Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«КАЛИНИНГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ»

Программирование

Учебно – методическое пособие по лабораторным работам

для студентов бакалавриата

по направлениям подготовки

" Информатика и вычислительная техника"

"Прикладная информатика"

(часть первая)

Калининград

Издательство ФГБОУ ВПО «КГТУ»

2014

УДК 681.3.06

УТВЕРЖДЕНО

Ректором ФГБОУ ВПО

«Калининградский

государственный технический

университет»

АВТОРЫ: Гудков А.Л., к.т.н., доцент кафедры систем управления и вычислительной техники» ФГБОУ ВПО «Калининградский государственный технический университет»;

Калинина С.А., доцент той же кафедры

Учебно – методическое пособие рассмотрено и одобрено кафедрой систем управления и вычислительной техники ФГБОУ ВПО «Калининградский государственный технический университет» 22 декабря 2014 г., протокол № 3.

Учебно – методическое пособие рекомендовано к печати методической комиссией факультета автоматизации производства и управления ФГБОУ ВПО «Калининградский государственный технический университет» 24 декабря 2014 г., протокол № 7.

**РЕЦЕНЗЕНТ –** кафедра систем управления и вычислительной техники ФГБОУ ВПО «Калининградский государственный технический университет»

**Введение**

Учебно – методическое пособие по лабораторным работам предназначено для студентов, изучающих на первом курсе основы алгоритмизации и процедурно - ориентированного программирования с использованием системы ("интегрированной среды") Турбо-Паскаль (Turbo Pascal) в рамках дисциплины "Программирование".

Представленный в данном пособии перечень лабораторных работ рассчитан на выполнение их в течение первого семестра обучения в университете. По лабораторным работам второго семестра следует использовать "вторую часть" учебно – методического пособия [8]. Параллельно с выполнением лабораторных работ студенты слушают лекционный курс по программированию, самостоятельно изучают язык программирования, используя литературу и учебные пособия.

При выполнении лабораторных работ, каждая из которых (кроме первой) предусматривает разработку Паскаль-программы решения на персональном компьютере определенной задачи, студент должен выполнить задание, руководствуясь соответствующими методическими указаниями, оформить отчет о проделанной работе и защитить её. В отчете приводится – задание, описание исходных для программируемой задачи данных и результатов её решения, схема алгоритма и текст программы. При защите лабораторной работы студент представляет преподавателю выполнение разработанной программы и отвечает на вопросы, демонстрируя полученные знания, умения и навыки по программированию.

Помимо заданий и необходимых для их выполнения сведений по каждой лабораторной работе в пособии в приложениях приводится справочный материал.

# Лабораторная работа №1

**Освоение системы программирования** Turbo Pascal

Цель работы: научиться использовать средства системы (интегрированной среды) программирования Turbo Pascal.

# Задание по лабораторной работе

Использовать средства системы Turbo Pascal для ввода в компьютер и выполнения простой программы

**Методические указания по выполнению работы**

Прежде всего следует ознакомиться с основными средствами системы Turbo Pascal, описанными ниже, или по соответствующей учебной литературе, список которой приведен в конце пособия, и загрузить систему для появления на экране монитора главного меню.

Далее необходимо:

1. Ввести в компьютер текст программы (рис. 1.1), предусматривающей вычисление площади прямоугольника по длинам сторон. Комментарии в фигурных скобках можно опустить, кроме второго, они предназначены для пояснений предложений программы.

**Program P1;** { Заголовок программы}

**{Фамилия студента, шифр группы}** {Комментарий}

**Var** {Описание переменных}

**a, b, s: real;**

{Значения переменных должны быть вещественными числами}

{ ИСПОЛНЯЕМАЯ ЧАСТЬ ПРОГРАММЫ}

**Begin**

**Writeln ('введите значения длин сторон ');**

**Readln (a);** {чтение с клавиатуры значения a }

**Readln (b****);** {чтение с клавиатуры значения b }

**s:=a\*b;** { вычисление площади прямоугольника}

**Writeln (‘ площадь= ‘, s:7:2);**

{ вывод на экран полученного результата}

**End.**

(\* Последнее предложение программы \*)

Рис. 1.1. Текст Паскаль - программы

1. Осуществить компиляцию программы командой Compile/ Compile. При получении сообщений об ошибках (приложение 1) отредактировать текст программы. Повторить компиляцию: <Alt>+<F9>.
2. Запустить выполнение программы командой Run/Run (<Ctrl>+<F9>).. При получении новых сообщений об ошибках (приложение 2) вновь внести изменение в текст программы, повторить пункты 2 и 3.
3. При выполнении программы следует ввести два числа (значения длин сторон прямоугольника).
4. Для просмотра результатов выполнения программы предназначена команда Debug/Output (<Alt>+<F5>). Для возврата в окно системы Turbo Pascal достаточно нажать любую клавишу.
5. Выполнить программу по шагам с помощью команд Run/Step over (затем <F8>).
6. Сохранить программу по команде File/Save (<F2>).
7. Организовать поиск переменной "**а"** командами Search/Find с фиксацией Whole words only (искать как слово). Командами Search/Find/Search again продолжить поиск. Найти фрагмент текста "**а"** командами Search/Find, очистив поле Whole words only.
8. Завершить работу с Turbo Pascal (File/Exit).
9. Загрузить Turbo Pascal, загрузить свою программу и любую другую (или копию своей программы) из текущей папки.
10. Последовательно выполнить команды, размещая окна каскадом (Window/Cascade), без перекрытия (Window/Tile), списком (Window/List).
11. Установить расположение окон вертикально вдоль экрана (слева и справа), используя команды Window/Size/Move. Для изменения размеров экрана применять клавишу <Shift> одновременно с клавишами: ←, ↑, →, ↓, для перемещения окна по полю экрана - клавиши со стрелками. Завершить работу следует нажатием клавиши <Enter>.
12. Увеличить размеры окон до всего поля экрана: Window/Zoom.
13. Завершить работу с Turbo Pascal (File/Exit).

# Система программирования Turbo Pascal

# Экран и меню

После загрузки системы Turbo Pascal, на экране появится главное меню и информация о данной версии. После нажатия любой клавиши эта информация исчезает. На рисунках 1.2 и 1.3 представлены окна программы для английского и русского вариантов Turbo Pascal.

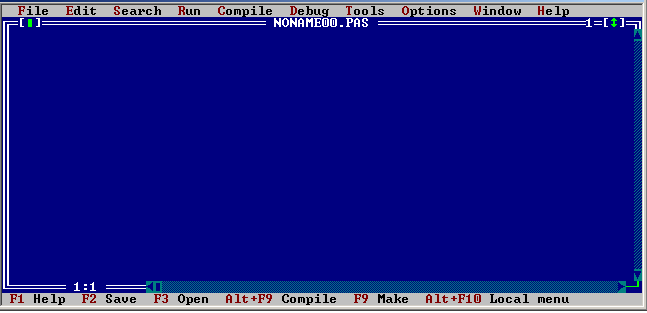


Рис. 1.2. Окно программы (английский вариант)

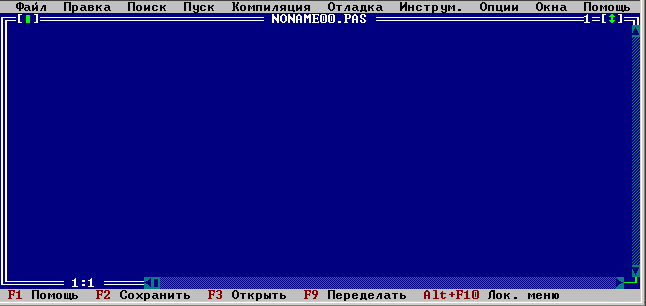


Рис. 1.3. Окно программы (русский вариант)

Поле экрана разделено на части:

* главное меню с командами Паскаля,
* окно редактора (с именем файла в заголовке) для ввода текста программы,
* указатель координат курсора (номер строки и позиция символа в строке),
* нижняя строка, показывающая, какие клавиши в данный момент активны и их назначение.

Для ориентации в интегрированной среде используйте следующую информацию:

Для выбора команд из меню пользуйтесь мышью, выделенными заглавными буквами или клавишами со стрелками.

Для выхода из меню нажимайте <Esc>.

Для перемещений между меню и активным окном нажимайте <F10>.

Если вы находитесь в главном или дополнительном меню, то для перемещения в окно, которое было ранее активно, нажмите <Esc>.

# Команды главного меню

В текущем разделе будут рассмотрены отдельные команды главного меню и подменю (английский вариант), без которых невозможно обойтись при выполнении достаточно простых программ.

**File** – команда взаимодействия с файлами.

New - открыть новое окно.

Open - загрузить существующий файл. Для определения списка файлов, один из которых вы хотите загрузить, можно использовать собирательные символы \* и ?, а также указывать дисковод, с которого должна производиться загрузка. При указании несуществующего дисковода 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 NONAME00, 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 вас спросят, надо ли его заменить.

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 (или Output), либо сочетание клавиш <Alt> + <F5>.

Для того чтобы прервать выполнение программы, например, в случае зависания или зацикливания, следует нажать комбинацию клавиш <Сtrl>+<Вгеаk>. 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.

**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, нажмите 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 (Меmоrу) 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 вашей программы, за текущим значением которых вы собираетесь наблюдать в режиме отладки. Наблюдаемые выражения вычисляются каждый раз заново, когда вы запускаете 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.

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 (Backround) всех элементов интегрированной среды Turbo Pascal.

**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 ←, ↑, →, ↓. Для перемещения окна по полю экрана следует воспользоваться клавишами ←, ↑, →, ↓. Завершить работу следует нажатием клавиши <Enter>.

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, в котором выделено слово, у которого совпадает с требуемым максимальное число начальных символов.

# Строка состояния

Какое бы окно или пункт меню вы не открыли, полоса, располагающаяся внизу, позволит получить справочную информацию о функциональных клавишах. Содержание нижней полосы меняется в зависимости от того, в каком из подменю вы находитесь.

Для главного меню или окна редактора содержимое строки 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 файла, последнего из загруженных в редактор. Turbo Pascal проверяет все файлы, от которых зависит компилируемый.

<Alt> + <FIO> - 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 команды, описанные выше: Cut, Copy, Paste, Clear. А так же:

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, в котором выделено 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.

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 статуса редактора (рис. 1.4). 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).

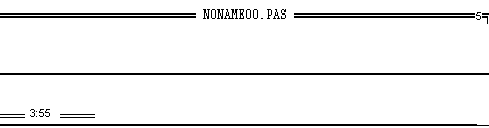


Рис. 1.4 Окно редактора

Если курсор на экране представлен как "**\_**" (символ подчеркивания), это означает, что включен режим вставки текста, если курсор изображается как "" (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.

* 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.
* Не следует в одной строке объединять несколько сложных или больших операторов, так как минимальный выполняемый блок команд в процессе отладки соответствует одной строке текста. Это осложняет поиск ошибок.
* Рекомендуется последовательность операторов, выполняющих какое-то законченное действие, отделять от предыдущих и последующих операторов пустыми строчками.
* 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.
* Рекомендуется использовать комментарии для пояснения отдельных разделов программы или операторов. Комментарий заключается в фигурные скобки или открывается символами (\* , закрывается \*). Например:

(\* Вывод результатов на экран \*) или

{ Группа 14 - ИЭ, студент Петровский Ю.А., вариант 7 }

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

**Алфавит Turbo Pascal**

Латинские буквы: a-z, A-Z

Цифры: 0-9

Знаки препинания: . , : ; ‘ \_

Скобки: ( ) { } [ ]

Знаки арифметических операций: + - \* /

Операции отношения: < > = <= >= <>

P.S. В символьных константах и комментариях можно использовать любой знак (символ) на клавиатуре.

**Структура программы**

**PROGRAM** <имя\_программы>; – заголовок

**VAR**

Описание объектов программы

**BEGIN**

Исполняемая часть – операторы языка, описывающий алгоритм обработки данных

**END.**

В Паскаль - программе описываются данные (в частности – переменные) и действия над ними. **Переменная** – именуемое данное, значение которого в ходе выполнения программы может быть изменено. Все переменные, используемые в программе, должны быть описаны в предложении VAR. Описать переменную – это значит указать её имя (идентификатор) и множество возможных значений переменной (тип). Имя состоит из латинских букв, цифр, знака подчеркивания (\_). Первый символ имени (возможно, единственный) – буква. Для Паскаля в идентификаторах нет разницы между заглавными и строчными буквами. SUMMA, Summa, summa – это одно имя.

**Тип переменной** – это множество возможных значений. Он может указываться в предложении Var своим именем. Существуют стандартные типы данных, для них определены соответствующие имена. Например, множество целых чисел в диапазоне от -32 768 до 32 367 имеет имя **integer**, а множество вещественных чисел (от -1.7\*1038 до 1.7\*1038) – имя **real**. Поэтому предложение

Var N: integer; S: real;

означает, что значениями переменной N могут быть целые числа, значением переменной S – вещественные числа из указанных диапазонов.

# Контрольные вопросы и задания

* 1. Как загрузить Turbo Pascal? Как завершить работу с Turbo Pascal?
  2. Как выделить фрагмент программы? Какие действия редактора предусмотрены для выделенного фрагмента?
  3. Как сохранить текст программы на диске? Как сохранить тот же текст программы с новым именем? Как сохранить тот же текст на флеш-накопителе?
  4. Какие варианты просмотра результатов на экране предусмотрены в языке Turbo Pascal?
  5. В чем назначение комментария?
  6. Как запустить программу для исполнения?
  7. Какие средства Turbo Pascal предназначены для отладки программ?
  8. Перечислите этапы решения задачи на компьютере.

# Лабораторная работа №2

**Разработка линейной программы**

Цель работы: получение первичных навыков программирования (разработки простейших программ линейной структуры).

# Задание по лабораторной работе

Составить и отладить программу, обеспечивающую решение задачи, формулировка которой приведена ниже (вариант указывается преподавателем). При тестировании программы организовать минимум трехкратное исполнение её для разных исходных данных.

# Варианты задач

1. По заданным коэффициентам уравнения прямой линии на плоскости (Ах + By + С = 0) должен вычисляться угол в градусной мере, образуемый этой прямой с положительным направлением оси ОХ (полагается, что A ≠О, В ≠ 0).

*2.* По заданным коэффициентам уравнения прямой линии на плоскости (Ах + By + С = 0) и координатам некоторой точки Р (xо, уо) должна вычисляться абсолютная величина отклонения точки от прямой.

3. По заданным координатам двух точек на плоскости Р1 (x1, y1) и Р2 (х2, у2) должно вычисляться расстояние между ними.

4. По заданным координатам точки Р (хо, уо) должно вычисляться расстояние точки от начала координат.

5. По заданным коэффициентам уравнений двух непараллельных прямых (A1x + B1y + C1 = О, А2х + В2у + С2 = 0) должны вычисляться координаты точки пересечения этих прямых.

6. По заданным координатам вершин треугольника Р1 (x1, y1), Р2 (х2, у2) и Р3 (х3, у3) должна вычисляться его площадь.

7. По заданным координатам вершин треугольника Р1 (x1, y1), Р2 (х2, у2) и Р3 (х3, у3) должен вычисляться его периметр.

8. По заданным диаметрам D1 и D2 двух окружностей с общим центром должна вычисляться площадь образуемого ими кольца.

9. По заданным координатам центра Р0 (хо, уо) и некоторой точки окружности Р1 (x1, y1) должны вычисляться длина окружности и площадь круга, образованного ею.

10. По заданным коэффициентам уравнения прямой линии на плоскости (Ах + By + С = 0) должны вычисляться площадь и периметр прямоугольного треугольника, образованного отрезком этой линии и полуосями координат, исходящими из их начала (полагается A ≠О, В ≠ 0, С ≠ 0).

11. По заданным коэффициентам уравнения прямой на плоскости (Ах + By + С = 0) должно вычисляться наименьшее расстояние между началом координат и прямой.

12. По заданным радиусу окружности с центром в начале координат R и координатам некоторой точки Р (хо, уо) должно вычисляться наименьшее расстояние от точки Р до окружности.

13. По заданным стороне А и диагонали D прямоугольника должны вычисляться его периметр и площадь.

14. По заданным основанию А и высоте Н равнобедренного треугольника должны вычисляться его периметр и площадь.

15. По заданным катету А и гипотенузе С прямоугольного треугольника должны вычисляться периметр и площадь треугольника.

16. По заданным декартовым координатам вершин четырехугольника Р1(x1,y1), P2(x2,y2), P3(x3,y3), P4(x4,y4) должен вычисляться его периметр.

# Методические указания по выполнению работы

1. Приступая к заданию, прежде всего, уясните сущность решаемой на компьютере задачи, определите исходные для нее данные, которые должны вводиться в оперативную память с помощью клавиатуры и установите, что должно быть результатом решения задачи. При этом следует уточнить постановку задачи, используя необходимые для ее решения функции, формулы (приложение 3).

2. На основании уточненной постановки задачи определите, какие операции, над какими данными, в какой последовательности необходимо выполнять для ее решения, т.е. разработайте алгоритм задачи. Алгоритм предложенной в данной работе задачи должен иметь линейную структуру.

3. Далее алгоритм решения задачи должен быть описан в виде программы на языке программирования Турбо-Паскаль. Для программирования в настоящей работе достаточно использовать следующие предложения:

- заголовок программы с именем программы (PROGRAM ….);

- комментарий с указанием фамилии и имени студента, шифра учебной группы, номера работы, номера варианта ( {……} );

- описание используемых в программе переменных (VAR);

- начало исполняемой части программы (BEGIN);

- операторы вывода пользователю подсказок, какие исходные данные он должен ввести в компьютер (WRITE или WRITELN);

- операторы ввода исходных данных (READ или READLN);

- операторы присваивания, предусматривающие необходимые вычисления;

- оператор вывода форматированных результатов на экран монитора (WRITE или WRITELN);

- завершающее текст программы предложение (END.).

4. Составив программу, организуйте ввод её в компьютер, подготовку к выполнению (преобразование исходного модуля в загрузочный), отладку и трехкратное (для разных вариантов исходных данных) исполнение машинной программы. При этом используйте соответствующие средства системы Турбо-Паскаль.

В приложении 4 указан список математических функций, которые могут потребоваться для вычислений по программе.

**Теоретическая (справочная 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 - ввод исходных данных, вычисление значений искомых переменных, вывод на экран результатов вычислений - выполняются однократно и последовательно друг за другом вне зависимости от исходных данных (см. рис. 2.1) .

Н

ввод данных

вывод

данных

процесс

(вычисление)

К

Рис. 2.1. Схема алгоритма линейного вычислительного процесса (графические символы схем алгоритмов представлены в приложении 5).

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

В результате выполнения **оператора присваивания** переменная принимает значение определенного выражения.

Примеры операторов присваивания:

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

м := 0.231;

Во всех случаях сначала вычисляется значение выражения, расположенного справа от знака присваивания (**:=**),а затем вычисленное значение присваивается переменной, указанной слева. Для того, чтобы оператор присваивания мог быть выполнен, необходимо, чтобы все переменные, которые входят в выражение, имели заданные ранее значения.

Для ввода данных в оперативную память компьютера и вывода результатов на экран монитора используются **операторы ввода и вывода**.

Операторы ввода (чтения данных с клавиатуры) могут выглядеть, например, так:

Read(a);

Readln (x1, x2,y);

При этом слова **Read** или **Readln** являются именами стандартных процедур ввода, в круглых скобках указывается список переменных, значения которых должны быть введены. Число переменных в списке может быть любым. Если переменных больше одной, то они разделяются запятыми. Рекомендуется использовать операторы, предусматривающие ввод одного значения.

При выполнении оператора ввода переменным присваиваются читаемые в оперативную память значения. Те числа, которые являются исходными данными, надо своевременно набрать на клавиатуре компьютера. Пусть на клавиатуре набрано число 3.6, тогда в результате выполнения оператора Read (a) переменная **а** получит значение 3.6, после чего начнет выполняться следующий оператор программы.

При вводе числовых данных между операторами Read и Readln нет различий.

Оператор вывода состоит из идентификатора (имени стандартной процедуры) **Write** и следующего за ним в круглых скобках списка переменных, выражений и констант, например:

**Write**(x, 2\*x-a, sqrt(x), ' Результат=', y);

Но рекомендуется предусматривать вывод значений переменных, полученных ранее, т.е. не использовать в списке вывода выражения.

В Паскале имеется оператор вывода **Writeln**, который выполняется так же, как Write, но после его выполнения последующий вывод данных будет начинаться с новой строки экрана. Переход на новую строку без вывода значений может быть предписан с помощью оператора Writeln без списка:

Writeln;

Например, последовательность операторов вывода

Writeln (‘x1=’, x1); Writeln (‘x2=‘,x2);

предусматривает вывод значений в следующем виде (если х1=25.78 и х2=-9.41):

x1= 2.5780000000Е 01

x2=-9.4100000000Е 00

Здесь вещественные числа представлены в стандартной (экспоненциальной) форме, в которой в 17-ти позициях размещаются мантисса и порядок числа. Но можно задавать более удобный формат вывода чисел. Для этого в списке вывода следует после имени переменной указывать необходимое количество позиций для размещения числа, а для вещественных чисел также количество позиций для дробной части числа.

Например, по оператору

Writeln ('n= ', n:7);  
значение целочисленной переменной n=-109 будет представлено в виде:

n= -109.

По оператору

Writeln ('x= ', x:7:2);   
значение вещественной переменной х=-109.58 будет представлено в виде:

х=-109.58

При использовании формата вывода предусмотрены определенные правила. 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 при выводе округляется до указанного количества позиций.

**Пример линейной программы**

Программа Р2 предусматривает вычисление площади треугольника (S) по длинам его сторон (а, b, c). Предполагается, что исходные данные могут быть длинами сторон (a>0, b>0, c>0, a<b+c, b<a+c, c <a+b).

|  |
| --- |
| Program P2;  Var a, b, c, p, s : real;  Begin  Writeln (’введите значения длин сторон’);  Write (’a= ’); Readln (a);  Write (’b= ’); Readln (b);  Write (’c= ’); Readln ( c);  p:= 0.5\*(a+b+c);  s:=sqrt(p\*(p-a)\*(p-b)\*(p-c));  Writeln (‘ Площадь треугольника=’, s:7:2);  End. |

В этой программе после описания используемых переменных (их значения должны быть вещественными числами) определены операторы ввода трех исходных значений, операторы присваивания для вычисления промежуточного значения (полупериметра треугольника) и результата, а также оператор вывода этого результата.

# Контрольные вопросы и задания

1. В чем заключается подготовка задачи к решению? Дайте определение понятий "алгоритм", "программа", "язык программирования".
2. В чем особенности линейных алгоритмов и программ?
3. Какие основные процедуры реализуются по линейным алгоритмам на компьютере?
4. Сформулируйте основные правила записи программ на языке Turbo Pascal.
5. Дайте определение понятий "константа", "переменная", "выражение", "указатель функции", "оператор".
6. Как представляются данные в Паскаль - программах?
7. Что понимают под вводом данных в компьютер? Под выводом данных? Как программируются эти процедуры на языке Паскаль в простейших случаях?
8. Что собой представляет "форматированный" вывод?
9. Что собой представляет оператор присваивания? Сформулируйте правила записи арифметических выражений на языке Паскаль.
10. Прокомментируйте смысл составленной Вами программы.

# Лабораторная работа № 3

# Разработка разветвляющей программы

Цель работы: получение практических навыков при алгоритмизации и программировании разветвляющихся процессов обработки данных (использования структур выбора в программах)

# Задание по лабораторной работе

Модифицировать полученную при выполнении предыдущей работы программу таким образом, чтобы при решении задачи на компьютере обеспечивалась при необходимости проверка вводимых исходных данных на допустимость - соответствие области определения, а также (в соответствии с предыдущим вариантом задачи):

1) учитывался случай В = 0 при А ≠ 0 и определялось, проходит ли заданная коэффициентами уравнения прямая через начало координат;

2) определялось, проходит ли прямая, заданная коэффициентами уравнения, между точкой Р и началом координат;

3) определялось, лежат ли точки Р1 и Р2 на прямой, проходящей через начало координат;

4) определялось, в каком квадранте плоскости расположена заданная декартовыми координатами точка Р;

5) определялось, являются ли прямые, заданные коэффициентами уравнений, перпендикулярными;

6) определялось, находится ли треугольник, заданный координатами его вершин, полностью в первом квадранте плоскости;

7) определялось, является ли треугольник, заданный координатами его вершин, равносторонним;

8) определялось, какая из окружностей является внешней, какая внутренней;

9) определялось, находится ли начало декартовых координат внутри окружности, вне ее или на окружности;

10) определялось, находится ли образованный пересечением прямой и осей координат треугольник в первом квадранте плоскости;

11) определялось, параллельна ли прямая, заданная коэффициентами уравнения, одной из осей координат и если параллельна, то какой;

12) определялось, находится ли точка Р внутри окружности, вне окружности или на окружности;

13) определялось, является ли прямоугольник квадратом;

14) определялось, является ли треугольник прямоугольным;

15) определялось, является ли треугольник равнобедренным;

16) определялось, является ли данный четырехугольник ромбом;

Тестирование программы осуществить несколько раз, используя различные варианты исходных данных для получения всех предусмотренных в программе возможных вариантов результатов решения задачи.

# Методические указания по выполнению работы

1. Задание предполагает пересмотр постановки сформулированной для выполнения лабораторной работы №2 задачи. Следует дополнить математическое описание, сформулировать условия, 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 монитора).

2. Алгоритм задачи в новой постановке должен содержать структуры выбора: предусматривать проверки условий, выполнение или невыполнение которых определяет те или иные действия (процедуры), т.е. быть "разветвляющимся".

Проверка условий допустимости введенных исходных данных в случае выполнения их должна приводить к определению искомых результатов задачи. В случае недопустимости введенных данных – к выводу сообщения о некорректности этих данных.

Проверка других условий (по варианту) должна приводить к выводу одного из возможных и предусмотренных алгоритмом сообщений на экране монитора.

3. При разработке разветвляющегося алгоритма (при наглядном представлении его в виде схемы) стремитесь к логической ясности, простоте и структурности его. Для этого рекомендуется использовать:

- структуры выбора, имеющие один "вход" и один "выход" (рис. 3.1), при необходимости вложенные друг в друга (рис. 3.2);

- переменную–признак возможности определения результатов (допустимости исходных данных) для "сведения" проверки нескольких условий этой возможности к проверке одного (см. пример программирования).

4. При описании полученного алгоритма на языке Турбо-Паскаль (составлении разветвляющейся Паскаль-программы) используйте условные операторы (операторы If) и составные операторы в их частях – "then" и "else". Стремитесь при этом к сохранению структурности программы.

При записи операторов If следует помнить, что "условия" выбора одной из ветвей алгоритма представляются в них в виде логических выражений (выражений типа boolean) - как одно или несколько отношений, соединенных знаками логических операций (and, or, not), сами же отношения должны заключаться в круглые скобки.

5. Для получения разветвляющейся Паскаль-программы нет необходимости полностью вводить ее текст в ПК. Можно с помощью редактора системы программирования модифицировать хранящуюся на диске линейную программу решения задачи, дополнив ее соответствующими новыми операторами. Конечно, затем следует откомпилировать новый исходный модуль и получить новый загрузочный модуль (используемую программу).

If <логическое выражение>

then <оператор 1>

else <оператор 2>;

If <логическое выражение>

then <оператор> ;

процедура 1

процедура 2

условие

нет

да

процедура

условие

нет

да

Рис. 3.1 Структура выбора с двумя ветвями и с одной.

**Теоретическая (справочная) часть**

В решении многих задач при выполнении определенных условий должны осуществляться одни процедуры, а при невыполнении их – другие. Такие процессы обработки данных и соответствующие алгоритмы (программы) называют разветвляющимися. Каждая из альтернативных процедур называется при этом ветвью алгоритма. Выбор той или иной ветви осуществляется в зависимости от результата проверки выполнения логического условия. В каждом конкретном случае процесс обработки данных реализуется только по одной из двух ветвей.

В алгоритмах разветвляющихся процессов обработки данных используются алгоритмические структуры, называемые "структурами выбора" (или "альтернативами"). На рис. 3.1 представлена структура выбора с двумя ветвями и с одной. Первая структура является инструкцией "Если проверяемое условие истинно, то выполнить процедуру 1 , иначе – процедуру 2". Вторая структура – инструкция "Если условие истинно, выполнить процедуру".

Для описания структур выбора на языке Турбо-Паскаль используется условный оператор – оператор If, форматы которого также представлены на рис. 3.1. При этом "условие" записывается в виде логического выражения (одного или нескольких отношений, соединенных знаками логических операций), результатом которого при выполнении программы является одно из двух возможных логических значений. При значении true ("условие" истинно) будет выполняться оператор за словом then, при значении false ("условие" ложно) – оператор за словом else (если часть else имеет место). После этого будет выполняться оператор, следующий в программе за условным оператором.

Например, по оператору

If a >= в then max := а else max := в;

если значение переменой **а** не меньше значения переменной **в**, переменной **max** должно присваиваться значение **а**, иначе (если значение **а** меньше значения **в**) – значение **в**. А по оператору

If c > max then max := c;

значение переменной **c** будет присвоено переменной **max** только, если **c** больше **max**. В приведенных примерах логические выражения после слова If представляют собой одиночные отношения двух значений. В логических выражениях, в которых используются несколько отношений, соединенных знаками логических операций, отношения заключаются в круглые скобки (см. учебную литературу по Турбо-Паскалю и пример программирования ниже).

При необходимости, в условном операторе следует записывать составные операторы (группы операторов, заключенные в операторные скобки begin, end). Например, оператор

IF x > = 0 then begin

Y : = sqrt (x);

Writeln (y:7:2);

end

else Writeln (’x <0 !’);

предусматривает при неотрицательном значении переменной х не только вычисление функции у = √⌐х, но и вывод результата вычислений на экран монитора (другие примеры использования составных операторах см. в примере программирования)

В алгоритмах обработки данных часто необходимо вкладывать одну структуру выбора в другую. На рис. 3.2 приведен пример вложенных структур выбора, в котором У1, У2, У3 являются обозначениями проверяемых условий, а Р1, Р2, Р3, Р1.1, Р1.2 - обозначениями процедур, выполняемых при выполнении (невыполнении) соответствующих условий.

**Пример программирования**

На рис. 3.3 приведена схема алгоритма вычисления площади треугольника S по длинам его сторон **а, в, с**, предусматривающего проверку исходных данных на допустимость (длины сторон должны быть положительными числами – **а > 0, в > 0, с > 0** и должны выполняться условия – **а < в + с, в < а + с; с < а + в**). Результатом этой проверки должно быть одно из двух возможных значений логической переменной – признака допустимости аргументов: r = true означает, что исходные данные допустимы для решения задачи, r = false - что они ошибочны. Таким образом, проверка четырех условий "сводится" в итоге к проверке одного, что обеспечивает последовательность и структурность алгоритма. Впрочем, возможны и другие варианты алгоритма этой задачи.

**Р1**

нет

да

У1

У2

Р1.1

Р1.2

да

У3

Р3

нет

да

нет

**Р2**

**Р1**

IF (Y1) THEN

IF (Y2 ) THEN оператор P1.1 процедура P1

ELSE оператор P1.2

ELSE IF(Y3) THEN оператор P3; процедура P2

Рис. 3.2. Вложенные структуры выбора

да

да

H

ввод

a, b,c

r=false

r=true

S

Усл 1

Усл 2

r=true

вывод S

вывод сооб-щения

K

да

нет

нет

нет

Рис. 3.3. Схема алгоритма вычисления площади треугольника с проверкой аргументов на допустимость

На рис. 3.3 используются следующие обозначения:

Усл. 1 - условия а > 0, в > 0, с > 0;

Усл. 2 - условия а < в + с, в < а + с; с < а + в;

"Сообщение" – сообщение об ошибке.

Паскаль-программа, соответствующая данному алгоритму, представлена на рис 3.4.

|  |
| --- |
| Program P3;  Var a, b, c, p, s : real;  r: boolean;  Begin  Writeln (’введите значения длин сторон’);  Write (’a= ’); Readln (a);  Write (’b= ’); Readln (b);  Write (’c= ’); Readln ( c);  r:=false;  If (a>0) and (b>0) and (c>0) then  begin  If (a<(b+c)) and (b<(a+c)) and (c<(a+b)) then r:=true;  end;  If r=true then begin  p:= 0.5\*(a+b+c);  s:=sqrt(p\*(p-a)\*(p-b)\*(p-c));  Writeln (‘Площадь треугольника=’ , s:7:2);  end  else Writeln (‘Ошибка в исходных данных’);  End. |

Рис. 3.4. Текст Паскаль – программы вычисления площади треугольника с проверкой аргументов на допустимость

**Контрольные вопросы и упражнения**

1. Что собой представляют разветвляющиеся алгоритмы, программы?
2. В чем сущность "структуры выбора"?
3. Какие требования предъявляются к разветвляющимся алгоритмам (программам)?
4. Какие средства языка Турбо-Паскаль используются при описании структур выбора (составления разветвляющихся программ)?
5. Что такое составной оператор, в каких случаях он необходим?
6. Прокомментируйте разработанные Вами алгоритм и программу (раскройте их смысл).

**Лабораторная работа № 4**

**Разработка циклической программы**

Цель работы: получение практических навыков при алгоритмизации и программировании циклических процессов обработки данных (использования структур повторения в программах).

**Задание по лабораторной работе**

Модифицировать полученную при выполнении работы № 3 программу таким образом, чтобы автоматически осуществлялись многократные решения задачи при изменении одного из исходных данных в заданном диапазоне (исходная переменная, значение которой должно изменяться, определяется студентом самостоятельно). Результаты решения должны выводиться для каждого значения изменяемого аргумента – в виде таблицы (форма таблицы согласовывается с преподавателем).

**Методические указания по выполнению работы**

1. Новый вариант программы должен обеспечивать циклический вычислительный процесс – многократные решения ранее сформулированной задачи при различных вариантах исходных данных. При этом в задании оговаривается, что изменение исходных данных для очередного определения искомых результатов (выполнения "цикла" из соответствующих операций) сводится к изменению лишь одного из них. Этот аргумент (обозначим его х) должен последовательно принимать значения в определенном диапазоне (хо – начальное значение, хк – конечное значение), каждое из которых больше предыдущего на постоянную величину (h). Эта величина h называется шагом изменения аргумента. Таким образом, вычисления должны предусматриваться для следующих значений:

х0, х1 = х0 + h, х2 = х1 + h, …, хi = хi-1 + h, …, хk = хk-1+ h

2. Алгоритм циклического процесса также должен иметь циклический характер, т.е. содержать структуру повторения, предусматривающую неоднократные выполнения определенной процедуры (цикла), пока истинно условие выполнения ее.

Для программируемой задачи следует использовать структуру повторения, изображенную на рис. 4.2, где под процедурой Р понимается определение и вывод результатов вычислений при очередном значении изменяющегося аргумента х (для первого вычисления должно использоваться начальное значение х = х0, а для каждого следующего – значение, больше предыдущего на установленный шаг h).

3. С учетом вышеизложенных решений по алгоритмизации задачи исходными данными, которые следует вводить в качестве значений соответствующих переменных, должны быть – начальное и конечное значения (х0, хк) изменяемого аргумента, а также неизменяемые значения остальных исходных величин задачи. Значение шага изменения x (h) может вычисляться для определенного количества значений x. Например, для 10-и значений x- h=(xk-x0)/9.

4. Для описание структуры повторения в Паскаль-программе следует использовать оператор While, в котором условием выполнения цикла в очередной раз является условие "непревышения" значения изменяемого аргумента задачи установленного конечного значения его (х ≤ хк).

**Теоретическая (справочная) часть**

При решении многих задач на компьютере отельные процедуры обработки данных должны многократно повторяться, при этом всякий раз должны использоваться новые значения исходных данных. Такие процессы называют циклическими, а повторяемые процедуры – 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 ограничения числа повторения выделяют циклы с заданным числом повторений и циклы, число повторений которых определяется в процессе обработки данных.

В алгоритмах (программах) циклы предусматриваются соответствующими **структурами повторения.**  Структура повторения в общем случае (рис. 4.1) предусматривает выполнение определенной процедуры, пока истинно условие ее выполнения (сначала в первый, а затем в очередной раз).

|  |  |
| --- | --- |
| да  Усло-вие  Процедура  нет | Повторять процедуру,  пока истинно условие  ее выполнения |
| **While <логическое выражение> do**  **<оператор>;** |

Рис. 4.1. Структура повторения ("повторять пока") и соответствующий оператор цикла ("цикл с предусловием")

В Паскаль-программах такая структура записывается в виде оператора цикла While, в котором условие выполнения процедуры (цикла) представляется в виде логического выражения (за словом While), а инструкцией выполнения процедуры в очередной раз является оператор, следующий после него за словом do (рис. 4.1). Предусмотренный в операторе цикла While оператор будет выполняться, если логическое выражение дает значение true. При значении логического выражения false выполнение оператора While завершается (имеет место "выход из цикла"). Например, по операторам (в них ***х*** – вещественная переменная):

Readln (x);

While x < 0 do Readln (x);

при чтении с клавиатуры отрицательных чисел процедура ввода будет повторяться и только при вводе неотрицательного числа исполнение процедуры Readln прекратится.

На рис. 4.2. представлена структура повторения, в которой используется особая исходная для вычислений переменная х, значение которой изменяется для каждого выполнения цикла от начального х0до конечного хk значения (увеличивается для следующего выполнения цикла на шаг h).

|  |  |
| --- | --- |
| нет  x=x0  Процедура Р  x=x+h  x<=xk  да | Задание начального значения исходной переменной цикла |
| Проверка условия выполнения цикла |
| "Тело" цикла |
| Изменение значения исходной переменной цикла |

Рис. 4.2. Структура повторения с изменяющимся значением исходной переменной

Такая структура на языке Турбо-Паскаль, если переменная ***х*** является вещественной, записывается следующим образом:

. . .

Var x, h, x0, xk : real;

. . .

х : = x0;

While х < = хk do begin

< оператор >; {выполнение процедуры Р }

х : = х + h;

еnd;

Здесь x0, хk – выражения вещественного типа, задающие начальное и конечное значения переменной **х**соответственно, h –шаг изменения x, <оператор> - оператор (в том числе составной), предусматривающий выполнение процедуры Р (необходимые вычисления и при необходимости вывод результатов их) при очередном выполнении цикла.

**Пример программирования**

На рис. 4.3 приведена схема алгоритма вычислений площади треугольника по длинам его сторон при изменении одной из длин (с) в задаваемом пользователем диапазоне (от c0 до ck) с вычисляемым шагом (ch).

Алгоритм предусматривает:

1. ввод двух неизменяемых при вычислениях длин сторон треугольника (а, в), начального и конечного значений длины третьей стороны как значений переменных (c0, ck);
2. проверку исходных данных на допустимость;
3. вычисление шага изменения длины стороны c (для 17-ти значений её);

4) структуру повторения (цикл): определение и вывода результата решения задачи для очередного значения длины с, изменяемой в диапазоне [с0, ск] с шагом ch.

Последние две процедуры предусматриваются при допустимых значениях исходных данных.

Программа, в которой описан этот алгоритм, приведена на рис.4.4.

В ней конкретизированы инструкции:

проверки данных на допустимость (по – прежнему используется логическая переменная – признак r);

определения и вывода результата для очередного значения с (при допустимом значении с – вычисление и вывод площади треугольника s, при недопустимом значении – вывод соответствующего сообщения).

нет

нет

да

да

Определение и вывод результата

ch=(ck-co)/16

c=c0

Н

К

Вывод сообщения

Ввод

a, b, c0, ck

c≤ck

Допу-стимы

c=c+ch

Рис. 4.3. Схема циклического алгоритма вычисления площади треугольника

|  |
| --- |
| Program P4;  Var a, b, c, c0, ck, ch, p, s : real;  r: boolean;  Begin  {Ввод длин сторон треугольника}  Writeln (’Введите значения длин сторон’ );  Write (’a= ’); Readln (a);  Write (’b= ’); Readln (b);  Write (’Начальное c0 = ’); Readln ( c0);  Write (’Конечное ck = ’); Readln ( ck);  {Проверка исходных данных на допустимость}  r:=false;  If (a>0) and (b>0) and (c0>0) and (ck>0) then  If (a<(b+c0)) and (b<(a+c0)) and (c0<(a+b))  then r:=true; {допустимы}  If r then begin {для допустимых исходных данных}    ch:=(ck-c0)/16; {вычисление шага для с}  c:=c0; {задание начального c}  {Цикл определения и вывода результата для с}  While (c<=ck) do  begin  Write (‘Для с=’ , c:7:2);  If (c<(a+b)) then  begin  {вычисление и вывод площади треугольника}  p:= 0.5\*(a+b+c);  s:=sqrt(p\*(p-a)\*(p-b)\*(p-c));  Writeln (‘ s=’ , s:7:2);  end  else {при c>a+b)}  Writeln (' недопустимая длина: c>= a+b');  c:=c+ch;  end  end  else { при ошибке в исходных данных }  Writeln (‘Ошибка в исходных данных’);  end. |

Рис. 4.4. Текст Паскаль - программы

**Контрольные вопросы и задания**

1) Что понимают под циклическими процессами при обработке данных на компьютере? В чем сущность их алгоритмов?

2) Что собой представляет алгоритмическая структура повторения? Какие основные процедуры (операции) она предусматривает?

3) Каким образом структура повторения "повторять пока" записывается в Паскаль-программах?

4) Что необходимо предусматривать для правильного исполнения структуры повторения?

5) В чем особенность структуры повторения, используемой в Вашей программе?

6) Обоснуйте необходимость предложений в разработанной Вами программе.

**Лабораторная работа № 5**

**Разработка и использование подпрограмм**

Цель работы: получение практических навыков по программированию с использованием подпрограмм.

**Задание по лабораторной работе**

Разработать подпрограмму определения результатов решения задачи, для которой Вами была написана программа в лабораторной работе № 2 (формулировка задачи может уточняться преподавателем). При этом в подпрограмме не предусматривать ввод исходных для задачи данных (они должны быть входными параметрами подпрограммы), а также вывод результатов (они должны быть выходными параметрами подпрограммы-процедуры или значением подпрограммы-функции). При необходимости в подпрограмме следует предусмотреть проверку аргументов задачи на допустимость.

Для исполнения подпрограммы при различных вариантах исходных данных следует вложить ее текст в описательную часть «основной» программы (модуль Program), содержащей:

- ввод исходных для решения задачи по подпрограмме данных;

- вызов подпрограммы;

- вывод полученных по подпрограмме результатов на экран монитора.

Основная программа должна предусматривать многократные вызовы разработанной подпрограммы.

**Методические указания по выполнению работы**

1. Приступая к лабораторной работе, следует уяснить, что собой представляют подпрограммы (в том числе в Турбо-Паскале), зачем они нужны и каким образом могут использоваться при программировании и обработке данных на компьютере. Важно при этом понять каким образом:

- подпрограммы записываются и вызываются к исполнению;

- передаются исходные данные подпрограмме, а результаты ее исполнения в место вызова подпрограммы.

2. Первым решением при разработке подпрограммы должен быть выбор ее вида. Если результатом решения задачи является одно одиночное значение (например, число), подпрограмма может быть подпрограммой-функцией («функцией»), в противном случае она должна быть подпрограммой–процедурой («процедурой»).

Затем следует определиться с заголовком подпрограммы, в т.ч. с входными параметрами и, если это «процедура», выходными параметрами подпрограммы (их именами и типами), а также, если подпрограмма является «функцией», типом получаемого по подпрограмме значения («типом функции»). Входные параметры подпрограммы (аргументы решаемой по подпрограмме задачи) должны быть параметрами-значениями. Выходные параметры в «процедурах» (результаты исполнения подпрограммы) – параметрами-переменными (они описываются в заголовке подпрограммы-процедуры в отличие от параметров-значений со словом Var).

3. В данной работе подпрограмма должна быть «внутренней», т.е. составлять с модулем Program единый текст, который должен размещаться в описательной части модуля. Для вызова подпрограммы к исполнению в исполняемой части модуля Program («основной» программе) - следует использовать соответствующие виду подпрограммы средства Турбо-Паскаля – для «процедур» отдельные операторы вызова, начинающиеся именем подпрограммы, а для «функций» - указатели функций в операторах присваивания. При этом рекомендуется использовать для фактических параметров подпрограмм, которые указываются при их вызовах, имена, отличающиеся от имен формальных параметров (имен параметров, используемых в заголовках подпрограмм).

4. Поскольку задание по лабораторной работе предусматривает неоднократные исполнения разработанной подпрограммы, в основной программе следует записать структуру повторения ввода аргументов задачи, вызова подпрограммы и вывода результатов ее очередного исполнения.

**Теоретическая (справочная) часть**

**Подпрограмма** – записанная по определенным правилам часть программы, которая предусматривает выполнение определенной процедуры и может вызываться к исполнению в разных местах программы для разных исходных данных. Программирование с использованием подпрограмм позволяет, с одной стороны, не копировать тексты одних и тех же алгоритмов в программе, с другой – структурировать программы, делая их легко читаемыми, удобными для тестирования, отладки и модификации. Например, в программе можно выделить подпрограммы ввода и проверки аргументов задачи, определения результатов, вывода результатов.

В Турбо-Паскале имеется широкий набор стандартных подпрограмм, которые программист может использовать, предусматривая их вызовы в своей программе. Однако можно разрабатывать и «собственные» подпрограммы.

На рис. 5.1 представлены структуры подпрограмм Турбо-Паскаля – подпрограммы-процедуры («процедуры») и подпрограммы-функции («функции»). В их заголовках указываются имена подпрограмм («процедур» и «функций»), которые используются для вызова подпрограмм к исполнению (они представляют собой допустимые в Турбо-Паскале идентификаторы). За именами подпрограмм в скобках записываются формальные параметры:

- для «процедуры» имена аргументов и результатов исполнения подпрограмм,

- для «функций» имена аргументов подпрограммы, которые используются в подпрограмме, с указанием их типов (для указания типов используются их имена).

В заголовке «функции» указывается также ее тип – имя типа получаемого по подпрограмме значения (функция это процедура, дающая в качестве результата одно одиночное значение).

|  |
| --- |
| **Подпрограмма-процедура** |
| Procedure <имя процедуры> (<формальные параметры>);  <Описание локальных объектов>  Begin  <Операторы>  End; |
| **Подпрограмма-функция** |
| Function <имя функции> (<формальные параметры>): <тип функции>;  <Описание локальных объектов>  Begin  <Операторы>  <имя функции>:= <итоговое значение>;  End; |

Рис. 5.1. Структуры подпрограмм Турбо-Паскаля

Например, заголовок процедуры

Procedure P1 (a, в, с: real, Var S: real);

является заголовком подпрограммы, которая предусматривает вычисление значения переменной S (вещественного числа) для трех исходных вещественных значений, являющихся значениями переменных а, в, с. Заголовок функции

Function F1 (a, в, с: real): real;

является заголовком подпрограммы для аналогичной процедуры, только результат по ней получается не как значение определенной переменной, а как «значение функции» с именем F1.

Приводя примеры заголовков подпрограмм, следует уточнить, что формальные аргументы в них описаны как «параметры-значения», а формальный результат (S) в заголовке подпрограммы-процедуры – как «параметр-переменная», что определяется словом Var. Именно параметрами-переменными и должны быть формальные параметры, являющиеся результатами процедур (сведения о формальных параметрах см. в учебной литературе).

После заголовка подпрограммы при необходимости описываются типы данных, константы и переменные, используемые в подпрограмме и имеющие смысл только внутри ее. В исполняемой части записываются операторы, описывающие алгоритм подпрограммы. В подпрограмме-функции получаемое по ней значение в итоге должно стать «значением функции», т.е. значением переменной, имя которой совпадает с именем подпрограммы в заголовке ее (см. примеры подпрограмм, приведенные после «теоретической (справочной) части»).

Для вызова подпрограммы-процедуры используется отдельный оператор следующего формата:

<имя процедуры> (<фактические параметры>);

По нему предусматривается исполнение подпрограммы для указанных фактических параметров. Последние должны соответствовать списку формальных параметров – по последовательности записи и типам. Например, оператор

Р1 (х, у, z, r);

(при Var, x, y, z, r: real;) является вызовом подпрограммы-процедуры Р1 для соответствующих фактических параметров, т.е. при исполнении подпрограммы а = х, в = у, с = z и в итоге r = s.

Для вызова подпрограммы-функции используется указатель функции (например, в операторе присваивания) следующего формата:

<имя функции> (<фактические аргументы>)

Например, в операторе

r :=F1(x, y, z);

(при Var x, y, z, r: real;) предусмотрен вызов подпрограммы-функции F1 для аргументов, являющихся значениями трех переменных х, у, z, c присваиванием результирующего значения переменной r.

**Примеры программирования**

На рис. 5.2 приведены подпрограммы вычисления площади треугольника по длинам его сторон – подпрограмма-процедура PSqT и подпрограмма-функция FSqT.

|  |
| --- |
| Procedure PSqT (a, b, c: real; Var S: real);  Var p: real;  Begin  s:= -1;  If (a>0) and (b>0) and (c>0) then  If (a<(b+c)) then  If (b<(a+c)) then  If (c<(a+b)) then  begin  p:= 0.5\*(a+b+c);  s:=sqrt(p\*(p-a)\*(p-b)\*(p-c));  end;  End; |
| Function FSqT (a, b, c: real): real;  Var p, s: real;  Begin  s:= -1;  If (a>0) and (b>0) and (c>0) then  If (a<(b+c)) then  If (b<(a+c)) then  If (c<(a+b)) then  begin  p:= 0.5\*(a+b+c);  s:=sqrt(p\*(p-a)\*(p-b)\*(p-c));  end;  FSqT:=s;  End; |

Рис. 5.2 Подпрограмма-процедура PSqT и подпрограмма-функция FSqT вычисления площади треугольника

Обе подпрограммы предусматривают в случае недопустимости аргументов получение в качестве результата отрицательного числа (-1), что является признаком данной ситуации. В подпрограмме-процедуре признак недопустимости аргументов можно определить в качестве отдельного выходного параметра с именем r (рис.5.3).

|  |
| --- |
| Procedure PSqT (a, b, c: real; Var S: real; Var r: boolean);  Var p: real;  Begin  s:= -1; r:=false {ошибка}  If (a>0) and (b>0) and (c>0) then  If (a<(b+c)) then  If (b<(a+c)) then  If (c<(a+b)) then  begin  p:= 0.5\*(a+b+c);  s:=sqrt(p\*(p-a)\*(p-b)\*(p-c));  r:=true; {ошибки нет}  end;  End; |

Рис. 5.3 Второй вариант подпрограммы-процедуры PSqT

На рис. 5.4 и 5.5 представлены программы, в которых используются эти подпрограммы (первые две). В них предусматривается многократное выполнение подпрограмм с помощью оператора While. По нему цикл ввода исходных данных, вызова подпрограммы к исполнению и вывода полученного результата выполняется, пока значение логической переменной enddata равно false (это значение присваивается переменной до первого исполнения цикла). Оно является таковым, если пользователь в качестве первого аргумента задачи набирает на клавиатуре число. Если же вместо числа одновременно нажимаются клавиши <Ctrl> и <Z>, что является признаком окончания вводимых с клавиатуры данных, значение стандартной функции Eof (до этого равное false) становится равным true и тогда остальные аргументы не вводятся, подпрограмма не вызывается, а значение переменной enddata становится равным true (´конец данных´). При enddata = true реализуется выход из цикла.

|  |
| --- |
| Program P5\_1; |
| Procedure PSqT (a, b, c: real; Var S: real);  Var p: real;  Begin  s:= -1;  If (a>0) and (b>0) and (c>0) then  If (a<(b+c)) then  If (b<(a+c)) then  If (c<(a+b)) then  begin  p:= 0.5\*(a+b+c);  s:=sqrt(p\*(p-a)\*(p-b)\*(p-c));  end;  End; |
| Var x, y, z, r: real;  enddate: boolean;  Begin  enddate:= false;  While not enddate do  begin  Writeln (’введите значения длин сторон);  Read (x);  If not Eof then  begin  Read (y); Read (z);  PSqT (x, y, z, r);  Write ( ‘ Значения сторон =’, x:6:1, y:6:1, z:6:1);  If r = -1 then Writeln (‘ Недопустимые данные’)  else Writeln (‘ Площадь:’, r:7:2);  end  else enddate:= true;  end;  End. |

Рис. 5.4. Программа с подпрограммой-процедурой вычисления площади треугольника

|  |
| --- |
| Program P5\_2; |
| Function FSqT (a, b, c: real): real;  Var p, s: real;  Begin  s:= -1;  If (a>0) and (b>0) and (c>0) then  If (a<(b+c)) then  If (b<(a+c)) then  If (c<(a+b)) then  begin  p:= 0.5\*(a+b+c);  s:=sqrt(p\*(p-a)\*(p-b)\*(p-c));  end;  FSqT:=s;  End; |
| Var x, y, z, r: real; enddate: boolean;  Begin  enddate:= false;  While not enddate do  begin  Writeln (’введите значения длин сторон);  Read (x);  If not Eof then  begin  Read (y); Read (z);  r:=FSqT (x, y, z);  Write ( ‘ Значения сторон =’, x:6:1, y:6:1, z:6:1);  if r = -1 then Writeln (‘ Недопустимые данные’)  else Writeln (‘ Площадь:’, r:7:2);  end  else enddate:= true;  end;  End. |

Рис. 5.5. Программа с подпрограммой-функцией вычисления площади треугольника

**Контрольные вопросы и задания**

1) В каких случаях целесообразно использование подпрограмм?

2) Какие подпрограммы возможны в Турбо-Паскале? В чем их отличия?

3) По каким правилам записываются заголовки подпрограмм?

4) Каким образом подпрограммы вызываются к исполнению?

5) Каков механизм передачи данных между подпрограммой и вызывающей ее программной единицей с помощью параметров?

6) В чем различия между параметрами-значениями и параметрами-переменными?

7) Какие требования предъявляются к фактическим параметрам подпрограмм?

8) Если разработанная Вами подпрограмма является «процедурой» («функцией»), напишите «функцию» («процедуру») для аналогичной задачи.

**Лабораторная работа № 6**

**Программирование создания и обработки массивов данных**

Цель работы: получение практических навыков алгоритмизации и программирования процессов создания и обработки одномерных массивов данных, основных процедур с их использованием, а также разработки и использования подпрограмм этих процедур.

**Задание по лабораторной работе**

Разработать подпрограмму обработки одномерного массива целых чисел для задачи:

1) подсчета количества элементов, которые равны наибольшему в массиве значению;

2) определения номеров элементов, равных заданному значению;

3) формирование нового массива, состоящего из чисел, не больше заданного значения;

4) определения, является ли массив упорядоченным по убыванию его элементов;

5) определения, равна ли сумма его элементов заданному значению;

6) определения, каких элементов в массиве больше – положительных или отрицательных;

7) определения номеров его элементов, которые являются четными числами;

8) определения, является ли первый элемент в массиве наименьшим;

9) определения, является ли последний элемент в массиве наибольшим;

10) определения, состоит ли массив из одинаковых элементов;

11) определения номера наибольшего по абсолютному значению элемента;

12) замены в массиве отрицательных элементов на нулевые значения;

13) формирование другого массива, в котором элементы исходного массива, равные заданному значению, заменены на элементы с другим заданным значением;

14) определения, равен ли последний элемент массива сумме предыдущих элементов;

15) определения номеров элементов, которые больше первого элемента;

16) подсчета количества элементов, которые равны наименьшему в массиве значению.

Для исполнения подпрограммы использовать модуль Program, в котором должны предусматриваться:

- создание одномерного массива чисел (ввод чисел в качестве элементов массива с помощью клавиатуры компьютера);

- вызов подпрограммы обработки массива;

- вывод результатов обработки массива на экран монитора.

**Методические указания по выполнению работы**

1. Для выполнения задания, прежде всего, следует уяснить понятие массива данных при обработке их на компьютере, каким образом в программах они описываются и как предусматриваются операции с элементами массивов (см. соответствующую учебную литературу, а также сведения, приведенные после методических указаний по лабораторной работе).

2. Необходимо также изучить синтаксис и семантику особого оператора цикла в Турбо-Паскале – оператора For, описывающего структуру повторения с целочисленной переменной ("управляющей циклом"), значение которой изменяется в заданном диапазоне с шагом 1. Оператор удобно использовать для целого ряда процедур создания и обработки массивов данных. При программировании этих процедур в операторе For в качестве "управляющей циклом" переменной предусматривается переменная, значением которой является индекс элемента массива, определяющий, какой элемент массива должен участвовать в соответствующей операции при очередном выполнении цикла по оператору For.

3. Алгоритм обработки данных в соответствии с заданием должен предусматривать:

1) циклический ввод элементов массива (и других исходных данных, если они имеют место) в оперативную память компьютера;

2) цикл обработки исходного массива чисел для получения требуемых результатов;

3) вывод результатов на экран монитора (если, результатом является массив чисел – цикл вывода его элементов).

4. Структура обработки исходного массива чисел определяется в зависимости от задачи (требуемого для нее результата) и может представлять собой алгоритм процедуры (или комбинации процедур):

1) подсчета количества определенных элементов в массиве;

2) поиска определенного элемента (или определенного отношения элементов) в массиве;

3) формирования нового массива на основе исходного массива;

4) определения наибольшего (наименьшего) элемента в массиве;

5) вычисления суммы элементов числового массива;

В зависимости от варианта алгоритм должен содержать структуры повторения, описываемые операторами While или For.

5. При описании алгоритма обработки массива данных в виде подпрограммы, прежде всего, следует определить вид подпрограммы. Напомним, что, если результатом процедуры является одно одиночное значение (число или логическое значение), подпрограмма может быть подпрограммой-функцией (Function), в противном случае (например, если результатом является массив чисел) - подпрограммой-процедурой (Procedure).

6. Затем необходимо определить и правильно описать формальные параметры подпрограммы. Следует отметить, что если результатом обработки исходного массива чисел должен быть другой одномерный массив, то в заголовке подпрограммы-процедуры необходимо указать два выходных параметра – имя формируемого по подпрограмме массива и имя переменной, значением которой будет количество элементов в этом массиве (выходные параметры описываются как параметры-переменные, т.е. со словом Var).

7. Поскольку одним из входных параметров подпрограммы в соответствии с заданием является массив чисел (а в ряде вариантов другой массив должен быть выходным параметром), тип массива (массивов) должен быть описан в предложении Type до заголовка подпрограммы. В заголовке же подпрограммы тип формального параметра–массива следует указывать именем типа из предложения Type. Напомним также, что тип фактического параметра-массива, для которого исполняется подпрограмма, должен быть таким же.

8. Процедуры ввода с клавиатуры элементов исходного массива и вывода элементов на экран монитора можно предусматривать с помощью оператора For.

**Теоретическая (справочная) часть**

**Массивом данных**  называют поименованную конечную последовательность значений одного и того же типа, одновременно представленную в оперативной памяти компьютера. Значения, составляющие массив, называются его элементами. Они последовательно представляются в полях оперативной памяти и обозначаются в алгоритмах и программах с использованием соответствующих индексов при именах массивов. Индексы являются "координатами" элементов в массивах, их значения определяют местоположение элементов в них. Считается, что **массив данных – это структурное значение нестандартного типа,**  а его элементы – значения индексированных переменных.

**Тип массива данных**  описывается в Паскаль-программе в предложении Туре с помощью конструкции "array". Например, предложение

Туре ta = array [1..20] of integer;

означает, что ta - имя типа одномерного массива целых чисел (чисел типа integer) с индексным диапазоном [1, 20]. Последнее означает, что возможными значениями индекса для элементов массива могут быть числа от 1 до 20 (максимально возможное количество элементов – 20). Тогда описание

Var x,y : ta;

позволяет предусматривать операции над элементами массивов указанного типа с именами **х** и **у**. Например, по оператору

If x[i] > 0 then begin j : = j + 1; y[j] : = x[i]; end;

если i-ый элемент массива **х** является положительным числом, то его значение должно стать значением j-го (очередного) элемента массива **у**. Конечно, имена используемых в операторе индексов элементов (они записываются в квадратных скобках) должны также быть описаны в программе – как целочисленные переменные (Var i, j : integer;).

При  **создании и обработке массивов данных**  предусматриваются процедуры, включающие в себя операции не только со значениями элементов массивов, но и со значениями индексов элементов. Изменения значений индексов необходимы для перехода от использования в операциях одного элемента массива к использованию другого. Так, при i = 1 значением x[i] будет первый элемент массива, а при i = 5 – пятый. Но значением x[1] всегда будет первый элемент (его значения могут быть разными).

Операции с элементами массивов и их индексами могут повторяться для разных элементов, в этом случае часто можно использовать **структуру-повторения с "управляющей переменной"**, (рис. 6.1), аналогичную изображенной на рис. 4.1 (она может представляться и в "свернутом" виде с помощью графического символа "Модификация"). Отличие её в том, что переменная, значение которой должно меняться при повторении цикла (на рис. 6.1 это переменная i) должна быть целочисленного типа, а шаг изменения ее значения должен быть равен 1 (h = 1). Такая структура может описываться в Паскаль-программе не только оператором цикла While, но и **особым оператором цикла Fоr,** в котором указывается диапазон изменения значений управляющей циклом переменной (начальное и конечные значения) и выполняемый оператор при каждом значении из этого диапазона.

Например, оператор

For i : = 1 to 20 do

begin

Write (´ Число:´); Read (x[i]);

end;

предусматривает поочередной ввод с клавиатуры 20-ти целых чисел в качестве элементов массива **х** (при соответствующем описании имен **х** и **i**). В нем в качестве управляющей циклом переменной используется переменная–индекс элемента массива, значение которой должно изменяться от 1 до 20.

|  |  |
| --- | --- |
| нет  i=i0  Процедура Р  i=i+1  i≤ik  да  …. | i=1,n  i=i0, ik  Процедура  …  **For i:=i0 to ik do <оператор>;**  i0 – начальное значение i;  ik – конечное значение i;  i0, ik – выражения целого типа |

Рис.6.1. Структура повторения с целочисленной управляющей переменной

Вышеприведенный пример является инструкциями, которые предусматривают двадцатикратное выполнение (не меньше и не больше) процедуры ввода элемента массива. Для ввода элементов массива, количество которых определяется пользователем программы при ее исполнении, необходимо либо предусматривать ввод этого параметра в качестве значения определенной переменной и указывать эту переменную в качестве конечного значения индекса элемента, либо (что удобнее для пользователя) использовать оператор цикла While и стандартную функцию Eof (см. пояснения к программе, представленной на рис. 5.3). Например, предложения:

…

Type ta = array [1..20] of integer;

Var x: ta; i, n: integer; enddate: boolean;

…

enddate := false; i:=0;

While (not enddate) and (i< 20) do

begin

i:= i+1; Write (´Число:´); Read (x[i]);

if Eof then enddate := true;

end;

n:=i-1;

…

предусматривают ввод чисел в качестве элементов массива **х** в количестве, необходимом пользователю (но не более 20-ти). Количество введенных элементов фиксируется в качестве значения переменной n.

На рис. 6.2 представлены **примеры алгоритмов процедур обработки одномерных числовых массивов данных** (в виде схем) **и их записи на языке Турбо-Паскаль**. При этом считается, что исходный массив **х** состоит из n целых чисел (х1, х2, …, хi,…, xn) и количество элементов, как и сам массив, определено до выполнения процедуры. Результатами же процедур являются:

- при подсчете в массиве положительных элементов значение переменной **m**;

- при формировании массива положительных чисел массив **у**  из **m** элементов;

- при определении наибольшего в массиве числа, результат определяется как значение переменной **хmax**;

- при вычислении суммы элементов массива, результат – значение переменной **s**.

- при поиске положительного элемента в массиве логическое значение **r** (значение переменной типа boobean): если в массиве **х** есть хотя бы один положительный элемент r = true , в противном случае r = false;

В **подпрограммах создания** и **обработки массивов данных** в качестве формальных параметров используются имена соответствующих массивов. При этом в заголовке подпрограммы тип параметра должен указываться явно – с помощью имени типа. Поэтому подпрограмме должно предшествовать предложение Туре с описанием типа массива, являющегося параметром подпрограммы. Например, так:

Туре ta = array [1..20] of integer;

Procedure Par (x : ta; n : integer; …);

Здесь **ta** - имя типа массива, **х** – формальный параметр-массив типа  **ta**, **n** – формальный параметр, значением которого при вызове подпрограммы к исполнению должно стать количество элементов в массиве **х**.

Ниже приведены примеры подпрограмм:

PMaxMin - подпрограмма-процедура определения наибольшего и наименьшего чисел в исходном массиве,

FSum - подпрограмма-функция вычисления суммы элементов числового массива.

Type ta= array [1..100] of integer;

Procedure PMaxMin ( x: ta; n: integer; Var xmax, xmin: integer);

Var i: integer;

Begin

xmax:= x[1]; xmin:= x[1];

For i:=2 to n do

If x[i] > xmax then xmax:= x[i]

else if x[i] < xmin then xmin:= x[i];

End;

Function FSum (x: ta; n: integer): integer;

Var i, s: integer;

Begin

s:=0;

For i:=1 to n do s:=s+x[i];

FSum := s;

End;

1. Определение количества положительных элементов одномерного массива чисел

|  |  |
| --- | --- |
| нет  m=0  m=m+1  xi>0  i=1,m  да | …  m:=0;  For i:=1 to n do  If x[i]> 0 then m:=m+1;  … |

1. Формирование из исходного массива нового массива, состоящего из положительных элементов

|  |  |
| --- | --- |
| нет  m=0  m=m+1  ym=xi  xi>0  i=1,n  да | …  m:=0;  For i:=1 to n do  If x[i]> 0 then  begin  m:=m+1;  y[m]:=x[i];  end;  … |

Рис. 6.2. Примеры алгоритмов процедур обработки числовых массивов

1. Выбор наибольшего элемента в одномерном массиве чисел

|  |  |
| --- | --- |
| да  xm=x1  xm=xi  xi>xm  i=2,n  нет | …  xm:=x[1];  For i:=2 to n do  If x[i]>xm then xm=x[i];  … |

1. Вычисление суммы элементов одномерного массива чисел

|  |  |
| --- | --- |
| s=0  s=s+xi  i=1,n | …  s:=0;  For i:=1 to n do s:=s+x[i];  … |

Рис. 6.2. Примеры алгоритмов процедур обработки числовых массивов (продолжение)

1. Поиск в исходном массиве хотя бы одного положительного числа

|  |  |
| --- | --- |
| нет  нет  да  да  r=false  i=1  r=true  i≤n r=false  xi>0  i=i+1 | …  r:=false;  i:=1;  While(i<=n)and(r=false) do  If x[i]> 0 then r:=true  else i:=i+1;  … |

Рис. 6.2. Примеры алгоритмов процедур обработки числовых массивов (окончание)

**Пример программирования**

На рис. 6.3 представлена программа, которая предусматривает:

1) ввод с помощью клавиатуры аттестационных оценок выпускника вуза по дисциплинам образовательной программы в качестве одномерного массива целых чисел (их значениями могут быть числа 3, 4 или 5).

2) определение возможного статуса диплома выпускника:

- диплом с отличием выдается, если среди оценок нет удовлетворительных ("троек"), а количество хороших оценок ("четверок") не превышает 25% от общего количества оценок.

При этом используется 6 подпрограмм:

- ввода оценок в качестве массиваb(PInput);

- поиска удовлетворительной оценки (FSearch);

- подсчета количества хороших оценок (FСalc);

- определения номеров хороших оценок, если таковые есть и их количество не более 25% (PDetNum), как дополнительный к основному результат задачи для диплома с отличием;

- вывода номеров хороших оценок, если они определялись (PОutput);

- вычисления среднего балла (FMid), если диплом без отличия, как

дополнительный результат задачи в этом случае.

|  |
| --- |
| Program P6;  Type ta= array [1..50] of integer; |
| {Подпрограмма ввода массива целых чисел}  Procedure PInput ( Var x: ta; Var n: integer);  Var i: integer;  Begin  Write (' Количество элементов: '); Readln (n);  For i:=1 to n do  begin Writeln (' Элемент: '); Read (x[i]); end;  End; |
| {Подпрограмма поиска элемента в массиве чисел}  Function FSearch (x: ta; n: integer; a: integer): boolean;  Var i: integer; r: boolean;  Begin  r:=false; i:=1;  While (i<=n) and (r=false) do  If x[i]=a then r:=true  else i:=i+1;  FSearch:=r;  End; |
| {Подпрограмма подсчета количества определенных элементов в массиве чисел}  Function FCalc (x: ta; n: integer; a: integer): integer;  Var i, m: integer;  Begin  m:=0;  For i:=1 to n do If x[i] = a then m:=m+1;  FCalc:=m;  End; |

Рис. 6.3. Программа определения статуса диплома выпускника

|  |
| --- |
| {Подпрограмма вывода массива целых чисел}  Procedure POutput ( x: ta; n: integer);  Var i: integer;  Begin  For i:=1 to n do Write (x[i]: 5); Writeln;  End; |
| {Подпрограмма формирование массива из номеров определенных элементов исходного массива}  Procedure PDetNum ( x: ta; n: integer; a: integer; Var y: ta; Var m: integer);  Var i: integer;  Begin  m:=0;  For i:=1 to n do If x[i]=a then  begin  m:=m+1;  y[m]:=i;  end;  End; |
| {Подпрограмма вычисления среднего арифметического значения}  Function FMid (x: ta; n: integer): real;  Var i, s: integer;  Begin  s:=0;  For i:=1 to n do s:=s+x[i];  FMid:= s/n;  End; |

Рис. 6.3. Программа определения статуса диплома выпускника (продолжение)

|  |
| --- |
| {Описание переменных основной программы}  Var b, num: ta;  b3, more25: boolean;  k, q4, q: integer;  bmid: real;  Begin  {Ввод оценок в массив b из k элементов}  PInput (b,k);  {Вывод оценок}  Writeln (' Оценки студента: ');  POutput (b, k);  {Поиск удовлетворительной оценки}  b3:= Fsearch (b, k, 3);  if not b3 then {удовлетворительных оценок нет}  begin {подсчет количества хороших оценок}  q4:= FCalc (b, k, 4);  if q4<= 0.25\*k then  {Количество хороших оценок не более 25%}  begin  Writeln (' Диплом с отличием ');  more25:=false  if q4= 0 then Writeln (' Все оценки - отличные ')  else Writeln (' Хороших оценок: ', q4:2 );  {Определение номеров хороших оценок}  PDetNum (b, k, 4, num, q);  {Вывод номеров хороших оценок}  Writeln (' Номера хороших оценок: ');  POutput (num, q);  end;  end  else {хороших оценок более 25%}  more25:=true;  if (b3 or more25) then  begin  {есть хотя бы одна удовлетворительная оценка}  {или хороших оценок более 25%}  Writeln (' Диплом без отличия ');  {Вычисление среднего балла}  bmid:= Fmid (x, k);  Writeln (' Средний балл: ', bmid:4:2 );  end;  End. |

Рис. 6.3. Программа определения статуса диплома выпускника (окончание)

**Контрольные вопросы и задания**

1) Что собой представляют массивы данных?

2) Каким образом описываются массивы данных в Паскаль-программах?

3) Какова роль индексированных переменных при программировании задач по обработке массивов данных?

4) Каким образом в алгоритмах и программах предусматривается переход к использованию и определению новых элементов массива?

5) Что собой представляют структуры повторения по обработке массивов данных?

6) В каких случая в программах создания и обработки массивов удобно использовать оператор For?

7) Какие ограничения накладываются на использование оператора For для описания циклов?

8) Прокомментируйте составленную Вами программу.

**Лабораторная работа № 7**

**Разработка программы со сложным циклом**

Цель работы: получение практических навыков по использованию вложенных друг в друга структур повторения при алгоритмизации и программировании, по разработке и использованию подпрограмм с такими структурами.

**Задание по лабораторной работе**

Разработать подпрограмму обработки матрицы чисел, представленной в виде двумерного массива, для задачи:

1) определения номеров строк матрицы, последние элементы которых равны сумме предыдущих элементов;

2) определения числа столбцов матрицы, сумма элементов которых равна заданному значению;

3) определения, все ли столбцы матрицы содержат заданное значение;

4) формирование одномерного массива из наибольших в строках матрицы чисел;

5) определения, являются ли суммы элементов столбцов матрицы одинаковыми;

6) определения, во всех ли столбах матрицы первые элементы являются наименьшими;

7) определения номеров столбцов матрицы, не содержащих в качестве элемента заданное значение;

8) формирование одномерного массива из наименьших в столбцах матрицы чисел;

9) подсчета количества строк матрицы, в которых сумма элементов больше заданного значения;

10) определения номеров столбцов матрицы, в которых элементы расположены по возрастанию;

11) определения номера строки матрицы, сумма элементов которой является наибольшей;

12) определения номера столбца матрицы, в котором заданное значение встречается наибольшее число раз;

13) формирования одномерного массива из элементов матрицы, принадлежащих заданному диапазону;

14) подсчета числа столбцов матрицы, в которых элементы располагаются по убыванию;

15) определения, содержит ли матрица хотя бы один столбец, в котором все элементы одинаковы.

16) формирования одномерного массива из элементов матрицы, не больше заданного значения.

**Методические указания по выполнению работы**

1. Поскольку задание предусматривает программирование обработки матрицы чисел, необходимо уяснить форму представления такого структурного значения в оперативной памяти компьютера. Оно представляется в виде двумерного массива чисел, каждый элемент которого обозначается с помощью двух индексов – номера строки и номера столбца матрицы, на «пересечении» которых располагается элемент матрицы (массива). Поэтому при описании типа двумерного массива в программах следует использовать два индексных диапазона – для каждого из индексов. Начальные значения индексов удобно принимать равными 1, конечными – максимально возможные значения числа строк (для первого индекса) и столбцов (для второго индекса) матрицы. В лабораторной работе достаточно использовать следующий тип массива:

Туре ta2 = array [1..10, 1..10] of integer;

Он может использоваться для матриц, в которых максимально возможное число строк (элементов в столбце) и столбцов (элементов в строке) равно 10.

2. Входными параметрами подпрограммы обработки матрицы (двумерного массива) чисел в ее заголовке должны быть – имя массива и имена переменных, значениями которых будут число строк и столбцов матрицы. Возможно, понадобятся и другие входные параметры в соответствии с заданием.

3. Алгоритм обработки матрицы должен представлять собой вложенные друг в друга структуры повторения («сложный цикл»). Во внутреннем цикле следует предусматривать обработку элементов столбца (изменение второго индекса элемента массива) или строки (изменение первого индекса) в зависимости от варианта задания. Во внешнем же – соответственно переход от одной строки к другой (изменении первого индекса элемента массива) или от одного столбца к другому (изменение второго индекса). При этом можно использовать структуры повторения для процедур, приведенных в теоретической (справочной) части описания предыдущей лабораторной работы.

Для сложного цикла важно, чтобы инструкции подготовки первого и каждого последующего выполнения внутреннего цикла находились в «теле» внешнего цикла.

4. Как и в предыдущей лабораторной работе для исполнения подпрограммы следует вложить ее в описательную часть модуля Program, в котором предусматриваются многократные выполнения:

- ввода исходных для подпрограммы данных (фактических параметров);

- вызова подпрограммы;

- вывода полученных при исполнении подпрограммы результатов.

**Теоретическая (справочная) часть**

**Матрица** чисел (прямоугольная таблица чисел – совокупность чисел, упорядоченных по строкам и столбцам таблицы) представляется в оперативной памяти компьютера в виде двумерного числового массива (рис. 7.1). Описание типа такого массива из целых чисел, если максимально возможное количество строк (элементов в столбце) матрицы 20, а максимально возможное количество столбцов (элементов в строке) матрицы 10, может быть следующим:

Туре ta2 = array[1..20, 1..10] of integer;

При этом обозначаются элементы массива с помощью двух индексов (номеров строки и столбца, на «пересечении» которых элемент располагается). Так, для массива

Var x : ta2;

значением x[1,1] будет первый элемент в первой строке, значением x[5, 10] - десятый элемент в 5-ой строке и т.п. Значением x[i, j] может быть любой элемент массива в зависимости от значений его индексов (i - номер строки, j - номер столбца или элемента в строке).

Матрица чисел X

|  |  |  |  |
| --- | --- | --- | --- |
| x= | x1,1 x1,2  … x1,n | | m – количество строк матрицы (элементов в столбце) |
| x2,1 x2,2 … x2,n | |
| … | | n – количество столбцов матрицы (элементов в строке) |
| xm,1 xm,2 … xm,n | |
|  | | xi,j - число, |
|  | | | xi,j  при i=2 и j=1 |  |

Двумерный массив чисел X (при m-20 и n=10)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x1,1 | x1,2 | … | x1,10 | x2,1 | x2,2 | **…** | x2,10 | **…** | x20,1 | x20,2 | **…** | x20,10 |

Поля оперативной памяти компьютера с числами – элементами двумерного массива X

Рис. 7.1. Матрица чисел X и двумерный массив чисел X, элементами которого являются элементы матрицы.

**Обработка двумерного массива** в общем случае предполагает выполнение операций с каждым элементом его. Для этого алгоритмы обработки должны содержать вложенные друг в друга структуры повторения (т.н. «сложные циклы»). В зависимости от порядка использования в операциях элементов массива возможны два варианта:

1) во **внутреннем цикле** осуществляются переходы **от одного элемента строки к другому** за счет изменения второго индекса элемента (номера элемента в строке), **во внешнем** – переходы **от одной строки к другой** за счет изменения первого индекса элемента (номер строки);

2) во **внутреннем цикле** осуществляются **переходы от одного элемента столбца к другому** за счет изменения первого индекса элемента (номера элемента в столбце), **во внешнем** – переходы **от одного столбца к другому** за счет изменения второго индекса элемента (номера столбца).

Например, предложения Турбо-Паскаля:

…

Type ta2 = array [1..20, 1..20] of integer;

ta1 = array [1..20] of integer;

Var x : ta2;

s : ta1;

m, n, i, j : integer;

…

For i: = 1 to m do

begin

s[i]: = 0;

For j: = 1 to n do s[i] : = s[i] + x[i, j];

end;

…

предусматривают первый вариант обработки массива **х** – вычисления сумм элементов для каждой строки его с формированием из них массива **s** (значения **m** и **n**  считаются известными, при этом 1≤ m ≤ 20 и 1 ≤ n≤ 20). Для вычисления сумм элементов столбцов массива нужны другие операторы (второй вариант обработки):

**…**

For j: = 1 to n do begin

s[j]: = 0;

For i: = 1 to m do s[j]: = s[j] + x[i,j];

end;

**…**

Правильное исполнение **сложного цикла** возможно, если инструкции подготовки первого выполнения внутреннего цикла входят в «тело» внешнего цикла. В вышеприведенных примерах это обеспечивается за счет записи оператора определения начального (нулевого) значения очередной суммы элементов до второго оператора For (до внутреннего цикла), но после первого оператора For (внутри внешнего цикла).

**Пример программирования**

На рис. 7.2 и 7.3 представлены алгоритм определения для исходной матрицы целых чисел номеров строк, все элементы которых принадлежат заданному диапазону, и программа с подпрограммой PMatrix. Входными параметрами подпрограммы являются:

x – исходный двумерный массив (матрица);

m – количество строк матрицы;

n – количество столбцов матрицы;

d1, d2 – левая и правая границы числового диапазона, используемые для определения номеров искомых строк.

Выходными параметрами подпрограммы определены:

num – одномерный массив номеров искомых строк матрицы;

к – количество элементов в сформированном массиве num.

Последний параметр нужен для поэлементной обработки получаемого массива, например, для циклического вывода его элементов на экран монитора.

Внешний цикл в подпрограмме представляет собой алгоритм формирования массива номеров искомых строк с именем num, внутренний – алгоритм поиска в строке элемента, не принадлежащего заданному диапазону [d1, d2] (рис. 7.2). Если в i-ой строке все элементы принадлежат диапазону, значение переменной r остается равным true, если в строке встречается элемент вне диапазона, значение r становится false. Для r = true номер строки (значение индекса i) становится очередным элементом в формируемом массиве num.

Основная программа предусматривает ввод числовой матрицы из 6 – ти строк по 4 элемента в каждой и пятикратное исполнение подпрограммы обработки матрицы PMatrix для разных границ числового диапазона (они должны вводиться пользователем программы). После каждого исполнения подпрограммы предусматривается вывод полученного по ней массива номеров искомых строк (если таковые есть).

нет

нет

да

да

нет

k=0

j=j+1

k=k+1

numk=i

r=true

j=1

r=false

К

Н

r=true j≤n

xij вне диапазона

i=1,m

r=true

да

Рис. 7.2. Схема алгоритма определения номеров строк матрицы, все элементы которых принадлежат заданному диапазону

|  |
| --- |
| Program P7;  Type ta1= array [1..20] of integer;  ta2= array [1..20, 1..20] of integer; |
| Procedure PMatrix (x: ta2; m, n: integer; d1, d2: integer;  Var num: ta1; Var k: integer);  Var i, j: integer; r: boolean;  Begin  {Формирование массива номеров искомых строк}  k:=0;  For i:=1 to m do  begin  {Поиск в строке элемента, не принадлежащего диапазону}  r:= true; j:=1;  While (j<=n) and (r= true) do  If (x[i, j] < d1) or (x[i, j] > d2) then r:= false  else j:=j+1;  If r= true then {все элементы в диапазоне}  begin  k:=k+1; num[k]:=i;  end;  end;  End; |
| Var a: ta2; b: ta1; c, i, j, g1, g2, f: integer; bmid: real;  Begin  Writeln (' Введите элементы матрицы по строкам ');  For i:= 1 to 6 do  For j:= 1 to 4 do read (a[I, j]);    Writeln (' Предусматривается пятикратное решение задачи: ');  For с:= 1 to 5 do  begin  Writeln (' Левая граница диапазона: '); readln (g1);  Writeln (' Правая граница диапазона: '); readln (g2);  PMatrix (a, 6, 4, g1, g2, b, f)  if f=0 then (' Искомых строк нет ')  else  begin  Writeln (' Номера искомых строк ' );  For i:= 1 to f do Write (b[i]:3); Writeln;  end;  end;  End. |

Рис. 7.3. Программа с подпрограммой определения номеров строк матрицы, все элементы которых принадлежат заданному диапазону

**Контрольные вопросы и задания**

1) Что понимают под сложным циклическим процессом?

2) Каковы правила записи вложенных друг в друга структур повторения (сложных циклов) на языке Турбо-Паскаль?

3) Что собой представляют двумерные массивы данных?

4) Каким образом предусматриваются операции над элементами их в алгоритмах и программах?

5) Как изменится Ваша подпрограмма, если результаты обработки матрицы нужно получать не для строк (столбцов), а для столбцов (строк)?

6). Прокомментируйте разработанную Вами программу.

# Приложения

###### Приложение 1

# Сообщения об ошибках периода компиляции

При обнаружении ошибки интегрированная среда Турбо Паскаля автоматически загружает в текстовый редактор исходный файл и помещает курсор около того места, где в исходном тексте обнаружена ошибка. При этом в командной строке редактора появляется диагностическое сообщение. Перевод сообщений приведен ниже:

1. Выход за границы памяти.

2. Не указан идентификатор.

3. Неизвестный идентификатор.

4. Двойной идентификатор.

5. Синтаксическая ошибка.

6. Ошибка в вещественной константе.

7. Ошибка в целой константе.

8. Строковая константа превышает допустимые размеры.

9. Слишком много вложенных файлов.

10. Не найден конец файла.

11. Слишком длинная строка.

12. Здесь нужен идентификатор типа.

13. Слишком много открыто файлов.

14. Неверное имя файла.

15. Файл не найден.

16. Диск заполнен.

17. Неправильная директива компилятора.

18. Слишком много файлов.

19. Неопределенный тип в объявлении указателя.

20. Отсутствует идентификатор переменной.

21. Ошибка в объявлении типа.

22. Слишком большая структура.

23. Базовый тип множества нарушает границы.

24. Компонентами файла не могут быть файлы или объекты.

25. Неверная длина строки.

26. Несоответствие типов.

27. Неправильный базовый тип для типа диапазона.

28. Нижняя граница больше верхней.

29. Нужен порядковый тип.

30. Нужна целая константа.

31. Нужна константа.

32. Нужна целая или вещественная константа.

33. Нужен идентификатор типа.

34. Неправильный тип результата функции.

35. Нужен идентификатор метки.

36. Нужен BEGIN.

37. Нужен END.

38. Нужно выражение типа INTEGER.

39. Нужно выражение порядкового типа.

40. Нужно выражение типа BOOLEAN.

41. Типы операндов не соответствуют типу операции.

42. Ошибка в выражении.

43. Неверное присваивание.

44. Нужен идентификатор поля.

45. Объектный файл слишком большой.

46. Неопределенная внешняя процедура.

47. Неправильная запись объектного файла.

48. Сегмент кода слишком большой.

49. Сегмент данных слишком велик.

50. Нужен оператор DO.

51. Неверное определение PUBLIC.

52. Неправильное определение EXTERNAL.

53. Слишком много определений типа EXTERNAL.

54. Требуется OF.

55. Требуется интерфейсная секция.

56. Недействительная перемещаемая ссылка.

57. Требуется THEN.

58. Требуется ТО или DOWNTO.

59. Неопределенное опережающее описание.

60. Слишком много процедур.

61. Неверное преобразование типа.

62. Деление на нуль.

63. Неверный файловый тип.

64. Нет возможности считать или записать переменные данного типа.

65. Нужно использовать переменную-указатель.

66. Нужна строковая константа.

67. Нужна строковая переменная.

68. Программный модуль не найден.

69. Несоответствие имен программных модулей.

70. Несоответствие версий программных модулей.

71. Повторное имя программного модуля.

72. Ошибка формата файла модуля.

73. Отсутствует исполняемая часть модуля.

74. Типы констант и тип выражения оператора CASE не соответствуют друг другу.

75. Нужна переменная типа запись.

76. Константа нарушает границы.

77. Нужна файловая переменная.

78. Нужно выражение типа указатель.

79. Нужно выражение типа REAL или INTEGER.

80. Метка не находится внутри текущего блока.

81. Метка уже определена.

82. Неопределенная метка в предшествующем разделе операторов.

83. Недействительный аргумент операции @.

84. Нужно служебное слово UNIT.

85. Нужно указать «;».

86. Нужно указать «:».

87. Нужно указать «,».

88. Нужно указать «(».

89. Нужно указать «)».

90. Нужно указать «=».

91. Нужно указать «:=».

92. Нужно указать «[» или «(.».

93. Нужно указать « ]» или «.)».

94. Нужно указать «.».

95. Нужно указать «..».

96. Слишком много переменных.

97. Неправильная управляющая переменная оператора FOR.

98. Нужна переменная целого типа.

99. Здесь не допускаются файлы.

100. Несоответствие длины.

101. Неверный порядок полей.

102. Нужна константа строкового типа.

103. Нужна переменная типа INTEGER или REAL.

104. Нужна переменная порядкового типа.

105. Ошибка в операторе INLINE.

106. Предшествующее выражение должно иметь символьный тип.

107. Слишком много перемещаемых элементов.

108. Недостаточно памяти для выполнения программы.

109. Невозможно найти тип .ЕХЕ.

110. Модуль выполнить нельзя.

111. Компиляция прервана.

112. Константа CASE нарушает допустимые границы.

113. Ошибка в операторе.

114. Невозможно вызвать процедуру прерывания.

115. При компиляции необходимо наличие сопроцессора 8087.

116. Для компиляции необходим режим 8087.

117. Указанный адрес не найден.

118. Здесь не допускаются включаемые файлы.

119. Ошибка формата файла .ТМР.

120. Нужен NIL

121. Неверный идентификатор.

122. Недействительная ссылка на переменную.

123. Слишком много символов.

124. Слишком большой раздел операторов.

125. В модуле нет отладочной информации.

126. Файлы должны передаваться как параметры-переменные.

127. Слишком много условных символов.

128. Пропущена условная директива.

129. Пропущена директива ENDIF.

130. Ошибка в условных определениях.

131. Заголовок не соответствует предыдущему определению.

132. Критическая ошибка диска.

133. Нельзя вычислить данное выражение.

134. Некорректное завершение выражения.

135. Неверный спецификатор формата.

136. Недопустимая косвенная ссылка.

137. Здесь нельзя использовать переменную структурированного типа.

138. Нельзя вычислить выражение без модуля SYSTEM.

139. Нет доступа к данному символу.

140. Недопустимая операция с плавающей запятой.

141. Нельзя выполнить компиляцию оверлеев в память.

142. Должна использоваться переменная процедурного типа.

143. Недопустимая ссылка на процедуру или функцию.

144. Этот модуль не может быть использован в качестве оверлейного.

147. Требуется объектный тип.

149. Требуется виртуальный тип.

151. Виртуальный конструктор не разрешается.

152. Требуется описание конструктора.

153. Требуется описание деструктора ("сборщика мусора").

###### Приложение 2

# Сообщения об ошибках периода выполнения

Ошибки, обнаруженные во время выполнения программы, приводят к появлению на экране сообщения вида:

Run time error nnn at xxxx:yyyy

(Ошибка во время выполнения nnn по адресу xxxx:yyyy)

После появления сообщения программа работу заканчивает.

ОШИБКИ ВВОДА/ВЫВОДА (от 1 до 199), вызывающие завершение программы:

1 - Файл не найден.

3 - Путь не найден.

4 - Открыто слишком много файлов.

5 - Отказано в доступе к файлу.

6 - Недопустимый файловый канал.

12 - Недействительный код доступа к файлу.

15 - Недопустимый номер дисковода.

16 - Нельзя удалить текущий каталог.

17 - Нельзя при переименовании указывать разные дисководы.

100 - Ошибка чтения с диска.

101 - Ошибка записи на диск.

102 - Файлу не присвоено имя.

103 - Файл не открыт.

104 - Файл не открыт для ввода.

105 - Файл не открыт для вывода.

106 - Неверный числовой формат.

150 - Диск защищен от записи.

151 - Неизвестный модуль.

152 - Дисковод в состоянии "не готов".

153 - Неопознанная команда.

154 - Ошибка в исходных данных.

160 - Ошибка при записи на устройство.

161 - Ошибка при чтении с устройства.

ГРУБЫЕ ОШИБКИ, всегда приводящие к остановке программы:

200 - Деление на нуль.

201 - Ошибка при проверке границ.

202 - Переполнение стека.

203 - Переполнение динамической памяти.

204 - Недействительная операция с указателем.

205 - Переполнение при операции с плавающей запятой.

206 - Исчезновение порядка при операции с плавающей запятой.

207 - Недопустимая операция с плавающей запятой.

###### Приложение 3

**Формулы вычислений**

1. Уравнение прямой, проходящей через две точки (x1,y1) и (x2,y2)
2. Расстояние от точки (x0,y0) до прямой Ax+By+C=0 (значение положительно, если начало координат и точка лежат по разные стороны от прямой, и отрицательно, если по одну сторону)
3. Две прямые A1 x+B1 y+C1=0 и A2 x+B2 y+C2=0 пересекаются в точке
4. Прямые параллельны, если
5. Прямые перпендикулярны, если
6. Расстояние между двумя точками (x1,y1) и (x2,y2)
7. Площадь треугольника со сторонами A, B, C

, где

###### Приложение 4

**Стандартные математические функции Турбо Паскаля**

|  |  |
| --- | --- |
| Указатель функции | Значение функции |
| Abs(x) | Модуль аргумента |
| ArcTan(x) | Арктангенс (значение в радианах) |
| Cos(x) | Косинус, угол в радианах |
| Exp(x) | Экспонента |
| Frac(x) | Дробная часть числа |
| Int(x) | Целая часть числа |
| Ln(x) | Логарифм натуральный |
| Pi | 3,141592653 |
| Sin(x) | Синус (угол в радианах) |
| Sqr(x) | Квадрат аргумента |
| Sqrt(x) | Корень квадратный |
| Round(x) | Округление до ближайшего целого |
| Trunc(x) | Отбрасывание дробной части числа |

###### Приложение 5

**Схемы алгоритмов**

При разработке программ алгоритмы обработки данных могут представляться наглядно в виде схем. Схема алгоритма – это изображение инструкций алгоритма в виде "графических символов" с указанием последовательности их исполнения с помощью линий ("линий потока") от одного символа к другому (см. расположенную ниже таблицу 1).

Нормальными направлениями линий потока являются направления сверху вниз и слева направо. Для указания других направлений в линиях потока используются стрелки. Конкретизация инструкций осуществляется с помощью соответствующих записей внутри символов-инструкций (в приведенной таблице 1 это первые пять символов).

Записи внутри графических символов должны быть четкими, без применения предложений языка программирования.

Символ "Решение" – единственный элемент, имеющий два выхода-перехода к соответствующим инструкциям. Рядом с каждым переходом следует указать результат проверки условия - "Да" и "Нет".

|  |  |  |
| --- | --- | --- |
|  | да  нет |  |

Символ "Модификация" имеет входы и выходы со следующими значениями:

|  |  |  |
| --- | --- | --- |
| возврат к следующему повторению инструкций |  | выход после окончания повторения соответствующих инструкций |
|  | переход к повторению инструкций |  |

Рекомендуется изображать схему алгоритма на одной странице, при необходимости "укрупняя" инструкции с последующей детализацией в виде других схем.

Таблица 1. Графические символы схем алгоритмов

| Графический символ | Наименование символа | Инструкция / смысл |
| --- | --- | --- |
|  | "Процесс" | Выполнить операцию или группу операций, в результате которых изменяется значение, форма представления или расположение данных (например, вычисления) |
|  | "Решение" | Проверить условие выбора следующих инструкций |
|  | "Ввод/ вывод" | Ввести исходные данные в оперативную память компьютера или вывести данные |
|  | "Модификация" | Повторить инструкции определенное число раз |
|  | "Предопределенный процесс" | Выполнить процедуру по подпрограмме |
|  | "Линия потока" | Определить переход от одного символа схемы к другому |
|  | "Начало"/ "Окончание" | Обозначение с символом "Н"/"К" всегда является первым/ последним элементом схемы алгоритма |
|  | "Внутристраничный соединитель" | Указание связи между прерванными линиями потока от одного символа схемы к другому. В нём размещаются буквы алфавита (два соответствующих друг другу соединителя имеют одинаковые буквы) или номера символов |
| − − − | "Комментарий" | Используется для символа схемы, который требует пояснения |

**Список учебной литературы**

1. Немнюгин С.А. Turbo Pascal: учеб. – СПб.: Питер, 2000, - 491 с.
2. Павловская Т.А. Паскаль. Программирование на языке высокого уровня: Учебник для вузов – СПб.: Питер, 2007. – 393 с.: ил.
3. Аляев Ю.А., Гладков В.П., Козлов О.А. Практикум по алгоритмизации и программированию на языке Паскаль: учебное пособие – М.: Финансы и статистика, 2004. – 528 с.
4. Аляев Ю.А., Козлов О.А. Алгоритмизация и языки программирования Pascal, C ++, Visual Basic: Учеб.-справ. пособие для курс. воен. учеб. завед. и студ. техн. вузов – М.: Финансы и статистика, 2002. – 319 с.
5. Сухарев М. Turbo Pascal 7.0. Теория и практика программирования. – 2-е изд., перераб. и доп. – СПб: Наука и техника, 2004. 640 с.
6. Фаронов В.В. Турбо-Паскаль: Учеб. пособие. – М: Питер, 2007. – 367 с.
7. Тарануха Н.А., Гринкруг Л.С., Бурменский А.Д., Ильина С.В. Обучение программированию: Язык Pascal. – М.: СОЛОН-ПРЕСС, 2009. – 384 с.
8. Учебно – методическое пособие по лабораторным работам для студентов бакалавриата по направлениям подготовки "Информатика и вычислительная техника" "Прикладная информатика" (часть вторая) – Калининград, ФГБОУ ВПО КГТУ, 2015. – с.

# Интернет-ресурсы (ссылки на учебники и учебно-методические пособия):

http://progbook.net/pascal

http://sprinter.ru/10038

# Оглавление

Введение 4

# Лабораторная работа №1. Освоение системы программирования

# Turbo Pascal 17

# Лабораторная работа №2. Разработка линейной программы 25

Лабораторная работа №3. Разработка разветвляющейся программы 28

# Лабораторная работа №4. Разработка циклической программы 35

# Лабораторная работа №5. Программирование и использование

# подпрограмм 43

# Лабораторная работа №6. Программирование создания и

# обработки массивов данных 53

# Лабораторная работа №7. Разработка программы со сложным 69

# циклом

# Приложения 78

###### Приложение 1. Сообщения об ошибках периода компиляции 78

###### Приложение 2. Сообщения об ошибках периода выполнения 82

###### Приложение 3. Формулы вычислений 83

###### Приложение 4. Стандартные математические функции

###### Турбо Паскаля 84

###### Приложение 5. Схемы алгоритмов 85

# Список учебной литературы 88