\cdot b}{\lvert\log_2 x\rvert}$	$\frac{\pi\cdot\lvert\lg x\rvert-a}{\cos^4 x}$
10	$\frac{\sqrt{c^2+a^2}+2c\cdot d\cdot tgx}{\lvert\ln x+e^c\rvert}$	$\frac{1.8\cdot\sqrt[3]{\lvert\sin x\rvert}+e^{\sqrt{\lvert x-1\rvert}}}{\sqrt{\lvert c\rvert}+\sqrt{\lvert a\rvert}}$

Методические указания

Приступая к заданию, прежде всего, необходимо уяснить сущность решаемой задачи, определить для нее исходные данные, которые должны вводиться в оперативную память с клавиатуры ПК и понять, что же должно быть результатом решения. При этом следует уточнить постановку задачи, используя необходимые для ее решения формулы.

Затем определяют, какие операции, над какими данными, в какой последовательности необходимо выполнять для решения задачи, т. е. разрабатывают программу. Алгоритм, разработанный в соответствии с заданием, должен иметь линейную структуру.

При программировании желательно использовать следующие конструкции:

1. Заголовок программы с именем программы (PROGRAM ...).
2. Комментарий с указанием фамилии и имени студента, номер работы, номер варианта ({……}).
3. Описание используемых в программе переменных (раздел VAR).
4. Начало исполняемой части программы (BEGIN).
5. Операторы вывода подсказок для пользователя – какие исходные данные он должен ввести в ПК с клавиатуры (WRITE).
6. Операторы ввода исходных данных (READ или READLN).
7. Операторы присваивания, предусматривающие необходимые вычисления.
8. Оператор вывода результатов (с использованием формата) на экран монитора (WRITE или WRITELN).
9. Завершающее текст программы слово END с точкой.

Составив программу, ввести ее в ПК, подготовить к выполнению (преобразовать исходный модуль в загрузочный), отладить и трехкратно (для разных вариантов исходных данных) выполнить программу. Необходимо при этом использовать соответствующие средства системы Турбо Паскаль.

Контрольные вопросы и упражнения

1. В чем заключается подготовка задачи к решению на ПК?

2. Дать определение понятий «алгоритм», «программа», «язык программирования».

3. В чем особенности линейных алгоритмов и программ? Какие основные процедуры реализуются по ним на ПК?

4. Сформулировать правила записи программ на языке Turbo Pascal.

5. Дать определение понятий «константа», «переменная», «выражение», «указатель функции», «оператор».

6. Как представляются данные в Паскаль-программах?

7. Что понимают под вводом и выводом данных в ПК? Как программируются эти процедуры на языке Паскаль в простейших случаях?

8. Что значит вывод с использованием формата?

9. Что собой представляет оператор присваивания? Сформулировать правила записи арифметических операторов присваивания (выражений) на языке Паскаль.

10. Прокомментировать смысл составленной вами программы.

11. Написать программу вычисления значения выражения z:

$$z = tg\sqrt{\left|y + 4x^2\right|} - \frac{a^2 + b^2}{\ln\left(a^2 + b^2\right)} \times e^{\sin^2 x}$$

При значениях $x = 10$, $y = 1.2$, $a = -5.8$, $b = -0.75$ должно получиться значение $z = -10.588$.

Лабораторная работа №3

Программирование разветвляющихся алгоритмов

Цель работы: получение практических навыков алгоритмизации и программирования разветвляющихся вычислительных процессов, записи логических выражений.

Теоретическая часть

В решении многих задач при выполнении определенных условий вычисления должны производиться по одним алгоритмам (или формулам), а при невыполнении их – по другим. Такие вычислительные процессы и соответствующие программы называют *разветвляющимися*. Каждое из направлений называется *ветвью вычислений*. Выбор той или иной ветви вычислений осуществляется проверкой выполнения логического условия, после чего вычислительный процесс реализуется только по одной ветви, а остальные не используются.

Для описания таких процессов служит **условный логический оператор**.

Например, при необходимости присвоить переменной MAX наибольшего из значений переменных $x1$ и $x2$, следует сравнить значения $x1$ и $x2$ и в зависимости от результата сравнения выполнить либо оператор MAX:= $x1$ либо MAX:= $x2$. Действия такого рода задаются условным логическим оператором:

IF B **THEN** P1 **ELSE** P2;,

где B – логическое выражение (условие), P1 и P2 – операторы. Если значение выражения B истинно(true), то выполняется P1, иначе выполняется P2 (см. рис. 3.1).

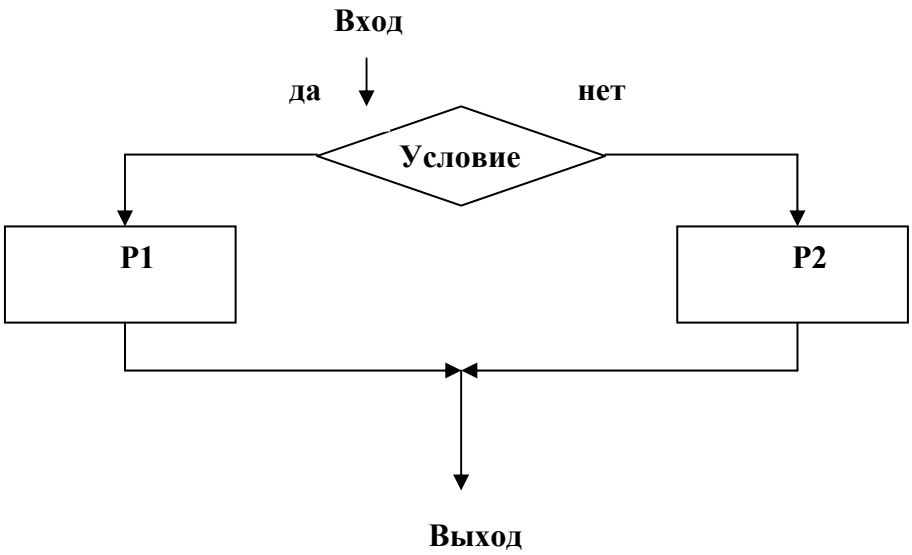


Рис. 3.1 Блок-схема выполнения условного логического оператора (полная форма)

В качестве логического выражения, в частности, могут использоваться отношения.

На клавиатурах современных компьютеров нет некоторых привычных со школьных лет знаков (например, не равно, больше, больше или равно, меньше, меньше или равно), и вместо этих знаков используются соответственно комбинации < >, >, >=, <, <=.

Для решения задачи о присваивании переменной MAX наибольшего из значений $x1$ и $x2$ достаточно выполнить условный логический оператор

if ($x1>x2$) then MAX:=$x1$ else MAX:=$x2$;

Допускается сокращенная форма условного логического оператора:

if B then P;

где B – логическое выражение, а P – оператор. В случае если B – истина (true),то выполняется оператор P, если же B – ложь (false), то P не выполняется (см. рис. 3.2).

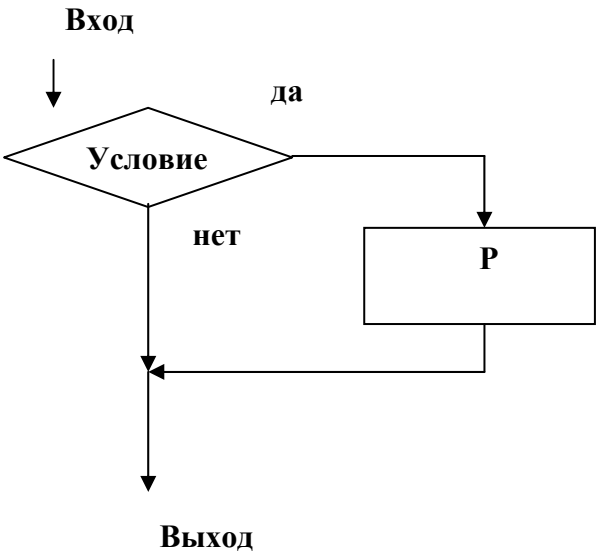


Рис. 3.2 Блок-схема выполнения условного логического оператора (сокращенная форма)

При записи операторов и блоков IF следует помнить, что упомянутые логические выражения часто представляются в виде отношения или комбинации несколько отношений, соединенных знаками логических операций: **AND** (логическое И, логическое умножение), **OR** (логическое ИЛИ, логическое сложение), **NOT** (логическое НЕ, логическое отрицание), **XOR** (исключающее ИЛИ).

Например, если структура оператора определяет, что для присваивания переменной *P* значения 1 необходимо, чтобы значение переменной *A* было положительно, а одно из значений – либо переменной *B*, либо переменной *C* – ненулевым, то условный оператор запишется в виде:

IF ((A > 0) AND ((B <> 0) OR (C <> 0))) THEN P = 1;

В условном логическом операторе после then и else можно помещать только по одному оператору. Однако часто необходимо в зависимости от результата проверки некоторого условия выполнить ту или иную группу операторов. В таком случае Паскаль предоставляет возможность сделать из группы операторов один **составной оператор**.

Структура составного оператора:

begin P1; P2;...; Pk **end;**

где P1; P2;...; Pk – любые операторы.

Как частный случай составного оператора возможно: begin *P* end, где *P* – любой оператор. Существенно, что оператор begin *P* end при любом операторе *P* не является условным логическим и может быть размещен после then (см. рис. 3.3 и 3.4).

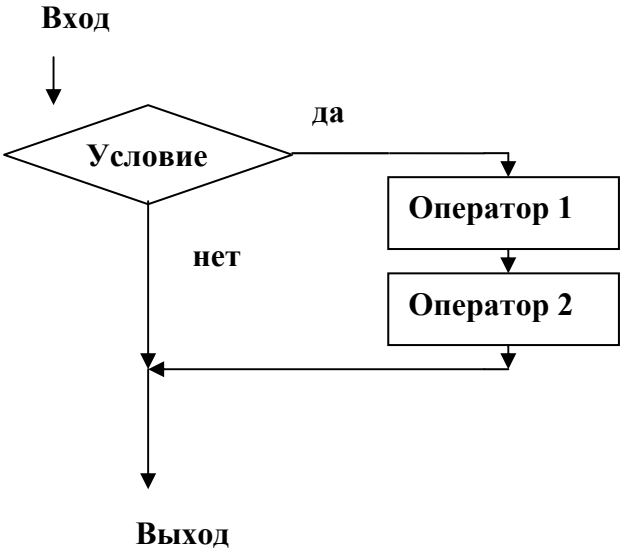


Рис. 3.3 Блок-схема сокращенного условного логического оператора (с составным оператором)

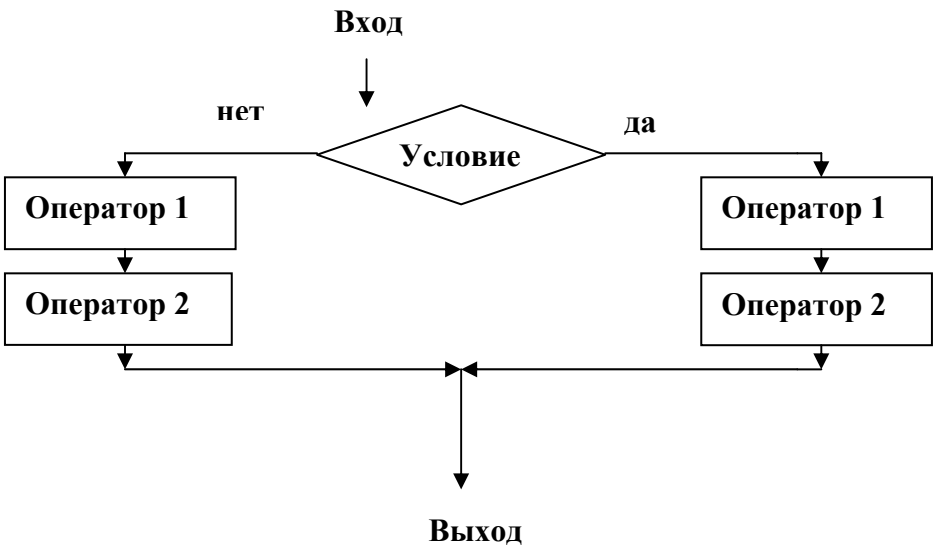


Рис. 3.4 Блок-схема условного логического оператора с составным оператором (полная форма)

Примером разветвляющегося вычислительного процесса могут служить вычисление значения некоторой величины по одной из заданных формул, когда выбор той или иной формулы для расчета определяется конкретным значением логического выражения в записи условного логического оператора.

Пример 3.1. Пусть квадратное уравнение $ax^2+bx+c = 0$ задано коэффициентами *a*, *b* и *c* (*a* не равно 0).

Ниже приведена программа, при выполнении которой исследуется дискриминант уравнения, выводится текст «решения не существует», если дискриминант отрицателен, и выводится пара корней, если дискриминант неотрицателен.

```
program korni;
var a, b, c, d, e, k1, k2: real;
begin
    writeln (‘Введите значения a, b, c’);
    readln (a, b, c);
    d:= sqr(b)-4*a*c;
    if d < 0      then writeln (‘решения не существует’)
    else   begin
                d:=sqrt(d);
                e:=2*a;
                k1:= (–b+d)/e;
                k2:=( –b–d)/e);
                writeln (‘корень 1 = ’, k1:10:2,
                ‘корень 2 = ’, k2:10:2);
           end;
    readln;
end.
```

Когда оператор IF появляется внутри другого оператора IF, они считаются вложенными. Такие вложенные друг в друга структуры приведены на рис. 3.5.

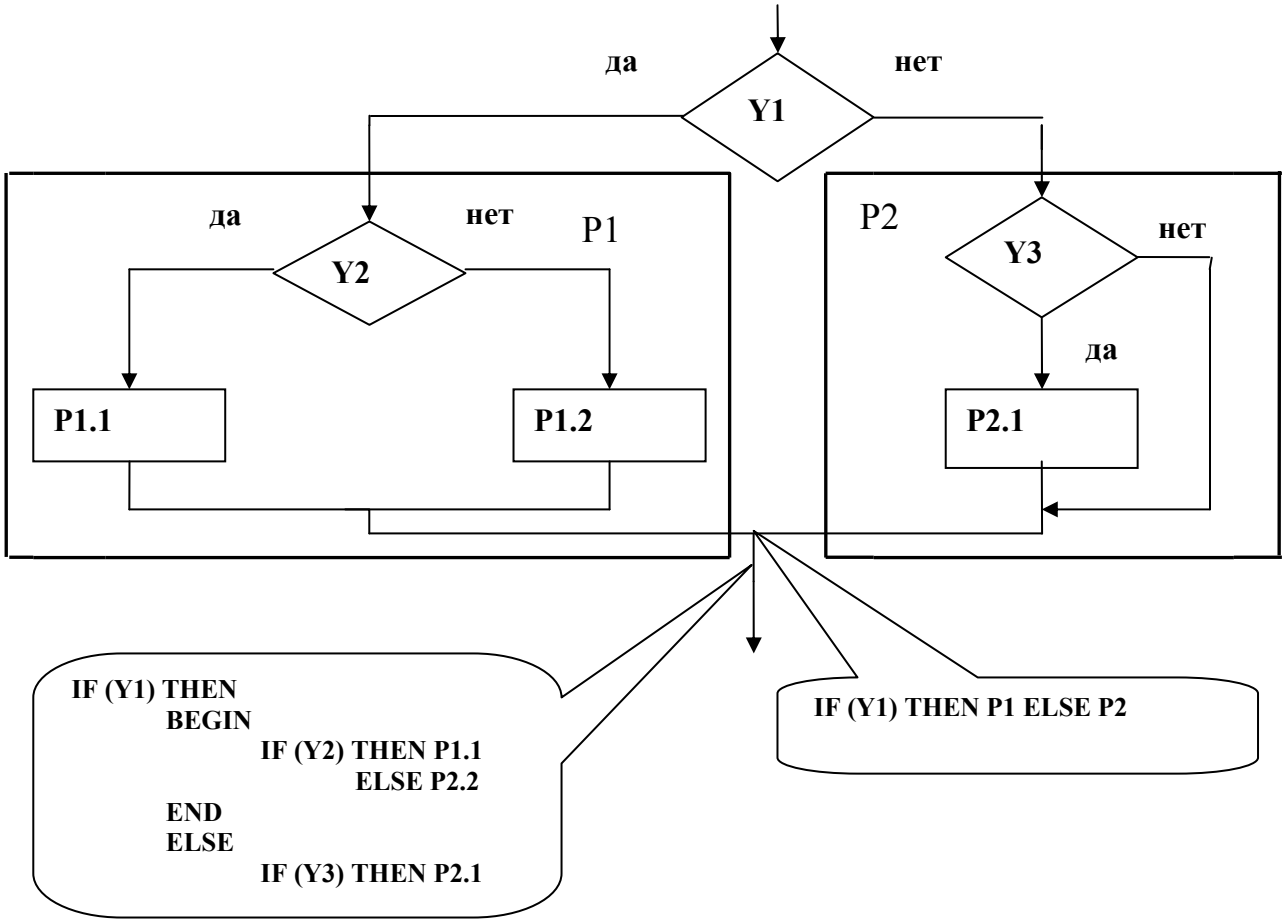


Рис. 3.5 Вложенные структуры с условными логическими операторами

Задание 1

Составить блок-схему алгоритма и программу расчета, применяя условный логический оператор. Вывести программу, исходные данные и результаты расчета на экран видеотерминала, а также на принтер. Функции для вычисления приведены в таблице 3.1.

Таблица 3.1

№ п/п	Функция
1	$Y=\begin{cases}\sqrt{\lvert a-x\rvert}\cdot\sin^2 x, & \text{если } x<b;\\ \left(\dfrac{x}{\lvert a+x\rvert}\right)\cdot\sqrt[3]{\lvert\sin x\rvert}, & \text{если } b\le x\le c;\\ e^{\sqrt{\lvert x\rvert}}, & \text{если } x>c;\end{cases}$
2	$Y=\begin{cases}ax+0.23x^2\log_2 a, & \text{если } x<b;\\ \left(\dfrac{xe^a}{\lvert a+x\rvert}\right)\cdot\sqrt{\lvert\cos x\rvert}, & \text{если } b\le x\le c;\\ x\cdot tga, & \text{если } x>c;\end{cases}$
3	$Y=\begin{cases}\left(a^2+x^2\right)\cdot e^x, & \text{если } x<b;\\ \sqrt{\lvert a\rvert}\cdot\sin^4 x, & \text{если } b\le x\le c;\\ e^{\frac{\lvert a-x\rvert\cdot\cos x4}{a}}, & \text{если } x>c;\end{cases}$
4	$Y=\begin{cases}\ln\lvert a+x\rvert\cdot\cos\lvert x^3\rvert, & \text{если } x<b;\\ e^{1.2}-\sqrt{\lvert a+x\rvert}, & \text{если } b\le x\le c;\\ \dfrac{\sqrt[3]{\lvert a+x\rvert}}{(a-x)}, & \text{если } x>c;\end{cases}$
5	$Y=\begin{cases}a^2+\sqrt{\lvert a^2+x\cdot\sin x\rvert}, & \text{если } x<b;\\ 2x^2+a^3\cdot tgx, & \text{если } b\le x\le c;\\ \dfrac{x^2}{\sqrt{\lvert a\rvert}}, & \text{если } x>c;\end{cases}$
6	$Y=\begin{cases}e^{0.2}+\sqrt{a+x}, & \text{если } x<b;\\ (a+x)\cdot\sqrt[3]{\lvert\sin x\rvert}, & \text{если } b\le x\le c;\\ \sqrt{\lvert a+x\rvert}, & \text{если } x>c;\end{cases}$

7	$Y=\begin{cases} 2.75e^{\lvert x+a\rvert}+\cos^4 x, & \text{если } x<b; \\ \dfrac{(x+a)\cdot tgx}{\lg\lvert x\rvert}, & \text{если } b\le x\le c; \\ e^{ax}+\dfrac{a\cdot\sin^2 x}{\sqrt[3]{\cos^2 x}}, & \text{если } x>c; \end{cases}$
8	$Y=\begin{cases} e^{0.15}+\dfrac{(a+x)^2}{\ln x}\cdot e^x, & \text{если } x<b; \\ (x^2-a^2x)\cdot tgx, & \text{если } b\le x\le c; \\ (a+x)\cdot e^{\sqrt{\lvert a+x\rvert}}, & \text{если } x>c; \end{cases}$
9	$Y=\begin{cases} \lvert\sin^3 a\rvert+\lvert\cos^3 a\rvert, & \text{если } x<b; \\ tg(ax)+\ln\lvert b\rvert, & \text{если } b\le x\le c; \\ \ln\lvert a-b\rvert-\ln\lvert(a-b)^3\rvert, & \text{если } x>c; \end{cases}$
10	$Y=\begin{cases} a^x-e^x+\cos(3x-0.2), & \text{если } x<b; \\ b\cdot\cos\left(a^3-\lvert x\rvert^3\right)-e^i, & \text{если } b\le x\le c; \\ tg4.5x+\dfrac{x-5}{\sin(a+b)^2}, & \text{если } x>c; \end{cases}$

Задание 2

Написать программу для вычисления функции по формулам. Исходными данными задаться из диапазона +/– (3.75–12.35).

Таблица 3.2

№ п/п	Функции	Условие
1	$B=\begin{cases} j^3+\cos ax, \\ -3tg(a-2)+e^{-x}, \\ \cos^4 x, \\ x, \end{cases}$	*если* $j<a \wedge x>j$ *если* $j<a \vee x>a$ *если* $(x<a)\wedge(x>j)$ *в остальных случаях*
2	$W=\begin{cases} \sqrt[3]{\lvert x\rvert^2}+\sqrt[4]{\lvert a\rvert}, \\ ae^{i}\cdot \sin x^2, \\ \ln\lvert x-7\rvert+\dfrac{a}{i}, \\ a\cdot\cos x, \end{cases}$	*если* $(x+a)<x^2$ *если* $x<i \vee a>i$ *если* $x<a \wedge x>i$ *в остальных случаях*
3	$Y=\begin{cases} kx+ab, \\ (kx+ab)\cdot\cos^2 x, \\ (kx+ab)\cdot e^{i}, \\ \lg\lvert x\rvert+i, \end{cases}$	*если* $k<x \vee a<b$ *если* $k>x \wedge a>b$ *если* $i>x \wedge a<b$ *в остальных случ.*
4	$P=\begin{cases} \sqrt{\lvert tga+tgb\rvert}, \\ e^{i}\cdot\cos(a+b)^2, \\ \ln\lvert a\rvert\cdot\sqrt{\lvert a+b\rvert}, \\ \log_2\lvert i\rvert, \end{cases}$	*если* $a<b$ *если* $a>b \vee i<a$ *если* $a<b \wedge i<a$ *в остальных случ.*
5	$L=\begin{cases} 3\lvert x\rvert-\cdot e^{a}(a/\lvert x-b\rvert), \\ \sqrt{\lvert x\rvert}/a+arctgx, \\ \lg\lvert a\rvert+\sin x^2, \\ (a+b)^2/\cos^2 x, \end{cases}$	*если* $x<a \wedge \neg(x>b)$ *если* $x>a \vee x>b$ *если* $x\neq a \wedge i\neq b$ *в остальных случ.*
6	$Y=\begin{cases} 0.5\cos ax+e^{i}/i^2, \\ ((x^2+a^2)/\lvert x-d\rvert)+\ln\lvert x\rvert-b, \\ \sqrt{\lvert\sin^2 dx+\cos^4 b\rvert}+(a/b), \\ a^2+2ab+\log_2 x, \end{cases}$	*если* $x>a \vee \neg(x>b)$ *если* $x>a \wedge (x>b)$ *если* $x\neq a \wedge i\neq b$ *в остальных случ.*
7	$I=\begin{cases} 6.75x^2+e^{i}\lg\lvert 7a\rvert, \\ 2tg2+\lvert b^3\rvert+x, \\ \sqrt[3]{\lvert x-a+b\rvert}+(x/\lvert a-b\rvert), \\ \ln\lvert x\rvert+\sin^3 x, \end{cases}$	*если* $a<b \wedge x\neq a$ *если* $a>b \vee \neg(x<a)$ *если* $a\neq b \wedge a\neq x$ *в остальных случ.*

8	$C=\begin{cases}\sin(\ln\lvert ax\rvert),\\ (e^i/\lvert x-a\rvert)+arctgb,\\ tg\lvert\sqrt{\lvert x\rvert}\rvert,\\ \sqrt{\lvert a+b+x\rvert},\end{cases}$	*если* $a\neq 0\wedge x\neq 0$ *если* $a<b\vee\neg(a<x)$ *если* $x<b\wedge x>a$ *в остальных случ.*
9	$J=\begin{cases}\sqrt{\lvert x\rvert}+\sqrt[3]{\lvert x\rvert}+e^i\ln b,\\ \sin^2 a+\sin\lvert b\rvert+tg\lvert x\rvert,\\ (\lvert x+a\rvert/\lvert x-a\rvert)+\cos^4 x+e^x/a,\\ (\sin^2 x)/\lvert 1-x\rvert,\end{cases}$	*если* $a>b\wedge b\neq 0$ *если* $x=b\vee x=a$ *если* $a<x\wedge\neg b>a$ *в остальных случ.*
10	$Y=\begin{cases}(x+b^2+a^2)\cdot\ln\lvert x\rvert,\\ (x-b)\cdot(\lvert x-a\rvert)\cdot\cos^3 x,\\ (tgx)/\sqrt{\lvert x\rvert}+e^a,\\ \lvert x-b\rvert/(x^2+e^i),\end{cases}$	*если* $a\neq b\wedge x=0$ *если* $x>a\vee x>b$ *если* $x\neq 0\vee\neg(a>b)$ *в остальных случ.*

Методические указания

Задание предполагает осмысление математического описания, формулировку условий, проверка которых в процессе решения задачи позволяет определить допустимость исходных данных (если это необходимо), а также дополнительный результат (одно из двух или более сообщений, высвечиваемых на экране видеотерминала).

Решение должно предусматривать проверку условий, выполнение или невыполнение которых определяет те или иные действия, т. е. быть разветвляющимся.

Ветвление – это предписание вида:

«ЕСЛИ условие истинно (выполняется),

то выполнить оператор 1,

ИНАЧЕ выполнить оператор 2».

Так, проверка условий допустимости введенных в ПК исходных данных в случае их корректности должна приводить к определению искомых результатов задачи.

В случае недопустимости введенных данных – к выводу сообщения о некорректности этих данных.

Проверка других условий должна приводить к выводу одного из возможных и предусмотренных алгоритмом сообщений ПК в качестве дополнительно требуемого результата решения задачи.

При разработке разветвляющегося алгоритма (наглядно представленного в виде блок-схемы) следует стремиться к его логической ясности, простоте, краткости и структурности.

Для этого рекомендуется:

1. Разрабатывать структуры, имеющие один «вход» и один «выход» (рис. 3.1–3.4).

2. Использовать при необходимости вложенные друг в друга структуры (рис. 3.5).

При описании полученного алгоритма на языке Паскаль (если реализуются разветвляющиеся участки в программе) используют условные логические операторы и составные блоки IF. Важно при этом сохранение структурности алгоритма.

Не рекомендуется использовать более одного – двух уровней вложенности IF. За вторым уровнем вложения труднее проследить последовательность проверки условий для каждого IF.

В настоящем пособии часто для получения разветвляющейся программы нет необходимости полностью вводить ее текст в ПК. Можно с помощью редактора модифицировать хранящиеся на диске ранее созданные алгоритмы, дополнив их соответствующими новыми операторами. Конечно же, затем следует откомпилировать новый исходный модуль и получить модуль загрузочный.

Контрольные вопросы и упражнения

1. Что собой представляют разветвляющиеся алгоритмы (программы)?
2. В чем сущность разветвления?
3. Какие требования предъявляются к разветвляющимся алгоритмам (программам)?
4. Какие средства языка Turbo Pascal используются при составлении разветвляющихся программ?
5. Что такое составной оператор, в каких случаях он необходим?
6. Прокомментировать разработанный алгоритм и программу (раскрыть их смысл).
7. Написать программу деления целого числа *M* на целое число *P*. Если *M* нацело делится на *P*, то вывести результат, в противном случае вывести сообщение «*M* на *P* нацело не делится».
8. Написать программу определения наибольшего, наименьшего и среднего из трех целых чисел. Средним считать число, которое больше наименьшего из данных чисел, но меньше наибольшего. Для тестирования использовать следующие наборы чисел:

11, 25, 30

3, 60, 3

78, 9, 78

5, 5, 5.

Лабораторная работа №4
Циклические алгоритмы

Цель работы: получение практических навыков алгоритмизации и программирования простых циклических алгоритмов и организации ввода-вывода данных на ПК.

Теоретическая часть

При программировании многих задач отдельные участки вычислений многократно повторяются, при этом всякий раз используются новые значения исходных данных. Такие вычислительные процессы называют *циклическими*, а повторяемые участки вычислений – *циклами*.

Различают простые циклы, т. е. не содержащие внутри себя других циклов, и сложные (вложенные) циклы, содержащие один или несколько других циклов.

В зависимости от ограничений на число повторений выделяют циклы с известным числом повторений и циклы, число повторений которых заранее не известно.

Примером может служить вычисление ряда значений функции в зависимости от аргумента.

В языке Паскаль имеется три оператора для конструирования циклов.

Многократно повторяемые действия могут быть, например, заданы оператором цикла:

WHILE *B* DO *P*;

где *B* – логическое выражение (в частности, быть может, отношение), *P* – любой оператор (называемый телом цикла, в том числе и составной оператор). Работает данный оператор цикла так: проверяется условие *B*, и если оно выполняется (принимает значение true), то срабатывает оператор P, а затем вновь проверяется выражение *B* и т. д. Если же на очередном шаге окажется, что B полу-

чит значение ложь (false), то выполнение оператора цикла прекратится. Блок-схема такого оператора приведена на рис. 4.1.

Тело этого оператора цикла может не выполниться ни разу, если B при первой же попытке зайти в тело цикла окажется ложным.

Так, например, при положительном значении X выполнение оператора

while X < = 0 do X:= X+1;

прекратится после первой же проверки условия X < = 0, и значение переменной X не изменится. Если же значение *X* неположительно, то к этому значению будет добавляться по единице до тех пор, пока значение не станет положительным.

В данном случае число повторений циклической части оператора заранее не известно.

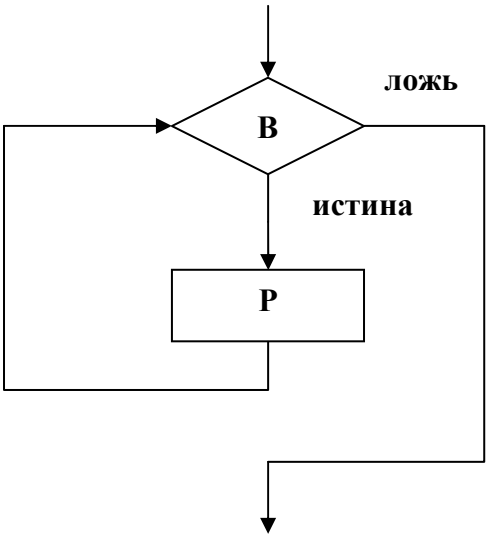


Рис. 4.1 Блок-схема оператора цикла типа WHILE

Ниже приведена программа, содержащая оператор цикла типа WHILE.

Пример 4.1. Пусть даны числа *a*, *b* (*a* > 1) и надо распечатать все степени числа *a*, значения которых меньше числа *b*.

Программа 4.1.

```
program stepen;
var a, b, St: real;
        k: integer;
begin
```

```
        k:=1;
        writeln (‘Введите a, b   ’);
        readln (a, b);
        writeln (‘   ’);
        St:= a;
        while St < b do
        begin
              writeln (‘При степени k = ', k:3,’ значение степени St = ', St:6:3);
              St:= St*a;
              k:= k+1;
        end;
        readln;
end.
```

При выполнении этой программы переменная St последовательно принимает значения $a1 = a$, $a2 = a*a$, $a3 = a*a*a$,... Изменение значения St происходит до тех пор, пока оно не станет больше или равно значению b. Если же с самого начала $a1 = a < b$, то не будет выведено ни одного члена последовательности $a1$, $a2$, $a3$...

Оператор цикла типа REPEAT также организует многократное выполнение последовательности операторов тела цикла с неизвестным числом повторений. Однако в отличие от только что рассмотренного случая тело цикла выполняется как минимум один раз. Выход из цикла осуществляется при истинности некоторого логического выражения B.

REPEAT $P1$; $P2$; $P3$;… UNTIL B;

где B – логическое выражение (в частности, быть может, отношение), $P1$, $P2$, $P3$,… – любые операторы, образующие тело цикла. Работает оператор так: выполняется $P1$, $P2$, $P3$,… проверяется условие B, и если оно истинно, то выполнение оператора цикла прекратится, если условие B не соблюдается, то вновь выполняются $P1$, $P2$, $P3$,…

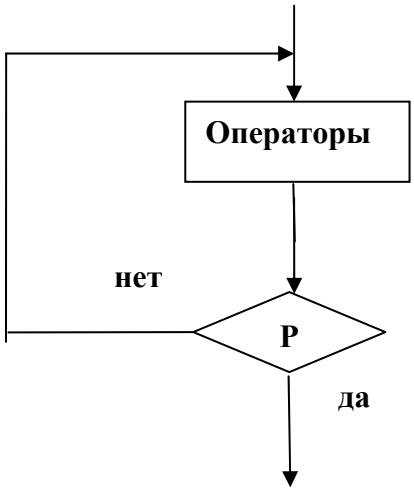


Рис. 4.2 Блок-схема оператора цикла типа REPEAT

Ниже приведена программа для той же задачи с одновременным выводом значения степени переменной:

Программа 4.2

```
program stepen;
var a, b, St: real;
k: integer:
begin
        writeln (‘Введите значения A и B ‘);
        readln (a, b);
        writeln (‘    ’);
        St:= a;
        k:= 1;
        repeat
                writeln (‘при степени K = ’,k:3, ‘     значение St = ’, St:6:3);
                St:= St*a;
                k:= k+1;
        until St> b
        readln;
end.
```

Следует учесть, что если с самого начала $a > b$, то все равно будет выведен первый член последовательности.

Третий оператор цикла, который называется *оператором цикла с параметром* (типа FOR), будет рассмотрен ниже.

Задание 1

Составить блок-схему алгоритма и программу для вычисления функций, приведенных в таблице 4.1. Сделать расчеты, распечатать листинг программы. Исходные данные и промежуточные результаты вывести на экран видеотерминала (дисплея) и на печатающее устройство.

Таблица 4.1

№ п/п	Функции
1	$Y = \begin{cases} \dfrac{(x^2 - a^2 x) \cdot tgx}{\sqrt{\lvert a + x \rvert}}, & \text{если } a \le x; \\ (a + x) \cdot e^{\sqrt{\lvert x + a \rvert}}, & \text{если } a > x; \end{cases}$
2	$Y = \begin{cases} \dfrac{(x + a) \cdot tgx}{\ln \lvert x \rvert}, & \text{если } a \le x; \\ e^{ax} + a \sin x^2, & \text{если } a > x; \end{cases}$
3	$Y = \begin{cases} \sqrt{a^2 + x^2}, & \text{если } a > x; \\ a - x \sqrt[3]{\sin^4 \lvert x \rvert}, & \text{если } a \le x; \end{cases}$
4	$Y = \begin{cases} \dfrac{x \cdot \sqrt{\lvert \sin x \rvert}}{\lvert a + x \rvert}, & \text{если } a \le x; \\ e^{\sqrt{\lvert x \rvert}}, & \text{если } a > x; \end{cases}$
5	$Y = \begin{cases} ax + 0.23x^2 \cdot \log_2 a, & \text{если } a \le x; \\ \dfrac{xe^a \cdot \cos x}{\lvert a + x \rvert}, & \text{если } a > x; \end{cases}$
6	$Y = \begin{cases} a^3 + \sqrt{a^2 + \lvert x \rvert} \cdot \sin^2 x, & \text{если } a > x; \\ 2x^2 + a^3 \cdot tgx, & \text{если } a \le x; \end{cases}$

7	$Y = \begin{cases} \dfrac{\sqrt{\lvert a+x \rvert}}{\lvert a^2+x \rvert}, & \text{если } a \le x; \\ e^{1.2} - \sqrt{\lvert ax \rvert}, & \text{если } a > x; \end{cases}$
8	$Y = \begin{cases} \dfrac{(x^2 - a^2 x) \cdot tgx}{\sqrt{\lvert a+x \rvert}}, & \text{если } a \le x; \\ (a+x) \cdot e^{\sqrt{\lvert x+a \rvert}}, & \text{если } a > x; \end{cases}$
9	$Y = \begin{cases} \dfrac{(x+a) \cdot tgx}{\ln\lvert x \rvert}, & \text{если } a \le x; \\ e^{ax} + a \sin x^2, & \text{если } a > x; \end{cases}$
10	$Y = \begin{cases} \sqrt{a^2 + x^2}, & \text{если } a > x; \\ a - x\sqrt[3]{\sin^4 \lvert x \rvert}, & \text{если } a \le x; \end{cases}$

Задание 2

Составить блок-схему алгоритма и программу для вычисления функций, приведенных в таблице 4.1. Исходные данные взять в таблице 4.2.

Таблица 4.2

Вариант	**Исходные данные**
1	$x \in [0.35,\ 0.43]$, *шаг* : 0.02; $a \in [0.50,\ 2.50]$, *шаг* : 0.50;
2	$x \in [0.54,\ 0.58]$, *шаг* : 0.01; $a \in [0.20,\ 1.00]$, *шаг* : 0.20;
3	$x \in [0.70 - 1.90]$, *шаг* : 0.30; $a \in [0.30 - 1.50]$, *шаг* : 0.30;
4	$x \in [0.80,\ 1.40]$, *шаг* : 0.20; $a \in [0.40,\ 2.00]$, *шаг* : 0.40;
5	$x \in [1.20,\ 2.00]$, *шаг* : 0.20; $a \in [0.40,\ 2.00]$, *шаг* : 0.40;
6	$x \in [1.50,\ 1.90]$, *шаг* : 0.10; $a \in [1.20,\ 2.00]$, *шаг* : 0.20;
7	$x \in [0.70,\ 1.30]$, *шаг* : 0.20; $a \in [0.30,\ 1.20]$, *шаг* : 0.20;

8	$x \in [0.30, 0.70]$, *шаг* : 0.10; $a \in [0.50, 1.10]$, *шаг* : 0.20;
9	$x \in [0.35, 0.43]$, *шаг* : 0.02; $a \in [0.50, 2.50]$, *шаг* : 0.50;
10	$x \in [0.54, 0.58]$, *шаг* : 0.01; $a \in [0.20, 1.00]$, *шаг* : 0.20;

Модифицировать полученную при выполнении работы № 3 программу таким образом, чтобы автоматически осуществлялись многократные решения задачи при изменении одного из исходных данных в заданном диапазоне. Шаг изменения аргумента должен быть выбран так, чтобы число повторений цикла лежало в диапазоне от 3 до 10 (исходная переменная, значение которой должно изменяться, определяется студентом самостоятельно).

Результаты решения выводить в виде таблицы, например:

Аргумент X0	**Функция**	**Площадь**	**Комментарий**
1.0	12.89	345.77	точка на окружности
1.2	17.90	785.42	точка вне окружности
…	…	…	…

Методические указания

Программа должна обеспечивать циклический вычислительный процесс – многократные решения сформулированной задачи при различных наборах исходных данных. При этом в задании оговаривается, что изменение исходных данных для очередного определения результатов (выполнения цикла из соответствующих операций) сводится к изменению лишь одного из них.

Этот аргумент (обозначим его здесь X) должен последовательно принимать значения в определенном диапазоне (X_o – начальное значение, $X_к$ – конечное значение), каждое из которых больше предыдущего на постоянную величину (h). Эта величина h называется *шагом изменения аргумента*

X_o, $X_1 = X_o+h$, $X_2 = X_1+h$,..., $X_i = X_{i-1}+h$.., $X_k = X_{k-1}+h$

Алгоритм циклического вычислительного процесса также должен содержать структуру повторения, предусматривающую неоднократные выполнения определенной последовательности операторов цикла (*P*), пока истинно условие выполнения ее.

Для программируемой задачи структура повторения может конкретизироваться в виде, изображенном на рис. 4.3, где под *P* понимается определение и вывод результатов вычислений при очередном значении изменяющегося аргумента *X*. Действительно, поскольку закон изменения аргумента (циклической переменной) известен, он может быть представлен как простая (неиндексированная) переменная *X*, а повторяемая в цикле инструкция присваивания $X = X+h$ будет обеспечивать вычисление очередного значения этой переменной путем увеличения предыдущего значения на заданный шаг ($X_i = X_{i-1}+h$). При этом для первого выполнения цикла (последовательности *P*) необходимо предусмотреть присваивание исходной переменной начального значения аргумента ($X = X_o$), а в качестве условия выполнения цикла рассматривать выражение $X <= X_k$, где X_k – конечное значение аргумента.

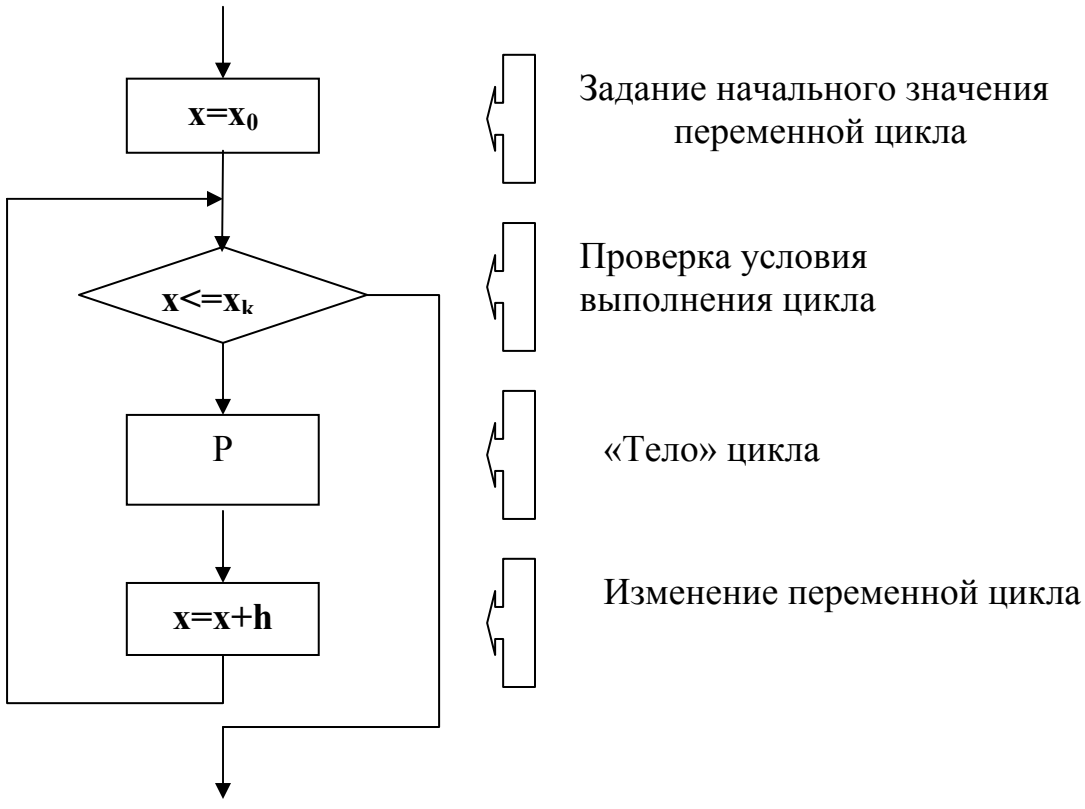


Рис 4.3 Блок-схема циклического вычислительного процесса

Более компактное изображение такой структуры повторения в виде блок-схемы алгоритма приведено на рис. 4.4, где описывается выполнение последовательности операторов *P* при изменении исходной переменной *X* от начального значения X_o до конечного значения X_k с шагом *h*.

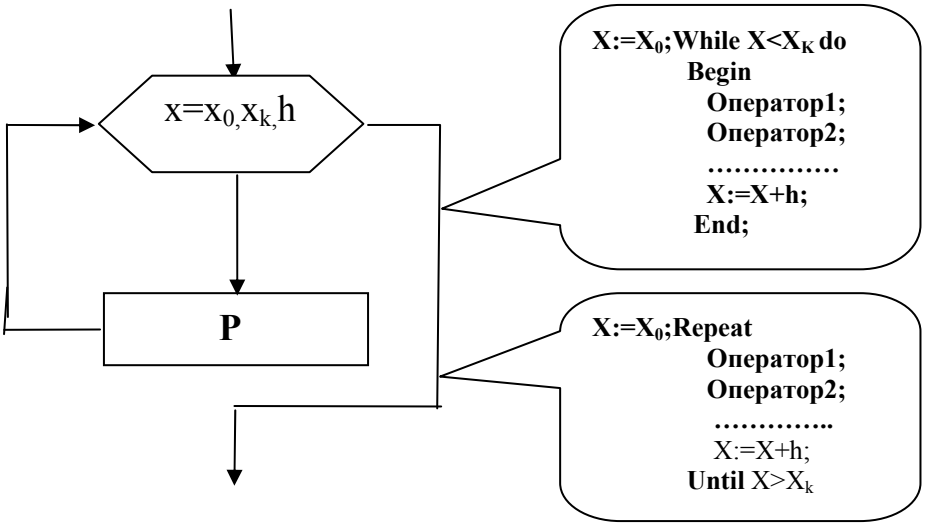


Рис. 4.4 Блок-схема циклического процесса

С учетом вышеизложенного исходными данными, которые следует вводить в ПК в качестве значений соответствующих переменных, должны быть – начальное и конечное значения (X_o, X_k) изменяемого аргумента, шаг изменения его значения (*h*), а также неизменяемые значения остальных исходных величин задачи.

Таким образом, для описания структуры повторения в Паскаль-программе возможно использование рассмотренных операторов цикла WHILE или REPEAT.

Контрольные вопросы и упражнения

1. Что понимают под циклическими процессами на ПК?
2. Что собой представляет алгоритмическая структура повторения? Какие основные операции она предусматривает?
3. В чем особенность структур повторения, в которых в качестве исходных и искомых переменных используются простые переменные?

4. Какие операторы используются при описании циклов? Какие ограничения накладываются на использование этих операторов?

5. При достижении каких условий заканчивается выполнение тела цикла?

6. Что означают понятия: тело цикла, параметр цикла, конечное значение параметра цикла, шаг изменения параметра цикла, программа зациклилась.

7. Прокомментировать разработанную циклическую программу.

8. Какими будут значения переменных *s* и *f* после выполнения операторов:

```
s:= 1; f:= 1;
While s <= 4 Do s:= s+1; f:= f+1;
```

9. Какими будут значения переменных s и f после выполнения операторов:

```
s:= 1; f:= 1;
While s <= 4 Do begin; s:= s+1; f:= f+1; end;
```

10. Написать программу, вычисляющую сумму цифр введенного целого числа.

Лабораторная работа №5

Основные алгоритмы обработки массивов данных

Цель работы: получение практических навыков алгоритмизации и программирования циклических вычислительных процессов при работе с одномерными массивами.

Теоретическая часть

Массивом называется упорядоченное (пронумерованное) множество элементов одного типа, обозначенных одним идентификатором (имеющих одно имя). Элементы множества называются *элементами массива*. Отношение порядка между элементами массива задается с помощью индексирования, т. е. положение элемента в массиве указывает индекс или совокупность индексов.

Массив может быть описан, например, так:

VAR **T: ARRAY [1…20] OF REAL;**

F: ARRAY [1…50] OF INTEGER;

Эти записи означают, что в программе будут использоваться массив t_k, состоящий из 20 вещественных элементов, и массив f_m, состоящий из 50 целых чисел.

Индексы заключаются в квадратные скобки. Например, запись T[1], T[2], …, T[K] соответствует индексированным переменным t_1, t_2, …, $t_к$.

Индекс, в частности, можно задавать целой константой, целой переменной или выражением, принимающим целые значения.

Значение индекса в программе должно быть не меньше нижней границы и не больше верхней.

В Паскале возможно использование также оператора цикла (третий вид), который называется *оператором цикла с параметром*:

FOR i:= A **TO** B **DO** P.

Здесь *i* – некоторая переменная порядкового типа (integer, char, boolean и др.), которая называется *параметром цикла*, *A* и *B* – выражения, имеющие упомянутый тип, *P* – оператор (тело цикла).

Оператор цикла с параметром выполняется так. Сначала вычисляются значения выражений *A* и *B*. Получаются два, например, целых числа, которые мы обозначим, соответственно, через $\underline{A}$ и $\underline{B}$. Если $\underline{A}$ меньше или равно $\underline{B}$, то переменная *i* последовательно принимает значения, равные $\underline{A}$, $\underline{A}$+1, $\underline{A}$+2,..., до тех пор, пока *i* не станет больше или равно *B*. Для каждого из упомянутых значений *i* выполняется оператор *P*. Если $\underline{A}$ изначально больше $\underline{B}$, то оператор *P* не будет выполнен ни разу и выполнение оператора цикла с параметром сразу же закончится. Для описания действия оператора цикла FOR используем блок модификация (см. рис. 5.1).

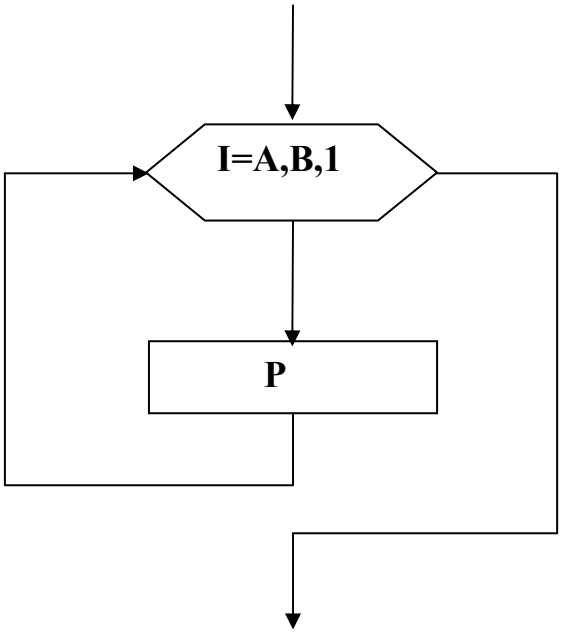


Рис. 5.1 Блок-схема оператора цикла с параметром (оператора FOR)

Например, для начального $s = 0$ и любого положительного n при выполнении оператора

for i:= 1 to n do s:= s+i*i*i

переменная *s* примет значение $1^3+2^3+...+n^3$.

Имеется еще один вариант оператора цикла с параметром:

FOR I := A **DOWNTO** B **DO P;**

Здесь *i* принимает последовательно значения *A*, *A*–1, ..., и для каждого из них выполняется оператор *P*, если же *A* меньше *B*, то он не выполняется ни разу.

Следует еще раз обратить внимание на то, что оператор цикла с параметром используется в тех случаях, когда начальное и конечное значения параметра цикла и сам параметр являются данными порядкового типа (integer, char, boolean и др.), а шаг изменения равен 1 или –1.

Работа с элементами массива обычно сводится к организации циклов, где параметром цикла является индекс и чаще всего используется оператор: for i:= N to K do S; , где *i* – индекс, *N* и *K* – начальное и конечное значение индекса, *S* – оператор (тело цикла).

Так, чтобы ввести значения t_1, t_2, t_3, …, t_{20} вместо предложения readln (t[1], t[2], t[3], ...,t[20]); достаточно указать: for I:= 1 to 20 do readln (t[i]);

Пример 5.1. Ввести последовательность a_1, ...,a_n, состоящую из целых чисел, где n<=15. Требуется определить a_k – наибольший член массива a_1, ...,a_n вместе с его порядковым номером *k*. В программе текст, выделенный жирным шрифтом, рекомендуется запомнить и использовать для всех программ, где требуется ввести одномерный массив.

Программа 5.1

```
program maximum;
var i, n, k, M:  integer;
        a: array [1..15] of integer;
begin
        write (‘Введите количество элементов массива N = ’);
        readln (n);
        write (‘Введите элементы массива = ’);
        for i:= 1 to n do
        read (a [i]);
        k:= 1; M:= a[1];
        readln;
        for i:= 2 to n do
```

```
        if a[i] > M then
            begin
                M:= a[i];
                k:= i
            end;
    writeln (‘наибольшее MAX = ’, M:5, ‘индекс K = ’,k:2);
    readln;
end.
```

Пример 5.2. Пример относится к упорядочиванию (сортировке) массива x_1, ..., x_n. Задача упорядочивания возникает при оформлении статистических сводок, справочных материалов и т. д. Этот процесс при большом объеме данных очень трудоемок, поэтому в таких случаях часто прибегают к помощи компьютеров. Здесь рассматривается задача упорядочивания в простейшей постановке: дан числовой массив x_1,..., x_n, элементы которого различны, требуется переставить элементы массива так, чтобы после перестановки они были упорядочены в порядке возрастания: x_1<...< x_n. Один из алгоритмов решения этой задачи приводится ниже.

Очевидно, что первое место в массиве должен занять наименьший элемент, второе место – наименьший из всех остальных элементов и т. д.

Схема решения может быть следующей (см. рис. 5.2):

1. Описание данных.
2. Ввод исходного массива x.
3. Поиск наименьшего элемента среди всего массива.
4. Перестановка местами с первым элементом.
5. Печать первого элемента.
6. Поиск наименьшего среди оставшихся элементов.
7. Перестановка местами со вторым элементом.
8. Печать второго элемента.
9. Выполнение аналогичных действий, вплоть до сравнения последних элементов массива x_{n-1} и x_n.

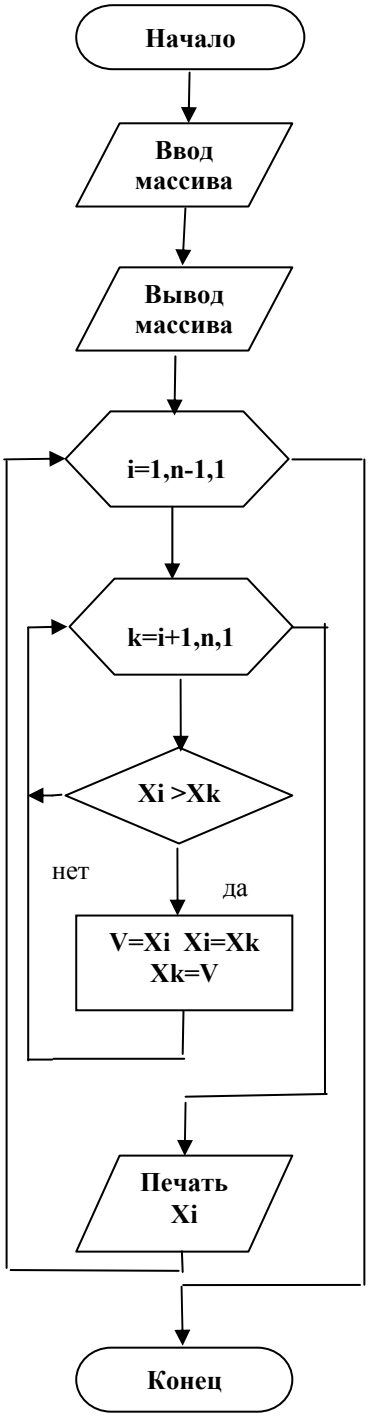


Рис. 5.2. Блок-схема программы сортировки элементов одномерного массива (Пример 5.2)

Пример 5.2.

```
program poradok;
var x: array[1..100] of real;
v: real;
n, i, j, k: integer;
begin
write (‘Введите количество элементов = ’);
readln (n);
write(‘Введите элементы = ’);
for i:=1 to n do read(x [i]);
readln;
writeln;
writeln;
writeln (‘ ’:10,‘Исходный массив ’);
for i:= 1 to n do writeln (‘x [‘,i:2,’] =‘, x[i]:6:2);
writeln;
writeln (‘ ’:10,‘Упорядоченный массив’);
writeln;
for i:= 1 to n do
{внешний цикл, где будут рассматриваться все элементы}
begin
for k:= i+1 to n do
{внутренний цикл по поиску минимального элемента }
if x[i] > x[k] then
begin
v:=x[i]; x[i]:=x[k]; x[k]:=v;
end;
writeln (‘ x [‘,i:2,’] = ’, x[i]:6:2);
end;
readln;
end.
```

Для перестановки текущего x[i] с найденным x[k] привлекается дополнительная переменная v: v:= x[i]; x[i]:= x[k]; x[k]:= v;

Задание

Модифицировать полученную при выполнении работы №3 программу таким образом, чтобы на ПК осуществлялось многократное вычисление функции при различных значениях одного из исходных данных, предварительно введенных в оперативную память ПК. Численные результаты должны запоминаться в памяти в виде одномерного массива (исходная переменная, ряд значений которой задается для многократного решения задачи, определяется студентом самостоятельно). Результат вывести в виде таблицы.

Методические указания

Задание на программирование предполагает организацию на ПК циклического вычислительного процесса, а определение результатов должно осуществляться для нескольких вариантов одного из данных. Все значения этого изменяемого аргумента могут заранее вводиться в оперативную память ПК в виде одномерного массива чисел или вычисляться в ходе выполнения цикла. Кроме того, в задании оговаривается, что и результаты каждого решения задачи должны запоминаться как численный массив. Таким образом, в процессе обработки данных на ПК должны выполняться операции над элементами одномерных массивов чисел.

Алгоритм обработки данных, отвечающий требованиям задания, должен предусматривать:

- ввод исходных данных, в том числе количества значений и самих значений массива;
- многократно выполняемый цикл определения и запоминания в памяти ПК результатов вычислений в виде массива значений;
- вывод результатов как элементов соответствующего массива.

Необходимо учитывать, что значение индекса определяет порядковый номер элемента в массиве (например, X_i означает первый элемент массива X, если $i = 1$, второй элемент этого же массива, если $i = 2$ и т. д.). Поэтому управляющей переменной циклического алгоритма должна быть переменная – индекс, используемая в теле цикла (вместе с именами массивов) для обозначения элементов исходного и искомого массивов. Значение же индекса при повторении цикла должно меняться от 1 до N с шагом 1, где N – количество значений аргумента.

В Паскаль-программе, предусматривающей ввод, обработку, формирование и вывод массивов данных, следует обязательно описать их соответствующими невыполняемыми операторами. В операторе описания одномерного массива указывается тип значений элементов массива с помощью определенного ключевого слова (INTEGER, REAL и т. п.), имя массива и количество элементов в массиве.

Вывод в программе следует предусмотреть, например, так:

```
WriteLn (‘N = ’); ReadLn (N);
for k:=1 to N do Write (X[k]);
```

Контрольные вопросы и упражнения

1. Что собой представляют массивы данных?
2. Какова роль индексированных переменных при программировании задач по обработке массивов данных?
3. Каким образом в алгоритмах и программах предусматривается переход к использованию и определению новых элементов массива?
4. Что собой представляют структуры повторения по обработке массивов данных?
5. В чем смысл и необходимость описания массивов в Паскаль-программах?
6. Все ли операторы цикла можно использовать для организации работы с элементами массивов?

7. Какие ограничения накладываются на использование оператора FOR для описания циклов?

8. Прокомментировать составленную программу.

9. Определить в массиве количество нулевых элементов, сумму положительных элементов, индексы отрицательных элементов.

10. Определить, есть ли в данном массиве два соседних положительных числа. Если есть, вывести индексы первой (последней) пары.

Лабораторная работа №6

Алгоритмы с вложенными циклами

Цель работы: используя массивы, получить практические навыки алгоритмизации и программирования вычислительных процессов с вложенными циклами.

Теоретическая часть

Вложенными называются такие циклы, которые состоят из двух и более включенных друг в друга циклов. При этом циклы, объемлющие (охватывающие) другие циклы, принято называть внешними, а циклы, входящие (включенные) во внешние циклы, – внутренними (вложенными).

Особенность выполнения вложенного цикла состоит в том, что одно выполнение внешнего цикла связано с многократным выполнением внутреннего цикла. При этом каждая фиксация параметра внешнего цикла предполагает изменение параметра внутреннего цикла таким образом, что он пробегает все свои значения.

Вложенные циклы используются в основном при программировании алгоритмов с многомерными массивами.

Следует отличать размерность массива от размера массива. Если *размерность массива* – это, по существу, число индексов, определяющих положение элемента в массиве, то *размер массива* – это число элементов в массиве.

Так, например, элементы матрицы

$$y = \begin{pmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \end{pmatrix}$$

представляются в программе как *Y* [1,1], *Y* [1,2], *Y* [1,3] и т.д., или в общем виде как *Y* [*I*, *K*], где *I* – номер строки, *K* – номер столбца. При этом переменная I может принимать значения 1, 2, а переменная *K* – значения 1, 2, 3. Поэтому массив *Y* имеет размерность 2 (количество индексов), а размер – 6 (количество

элементов). Индексы указывают не только положение элемента в массиве, но и количество измерений массива.

В составляемых программах элементы двумерного массива необходимо задавать и выводить построчно: вначале элементы первой строки, затем – второй и т. д.

При алгоритмизации и программировании сложного циклического процесса используются вложенные друг в друга типовые структуры. При этом, как уже отмечалось, за одно использование внешнего цикла внутренний цикл повторяется многократно (см. рис. 6.1 и таблицу 6.1).

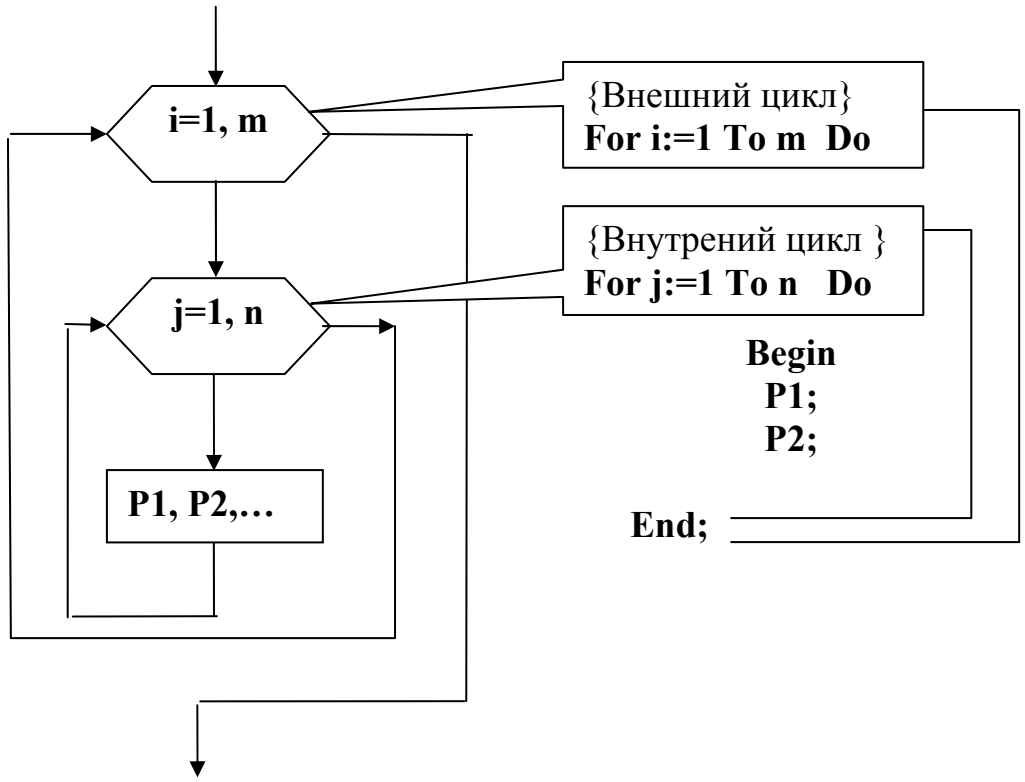


Рис. 6.1 Блок-схема алгоритма вложенного цикла

Для $m = 2$ и $n = 3$ характер изменения параметров i и j представлен в таблице 6.1.

Таблица 6.1

i	1			2		
j	1	2	3	1	2	3

Ниже рассмотрены примеры решения задач с использованием сложных циклов и индексированных переменных.

Пример 6.1. Пусть дана матрица размера $N \times K$ (значение N – не более 10), состоящая из действительных чисел, надо преобразовать матрицу так: поэлементно вычесть последний столбец из всех столбцов, кроме последнего (столбец с номером j – это элементы $a[1, j]$, $a[2, j]$,..., $a[n, j]$).

Текст, выделенный жирным шрифтом, можно использовать для ввода и вывода матрицы в своих программах.

Программа 6.1

```
program COLUMN;
var a: array[1..10,1..10] of real;
    i, j, n, k : integer;
begin
    write('Введите размеры матрицы: ');
    readln(n, k);
    writeln('Введите элементы матрицы построчно:');
    for i:= 1 to n do
        for j:= 1 to k do
            read (a[i, j]);
        writeln (' ':6,' Исходная матрица: ');
        for i:=1 to n do
            begin
                for j:=1 to k do
                    write (a[i,j]:10:2);
                writeln (' ');
            end;
    for j:= 1 to k–1 do
        for i:= 1 to n do
            a[i, j]):= a[i, j]) – a[i, k];
    writeln (' ':6,' Результат выполнения программы: ');
    for i:= 1 to n do
        begin
```

```
                for j:= 1 to k do
                        write (a[i, j]:10:2);
                writeln(‘ ’);
            end;
        readln;
        readln;
end.
```

Пример 6.2. Дана матрица размера *N*×*M*, состоящая из вещественных чисел. Требуется упорядочить (переставить) строки матрицы по возрастанию первых элементов строк. Взяв за образец схему программы poradok из предыдущей работы, указав 5×6 в качестве размера матрицы, можно получить следующее:

Программа 6.2

```
program poradok;
const n=5;
        m=6;
var x: array[1..n,1..m] of real;
        i, j, l: integer;
        v: real;
begin
        writeln (‘Введите элементы матрицы построчно’);
        writeln (‘Количество строк=’, n:2,
        writeln (‘Количество столбцов=’, m:2);
        for i:= 1 to n do
            for j:= 1 to m do
                read (x[i, j]);
        writeln (‘Исходная матрица: ’);
        for i:=1 to n do
            begin
                for j:=1 to m do
                    write (x[i, j]:8:2);
```

```
                writeln (‘ ’);
            end;
        for i:= 1 to n-1 do
            for j:= i+1 to n do
                if x[j, l] < x[i, l] then
                    for l:= 1 to m do
                        begin
                            v:=x[i,l]; x[i,l]:=x[j,l]; x[j,l]:=v;
                        end;
        writeln (‘Упорядоченная матрица’);
        for i:= 1 to n do
        begin
            for j:= 1 to m do
                write (x[i, j]:8:2);
            writeln;
        end;
    writeln;
    readln;
    readln;
    end.
```

Пример 6.3

Написать программу вычисления значений функции

$$y = \begin{cases} e^{x}+(a+x), \text{ если } x < b; \\ (a^2+x^2)e^{x}, \text{ если } b<x<c; \\ a^{x}+a\sin^2(x), \text{ если } x>c \end{cases}$$

при a =[0.50, 2.00], шаг h_a=0.50 и при x=[–1.00, 1.00],шаг h_x=0.40.

Значения функции записать в массив Y, который вывести затем в виде прямоугольной таблицы чисел. При решении задачи использовать оператор цикла с параметром (цикл типа FOR).

Программа 6.3

```
Program Massiv;
Var a, b, c, x, ha, hx: real;
        i, j:integer;
        y: array [1..4, 1..6] of real;
begin
    ha:= 0.50; hx:= 0.40;
    b:= –0.50; c:= 0.50;
    a:= 0.50;{Инициализация циклического параметра внешнего цикла}
    for i:=1 to 4 do
        begin
            x:= –1.00;{Инициализация циклического параметра
            внутреннего цикла}
            for j:=1 to 6 do
                begin
                    if (x<b) then y[i, j]:=exp(x)+(a+x)
                    else if (x>c) then
                        y[i,j]:=exp(x*ln(a))+a*sqr(sin(x))
                            else y[i,j]:=(a*a+x*x)*exp(x);
                                x:=x+hx;
                end;
                a:=a+ha;
        end;
        writeln (‘ ’:16,‘Массив результатов’);
        for i:= 1 to 4 do
            begin
                for j:= 1 to 6 do
                    write (y[i, j]:6:1, ‘ ’:2);
                writeln;
            end;
    readln;
end.
```

Задание 1

Модифицировать полученную при выполнении работы №5 программу таким образом, чтобы на ПК автоматически осуществлялись многократные решения задачи при изменении значений двух исходных переменных (для каждой пары заданных значений). Численные результаты должны быть получены в виде двумерного массива данных (исходные переменные, значения которых должны изменяться при многократном решении задачи, определяются преподавателем или студентом самостоятельно). Вывод по-прежнему в виде таблицы.

Задание 2

1. Найти наибольший элемент главной диагонали матрицы *A* (*N***N*) и вывести всю строку, в которой он находится.

2. Переписать положительные элементы главной диагонали матрицы *A* (*N***N*) в одномерный массив.

3. Найти наибольший по абсолютной величине элемент второй строки матрицы *A* (*N***N)*.

4. Найти наименьший элемент главной диагонали матрицы *A* (*N***N*) и вывести весь столбец, в котором он находится.

5. Найти сумму элементов главной (*S*1) и побочной (*S*2) диагоналей матрицы *A* (*N***N*).

6. Дана прямоугольная матрица *A* (*N***M*). Получить новую матрицу с элементами (*i*+*j*)**A* (*i*, *j*).

7. Дана прямоугольная матрица *A* (*N***M*). Подсчитать сумму всех элементов, больших, чем 0,5.

8. Дана матрица *A* (*N***M*). Подсчитать сумму квадратов элементов главной диагонали.

9. Дана матрица *A* (*N***M*). Переменной *B* присвоить значение, равное количеству строк матрицы *A*, содержащих хотя бы одну нулевую компоненту.

10. Определить, сколько элементов матрицы *A* (*N***M*) превышает число 3,345.

11. Среди строк заданной целочисленной матрицы, содержащих только нечетные элементы, найти строку с максимальной суммой модулей элементов.

12. Среди строк заданной целочисленной матрицы, содержащих только такие элементы, которые по модулю не больше 10, найти столбец с минимальным произведением элементов.

13. В заданной матрице *A* (*N***N*) найти количество строк, не содержащих отрицательных чисел.

14. Найти произведение элементов, расположенных на побочной диагонали матрицы *A* (*N***N*).

15. 15. Дана матрица *A*(*N***N*) и массив *X*(*N*). Заменить нечетные строки матрицы элементами массива *X*(*N*).

16. Дана матрица *A* (*N***N*). Сформировать новую матрицу *B* (*N***N*), в которой все элементы совпадают с элементами матрицы *A* за исключением: если элементA_{ij} превышает значение 20, то соответствующему элементу из *B* присваивается значение 20.

17. Дана матрица *A* (*N***N*). Сформировать новую матрицу *B* (*N***N*), в которой диагонали совпадают с соответствующими диагоналями матрицы *A*, а все остальные элементы равны 0.

18. Переписать положительные элементы матрицы *A* (*N***N*) в одномерный массив *X* (*M*).

19. Найти наибольший элемент в *k*-м столбце матрицы *A* (*N***N*).

20. Найти сумму всех положительных элементов матрицы *A* (*N***M*).

Методические указания

Задание предполагает организацию на персональном компьютере циклического вычислительного процесса, в ходе которого результаты решения задачи (обозначены здесь *R*) определяются многократно для разных значений двух аргументов (например, *X*, *Y*):

$R_{i,j} = P\ (X_i,\ Y_j,\ ...),$

$i = 1,2,\ ...,\ N;$

$j = 1,2,\ ...,\ M.$

Данный циклический процесс иногда называют сложным, т. к. он содержит два вложенных друг в друга цикла. Повторение внутреннего цикла *M* раз приводит к определению результатов при некотором фиксированном значении одного из изменяющихся аргументов (например, при X_i) для каждого значения второго аргумента (Y_1, Y_2, ..., Y_m):

$R_{i,1} = P\ (X_i,\ Y_1,...),$

$R_{i,2} = P\ (X_i, Y_2,...),$

…

$R_{i,m} = P\ (X_i, Y_m,...).$

В качестве исходных переменных, значения которых должны изменяться в ходе решения задачи, используют индексированные переменные. В этом случае программа должна предусматривать ввод всех необходимых для вычислений значений этих переменных в память ПК или определение значений элементов массивов с помощью генератора случайных чисел.

Результаты же решения задачи для каждого варианта исходных данных (этого требует задание) должны представляться как элементы двумерных массивов чисел. Следовательно, искомыми переменными будут индексированные переменные с двумя индексами (например, $R_{i,j}$).

С учетом вышеизложенного внешняя и внутренняя структуры вложенных циклов задачи в качестве управляющих переменных должны содержать переменные – индексы (например, *I*, *J*), используемые для обращения к элементам

исходных и искомых массивов чисел (рис. 6.1). Ясно, что внутренний цикл, должен полностью находиться между операторами внешнего цикла.

Контрольные вопросы и упражнения

1. Что понимают под вложенным циклом?
2. Каковы особенности программирования сложных циклических процессов?
3. Каковы правила записи вложенных друг в друга структур повторения (сложных циклов) на языке Паскаль.
4. Что собой представляют двумерные массивы данных?
5. Каким образом предусматриваются операции над элементами двумерных массивов в программах?
6. Прокомментировать разработанную программу.
7. Написать программу вывода элементов главной диагонали квадратной матрицы и вычисления суммы этих элементов.
8. Поменять местами первый максимальный и последний минимальный элементы двумерного массива.
9. Составить программу вывода на экран арифметического квадрата, в котором первый столбец и первая строка заполнены 1 (единицами), а каждый из остальных элементов равен сумме соседних сверху и слева.

ПРИЛОЖЕНИЕ 1

Образец оформления титульного лист

Федеральное агентство по образованию

ГОУ ВПО «Уральский государственный технический университет – УПИ»

имени первого Президента России Б.Н. Ельцина

Кафедра «Интеллектуальных информационных технологий»

Дисциплина «Информатика»

Отчет по лабораторным работам

Решение задач в интегрированной среде

Турбо Паскаль

Преподаватель Паклина В.М

Студент

группы СМ-16032 Иванов Б. Е.

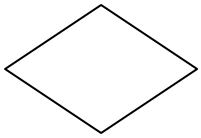
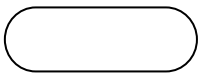
Екатеринбург

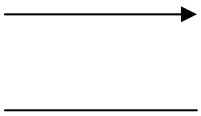
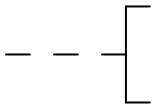
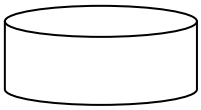
2009

ПРИЛОЖЕНИЕ 2

Блок-схемы

Блок-схемы представляет собой графическое изображение логики решения задачи.

Фигура	Обозначение	Описание
	Процесс	Выполнение операции или группы операций, в результате которых изменяется значение, форма представление или расположение данных, например, вычисления
	Решение	Выбор направления выполнения алгоритма или программы в зависимости от некоторых условий, например, для разветвления
	Ввод / вывод	Обозначение ввода исходных данных или отображения результата, например, печать
	Модификация	Выполнение операций, меняющих команды или группы команд, меняющих программу, например, для организации циклов
	Начало/ окончание	Обозначение с символом «Н»/«К» всегда является первым / последним элементом алгоритма
	Внутристраничный соединитель	Указание связи между прерванными линиями потока, в указателе, как правило, располагаются буквы алфавита, два соответствующих друг другу соединителя имеют одинаковые буквы
	Межстраничный соединитель	Используется в конце страницы блок-схемы, если она продолжается на другой, внутри элемента указывается номер страницы с продолжением, вверху страницы с продолжением указывается номер предыдущей страницы

	Предопредленный процесс	Использование ранее созданных и отдельно описанных программ или алгоритмов, например, ссылка на подпрограммы.
	Направление потока	Стрелки или линии стоят у каждого элемента блок-схемы, указывают направление потока выполнения программы.
	Комментарий	Используется для элементов блок-схемы, которые требуют пояснения
	Магнитный диск	Показывает место хранения данных.

Блок-схемы состоят из стандартизированных элементов. Для рисования можно воспользоваться линейкой-шаблоном или программными средствами, на крайний случай, обычной линейкой.

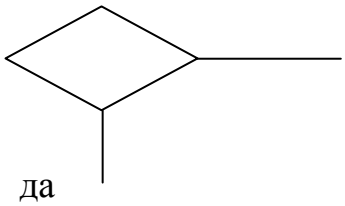
Структура алгоритма должна изображаться на странице сверху вниз и слева направо. В этом случае блоки соединяются линиями, при нарушении направлений следует использовать стрелки.

Расстояние между параллельными линиями потока не менее 3 мм, между остальными символами схемы не менее 5 мм.

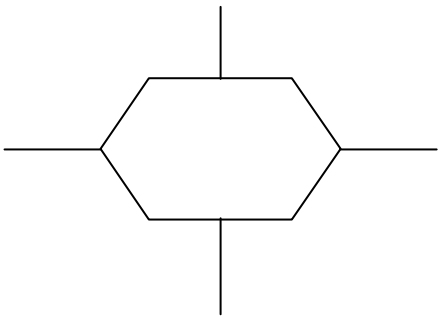
Вход в блок внутристраничного соединителя допускается в любом месте (выхода этот блок не имеет):

Вход в блок межстраничного соединителя допускается только сверху, выход из этого блока – только снизу:

Элемент «Решение» – единственный элемент, имеющий два выхода, в этом месте выбирается один из вариантов продолжения. Рядом с каждым выходом следует указать «Да» и «Нет» (истина и ложь).



Блок «модификация» имеет выходы и входы (кроме входа в блок) со следующими значениями:



возврат к началу цикла, пока параметр цикла не превысил максимального значения

выход из цикла, когда параметр цикла превысил максимальное значение

вход в тело цикла

Инструкции внутри условных обозначений должны иметь четкую формулировку на разговорном языке, без применения операторов языка программирования.

Библиографический список

1. **Информатика.** Базовый курс: учеб. пособие для студентов вузов / С.В. Симонович [и др.]. – 2-е изд. – М., 2007. – 640 с.

2. **Немнюгин, С.А.** Turbo Pascal: Практикум / С.А. Немнюгин. – М. : Питер, 2003. – 256 с.

3. **Немнюгин, С.А.** Turbo Pascal. Программирование на языке высокого уровня: Учебник для студентов вузов, обучающихся по направлению подгот. дипломир. специалистов «Информатика и вычислительная техника» / С.А. Немнюгин. – 2-е изд. – М., 2003. – 544 с.

4. **Фаронов, В.В.** Турбо Паскаль. Начальный курс: Учебное пособие / В.В. Фаронов. – 7-е изд., перераб. – М. : Нолидж: ОМД Групп, 2002. – 576 с.

5. **Гусева, А.И.** Учимся информатике: задачи и методы их решения: Учеб. пособие / А.И. Гусева. – 2-е изд., испр. и доп. – М. : Диалог-МИФИ, 2001. – 384 с.

Учебное электронное текстовое издание

Томашевич Виктор Григорьевич

РЕШЕНИЕ ЗАДАЧ В ИНТЕГРИРОВАННОЙ СРЕДЕ ТУРБО ПАСКАЛЬ

Редактор *А. В. Ерофеева*
Компьютерный набор *авторский*

Рекомендовано РИС ГОУ ВПО УГТУ–УПИ
Разрешен к публикации 21.05.09
Электронный формат – pdf
Объем 3,9 уч.-изд. л.

Издательство ГОУ ВПО УГТУ–УПИ
620002, Екатеринбург, ул. Мира, 19

Информационный портал
ГОУ ВПО УГТУ–УПИ
http://www.ustu.ru