Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»

Е.А.ДЕЙ

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ТУРБО-ПАСКАЛЬ И ЧИСЛЕННОЕ РЕШЕНИЕ ФИЗИЧЕСКИХ ЗАДАЧ

Гомель 2005

Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»

Е.А.ДЕЙ

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ТУРБО-ПАСКАЛЬ И ЧИСЛЕННОЕ РЕШЕНИЕ ФИЗИЧЕСКИХ ЗАДАЧ

ПРАКТИЧЕСКОЕ ПОСОБИЕ
для студентов 1 курса
специальности 1-31 04 01 02 – «Физика»

Гомель 2005

УДК 681.3.06(075)
ББК 32.973я7
Д27

Рецензенты:
В.Ф. Шолох, доцент, кандидат физико-математических наук;
кафедра теоретической физики учреждения образования
«Гомельский государственный университет имени
Франциска Скорины»

Рекомендовано к изданию научно-методическим советом учреждения образования «Гомельский государственный университет имени Франциска Скорины» 26.10.2005 (№ 192).

Д27 Программирование на языке Турбо-Паскаль и численное решение физических задач: Практическое пособие для студентов 1 курса специальности 1-31 04 01 02 – «Физика» / Е.А.Дей: Мин-во обр. РБ. – Гомель: УО «ГГУ им. Ф. Скорины», 2005. – 89 с.

Практическое пособие ставит своей целью обучение навыкам разработки программ на языке Турбо-Паскаль и решения физических задач с помощью компьютерных вычислений и адресовано студентам 1 курса специальности «Физика» для использования на лабораторных занятиях по курсу «Программирование и математическое моделирование».

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Умение разрабатывать собственные программы для решения физических задач с использованием современных средств программирования является необходимым элементом подготовки студента-физика.

Язык программирования Турбо-Паскаль получил широкое распространение, во-первых, как основной язык школьного курса информатики, и, во-вторых, как язык разработки прикладных программ. Это объясняется тем, что Турбо-Паскаль содержит все элементы, присущие современному языку программирования высокого уровня: структурные операторы управления, подпрограммы-процедуры и подпрограммы-функции, возможность создания библиотек подпрограмм (модулей), удобные, лаконичные средства управления экраном, графикой, звуком. В последние годы важность изучения языка возросла в связи с развитием системы визуального программирования Delphi, являющейся дальнейшим развитием языка Турбо-Паскаль.

Практическое пособие адресовано студентам специальности 1-31 04 01 «Физика» в рамках изучения первой части курса «Программирование и математическое моделирование».

Практическое пособие содержит изложение основных операторов языка Турбо-Паскаль и принципов программной реализации вычислений и имеет целью сформировать у студентов знание основных структур данных и операторов языка Турбо-Паскаль и выработать навыки их применения для численного решения физических задач.

По каждой теме приводятся теоретические сведения, описания алгоритмов, примеры программ, а также вопросы для самоконтроля и задания для самостоятельного выполнения.

Автор выражает признательность заведующему кафедрой теоретической физики профессору Максименко Н.В., преподавателям и сотрудникам кафедры теоретической физики: Андрееву В.В., Афонину А.А., Денисову Д.О., Дерюжковой О.М., Кагановичу В.Е., Капшаю В.Н., Климову М.Х., Марченко Т.Д., Тюменкову Г.Ю., Шульге С.Г. за доброжелательное отношение и поддержку в работе.

ТЕМА 1. ОСНОВНЫЕ КОМАНДЫ ОПЕРАЦИОННОЙ СИСТЕМЫ MS-DOS

Физика и компьютер. Персональный компьютер (ПК) – это комплекс технических устройств, позволяющих хранить и обрабатывать числовую, текстовую, графическую и звуковую информацию. Слово «компьютер» означает «вычислитель», то есть, устройство для вычислений. Для решения каждой задачи *необходима* последовательность команд – программа, по которой процессор будет выполнять нужные действия над исходными данными в требуемом порядке с целью получения результатов.

Создание и совершенствование ПК основано на достижениях различных разделов физики (механики, оптики, физики полупроводников и других), информатики и технологии.

Основные функциональные элементы ПК: а) процессор, выполняющий команды, записанные в двоичной форме, над числами, записанными в двоичной форме; б) микросхемы памяти, хранящие числа и команды в двоичной форме; в) внешние устройства, позволяющие хранить информацию, вводить ее в память компьютера или выводить на экран дисплея.

В компьютере любая информация (тексты, звуки, изображения) хранится и обрабатывается в виде двоичных чисел, записанных цифрами 0 и 1. Двоичный разряд, принимающий одно из двух значений: 0 или 1, называется *бит*. Физически двоичной цифре сопоставляется одно из двух возможных состояний элемента памяти. Микросхемы памяти содержат несколько миллионов таких элементов.

Восемь битов образуют *байт*, имеющий 2^8=256 различных состояний. Состояниям одного байта можно сопоставить или целые числа от 0 до 255, или один из 256 различных символов, или один из 256 различных цветов. Для хранения чисел используются также группы из 2 (полуслово), 4 (слово) и 6 байтов. В каждом случае важно указать, какой *тип* имеет информация, хранящаяся в памяти.

Для описания объема памяти используют более крупные единицы: один *килобайт* (Кб) содержит 2^{10}=1024 байт, один *мегабайт* (Мб) содержит 1024 килобайт.

Процессор – это сверхбольшая интегральная электронная схема, в которой содержится блок управления, арифметико-логическое устройство, вспомогательная память (кэш) и рабочие ячейки памяти (регистры) для хранения текущих команд и данных. Процессор

обрабатывает 16 или 32 бита одновременно и может выполнять несколько сот различных операций. Тактовая частота процессора 1ГГц означает выполнение 10^9 элементарных операций в секунду.

Загрузка MS-DOS. *Операционная система* (ОС) - это комплекс специальных программ, которые записываются («*загружаются*») в память компьютера при его включении и реализуют управление работой аппаратной части ПК и взаимодействие пользователя с ПК.

Основной ОС для IBM-совместимых компьютеров с 1981 г. вплоть до начала 90-х годов была операционная система MS-DOS (MicroSoft Disk Operating System). Последняя ее версия 6.22 выпущена в 1994 г.

ОС выполняет множество зачастую полностью скрытых от пользователя элементарных, но очень громоздких операций по управлению ресурсами и устройствами компьютера при его работе. Так, для обеспечения, например, простой операции копирования информации с одного диска на другой процессор должен совершить под управлением ОС несколько сотен элементарных действий. Пользователь же для этой цели выполняет всего одну команду операционной системы.

Загрузка операционной системы выполняется автоматически при включении ПК в электрическую сеть. Процесс загрузки сопровождается выдачей сообщений, связанных с конкретной конфигурацией системы и настройкой на определенную рабочую обстановку.

На завершение процесса загрузки и готовность ПК к работе указывает появление на экране командной строки MS-DOS, обозначаемой символом “>”. Он указывает, что в этой строке можно вводить команду, которая будет немедленно выполнена.

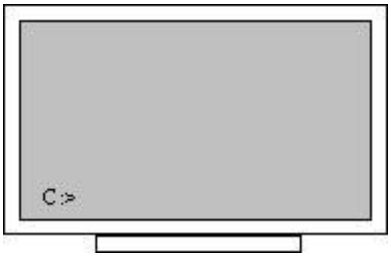


Рис. 1.1. Командная строка (приглашение) MS-DOS

Обозначение файлов в MS-DOS. Каждый накопитель на магнитных дисках в MS-DOS обозначается латинской буквой, за которой следует двоеточие, например A:, B: - гибкие магнитные диски; C:, D: и далее – логические разделы жесткого диска.

Программы пользователя, данные и результаты вычислений хранятся на магнитных дисках под различными *именами* и являются файлами. *Файл* - это содержащая информацию область внешней памяти, которой присвоено конкретное имя. Файл представляет собой совокупность записей, однотипных по структуре и методу доступа.

С любым файлом как с единицей информации на внешнем носителе пользователь может совершать различные действия: создание, переименование, перемещение, чтение и запись, удаление. К файловой системе также имеет доступ любая прикладная программа.

Обозначение файла состоит из имени и расширения, разделенных точкой. Имя может содержать до 8 символов, расширение – не более трех.

```
<ИМЯ>.<РАСШИРЕНИЕ>
(не более 8) (не более 3)
```

Имя и расширение могут состоять из латинских букв, цифр и специальных символов « _ $ # & @ ! % ( ) { } ` ' ~ ^».

В обозначении файла запрещается использовать следующие символы:

< > . : ; - ? *

Расширение имени файла, как правило, отражает назначение информации, содержащийся в файле. Например:

.EXE - исполняемый файл, содержащий двоичные команды, выполняемые процессором;

.SYS - системный файл;

.TXT - текстовый файл;

.TPU – файл модуля (библиотеки подпрограмм Турбо-Паскаль)

.BAT – командный (пакетный) файл;

.PAS – файл, содержащий текст программы на языке Турбо-Паскаль;

.BAK – копия файла, созданная перед его изменением (старая версия программы);

Полное имя файла должно быть уникальным для каждого файла. Как правило, файлы, относящиеся к одной задаче, имеют одинаковые имена, но разные расширения, например:

```
PROG.PAS – программный файл на языке Паскаль;
PROG.EXE – программа, готовая к выполнению;
```

PROG.DAT – результаты выполнения программы.

Некоторые специальные (зарезервированные) имена позволяют осуществить ввод или вывод информации обращаться на устройства компьютера как в отдельные файлы. Например:

PRN - используется для обращения к принтеру;

CON – консоль, при вводе означает клавиатуру, при выводе - дисплей;

NUL - "пустое" устройство; все операции ввода-вывода для этого устройства игнорируются.

Обозначение группы файлов. Для обозначения сразу нескольких файлов (но не каталогов!) используют запись имени файла с использованием символов-заменителей «*», «?» (такая запись часто называется «маска» или «шаблон»).

Символ «*» в имени или расширении файла заменяет любое число любых символов, а знак «?» - любой одиночный символ или отсутствие такового. Примеры шаблонов:

D:\FIZ*.PAS - все файлы типа PAS в каталоге FIZ диска D:

. - все файлы текущего каталога;

d*.* - все файлы с именами, начинающимися с d;

d. - все файлы, имена которых оканчиваются на d;

?A.EXE - все файлы типа EXE, у которых имя файла состоит из одного или двух символов и последний символ имени буква A;

A??B.* - все файлы, имена которых состоят из двух, трех или четырех символов, начинаются с A и оканчиваются буквой B;

Структура файловой системы MS-DOS. Каждый файл регистрируется в каталоге (директории), который сам является файлом специального вида. *Каталог* - это список файлов, содержащий сведения об имени, размере, дате создания и других характеристиках каждого файла. Если в каталоге записано имя файла, то говорят, что файл находится в данном каталоге. В каталог удобно включать группу файлов на одном носителе, объединенных по какому-либо признаку.

На каждом диске имеется один особый, главный каталог, называемый *корневым* каталогом. Он имеет имя дисковода, на котором находится (A:, C: и т, д). Корневой каталог создается операционной системой в процессе форматирования дисков, и его невозможно удалить или переименовать.

Имя каталога также может содержать не более 8 символов. Расширение в имени каталога обычно не указывается.

Каталог, в свою очередь, может быть зарегистрирован в другом каталоге. Это значит, что он включен в последний как целое, и тогда говорят, что он является внутренним, подчиненным каталогом (*подкаталогом*). В результате образуется иерархическая древовидная файловая система (рис. 1.2).

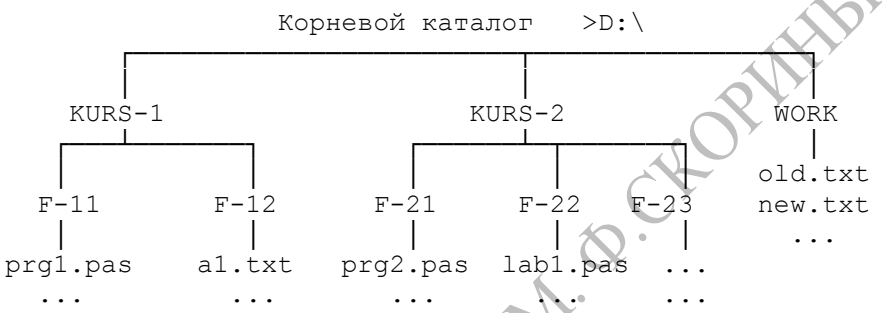


Рис. 1.2. Пример структуры файловой системы

Каталог, с которым в настоящий момент работает пользователь, называется *текущим.* Если в команде указать имя файла, то этот файл будет создаваться или отыскиваться в текущем каталоге.

Путь (маршрут) перехода из одного каталога в другой – это перечисление каталогов, которые необходимо пройти, разделенных символом «\». Если выполняется вход во внутренний каталог, то указывается имя этого каталога, но если выполняется выход из каталога на верхний уровень, то указывается описание «**. .**».

Например, чтобы из текущего каталога F-11 перейти в каталог F-23, необходимо указать следующий маршрут

```
..\..\KURS-2\F-23
```

Если путь начинается с символа «\», то маршрут вычисляется от корневого каталога диска, иначе - от текущего каталога.

Полное имя файла состоит из имени диска, пути от корневого каталога и самого имени файла. Полное имя используется при ссылках на нужный файл в различных командах. Например:

```
D:\WORK\new.txt
D:\KURS-1\F-12\a1.txt
```

Если дисковод не указан, то подразумевается текущий дисковод. Если путь не указан, то подразумевается текущий каталог.

Вспомогательные команды MS-DOS. Пользователь управляет работой компьютера с помощью команд, которые набираются посредством клавиатуры в командной строке. Команды являются словами или сокращениями слов разговорного английского языка. Операционная система выводит на экран результаты выполнения команды и необходимые сообщения.

Ввести команду – это значит набрать ее в командной строке и нажать клавишу [Enter].

В командной строке присутствует *курсор* (указатель) – мигающий знак подчеркивания. Он показывает, в каком месте строки появится символ, набираемый на клавиатуре. Текст команды можно редактировать такими клавишами:

[BackSpace] (иногда обозначается [←]) – стирание предыдущего символа;

[Del] – удаление текущего символа;

[F3] – вызов в командную строку предыдущей команды;

[Esc] – очистка всей командной строки.

Прописные и строчные латинские буквы в MS-DOS являются эквивалентными.

Команды могут иметь параметры (необязательные параметры перечисляются в квадратных скобках, но при вводе команды сами скобки не набираются).

Таблица 1.1. Вспомогательные команды MS-DOS

Вывести на экран системное время (time – время)	>TIME [Enter]
Изменить системное время – задать новое значение в том же формате	>TIME *Новое_значение*
Вывести на экран системную дату (date – дата)	>DATE [Enter]
Изменить системную дату – задать новое значение в том же формате	>DATE *Новое_значение*
Вывести на экран номер используемой версии MS-DOS (version - версия)	>VER
Запись личного названия (метки) диска (Label – метка) (не более 11 символов)	>LABEL *Метка*

Вывести на экран метку текущего диска (volume – том, диск)	>VOL
Вызвать на экран справочник по всем командам MS-DOS	>HELP
Очистить экран от предыдущих сообщений (CLean Screen – очистить экран)	>CLS

Информацию, выводимую на экран при выполнении любой команды, можно автоматически записать в текстовый файл. Для этого в конце команды нужно указать символ перенаправления ввода-вывода “>” и имя файла, который будет создан, например: >DIR > a.txt (вывод оглавления текущего каталога в файл a.txt).

Таблица 1.2. Команды MS-DOS для работы с каталогами (Name – обозначает имя нужного каталога).

Сделать текущим диск A: Сделать текущим диск D:	>A: >D:
Создать новый каталог с именем Name внутри текущего (Make Directory – создать каталог)	>MD Name
Войти во внутренний каталог с именем Name (Change Directory – изменить каталог)	>CD Name
Выйти из каталога в каталог верхнего уровня	>CD **..**
Выйти из каталога в корневой каталог	>CD \
Сообщить имя текущего диска и каталога	>CD
Удалить пустой каталог с именем Name внутри текущего (Remove Directory – удалить каталог)	>RD Name
Вывести список файлов и каталогов, хранящихся в текущем каталоге (Directory – каталог)	>DIR
- вывести список файлов постранично (Page – страница)	>DIR /P
- с записью списка в текстовый файл f.txt, который будет создан в текущем каталоге	>DIR > f.txt
- вывести список файлов по заданному шаблону	>DIR *.pas
Изменить имя подкаталога A:\Z\D1 на имя D2 (Move – перенести, здесь: переименовать)	>MOVE A:\Z\D1 D2
Вывести на экран дерево внутренних каталогов, хранящихся в текущем (Tree – дерево)	>TREE

- с выводом имен файлов	>TREE \F
- с записью списка в текстовый файл f.txt, который будет создан в текущем каталоге	>TREE > f.txt

Таблица 1.3. Команды MS-DOS для работы с файлами

Запустить программу с именем Name.exe на выполнение	>NAME
Создать копию файла f1.nnn под именем f2.mmm (Copy – копировать)	>COPY f1.nnn f2.mmm
Перенести файл f1.nnn из текущего каталога в каталог A:\MC (Move – переносить)	>MOVE f1.nnn A:\MC
Изменить имя файла с f1.nnn на f2.mmm (Rename – переименовать)	>REN f1.nnn f2.mmm
Показать на экране информацию, записанную в файле ff.nnn (Type – печатать)	>TYPE ff.nnn
Удалить файл с именем ff.nnn (Delete – удалить)	>DEL ff.nnn

Пояснения:

1. Если обрабатываемые файлы находятся в текущем каталоге, то указываются только их имена. Если какой-либо файл находится в другом каталоге, необходимо указать полное имя файла.

2. В именах файлов можно использовать шаблоны. При выполнении команды будут обработаны все файлы, соответствующие шаблону.

3. Если символы * и ? имеются во втором имени файла в команде, то символы имен файлов на соответствующих позициях не изменяются.

4. Общий формат команды копирования:

>COPY [диск:] [\путь] имя-файла1 [диск:] [\путь][имя-файла2]

В случае, если имя-файла2 опущено, копирование производится без изменения имени.

5. Командой COPY можно воспользоваться для объединения нескольких файлов и записи результата в один новый файл. Для этого сначала имена соединяемых файлов перечисляют через знак +, а затем указывают имя файла, в котором будет записан результат. Например

```
>COPY F1.txt+F2.txt+F3.txt F4.txt
```

Файл с именем F4.t образуется путем соединения файлов Fl.txt, F2.txt и F3.txt. Если результирующий файл не указан, то соединенные файлы будут записаны в файл, имя которого задано первым (в приведенном примере - в файл Fl.txt).

6. С помощью команды COPY можно создать текстовый файл, в который будут записаны строки, набираемые на клавиатуре. В этом случае в качестве первого имени файла задается устройство CON (копирование с консоли-клавиатуры). Например, для создания файла с именем info.txt следует выполнить действия:

```
>COPY CON INFO.TXT
набрать строку текста, нажать [Enter]
набрать строку текста, нажать [Enter]
. . .
нажать [Ctrl-Z] или [F6]
```

7. С помощью команды COPY можно отпечатать файл на принтере. В этом случае в качестве второго имени файла задается устройство PRN (принтер):

```
>COPY help.txt PRN
```

Файл с именем help.txt копируется на устройство PRN, т.е. печатается на принтере. Принтер должен быть готов к печати. Этот способ относится только к текстовым файлам.

8. При выполнении команды >TYPE **в**ывод на экран можно приостановить нажатием клавиш [Crtl-S], продолжение вывода - по нажатию любой клавиши. Прекратить вывод на экран можно, нажав [Ctrl-C]. Шаблоны имен файлов в команде TYPE не допускаются.

9. Вывести файл на экран можно и с помощью команды

>COPY *имя_файла* CON

10. Для создания системной дискеты используется команда

>FORMAT A: /S

При этом после выполнения обычного форматирования на дискету копируются системные файлы MS-DOS, реализующие начальную загрузку системы с дискеты.

Примеры команд MS-DOS:

>D: - сделать диск D: текущим;

>CD \1\F-12 - переход в каталог \1\F-12;

>DIR A:\ /P – вывести оглавление корневого каталога на диске A: постранично;

>MD A:\WORK – создание подкаталога WORK в корневом каталоге диска A:

> DEL *.bak – удаление всех файлов с расширением .bak из текущего каталога;

>DEL a.txt - удаление файла a.txt из текущего каталога;

>COPY info.txt t1.txt

>COPY \TMP*.pas a: - копирование всех файлов с расширением .pas из каталога \TMP текущего диска на диск a:.

>COPY *.txt new.prn – объединение содержимого всех файлов с расширением .txt в один файл new.prn;

>REN *.txt *.doc – переименование всех файлов с расширением .txt в файлы с теми же именами, но расширением .doc;

>REN t1.txt a1.txt

>MD Physics - создать внутренний каталог с именем Physics

>CD Physics - войти в каталог с именем Physics

>DIR /P – вывести на экран список файлов постранично

>CLS – очистить экран

Вопросы для самоконтроля

Как перезагрузить MS-DOS?

Что называется файлом? Как обозначаются файлы?

Какие вы знаете стандартные расширения файлов?

В чем отличие знаков ? и * в шаблонах файлов? Приведите примеры.

Что называется каталогом? Иерархическая структура каталогов на диске.

Какими способами в DOS можно создать текстовый файл?

Как в DOS посмотреть содержимое диска, каталога?

Как в DOS посмотреть содержимое файла?

Полное имя файла. Указание пути к файлу.

Содержание задания

Упражнение 1. Переход в режим MS-DOS.

Все компьютеры настроены таким образом, что сразу после загрузки операционной системы MS-DOS запускается программа Norton Commander.

Поэтому для работы собственно в MS-DOS следует свернуть рабочие панели NC клавишной командой [Ctrl+O].

Состояние пустого черного экрана с одной строкой внизу – это все, что в первые годы предлагала MS-DOS пользователю.

Нижняя строка экрана имеет вид C:\> и называется командная строка. После символа > набираются команды, которые сразу же выполнняются.

Упражнение 2. Выполнение простейших команд MS DOS. Упражнение состоит в наборе команд и просмотре результатов их выполнения.

```
DIR                 TREE
DIR > a.txt         TREE > b.txt
CLS                 CLS
TYPE a.txt          TYPE b.txt
```

Упражнение 3. Работа с файлами и каталогами в MS-DOS.

Указание: Для детального анализа результатов выполняемых команд после выполнения каждого шага выполняйте команду DIR для того чтобы увидеть как изменяется содержимое текущего каталога.

1. Перейдите на диск D.
2. Создайте каталог физического факультета PHYSICS.
3. Перейдите в каталог с именем 1, если его нет, то создайте его.
4. Создайте каталог группы (F-13 или F-14 соответственно).
5. Перейдите в каталог группы.
6. Создайте собственный каталог с именем, образованным от собственной фамилии. В имени каталога не должно быть больше 8 символов, и оно должно состоять из букв латинского алфавита (например, PETROV).

Впредь работайте только в своем каталоге! Создавать свои новые файлы можно только в своем каталоге!

При создании собственных файлов используйте это обозначение в именах файлов, причем в качестве последнего символа имени удобно использовать цифру – для нумерации файлов, связанных с разными задачами (например, petrov01.txt, petrov02.txt и т.д).

7. Выполните команду очистки экрана.
8. Командой COPY CON . . . создайте в своем каталоге текстовый файл, который должен содержать Вашу фамилию, имя и отчество, а также номер группы и номер вашего варианта (номер варианта соответствует номеру в журнале группы). Переход на русский

алфавит – клавишами [Shift-Shift] (только для набора текста файла. Все команды записываются латинскими буквами!).

9. Создайте копию этого файла с именем file1.txt. Просмотрите содержимое файла file1.txt на экране.

10. Просмотрите текст файла file1.txt на экране.

11. Создайте в своем каталоге еще один текстовый файл, в котором наберите список основных команд MS-DOS – по одной команде в строке:

TIME DATE CLS MD RD CD DIR COPY RENAME DEL

12. Создайте копию этого файла с именем file2.txt. Просмотрите содержимое файла file2.txt на экране.

13. Просмотрите список файлов в вашем каталоге.

14. Переименуйте одной командой файлы, имена которых начинаются “file.. “, в файлы, начинающиеся символами “myfile..”.

15. Просмотрите список файлов в вашем каталоге.

16. Покажите результат проделанной работы преподавателю.

17. Удалите одной командой все файлы, имя которых начинается с “m”.

18. Просмотрите список файлов в вашем каталоге.

19. Удалите все созданные файлы в вашем каталоге.

20. Перейдите на диск C:.

Восстановите рабочие панели NC клавишной командой [Ctrl+O].

ТЕМА 2. ПРОГРАММА-ОБОЛОЧКА NORTON COMMANDER

Общая характеристика программы NC. Для работы с операционной системой MS-DOS наиболее удобна программа-оболочка Norton Commander (NC). Она обеспечивает визуальное отображение файловой структуры на экране, применение клавишных команд и окон диалога для обработки файлов и каталогов.

Вместо набора команды MS-DOS достаточно нажать одну или две определенные клавиши, и команда начинает выполняться. Соответствующую команду DOS программа NC генерирует

автоматически. Таким образом исключаются грамматические ошибки и сокращается время для ввода команды.

Имя программы NC.EXE, она обычно расположена в каталоге C:\NC\, так что команда запуска программы имеет вид >C:\NC\NC. Практически на всех компьютерах вызов этой программы предусмотрен в файле autoexec.bat и происходит автоматически при включении компьютера.

Для выхода из программы NC следует нажать клавишу [F10] и на запрос подтверждения выхода - клавишу [Enter] или "Y".

Режимы панелей NC. При работе NC информация появляется на экране в двух прямоугольных рамках (панелях). Это позволяет видеть сразу содержимое двух дисков. Каждая панель имеет заголовок: имя текущего диска или каталога.

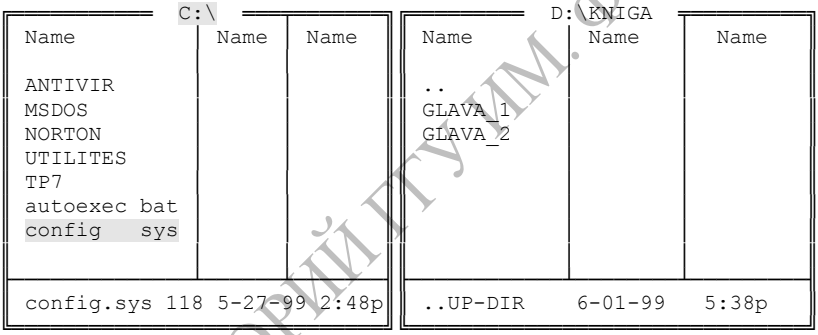


Рис.2.1. Пример экрана после загрузки Norton Commander

Панели могут отображать информацию в трех различных режимах:

- список файлов и каталогов (стандартный режим)
- дерева каталогов (режим Tree)
- информационный (режим Info)

Общий вид экрана операционной оболочки Norton Commander изображен на рис.2.1. Одна из панелей является текущей, т.е. активной. Заголовок текущей панели выделен цветом. Один из файлов или каталогов текущей панели также выделен (говорят, что на него

установлен указатель). Это означает, что клавишные команды, подаваемые пользователем, будут относиться к выделенному файлу или каталогу на активной панели. В пределах панели указатель перемещается клавишами [↑], [↓], [PageUp], [PageDn].

Под панелями располагается обычная командная строка MS-DOS. В этой строке можно вводить любую команду операционной системы.

В частности, на Рис.2.1 левая панель имеет заголовок "C:\". В ней высвечивается корневой каталог, содержащий:

- каталоги ANTIVIR, MSDOS, NORTON, UTILITES, TP7;
- файлы autoexec.bat, config.sys.

Правая панель имеет заголовок "D:\KNIGA", т.е. в ней высвечивается содержимое каталога "KNIGA" диска "D": каталоги GLAVA_1 и GLAVA_2. Две точки в первой строке указывают на то, что каталог не является корневым.

Информация о выделенном файле представлена в нижней части панели. Для файла указываются имя, размер в байтах, дата создания (или модификации) и время. При выводе даты сначала указывается месяц, затем число и год.

Нажатие ряда специальных клавиш в NC воспринимается как команда. В частности, при работе с панелями можно:

[TAB] - перевести указатель на другую панель;

[Enter] - войти внутрь выделенного каталога (для выхода использовать строку [. .])

[ALT - F1] - выбрать рабочий диск на ЛЕВОЙ панели;

[ALT - F2] - выбрать рабочий диск на ПРАВОЙ панели;

[ALT - F10] - включить режим изображения дерева каталогов (Рис. 2) и перемещения по нему ([Esc] – отмена). Выводится структура каталогов активного диска, причем указатель можно быстро перемещать к нужному каталогу и войти в него, нажав [Enter].

[CTRL - O] - убрать/установить обе панели (используется для просмотра результатов выполнения команд);

[Ctrl-Q] – включение режима быстрого просмотра (повторное нажатие – выключение). В этом режиме содержимое выделенного файла выводится на соседней панели. Если выделен каталог, то в соседней панели выводится информация о количестве и суммарном размере всех файлов, хранящихся в нем, включая подкаталоги;

[Ctrl-L] - включить режим вывода сводной информации об использовании памяти и дискового пространства (повторное нажатие – отмена) (рис. 2.2).

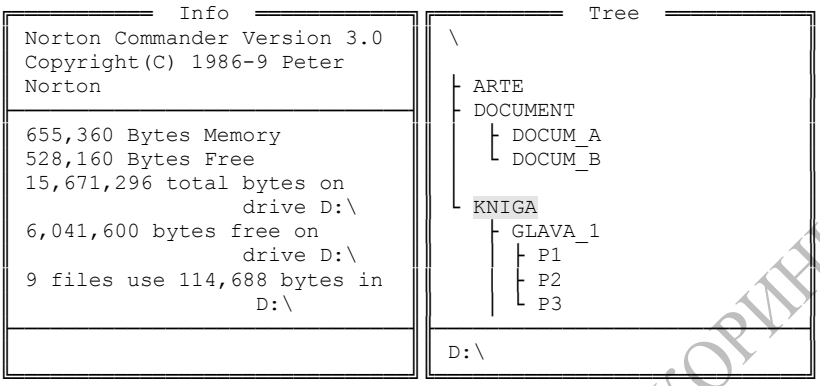


Рис. 2.2. Панель Info и панель Tree

Назначение функциональных клавиш. В самой нижней строке экрана Norton Commander выводится напоминание о назначении функциональных клавиш. Цифрам 1...10 соответствуют функциональные клавиши F1...F10. Нажатие каждой такой клавиши рассматривается программой как команда, которая сразу же выполняется. В частности:

F1 - Help – вызов справочной информации (помощь). При нажатии этой клавиши на экране высвечивается справочная информация, описывающая назначение клавиш и команд.

F2 - User – вызов списка команд MS-DOS, сформированного пользователем (пользовательское меню).

F3 - View – просмотр на экране содержимого выделенного файла.

F4 - Edit – редактирование файла с помощью встроенного в NC текстового редактора.

F5 - Copy - копирование файла или группы файлов, выделенных в одной панели, в каталог, указанный в другой панели.

F6 - RenMov – переименование файла (если обе панели настроены на один каталог) или перенос файла в другой каталог (указанный на соседней панели).

F7 - Mkdir - создание нового каталога внутри текущего.

F8 - Delete – удаление выделенного файла или каталога.

F9 - PullDn - вызов главного меню самой программы NC (появляется в самой верхней строке экрана).

F10 - Quit - выход. Завершение работы программы NC. При запросе подтверждения следует выбрать кнопку диалога "Yes".

Клавишные команды Norton Commander позволяют производить действия только с каталогами и файлами текущей панели.

При одновременном нажатии клавиши [Alt] и клавиш [F1]..[F10] получается набор других команд, и это отражается в строке-подсказке.

[Alt - F1] - выбор диска на левой панели;

[Alt - F2] - выбор диска на правой панели;

[Alt - F7] - поиск файла на диске ;

[Alt - F8] – просмотр списка ранее введенных команд MS-DOS и выполнение выделенной команды;

[Alt - F10] – вывод дерева каталогов и работа с ним.

Основные действия с файлами и каталогами в NC. Главное удобство наличия двух панелей - возможность видеть одновременно и каталог-отправитель (вместе с выделенным отправляемым файлом) и каталог-получатель.

Настройка панелей:

1) выбор диска: Alt-F1 - на левой панели; Alt-F2 - на правой (нажать клавишу с буквой – именем диска).

2) настройка каталога: - установить указатель и нажать Enter; для выхода из каталога – указать строку [..] и нажать [Enter]

3) клавиша Tab – перевод указателя на другую панель.

Перенос файла из одного каталога в другой выполняется с помощью следующих действий:

1) настроить одну панель на тот диск и каталог, КУДА переносится файл;
2) другую панель настроить на тот диск и каталог, ОТКУДА переносится файл;
3) указать переносимый файл, нажать клавишу [F6] и ,в подтверждение, [Enter].

Курсор должен находиться на той панели, ОТКУДА будет производиться перемещение. Для отмены команды - нажать клавишу [Esc].

При переносе файл копируется на новое место и удаляется на старом. Это сразу будет видно на экране NC, в отличие от выполнения команды в MS-DOS.

Создание нового каталога внутри текущего. Для создания каталога на рабочей панели достаточно нажать клавишу [F7] и в появившемся окне диалога ввести имя нового каталога.

Замечание: Реакция NC на нажатие [Enter] :

* если в командной строке DOS написано что-нибудь (команда), то выполняется команда;

* если в командной строке DOS ничего не написано, то реакция определяется тем, какое имя выделено на рабочей панели: в частности,

если выделен каталог, то происходит вход в каталог,

если выделен файл с типом .PAS, то вызывается Turbo Pascal,

если выделен файл с типом .TXT, то вызывается Multi Edit,

если выделен файл с типом .EXE, .COM, .BAT, то запускаются программы.

Создание текстового файла с помощью встроенного редактора: нажать [Shift-F4] и указать имя создаваемого файла. Сохранение текста – [F2], выход – [Esc].

Выделение группы файлов. Некоторые действия над файлами, такие как копирование, удаление, можно производить не только с одним файлом, но также с группой файлов (выделяются желтым цветом):

– включить файл в группу - нажать клавишу [Ins] (повторное нажатие [Ins] снимает выделение);

– включить в группу файлы по шаблону – нажать клавишу [+] на правой (калькуляторной) клавиатуре и задать шаблон;

– отменить включение в группу файлов по шаблону – нажать клавишу [-] на правой группе клавиш.

Для удаления группы файлов следует: а) выделить группу; б) нажать F8 и ответить на вопросы меню.

Если надо удалить только один файл, то его помечать не надо. Достаточно его выделить и далее сделать F8.

Для просмотра содержимого текстового файла следует указать файл на рабочей панели и нажать клавишу F3.

Главное меню программы NC. Операционная оболочка Norton Commander обладает большими возможностями и все ее функции можно увидеть, войдя в режим "PullDn" (полного спускающегося меню). Меню представляет собой совокупность пунктов (разделов),

каждый из которых может содержать свою совокупность пунктов. Раздел меню предназначен для выполнения каких-либо действий или установки режимов, параметров, значений и пр. Меню содержит пункты:

Left (левая), определяет вид и содержание левой панели;

Files (файлы), выполнение операций над файлами;

Commands (команды), выполняет дополнительные команды (NC);

Options (опции), определение внешнего вида экрана (NC);

Right (правая), определяет вид и содержание правой панели.

Возврат из выбранного пункта в главное меню осуществляется нажатием клавиши <Esc>. Для отказа от меню или снятия промежуточных окон используется клавиша <Esc>.

СОДЕРЖАНИЕ ЗАДАНИЯ

Выполните все действия, перечисленные в Упражнении 2 и Упражнении 3 предыдущего задания с помощью клавишных команд программы NC. Во всех практических заданиях используется в качестве рабочего - диск "D:".

ТЕМА 3. СЕАНС РАБОТЫ В ИНТЕГРИРОВАННОЙ СРЕДЕ ТУРБО-ПАСКАЛЬ

Общая характеристика системы программирования Турбо-Паскаль. Система Turbo-Pascal 7.0 включает в себя среду программирования и инструмент программирования - язык программирования Turbo-Pascal.

Язык программирования Pascal был разработан профессором Никлаусом Виртом в техническом институте в Цюрихе и опубликован в 1971 году. Язык назван в честь французского ученого Блеза Паскаля, разработавшего одно из первых суммирующих устройств. Pascal предназначался для обучения студентов программированию, и был первым языком, в котором были явно выражены концепции структурного программирования. Международный стандарт языка Pascal был утвержден в 1982 году. Последующие совершенствования

языка сделали его одним из самых распространенных в мире профессиональных алгоритмических языков.

Наиболее мощное развитие язык получил в реализации интерактивной среды программирования Турбо-Паскаль.

В настоящее время язык Турбо-Паскаль является составной частью языка Object Pascal, используемой в среде визуального программирования Delphi.

Система Turbo-Pascal включает в себя:

1. Текстовый редактор
2. Компилятор Turbo-Pascal (переводит команды в двоичные коды)
3. Отладчик, позволяющий при выполнении программы выявлять все ошибки и неточности
4. Справочная система, позволяющая находить ответы на интересующие программиста вопросы (работает по нажатию кнопки F1)
5. Вспомогательные средства (встроенные процедуры и функции)

Этапы создания программы:

- написание текста программы;
- набор и редактирование текста программы;
- отладка программы (выявление синтаксических ошибок). Выполняется по команде Alt-F9 - компиляция
- тестирование программы (выполнение при исходных данных, для которых результат известен). Выявление логических ошибок;
- выполнение программы Ctrl-F9.
- создание самостоятельной программы (EXE-файла)

Сеанс работы в среде программирования. Для начала работы нужно войти в свой рабочий каталог, например:

F:\2\WORK-TP\F-23\IVANOV

Для этого выполнить с программе NC следующие действия:

- выбор диска: Alt-F1 или Alt-F2
- выбор каталога: Указать --> Enter
- (при необходимости) создание нового каталога: F7 --> Имя

Находясь в рабочем каталоге, нужно вызвать Турбо-Паскаль для обработки своей программы:

а) для создания нового файла программы набрать в командной строке >turbo Имя (расширение .pas можно не указывать).

б) для доработки файла существующей программы указать файл в панели NC --> Enter.

В результате начнется выполнение программы – компилятора Турбо-Паскаля, на экране появится окно редактора программ и в окне редактора откроется файл name.pas

Если файл name.pas уже был записан в каталоге, он будет открыт для обработки.

Если такого файла не было, он будет создан (но вначале он не содержит информации).

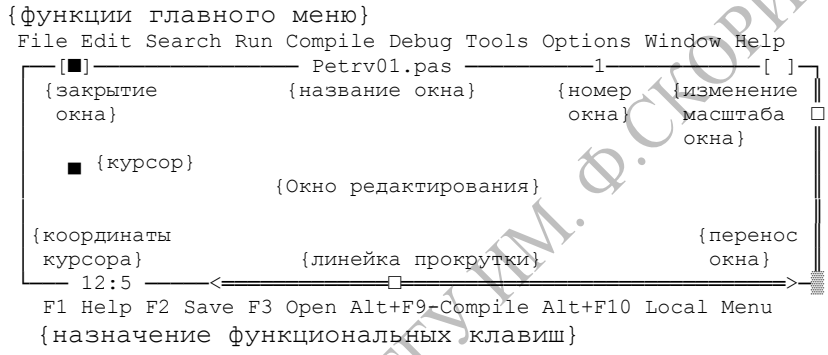


Рис. 3.1 Рабочее состояние экрана - экранный редактор

Для открытия другого файла достаточно перейти в главное меню (нажать F10) и в пункте <File> выбрать элемент <Open> и в открывшемся списке – выбрать нужный файл.

Для создания нового файла достаточно перейти в главное меню (нажать F10) и в пункте <File> выбрать элемент <New> и в открывшемся новом окне набирать текст нового файла.

Далее файлу необходимо присвоить конкретное имя с помощью пункта меню <File>–<Save as..>.

В ходе набора текста программы следует регулярно сохранять на диске сделанные изменения, нажимая клавишу [F2] (что соответствует меню <File>–<Save>).

Быстрый переход к нужной функции главного меню: ALT+Первая буква, например: ALT+F - переход к функции FILE.

Завершение работы: Alt-X.

Команды экранного редактора (функция EDIT). При написании программы в редакторе ТП справедливы стандартные правила использования клавиш:

[Tab] – фиксированный отступ на несколько пробелов;
[Shift]+Буква – Большая буква
Левый [Shift]+[Ctrl] – английская раскладка клавиатуры
Правый [Shift]+[Ctrl] – русская раскладка клавиатуры
←↑↓→ - клавиши управления курсором
[Home] – на начало строки
[End] – в конец строки
[PgUp] – переместить текст на страницу вверх
[PgDn] - переместить текст на страницу вниз
[Del] – удалить символ, указанный курсором

Блок - это любой выделенный фрагмент текста, от единственного символа до сотен строк.

Одновременно в окне может быть только один блок. Помеченный блок можно копировать, передвигать, удалять, выводить на печать или записывать в файл.

В режиме редактирования имеется вспомогательная область памяти (*буфер*), в которую можно записать блок для последующего вывода в любом окне редактирования.

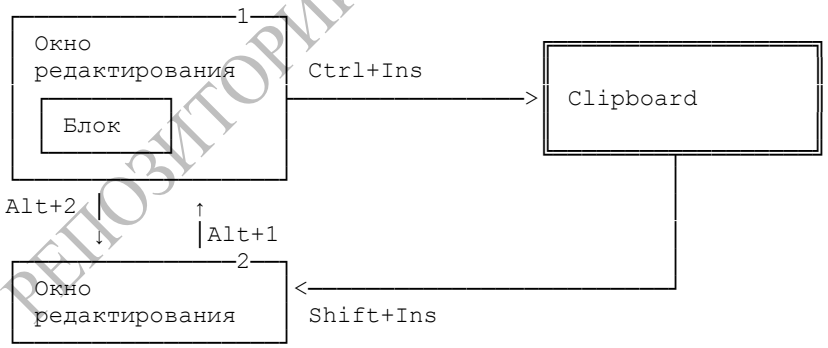


Рис. 3.2 Схема работы с буфером при редактировании двух файлов

Для выделения блока и работы с буферной памятью используются команды:

[Shift]+ → - выделить фрагмент строки

[Shift]+↓ - Выделить несколько строк вниз от начальной

[Ctrl]+[Ins] – скопировать выделенный фрагмент в буфер обмена (область памяти для временного хранения информации);

[Shift]+[Ins] – вставить фрагмент из буфера в текст программы

[Shift]+[Del] – вырезать выделенный фрагмент (перенести в буфер)

[Ctrl]+[Del] – удалить вообще

Компиляция и выполнение программы. ТП – язык программирования высокого уровня, то есть, использующий наглядную и легковоспринимаемую человеком систему записи команд, то необходимо переписать программу на язык, понятный компьютеру.

Перевод программы называется компиляцией. Чтобы откомпилировать программу, нужно нажать [Alt]+[F9] или выбрать соответствующий пункт в меню.

При переводе на машинный язык программа-компилятор ТП проверяет текст программы на наличие синтаксических ошибок. Если ошибок нет, то на экране появляется диалоговое окно, в котором будет указано, что компиляция прошла успешно и для продолжения работы нужно нажать любую клавишу

```
Compile succesfull
Press any key
```

Если компилятор нашел ошибки в тексте программы, то компиляция останавливается, и управление передается редактору, так что в окне редактора выводится сообщение об ошибке. При этом курсор устанавливается на место, где компилятор нашел ошибку. Для продолжения работы и исправления ошибки необходимо нажать любую клавишу. После успешной компиляции программу можно запустить на выполнение комбинацией клавиш [Ctrl]+[F9] или пунктом меню <Run>.

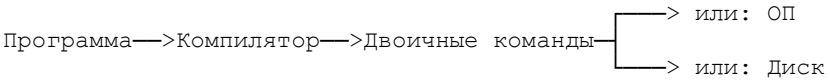
В результате выполнения программы на экране должна выводится информаци, для ее просмотра нужно свернуть среду ТП (перейти в окно результатов) с помощью клавишной команды [Alt]+[F5].

Alt-F9 - компиляция программы

Ctrl-F9 - компиляция и выполнение программы

Создание EXE-файла. После успешной компиляции и выполнения программа еще зависит от среды ТП и не может без нее использоваться. Для того, чтобы получить программу, не зависящую

от среды программирования, нужно результат компиляции программы сохранить не в оперативной памяти, а на диске. При этом создается файл, содержащий двоичные команды управления процессором. Файл имеет расширение .EXE (Execute - выполнить).



Для создание exe-файла необходимо:

1) изменить размещение (Destination) результата компиляции в меню Compile:

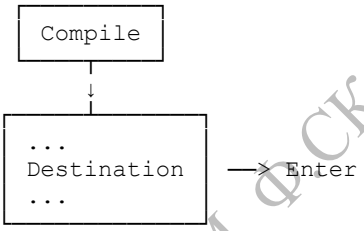


Рис. 3.3 Изменение размещения результата компиляции

2) выполнить команду компиляции Alt-F9

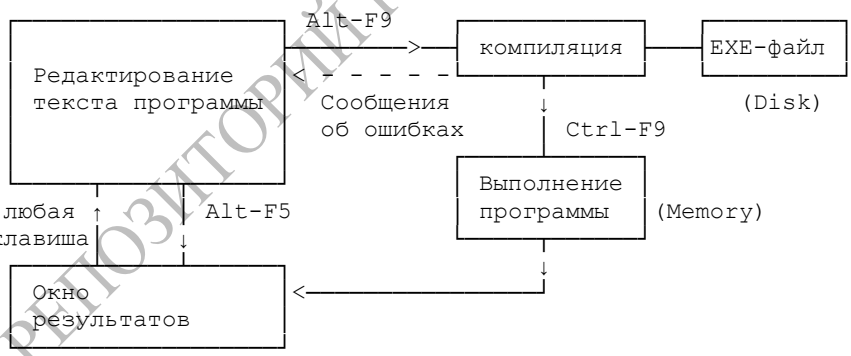


Рис. 3.4 Общая схема компиляции программы

Для запуска программы на выполнение достаточно указать его в панели NC и нажать [Enter].

Рабочие окна среды ТП. Редактор ТП позволяет работать сразу с несколькими окнами, в которых могут размещаться тексты других программ либо вспомогательная информация. Все команды работы с окнами находятся в пункте меню Window.

Открыв окно *Watch*, мы просмотрим значение переменных, используемых в программе.

В окне *Output* можно без сворачивания среды ТП увидеть содержимое экрана.

Если необходимо видеть одновременно все открытые окна, то из пункта меню выбирают *Window/File* (F10). Чтобы перейти от одного окна к другому, нужно нажать F6, окна будут активизированы по очереди.

Клавиша F5 разворачивает окно на весь экран, повторное ее нажатие сворачивает его до предыдущего размера.

Размер окна удобно изменять мышью, для чего необходимо взять окно за правый нижний угол и двигать мышь, удерживая левую клавишу.

Комбинация клавиш Alt+F3 закрывает активное окно.

Результаты расчетов можно отслеживать в окне Watch. Для того, чтобы узнать, чему равна та или иная переменная, ее нужно поместить в это окно. Для этого нужно нажать Ctrl+F7, и в диалоговом окне Add Watch ввести имя этой переменной. После этого открывается окно Watches в котором будет указано значение введенной переменной. Это окно не редактируется за исключением того, что переменную из этого окна можно удалить клавишей Delete. Если это окно активно, то для добавления новой переменнной нужно нажать клавишу [Ins]. Окно Watches используется при *отладке программы*.

Диалоговые окна. Многоточие в конце команды меню (например, «Save as ...») указывает на то, что команда открывает диалоговое окно. Диалоговое окно - это наиболее удобный способ показать и установить многочисленные параметры выполнения команды. Различные элементы диалогового окна выделяются цветом.

Выделяют пять основных типов управления экрана: зависимые кнопки, независимые кнопки, кнопки действия, окна ввода и окна списка. Ниже приведено типичное диалоговое окно, которое иллюстрирует некоторые из этих элементов:

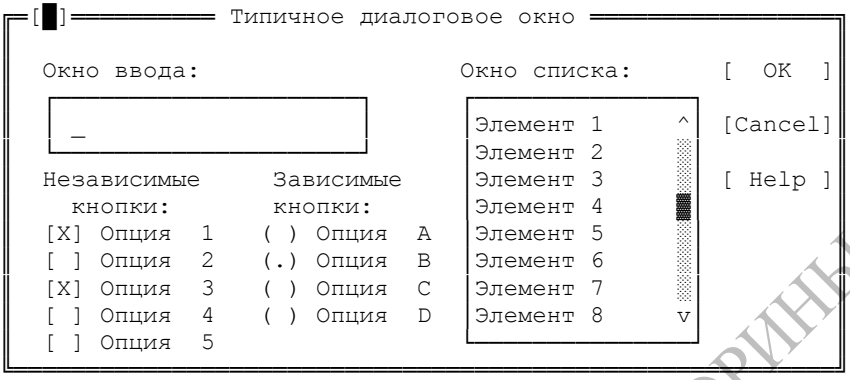


Рис. 3.5. Основные элементы окна диалога

Это диалоговое окно имеет три стандартные кнопки: OK, Cancel, Help. При нажатии OK будут выполнены все выборы в диалоговом окне; при выборе Cancel никаких изменений и действий сделано не будет, а диалоговое окно исчезнет. Esc всегда является быстрым вариантом клавиатуры для Cancel (даже если Cancel кнопка не присутствует).

Выбор и нажатие кнопок можно выполнить:

- мышью (перевод указателя и щелчок левой кнопкой);
- клавиша [Tab] – перевод указателя, [Enter] – нажатие кнопки. Нажатие [Shift-Tab] – перевод указателя в обратной последовательности.

Если название элемента имеет выделенную цветом букву, то нажатие клавиши [Alt] совместно с этой буквой эквивалентно включению элемента диалога.

Встроенная справочная подсистема. Справочную подсистему вызывают клавишей "F1". Справка является контекстно-зависимой. Это значит, что на экран выводится справка об обьекте, указываемом курсором в тексте программы или в меню, либо о текущей ситуации в системе. В большинстве случаев, передвигая курсор по тексту справки, выбирая те или иные отмеченные элементы текста справки и нажимая затем клавишу "Enter", можно получить дальнейшие (более подробные) сведения. Целесообразно пользоваться справочной подсистемой и при появлении сообщений об ошибках. Это избавляет от необходимости выяснять их причины по руководствам и справочникам.

Во время работы с Редакторм текста в ТП Вы имеете возможность с помощью клавиатурной комбинации "Ctrl-F1" или правой кнопки мыши (если она, конечно, не установлена по-другому в "Options/Environment/Mouse...") получать справочную информацию об операторе программы, на который указывает курсор. Установите курсор на строку, содержащую "WriteLn" (непосредственно на само имя процедуры), и нажмите "Ctrl-F1". Вы получите справку об этой Паскаль-процедуре.

Если курсор не находится на зарезервированном слове и общая ошибочная ситуация не имеет места, то нажатие клавиатурной комбинации "Ctrl-F1" приводит к выдаче индексной страницы (предметного указателя справок), перемещаяясь из которой по справкам внутри справочной подсистемы, можно добиться получения информации о любой процедуре и функции, даже если ее имя и неточно указано в тексте программы или вообще там отсутствует.

В ТП, начиная с версии 5.5, имеется возможность выделения и копирования фрагмента текста из окна справки в программу. Это полезно для переноса иллюстративных примеров программирования в окно редактора для их выполнения и изучения.

СОДЕРЖАНИЕ ЗАДАНИЯ

Выполните набор, сохранение на диск и компиляцию примера простой программы. Выполните не менее 5 расчетов с различными исходными данными.

```
PROGRAM Sum1;
{Лабораторная работа 1. Задача 1}
{Ппрограмма для вычисления суммы двух чисел}
VAR alfa,beta,gamma,sigma:real;
BEGIN
 WRITELN('Программа вычисления суммы двух вещественных
 чисел');
 WRITELN;
 WRITELN('Введите значения двух слагаемых:');
 READLN(alfa,beta);
 gamma:=alfa+beta;
 WRITELN('Сумма заданных чисел = ',gamma);
sigma:=alfa*beta;
```

```
WRITELN('Произведение заданных чисел = ',sigma);
 READLN;
END.
```

ТЕМА 4. ПРОГРАММИРОВАНИЕ ВЫЧИСЛЕНИЙ С ЛИНЕЙНЫМ ПОРЯДКОМ ДЕЙСТВИЙ НА ЯЗЫКЕ ТУРБО-ПАСКАЛЬ

Англо-русский словарь ключевых слов (технический перевод). Язык программирования служит для записи команд, понятных компьютеру. Отдельные части команд - английские слова, из которых строятся команды. Далее приводится англо-русский словарь программных терминов:

AND - и (Логическая операция "И")
ARRAY - массив
BEGIN - начать (выполнение записанных команд)
CASE - в случае
CONST - константа
DIV - деление (division) (Операция деления целых чисел с получением целого результата)
DO - выполнить
DOWNTO - назад (изменение параметра цикла от максимального к минимальному значению)
ELSE - иначе
END - закончить (выполнение записанных команд)
FILE - файл
FOR - для (Начало цикла с известным числом повторений)
FUNCTION - функция (Начало подпрограммы-функции)
IF - если (Начало условного оператора IF)
MOD - модуль (Остаток от деления целых чисел)
NOT - нет (Логическое отрицание)
OF - из (из каких элементов или выбор из многих вариантов)
OR - или (Логическая операция ИЛИ)
PROCEDURE - процедура (Начало подпрограммы-процедуры)
PROGRAM - программа (Начало программы)

READ - прочитать (с клавиатуры)
REPEAT - повторять (Начало цикла REPEAT)
STRING - строка
THEN - то, тогда (Раздел ТОГДА условного оператора IF)
TO - к, до (изменение параметра цикла от минимального к максимальному значению)
TYPE - тип
UNIT - блок, модуль
UNTIL – до получения (Конец цикла REPEAT)
USES - используемый (Указатель используемых модулей)
VAR – переменная (variable)
WHILE - пока (Начало цикла WHILE)
WRITE - написать (на экране)

Элементы программы на языке Турбо-Паскаль.

Программа - это упорядоченная последовательность команд, записанная по определенным правилам и реализующая алгоритм решения задачи на каком-либо языке программирования.

Модулем называется самостоятельно компилируемый файл ТП, в котором содержатся описания констант, переменных, типов, процедур и функций. Все описанные в модуле элементы после его подключения можно использовать в программе. Чтобы подключить модули, достаточно перечислить их имена после слова Uses. Например, Uses Crt, Graph;

В любом месте программы можно записать *комментарий*, то есть пояснение к тексту программы либо в {. .} либо в (*. .*) скобках. Комментарии не учитываются компилятором при обработке программы.

При необходимости выполнить программу без группы операторов удобно их закомментировать, не удаляя.

Описание констант. Константой называется величина, значение которой не будет меняться в ходе выполнения программы. В качестве константы в ТП могут использоваться целые и вещественные числа, логические константы (True, False), символы и строки символов (записываются в апострофах ‘f’, ‘stroka’).

Все константы должны быть перечислены в разделе описания констант

Const Имя=Значение;

В разделе описания констант можно описать любое количество констант различного типа, например:

```
Const  g=9.81; n=200; - числовые константы
       sym='A' ; - символьный тип
       stroka='Ф-12' ; - строковый тип
       kod=$124 ; - шестнадцатиричный тип
       log=True; логический тип
```

При описании констант вещественного типа для указания значения «десять в степени .. » используют букву e (или E): 1.23e3 соответствует числу 1230.

Константы не требуют описания типа (нетипизированные константы). Тип автоматически распознается по написанию константы.

Стандартные типы переменных. *Тип* - это способ организации оперативной памяти для хранения данных. *Переменная* - это именованный участок оперативной памяти, в котором хранятся данные.

Тип переменной однозначно определяет: - объем занимаемой памяти; - диапазон его возможных значений; - набор допустимых операций.

Таблица 4.1. Стандартные целочисленные типы переменных в языке Турбо-Паскаль

Обозначение типа	**Наименование**	**Размер занимаемой ОП**	**Диапазон значений**
Byte	Байт	1 байт	0..255
Shortint	Короткое целое	1 байт	-128..127
Integer	Целое	2 байта	-32768..32767
Word	Слово	2 байта	0..65535
Longint	Длинное целое	4 байта	-2147483648 ..2147483647

Таблица 4.2. Вещественные типы переменных Турбо-Паскаля

Обозначение типа	Наименование	Число значащих цифр	Размер в ОП	Диапазон допустимых значений
Real	Вещественный	12	6 байт	$2.9*10^{-39}..1.7*10^{38}$
Single	Одинарной точности	8	4 байта	$1.5*10^{-45}..3.4*10^{38}$
Double	Двойной точности	16	8 байт	$5*10^{-324}..1.7*10^{308}$
Extended	Повышенной точности	18	10 байт	$1.9*10^{-4951}..$ $1.1*10^{4932}$
Comp	Сложный (длинное целое)	19	8 байт	$-2*10^{18}..2*10^{18}$

Описание переменных. Все переменные, используемые в программе, должны быть описаны в описании VAR. При этом указывается имя и тип каждой переменной.

```
Var  Имя:Тип;
```

Переменные одинакового типа можно перечислять через запятую. Например:

```
Var  a,b:real;
     m,n: integer;
     b1,b2:byte;
```

По описанию переменных компилятор резервирует для хранения их значений место (участок) в оперативной памяти. С именем переменной автоматически соотносится физический адрес байтов оперативной памяти, выделенных для ее хранения. Тип переменной указывает, сколько байтов необходимо отвести в памяти для хранения ее значения.

Структура и оформление программ в языке Турбо-Паскаль. Каждая программа должна отвечать жестким требованиям оформления. Написания программы начинается следующим образом:

```
<Заголовок>
<Раздел описаний>
<Раздел команд>
```

Заголовок представляет собой служебное слово Program и имя программы, например Program Prog1. Имя программы должно быть написано латинскими буквами без пробела.

Раздел описаний содержит указания на те свойства используемых элементов, которые определяют объем необходимой для их хранения оперативной памяти и характер использования этих элементов при выполнении вычислений:

```
PROGRAM <ИМЯ>;                    {--- заголовок программы}
                                {--- Раздел описаний:}
USES ... {список используемых модулей}
CONST ... {список констант, используемых в программе }
TYPE ... {описание конструируемых типов данных}
VAR ... {список используемых переменных с указанием
типа}
...
PROCEDURE ИМЯ(параметры); {тексты собственных процедур}
...
FUNCTION ИМЯ(параметры):тип; {тексты собственных ф-ций}
...
                                {--- Раздел выполнения:}
BEGIN {начало исполняемой части}
... {исполняемые операторы программы}
END. {конец программы}
```

Каждый элемент программы должен иметь уникальное в пределах программы имя (идентификатор) и быть описан в разделе описаний.

Имя должно состоять из букв латинского алфавита, цифр и знака подчеркивания, причем должен начинаться с буквы и не содержать пробелов.

Каждая часть раздела описаний начинается служебным словом, обозначающим характер описываемых элементов.

Если какие-то элементы отсутствуют в программе, то и служебное слово не приводится.

Кроме того, в раздел описаний включаются полные тексты используемых в программе подпрограмм (функций и процедур).

Раздел операторов любой программы начинается с ключевого слова *Begin*, после которого начинаются эти самые операторы. После *Begin* знак «;» не ставится. Заканчивается текст программы ключевым словом *End.*

В конце программы ставится точка. Все, что написано после точки не включается в текст программы и не воспринимается компилятором.

Конструкция *begin . . end* называется операторными скобками (составным оператором). Все операторы, записанные внутри, рассматриваются как единое целое и выполняются полностью в процессе работы программы.

Пояснения:

1. Каждый оператор оператора программы заканчивается знаком «точка с запятой» **(;)**
2. Ключевые слова отделяются пробелами.
3. Комментарий ограничивается фигурными скобками **{**…..**}** или (*...*)
4. Текст можно набирать как большими, так и малыми буквами.
5. Имя переменной может содержать до 63 символов.
6. Проверить программу - значит мысленно выполнить ее #.

Оператор присваивания. Оператор присваивания используется для назначения (присваивания) переменной определенного значения. Это основной оператор работы с оперативной памятью компьютера в программе.

Оператор присваивания обозначается «**:=**» и всегда имеет две части: справа – аналитическое выражение (формула), которое должно быть вычислено, слева – имя переменной, которой будет присвоен результат.

Имя := Выражение;

Оператор присваивания выполняется в 2 этапа:

- вычисление выражения в правой части (при этом вместо имен переменных подставляются их ЗНАЧЕНИЯ, хранящиеся в оперативной памяти (ОП);
- запись результата в ячейку памяти, обозначенную именем, указанным в левой части оператора;

Имя переменной в *левой части* оператора присваивания указывает, что в данном месте нужно использовать ее *значение*.

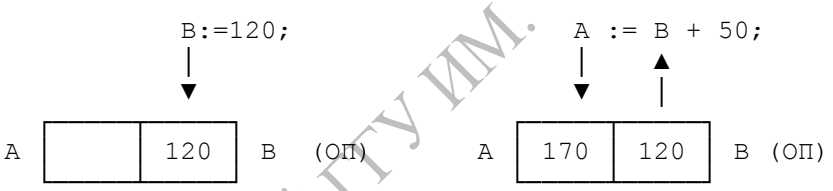
Имя переменной в *правой части* оператора присваивания указывает, по какому адресу (обозначенному именем) в оперативной памяти записать результат.

Когда мы присваиваем переменной значение, это приводит к тому, что значение будет записано в оперативную память по адресу, который соответствует имени переменной.

Пример 1. Рассмотрим выполнение простейших вычислений для фрагмента программы:

```
Program Prim21;
Var a,b,c:real;
Begin
 B:=120;
 A:=B+50;
 C:=A+B;
 . . . .
```

В этом примере описанием переменных зарезервировано в памяти 3 ячейки по 6 байт. При этом выполняется запись чисел в оперативную память:



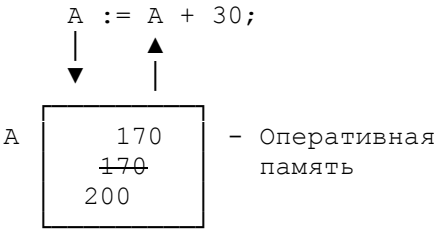
Примечание. Тип выражения в правой части оператора присваивания должен совпадать с типом переменной в левой части, в противном случае на экране появится сообщение об ошибке

Error 30: Type Mismatch – несоответствие типов.

Оператор присваивания может иметь вид

```
A := A + 30;
```

Это значит, что из оперативной памяти извлекается значение переменной, указанной в правой части (старое значение), вычисляется результат, и далее результат (новое значение) записывается в ту же самую ячейку памяти вместо старого



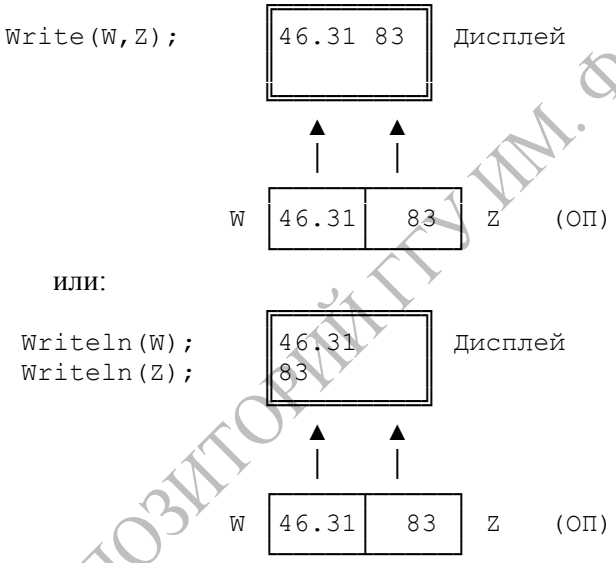
Операторы вывода результатов на экран дисплея. Значения, хранящиеся в оперативной памяти, не видны программисту. Но их можно показать на экране, если записать в программе специальную *процедуру вывода:*

```
Write(x1,x2, . .,xn);
Writeln(x1,x2, . ., xn);
```

Где x1,x2, . ., xn – параметры процедуры. Это могут быть имена переменных, обращения к функциям, символьные или строковые константы.

```
Write(список переменных);
```
или
```
Writeln(список переменных);
```

Например:



или:

После выполнения этих операторов на экран начиная с текущей позиции курсора будут выведены значения параметров x1, x2, . ., xn.

Примеры программ:

```
Program Prim22;                    Экран:
Var a:integer;                      10
Begin                               a
  A:=10; Writeln(a);
  Writeln('a');
End.
```

```
Program Prim23;                Экран:
Var a:integer;                   a= 10
Begin                            Возведение в квадрат= 100
  A:=10;
  Writeln('a= ',a);
  Writeln('Возведение в квадрат= ',a*a);
End.
```

```
Program Prim 24;                    Экран:
Var r,s:real;               s= 6.2831853071E+0000
Begin
 r:=1;
 s:=2*Pi*r;
 writeln('s= ',s);
end.
```

```
Program Prim25;
Begin
  Writeln(sin(1.08));
End.                    {Результат на экране:  0,881957}
```

Форматирование вывода. После имени переменной можно указать формат отображения ее значения на экране

```
Write(a:w:d);
```

где w - общее количество знакомест, занимаемых числом на экране (вместе с десятичной точкой); d – количество знакомест для дробной части. Точка занимает одно знакоместо.

Если переменная целочисленная или строковая, то указывается только общее количество символов.

В списке вывода можно использовать строковые константы, которые копируются на экране

```
Program Prim26;
Var a:real;  b:integer;  c:string;
Begin
 A:=1.234;  b:=12;   c:='computer';
 Writeln('Значение a = ',a:8:2);
 Writeln('a = ',a:5:3);
 Writeln(b);
```

```
 Writeln(b:10);
 Writeln(c);
 Writeln(c:12);
End.
```

Операторы ввода данных с клавиатуры. В предыдущих примерах начальные значения переменных задавались с помощью оператора присваивания. Это делает программу простой и наглядной, но пригодной для решения всего одной задачи – именно с этими числами.

В то же время, цель программирования – создание как можно более универсальных программ, пригодных без переделки к решению многих задач.

Один из способов создания программ, решающих целый класс однотипных задач – указание значений не в самой программе, а в ходе ее выполнения.

Для этого используются операторы ввода – команды, по которым компьютер приостанавливает выполение и ждет набора чисел на клавиатуре.

В этом случае программа решает задачу как бы в общем виде, для любых значений известных величин.

Важно понимать, что пользователь может обозначать величины, с которыми он работает, своими именами (обозначения пользователя). В тексте программы можно использовать любые другие имена (обозначения программиста), потому что они все равно не видны пользователю. Ему важно, чтобы над величинами, которые он ввел в определенном порядке, выполнялись нужные ему действия.

Этот прием полностью реализуется в дальнейшем при разработке подпрограмм.

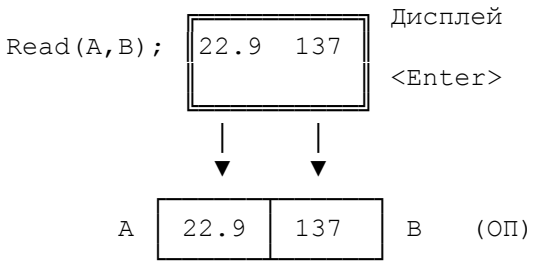
Для того, чтобы программа была универсальной, параметры должны вводиться с клавиатуры.

Для ввода данных используют процедуру (оператор) readln (реже – read).

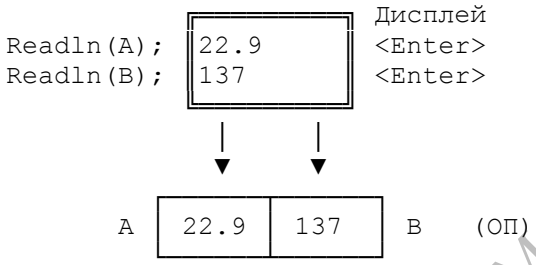
```
Readln(x1,x2, . .,xn);
```

Оператор считывает с клавиатуры вводимые значения и присваивает их указанным в списке ввода переменным x1,x2, . .,xn соответственно. Вводить значения необходимо через пробел.

Например:



или:



Удобно всегда при вводе исходных данных с клавиатуры перед оператором ввода Readln использовать *поясняющий* оператор вывода Writeln, в котором с помощью строковой константы на экране появляется указание, что именно следует ввести в данный момент.

```
Program Prim27;
Var d1,d2,d3:real;
Begin
Write('введите значение a_');
Readln(d1);
Write('введите значение b_');
Readln(d2);
D3:=d1+d2;
Write('сумма a+b = ', d3:8:2);
Writeln('Для завершения нажмите <Enter>');
Readln;
End.
```

Для удобства просмотра результатов в данном примере в конце программы использован пустой оператор Readln, приостанавливающий работу программы до нажатия клавиши [Enter].

Пример составления программы на языке Турбо-Паскаль. *Задача.* Составить программу для вычисления суммы двух чисел.

Решение.

А) Система математических соотношений

$$C=a+b$$

Б) Таблица имен переменных.

Имя переменной служит для указания ячейки памяти, в которой хранится число. Имя может быть любым, но всегда следует помнить, какой величине оно соответствует. Для этого удобно составить таблицу имен переменных:

Назначение	Математическое обозначение	Программное имя
1-е слагаем	a	alfa
2-е слагаем	b	beta
сумма	c	gamma

```
PROGRAM Sum1;
{Лабораторная работа 1. Задача 1.
Составить программу для вычисления суммы двух чисел.
Составил: ...}
VAR alfa,beta,gamma:real;
BEGIN
 WRITELN('Программа вычисления суммы двух вещественных
 чисел');
 WRITELN(' Составил ... 12.10.2004}
 WRITELN;
 WRITELN('Введите значения двух слагаемых:');
 READLN(alfa,beta);
 gamma:=alfa+beta;
 WRITELN('Сумма заданных чисел = ',gamma);
 READLN;
END.
```

Вопросы для самоконтроля

1. Структура программы на языке Турбо-Паскаль.
2. Операторы вывода информации на экран Write и Writeln.
3. Форматирование выводимой на экран информации.
4. Оператор присваивания.

5. Операторы ввода информации с клавиатуры Read и Readln.

6. Форматированный вывод числовых величин.

6. Встроенные вычислительные функции языка Турбо-Паскаль.

7. Основные действия при наборе текста программы и запуске программы на выполнение.

СОДЕРЖАНИЕ ЗАДАНИЯ

Для каждой задачи выписать расчетные формулы, записать таблицу переменных и составить программу, содержащую необходимые комментарии, подробный запрос на ввод данных, вычисления и полный ответ при выводе результатов.

Набрать, отладить и выполнить программу. Создать EXE-файл программы.

Провести не менне 5 расчетов с числовыми данными, подобранными самостоятельно, и записать результаты в отчет по работе.

1. Вычислить время падения камня на поверхность земли с высоты H и его скорость в момент падения. Сопротивлением воздуха пренебречь.

2. В сеть напряжением 220 В включены последовательно лампочка мощностью P1 Вт и электрический звонок мощностью P2 Вт. Вычислить величину тока в цепи.

3. Вычислить тормозной путь автомобиля, движущегося со скоростью v0, и имеющего массу m. Коэффициент трения k.

ТЕМА 5. ВСТРОЕННЫЕ МАТЕМАТИЧЕСКИЕ ФУНКЦИИ

Встроенные алгебраические функции. В языке Turbo-Pascal имеется модуль SYSTEM, содержащий процедуры и функции для вычисления математических функций или помогающие упростить вычисление или обработку данных во время работы программы.

Модуль подключается к программе автоматически, поэтому его имя не указывается в разделе USES. По этой причине программные

средства модуля называют *встроенными*, так как их использование не требует никаких дополнительных описаний.

Встроенная константа pi= 3.1415926535897932385

Алгебраические функции:

ABS (x:real):real; - вычисление модуля числа.
FRAC(x:real):real; - выделяет дробную часть числа.
INT(x:real):real; - выделяет целую часть числа.
SQR(x:real):real; - возведение в квадрат числа.
SQRT(x:real):real; - извлечение квадратного корня числа.
EXP(x:real):real; - возведение экспоненты в степень.
LN(x:real):real; - возвращает значение натурального логарифма.

Тригонометрические функции:

SIN(x:real):real; - производит вычисление синуса угла (угол x задается в радианах)

COS(x:real); - производит вычисление косинуса угла (угол x задается в радианах)

ARCTAN(x:real):real; - производит вычисление угла по его тангенсу (угол получается в радианах)

Аргумент тригонометрических функций расположен в I и IV квадрантах: $x \in [-\pi/2, +\pi/2]$.

Функция – генератор случайных чисел:

Random:real; - выдает случайным образом вещественные числа в пределах от 0 до 1.

Random(n:integer):integer; – выдает случайным образом целые числа в пределах от 0 до n-1.

Randomize; – процедура, изменяющая начальные установки генератора случайных числе при каждом запуске программы. Без нее каждый раз будет получаться одна и та же последовательность случайных чисел. Пример использования:

```
Randomize;
For i:=1 to 10 do writeln('число=',3+random(5));
```

Функции преобразования типа:

Trunc(x:real):integer; - отбрасывает дробную часть числа

Round(:real):integer; - округляет число с повышением

Odd(x:real):boolean; - принимает значения true при x - нечетном, false – при x - четном. Пример использования:

```
For i:=1 to 10 do begin
  if Odd(i) then writeln('i - нечетное')
     else writeln ('i - четное');
end;
```

Вычисление других функций с помощью встроенных. При решении физических задач используются и другие элементарные функции, не включенные в состав встроенных в языке Турбо-Паскаль. При необходимости их можно вычислить, выразив через встроенные функции.

Вычисление вещественной степени положительного числа:

$$a^b = e^{b\ln a}, \quad a > 0;$$

Вычисление логарифма по заданному основанию:

$$\log_a b = \frac{\ln b}{\ln a}$$

Перевод угла из градусной меры в радианную, из радианной меры в градусную:

$$r = \frac{g}{180°}\pi; \qquad g = \frac{r}{\pi}180°.$$

Остальные тригонометрические функции:

$$tgx = \frac{\sin x}{\cos x}, \quad x \neq \frac{\pi}{2} + \pi k \qquad ctgx = \frac{\cos x}{\sin x}, \quad x \neq \pi k$$

Обратные тригонометрические функции:

$$\arcsin x = arctg\left(\frac{x}{\sqrt{1-x^2}}\right); \quad \arccos x = arctg\left(\frac{\sqrt{1-x^2}}{x}\right);$$

$$arcctgx = arctg\left(\frac{1}{x}\right);$$

Гиперболические функции:

$$shx = \frac{e^x - e^{-x}}{2}; \quad chx = \frac{e^x + e^{-x}}{2};$$

$$thx = \frac{e^x - e^{-x}}{e^x + e^{-x}}; \quad cthx = \frac{e^x + e^{-x}}{e^x - e^{-x}};$$

Обратные гиперболические функции:

$$arshx = \ln\left(x + \sqrt{x^2 + 1}\right); \quad archx = \ln\left(x + \sqrt{x^2 - 1}\right);$$

$$arthx = \frac{1}{2}\ln\left(\frac{1+x}{1-x}\right); \quad arcthx = \frac{1}{2}\ln\left(\frac{x+1}{x-1}\right).$$

Упражнения

1. Записать операторы для выполнения вычислений:

y=4z-1,2; f=sin(x^3) - 0.5*ln*(cosD); $r = t^{2,3} + \frac{a-b}{a+b}$;

2. Какие из приведенных операторов присваивания записаны неправильно и почему ?

```
1) ab:=arcsin(x)+1.2;        2) k1:=k1+1;
3) alfa=124;                 4) x2:=cosW;
```

3. Указать ошибки в записи оператора:

```
w = a**x/sin(x)+b(cos(a)+t*d)-3x
```


ТЕМА 6. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ОПЕРАТОРОВ ВЫБОРА

Логические отношения и логические операции. Операторы выбора позволяют выбрать один из нескольких вариантов вычислений в зависимости от выполнения заранее сформулированного условия. В языках программирования условия шифруются с помощью логических операций и логических отношений.

Логическое отношение требует от компилятора проверить на истинность соотношение между двумя однотипными величинами.

В тексте программы различные соотношения указываются с помощью знаков сравнения:

> больше <= меньше или равно

>= больше или равно <> не равно

< меньше = равно

Например, логическое отношение записывается: a > (b+c). На практике такая запись означает: верно ли, что a > (b+c) ?

Результат такой проверки является логическим утверждением (тип boolean), принимающего значения TRUE (истина) или FALSE (ложь).

Пример 1:

```
var
 a,b:real; ( a>b --> false )
 L:boolean; ( a<b --> true )
begin ( a=b --> false )
 a:=25;
 b:=26;
 L:=a>b;
 print(L); ===> {False}
end.
```

Для записи сложных условий, моделирующих рассуждения человека при анализе ситуации, используются *логические операции*, то есть, действия над логическими величинами.

Операция NOT (отрицание) действует на отдельную логическую величину и меняет ее значение на противоположное:

Операция AND - логическое "И": результат "истина" (TRUE), если И первая, И вторая переменные истинны);

Операция OR - логическое "ИЛИ": результат "истина" (TRUE), если хотя бы одна логическая величина истинна - ИЛИ первая, ИЛИ вторая;

Операция XOR - исключающее "ИЛИ": результат "истина" (TRUE), если хотя бы одна логическая величина истинна, за исключением случая, когда истинны обе одновременно.

Таблица истинности для логических операций (F-False, T-True):

A B	A AND B	A OR B	A XOR B	NOT A	NOT B
T T	T	T	F	F	F
T F	F	T	T	F	T
F T	F	T	T	T	F
F F	F	F	F	T	T

Логические операции часто используются для проверки принадлежности числовой переменной заданному интервалу:

математически:	в программе:
$-1 \le x \le 1$	(x>= -1) AND (x<=1)

Так как логические операции выполняются раньше логических отношений, необходимо отношения записывать в скобках.

Сравнение символов соответствует сравнению их числовых кодов:

'R' < 'W' {=> True}

Оператор выбора IF. Общий вид оператора:

```
IF <Условие> THEN <Оператор1> ELSE <Оператор2>;
```

Если условие принимает значение TRUE, то выполняется Оператор1, иначе - выполняется Оператор2.

На месте, обозначенном Оператор1 и Оператор2, может быть только один оператор языка Турбо-Паскаль, или же составной оператор (см. следующий пункт).

Существует модификация оператора, в которой отсутствует Оператор2, - что соответствует структуре "Обход":

```
IF <Условие> THEN <Оператор>;
```

Пример 2: вычисление модуля вещественного числа

$$|x| = \begin{cases} x, & x \ge 0 \\ -x, & x < 0 \end{cases}$$

```
IF x<0 then modul:=-x ELSE modul:=x;
```

Отметим, что здесь заданы взаимоисключающие условия, то есть, при выполнении одного условия второе не может быть выполнено.

Пример 3: Вычисление кусочно-непрерывной функции

$$f = \begin{cases} a \cdot b, & y < 3{,}4 \\ a + b, & y \ge 3{,}4 \end{cases}$$

```
IF y>=3.4 THEN f:=a*b ELSE f:=a+b;
```

Пример 4: совершение действия при выполнении сложного условия.

F=2*z, если a>0, p=s, z='Y'

F=0, если хотя бы одно условие не выполнено

```
IF (a>0) AND (p=s) AND (z='Y') THEN f:=2*z ELSEf:=0;
```

Пример 5: Вычисление кусочно-непрерывной функции с большим количеством элементов.

$$f=\begin{cases} t+w, & x\le -1 \\ \sin t+w, & -1<x\le 1 \\ t\cdot w, & 1<x\le 5 \\ \dfrac{1}{t+w+1}, & x>5 \end{cases}$$

```
IF x<=-1 THEN f:=t+w;
IF (x>-1) AND (x<=1) THEN
   f:=sin(t)+w;
IF (x>1) AND (x<=5) THEN
   f:=t*w;
IF x>5 THEN f:=1/(t+w+1);
```

Такая последовательность операторов позволяет организовать множественный выбор нужного действия в зависимости от значения *вещественного* параметра x.

Составной оператор. Составной оператор - это последовательность любых выполняемых операторов, заключенных в операторные скобки BEGIN ... END;

Составной оператор используется в тех случаях, когда в рабочей области оператора управления может находиться ТОЛЬКО ОДИН оператор, а по логике решения задачи необходимо выполнить несколько операторов. Общий вид составного оператора:

```
begin
  <оператор 1>;
  <оператор 2>;
  <... >;
  <оператор n>;
end;
```

Пример 6: выбор одного из двух вариантов многошаговых вычислений. Вычислить значения величин k, m, n при заданном x по правилу:

t=cos(x); при t≤0.5 k=a+b, m=a*b, n=sin(a)

```
.....
t:=cos(x);
IF t<=0.5 THEN
 begin
   k:=a+b;
   m:=a*b;
```

```
| при t>0.5 k=cos(b), m=a-b, n=a*a        n:=sin(a);
                                         end
                                       ELSE
                                        begin
                                          k:=cos(b);
                                          m:=a-b;
                                          n:=a*a;
                                        end;
```

Вопросы для самоконтроля

1. Логические отношения.
1. Логические операции.
2. Структура и порядок выполнения оператора выбора IF.
3. Ситуации, требующие использования составного оператора.

СОДЕРЖАНИЕ ЗАДАНИЯ

ЗАДАЧА 1. Для заданного варианта N составить программу, содержащую необходимые комментарии, подробный запрос на ввод данных с клавиатуры, вычисления и полный ответ при выводе результатов на экран.

Набрать и отладить программу.

Выполнить тестирование программы – вычисления с исходными данными, для которых легко предсказать результат. Выполнить не менее 3 тестов. Результаты записать в отчет.

Создать EXE-файл программы. Выполнить вычисления для 3 различных вариантов исходных данных (приведенных в задании и подобранных самостоятельно). Результаты записать в отчет.

1. Два мяча, массы которых m1, m2 были брошены вертикально вверх с начальными скоростями v1, v2 соответственно. Определить, какой из них поднимется на большую максимальную высоту.

2. Задана площадь круга S1 и площадь квадрата S2. Определить, поместится ли круг внутри квадрата. Ответ вывести в текстовом виде.

3. Дано квадратное уравнение $ax^2+bx+c=0$. Составить программу для вычисления вещественных корней уравнения. В случае, когда вещественных корней нет, вывести текстовое сообщение.

ЗАДАЧА 2. Составить программу для вычисления значений кусочно-непрерывной функции. Исходные данные ввести с клавиатуры. Результат вычислений вывести на экран.

Выполнить тестирование программы – вычисления для таких значений исходных данных, для которых результат легко можно предсказать. Выполнить не менее 3 тестов. Результаты записать в отчет.

Создать EXE-файл программы. Выполнить вычисления для 3 различных вариантов исходных данных. Результаты записать в отчет.

$$c = \begin{cases} x^2 + y^2 + \sin(x), x - y = 0 \\ (x - y)^2 + \cos(x), x - y > 0 \\ (y - x)^2 + tg(x), x - y < 0 \end{cases}.$$

ТЕМА 7. СТРУКТУРНЫЕ ОПЕРАТОРЫ ЦИКЛА В ЯЗЫКЕ ТУРБО-ПАСКАЛЬ

Циклические алгоритмы и операторы цикла. Алгоритм называется циклическим, если он содержит многократное выполнение одних и тех же действий при различных значениях одних и тех же величин. Число повторений может быть задано в явной или неявной форме.

Существует три основных вида операторов цикла: цикл while, цикл repeat и цикл for.

Формат оператора цикла с предусловием

```
WHILE <условие> DO <оператор>;
{ПОКА условие истинно ВЫПОЛНЯТЬ оператор}
```

- в условие должны входить величины, определенные ранее (перед оператором цикла);

- параметры, входящие в условие, должны изменяться внутри цикла.

Выражение, с помощью которого производится управление повторением оператора, должно иметь логический тип. Вычисление его производится до того, как внутренний оператор будет выполнен.

Внутренний оператор выполняется повторно до тех пор, пока выражение принимает значение True.

Если выражение с самого начала принимает значение False, то оператор, содержащийся внутри оператора цикла while, не выполняется ни разу.

Элементы, составляющие цикл:

1 - начальные условия

2 - оператор цикла

3 - рабочая область

4 - модификация параметра

Пример: вывести таблицу квадратов чисел от 100 до 200 с шагом 10 n=100..200 (10)

```
Program pr1;
var n:byte;
n:=100;                                {1}
WHILE n<=200 DO                        {2}
 begin
  writeln('n= ',' квадрат = ',n*n); {3}
  n:=n+10;                             {4}
 end;
```

Пример: вычисление суммы ряда с заданной точностью $S=\sum_{k=1}^{\infty}\frac{1}{k^2}$, $\varepsilon=10^{-5}$.

```
Program pr2;
{вычисление суммы ряда с заданной точностью}
const
 delta=1e-5;
var
 k:integer;
 s,a:real;
begin
 s:=0; k:=1; a:=1/sqr(k);
 WHILE a>delta DO
 begin
 s:=s+a;
 k:=k+1;
 a:=1/sqr(k);
 end; {while}
```

```
 writeln('учтено ',k-1,' слагаемых сумма = ',s);
end.
```

Оператор цикла с постусловием Repeat....Until. Формат оператора цикла "ПОВТОРЯТЬ ДО ПОЛУЧЕНИЯ УСЛОВИЯ":

```
 REPEAT          {Повторять}
 оператор;
 оператор;
 ........
 оператор;
 UNTIL Условие;  {До выполнения условия}
```

Пример: организация контроля корректности вводимых данных с помощью оператора REPEAT ... UNTIL

```
Program Pr5;
{контроль вводимых данных x>0, n<10 и т.п.}
...
REPEAT
 writeln('Задайте параметр x>0');
 Readln(x);
 writeln('Задайте показатель n<10');
 Readln(n);
UNTIL (x>0) and (n<10);
...
```

Результатом выражения должен быть результат булевского типа. Операторы, заключенные между ключевыми словами repeat и until, выполняются последовательно до тех пор, пока результат выражения не примет значения True. Последовательность операторов выполняется по крайней мере один раз, поскольку вычисление выражения производится после каждого выполнения последовательности операторов.

Приведем примеры:

```
 repeat
 Write('Enter Value(0..9):');
 Readln(I);
 until(I>=0) and (I<=9);

 program Ratio;
 var
 A, B: Integer;
 Ratio: Real;
```

```
Ans: Char;
begin
repeat
Write('ВВедите два числа');
Readln(A,B);
Ratio:=A/B;
Writeln('Отношение равно', Ratio);
Writeln('Повторить? (Y/N)');
Readln(Ans);
until Upcase(Ans)='N';
end.
```

В этой программе повторяется выполнение операторов, пока ответ на вопрос - n или N (Повторить? Y/N). Другими словами repeat и until, повторяются, до тех пор, пока значение выражения при until не будет True.

Существуют три основных отличия от цикла while:

- операторы в цикле repeat выполняются хотя бы один раз, потому что проверка выражения осуществляется в конце тела цикла. В цикле while, если значение выражения False, тело его пропускается сразу.

- цикл repeat выполняется пока выражение не станет True, в то время, как цикл while выполняется до тех пор, пока выражение имеет значение True.

- оператор WHILE обеспечивает повторение ОДНОГО оператора (или составного оператора), оператор REPEAT . . UNTIL может содержать внутри любое количество операторов. При использовании этого цикла не используются слова begin...end, как в случае с циклом while.

При замене одного типа цикла на другой необходимо на это обращать особое внимание. Рассмотрим программу Prim2, где цикл while заменен на цикл repeat:

```
program Prim2;
var
Count: Integer;
begin
Count:=1;
repeat
Writeln('Параметр цикла = ',Count);
Count:=Count+1;
until Count > 10;
```

```
Writeln('Цикл завершен');
end.
```

Отметим, что теперь переменная Count проверяется на значение больше 10 (а в while было Count <= 10).

Оператор арифметического цикла for. Организует цикл для целочисленных значений параметра цикла.

Общий вид оператора:

```
FOR параметр:= Начальн. TO Конечн. DO оператор;
{Для каждого значения параметра от Начального до
        Конечного выполнять Оператор}
```

Параметр - должен быть переменной целого типа. Когда оператор FOR использует ключевое слово TO, значение управляющей переменной увеличивается при каждом повторении на 1. Если начальное значение превышает конечное значение, то содержащийся в теле оператора for оператор не выполняется.

Оператор FOR можно рассматривать как специальный случай оператора WHILE для целочисленного параметра цикла

```
k:=1;
while k<=10
do begin          То же    For k:=1 TO 10 DO Оператор;
 Оператор         самое:
 k:=k+1;
end;
```

Если необходимо при выполнении цикла уменьшать значение параметра (также на 1), то используется ключевое слово DOWNTO. Если начальное значение в таком операторе меньше, чем конечное значение, то содержащийся в теле оператора цикла оператор не выполняется.

```
FOR параметр:= Начальн. DOWNTO Конечн. DO оператор;
```

Пример: вычисление квадратов целых чисел

```
Program pr1;
var n:byte;
begin
```

```
   FOR n:=10 to 20 do writeln('n= ',' квадрат = ',n*n);
  end.
```

Если оператор, содержащийся в теле оператора for, изменяет значение управляющей переменной, то это является ошибкой.

После выполнения оператора for значение управляющей переменной становится неопределенным, если только выполнение оператора for не было прервано с помощью оператора перехода.

Главный недостаток цикла for - это возможность уменьшить или увеличить индекс только на 1.

Основные преимущества - краткость, возможность использования символьного и перечислимого типа в диапазоне значений.

Если число повторений заранее известно, то подходящей конструкцией является оператор for. В противном случае следует использовать операторы while или repeat.

Пример практического применения операторов цикла. Пример: Составим программу для вычисления и вывода на печать таблицы значений функции Y=a*sin(b*x) при x=0, 0.2, 0.4, ..., 5 для a=2.5, b=0.34. С использованием оператора WHILE программа имеет вид

```
 Program Prim1;
 {комментарий к программе}
 const x1=0.0; x2=5.0; h=0.2;
 var a,b,x,y : real;
 begin
 writeln('Программа строит таблицу значений функции');
 writeln('Y=a*sin(b*x)');
 writeln('Задайте значения параметров функции a, b');
 readln(a,b);
{1-----------------------}
 x:=x1;
 while x<=x2 do begin
 y:=a*sin(b*x);
 writeln('x= ',x:6:3,' ':5,'y= ',y:6:4);
 x:=x+h;
 end;
{2-------------------------}
 end.
```

С использованием оператора REPEAT фрагмент программы {1}-{2} выглядит следующим образом:

```
{1-------------------------}
 x:=x1;
 repeat
 y:=a*sin(b*x);
 writeln('x= ',x:6:3,' ':5,'y= ',y:6:4);
 x:=x+h;
 until x>x2;
{2-------------------------}
```

С использованием оператора FOR фрагмент программы {1}-{2} выглядит следующим образом:

```
{1-------------------------}
 n:=(x2-x1)/h;
 for i:=0 to n do begin
 x:=x1+i*h;
 y:=a*sin(b*x);
 writeln('x= ',x:6:3,' ':5,'y= ',y:6:4);
 end;
{2-------------------------}
```

Вопросы для самоконтроля

1. Как записывается и как работает оператор FOR ?
2. Для организации каких циклов применим оператор FOR ?
3. В чем отличие оператора WHILE от оператора REPEAT ?
4. Как программируются циклические алгоритмы с явно заданным числом повторений цикла (три варианта) ?
5. Как программируются циклические алгоритмы с незаданным числом повторений цикла (два варианта) ?

СОДЕРЖАНИЕ ЗАДАНИЯ

ЗАДАЧА 1. Для решения задачи варианта составить три варианта программы с использованием различных операторов организации циклов. Результаты вывести на экран и записать в файл.

1. Составить программу для перевода градусов в радианы, выводящую таблицу перевода от 0° до 180° через каждые 5°.
2. Вычислить площади кругов диаметрами 0.1, 0.2, … 1.0 м.
3. Составить программу для вычисления значения выражения b= b=1+1/2^2 +1/3^2+1/4^2+...+1/N^2. Значение N ввести с клавиатуры. Вычислить результат для N=5 и N=20.
4. Вывести на печать таблицу n значений функции $y = ax^2 + b \cdot x + c$ при изменении x от x_1 до x_2 с шагом h = (x_2-x_1)/n. Коэффициенты a, b, c, границы интервала x_1, x_2 и число n ввести с клавиатуры. По таблице значений определить участки, содержащие корни уравнения, а также максимальное и минимальное значение функции для a = -1,14; b = -4,21; c = 3,25; x1=-2; x2=2; n=25.
5. Вычислить и вывести на экран таблицу значений двух функций y1=sin(x); y2=cos(x) для $x_{нач.} \le x \le x_{кон.}$ с шагом h. Исходные данные: $x_{нач.}$= -3; $x_{кон}$ = 3; h = 0,2.

ТЕМА 8. РЕШЕНИЕ ЗАДАЧ НА ОБРАБОТКУ МАССИВОВ

Определение. *Массивом* называется набор однотипных пронумерованных элементов, обозначенных общим именем. Элементы массива могут иметь любой базовый (определенный в ТП) тип, например, real, integer, byte и т.п.

Обозначение элемента состоит из имени всего массива и номера элемента в нем, записываемого в квадратных скобках.

Элементы *одномерного* массива нумеруются одним индексом:

Z	10	20	30	40	50
	Z[1]	Z[2]	Z[3]	Z[4]	Z[5]

Двумерный массив (матрица) состоит из строк и столбцов, номера которых указываются раздельно:

W							
	W[1,1]	W[1,2]	W[1,3]	W[1,4]	W[1,5]	W[1,6]	W[1,7]
	W[2,1]	W[2,2]	W[2,3]	W[2,4]	W[2,5]	W[2,6]	W[2,7]
	W[3,1]	W[3,2]	W[3,3]	W[3,4]	W[3,5]	W[3,6]	W[3,7]

Массивы удобно использовать для хранения и обработки большого количества чисел, в частности, при обработке результатов физического эксперимента, а также при реализации на ПК численных методов.

Способы описания массивов. Имя, количество и базовый тип элементов в массиве выбирает программист с учетом решаемой задачи, поэтому перед использованием массив должен быть описан в разделе описаний программы.

Способ 1. При составлении несложных программ достаточно описать массив в разделе описания переменных:

```
Var  <ИМЯ> : ARRAY [диапазон индексов] OF <Тип>;
```

Например:

```
Var Z:array [1..5] of integer;
    W:array [1..3,1..7] of real;
```

Способ 2. При использовании массивов в подпрограммах необходимо отдельно описать устройство массива как самостоятельный тип, а затем уже описать массив как переменную этого типа (имя типа выбирает программист):

```
Type <имя_типа>=ARRAY [диапазон индексов] OF <Тип>;
Var <имя_переменной> : <имя_типа>;
```

Например:

```
Type
 massiv=array [1..5] of integer;
 matrica=array [1..3,1..7] of real;
Var
 Z:massiv;
 W:matrica;
```

Описание типа задает только *способ* размещения данных в оперативной памяти и допустимые операции над ними, а описание переменных *практически* выделяет ячейки памяти для их хранения.

Использование элементов массива. Отдельный элемент массива используется точно так же, как и переменная такого типа:

```
. . .
w[1,1]:=0.123;              - присваивание
readln(w[3,7]);             - ввод с клавиатуры
w[1,5]:=sin(w[2,4])+1;      - использование в вычислениях
writeln(z[4]);              - вывод значения на экран
If z[2]>z[3] then k:=k+1; - сравнение
...
```

Программирование действий по обработке массивов. В большинстве случаев требуется выполнить одинаковые действия для всех элементов массива. Это значит: обработать 1-й элемент, затем 2-й, затем 3-й, и так далее. *Основная идея* программирования таких процессов заключается в том, что для описания *номера* элемента используется *параметр цикла*, который изменяется в нужных пределах.

Например, присвоение всем элементам массива Z возрастающих значений можно реализовать

- или так:

```
Z[1]:=10;
Z[2]:=20;
...
z[5]:=50;
```

- или вот так:

```
FOR k:=0 to 5 DO z[k]:=k*10;
```

В данном случае меняются номера элементов и их значения, но *действие* – одно и то же: присвоить элементу массива значение, равное его номеру, умноженному на 10.

Оператор цикла обеспечивает выполнение *всех* повторяющихся действий. Выигрыш в том, что программист должен записать не 5 строк программы, а всего одну!

Поэтому всегда при составлении программ стараются выделить повторяющиеся действия и запрограммировать их с помощью нескольких строк оператора цикла. Компьютер выполняет по команде оператора цикла все действия, но программируются они в сжатой форме, с помощью индекса цикла – номера элемента массива.

Способы заполнения массива. Для описанных выше одномерного массива Z и матрицы W опишем различные способы заполнения (просвоения значений элементам массива) и вывода значений на экран. Предполагается, что все используемые переменные описаны в программе.

А) Заполнение массива заданными значениями

```
. . .
for k:=1 to 5 do Data[k]:=k*10;
. . .
for i:=1 to 3 do
 for j:=1 to 7 do W[i,j]:=2*sin(i)+cos(j);
. . .
```

Б) Заполнение массива значениями, вводимыми с клавиатуры. При вводе следует организовать вывод поясняющей информации, чтобы пользователь видел, какой элемент следует ввести

```
. . .
for k:=1 to 5 do begin
  write('Введите ',i,'-ый элемент массива: ');
  readln(z[i]);
end;
. . .
for i:=1 to 3 do
 for j:=1 to 7 do begin
   write('элемент матрицы w[',i,j,']= ');
   readln(w[i,j]);
end;
. . .
```

В) Заполнение массива случайными значениями при помощи генератора случайных чисел.

```
Program Random_num;
Var z:array[1..5] of integer;
    W:array [1..3,1..7] of real;
    i:integer;
Begin
. . .
Randomize;
For i:=1 to 5 do z[i]:=Random(50);
. . .
For i:=1 to 3 do
 For j:=1 to 7 do w[i,j]:=Random;
. . .
```

Примечание 1. Процедура Randomize обеспечивает получение новой серии псевдослучайных чисел при каждом запуске программы.

Примечание 2. Если необходимо произвести заполнение вещественного массива числами из определенного отрезка [a;b], то применяем формулу:

```
w[i,j]:= a + random*(b-a);
```

Границы отрезка нужно будет описать в программе и присвоить им значения.

Вывод значений элементов массивов на экран. Вывод элементов также организуется в цикле, причем использование

операторов Write или Writeln позволяет вывести элементы или в строке экрана, или в столбик. Это важно учитывать при выводе матрицы, чтобы расположение элементов соответствовало структуре матрицы.

```
. . .
for k:=1 to 5 do write('z[',k,']= ',z[k]);
. . .
for i:=1 to 3 do begin
 for j:=1 to 7 do write(w[i,j]:0:4,'    ');
 writeln;
end;
. . .
```

Стандартные алгоритмы обработки массивов. При обработке массивов вычисляются различные их характеристики: сумма элементов, максимальное значение и т.д. Для хранения их в памяти необходимо в программе предусмотреть отдельные переменные.

А) Суммирование всех элементов массива. Для хранения значения *суммы* элементов выделяется отдельная переменная того же типа, что и элементы массива (или с большим диапазоном значений). Начальное значение ее полагается равным 0.

В операции суммирования S=z[1]+z[2]+z[3]+z[4]+z[5] выделяется повторяющиеся шаги, которые реализуются с помощью оператора цикла:

```
S:=0;                  {заполнение массива}
S:=S+z[1];             . . .
S:=S+z[2];     =>      S:=0;
S:=S+z[3];             For i:=1 to 5 do S:=S+z[k];
S:=S+z[4];             . . .
S:=S+z[5];
```

Для вычисления *произведения* используется такой же подход, но начальное значение произведения полагается равным 1, а в цикле произведение домножается каждый раз на очередной элемент

```
P:=1;
For i:=1 to 5 do P:=P*z[k];
```

Б) Нахождение максимального элемента матрицы и его индексов. Для хранения значения максимума, а также номера строки и столбца,

в которых расположен этот максимальный элемент, в памяти нужно выделить отдельные переменные.

```
{. . . заполнение массива . . .}
max:=z[1]; nmax:=1;
for i:=2 to 5 do begin
 if z[i]>max then begin
   max:=z[i];  nmax:=i;
end;

{. . . заполнение матрицы . . .}
mx:=w[1,1]; im:=1;  jm:=1;
FOR i:=1 TO 3 DO
 FOR j:=1 TO 7 DO
  If mx>w[i,j] then begin
    mx:=w[i,j];
    im:=i;
    jm:=j;
  end;
writeln(mx,im,jm);
```

Для нахождения минимального элемента выполняются те же действия, но знак неравенства в операторе выбора меняется на противоположный.

```
{. . . заполнение массива}
min:=z[1]; nmin:=1;
for i:=2 to 5 do begin
 if z[i]<min then begin
   min:=z[i];  nmin:=i;
end;
```

В) Перестановка элементов массива (обмен элементов местами) выполняется с использованием вспомогательной переменной, в которой временно хранится значение одного из элементов. Например, обмен 1-го и 5-го элементов:

```
temp:=z[1];
z[1]:=z[5];
z[5]:=temp;
```

Описание массивов и универсальность программ. В языке ТП количество элементов в массиве должно быть *явно указано* при

составлении программы и не может меняться при выполнении программы.

Это требование является не очень удобным с точки зрения универсальности программ. Действительно, в одном случае потребуется обработать всего 5 элементов, в другом 100, в третьем 48 и т.д.

Выход заключается в указании при описании массивов *наибольшего* для некоторого круга задач числа элементов, например, 1000, и использовании дополнительной переменной, которая будет указывать реально используемое число элементов.

Вопросы для самоконтроля

1. Описать типы данных и переменные, имеющие эти типы:

- массив из 20 элементов типа char, нумерация элементов начинается с 0;
- массив из 12 элементов типа byte; начальный элемент имеет номер 5;
- вещественная матрица 6*8, нумерация элементов начинается с 1.

2. Составить фрагмент программы, в котором всем элементам массива из 60 целых чисел присваивается значение 100.

3. Составить фрагмент программы, в котором элементы массива из 8 чисел типа byte вводятся с клавиатуры.

4. Составить фрагмент программы, в котором массив из 100 вещественных чисел заполняется с помощью генератора случайных чисел (ГСЧ).

5. Дополнить предыдущий фрагмент так, чтобы полученные элементы выводились поочередно на экран.

6. Дополнить предыдущий фрагмент вычислением суммы элементов заполненного массива и выводом результата.

7. Составить фрагмент программы, в котором матрица 12*12 целых чисел выводится на экран.

СОДЕРЖАНИЕ ЗАДАНИЯ

ЗАДАЧА 1. Составить три варианта программы с использованием различных операторов организации циклов. Результаты вывести на экран.

1. Массив из 15 целых чисел заполнить случайными целыми числами, принимающими значение от 2 до 8.

2. Массив из 8 целых чисел заполнить вводимыми с клавиатуры данными.

3. Массив из 15 целых чисел заполнить, присваивая элементам с четными номерами значение 20, а элементам с нечетными номерами - значение -1.

4. Массив из 12 целых чисел заполнить числами, начинающимися с 12 в обратном порядке.

5. Массив символов из 26 элементов заполнить прописными буквами английского алфавита.

ЗАДАЧА 2. Составить программу, выполняющую указанные действия. Элементы исходных массивов вводить с клавиатуры. Исходный массив и результаты вывести на экран.

1. Дан массив из 10 чисел. Получить новый массив путем умножения всех элементов исходного массива на его элемент, наибольший по абсолютной величине.

2. Дан массив из 20 чисел. Найти минимальное и максимальное из них.

3. Дан массив из 16 чисел. Найти сумму всех положительных и сумму всех отрицательных чисел.

4. В массиве M[1..12] среди элементов с номерами от 3 до 10 найти количество элементов, больших среднего арифметического элементов всего массива.

5. Найти куб минимального элемента массива с элементами a[i]:=0.34*i-sin(4.5*i);

6. Дан массив из 11 чисел. Найти количество и сумму квадратов положительных элементов массива, имеющих четные номера.

7. Дан массив из 30 чисел. Найти количество элементов, модуль которых не превышает 2.

ТЕМА 9. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ПРОЦЕДУР И ФУНКЦИЙ

Подпрограммы в языке Турбо-Паскаль. Часто в программе присутствуют группы операторов, выполняющих одинаковые действия, хотя обрабатываемые переменные и соответственно результат этих действий имеют разные значения. Такие одинаковые участки могут присутствовать и в разных программах.

В таких случаях удобно запрограммировать алгоритм указанных вычислений один раз в общем виде, то есть, описать как подпрограмму, с общим обозначением обрабатываемых величин в виде параметров, и в нужный момент использовать ее, вызывая из программы.

В языке Паскаль выделяют два вида подпрограмм: ПРОЦЕДУРЫ и ФУНКЦИИ. Основное отличие между ними состоит в том, что функция возвращает одно результирующее значение и может использоваться в выражениях наряду с переменными.

Структура процедур и функций. Полные тексты используемых подпрограмм располагаются в разделе описаний главной программы, до начала ее исполняемой части.

Структура процедур и функций такая же, как и структура основной программы, за исключением заголовка.

Для процедур используется следующий формат:

```
Procedure ИМЯ(формальные параметры);
   Раздел описаний
Begin
   Раздел операторов
End;
```

Здесь ИМЯ - имя процедуры. Раздел описаний, как и в случае программы, может содержать следующие элементы: описание меток, констант, типов, переменных, внутренние процедуры и функции.

Функции имеют такой же формат, что и процедуры, за исключением того, что они начинаются с заголовка FUNCTION, который заканчивается описанием типа данных для возвращаемого значения функции:

```
Function ИМЯ(параметры): тип результата;
   Раздел описаний
Begin
   Раздел операторов
End;
```

В разделе операторов функции должен содержаться по крайней мере один оператор присваивания, в котором имени (идентификатору) функции присваивается значение. Результатом функции является последнее присвоенное значение. Если такой оператор присваивания отсутствует или он не был выполнен, то значение, возвращаемое функцией, неопределено.

Наиболее распространенными ошибками при программировании функций являются:

- отсутствие оператора присваивания результата имени функции;

- использование имени функции в ходе вычислений возвращаемо го значения (невольная рекурсия);

```
Function St(x:real; n:byte):real;
{ошибка при компиляции: невольная рекурсия}
{вычисление целой степени вещественного числа}
var k:byte;
begin
  St:=1.0;
  For k:=1 to n do St:=St*x;
end;
```

Правильный вариант программирования:

```
Function St(x:real; n:byte):real;
{вычисление целой степени вещественного числа}
var k:byte;
    z:real;
begin
  z=1.0;
  For k:=1 to n do z:=z*x;
  St:=z;
end;
```

Параметры подпрограмм. Существует три типа параметров: значение, переменная и нетипизованная переменная. Они характеризуются следующим:

-группа параметров, перед которыми отсутствует ключевое слово VAR и за которыми следует тип, является списком параметров значений;

-группа параметров, перед которыми следует ключевое слово VAR и за которыми следует описание типа, является списком параметров-переменных;

-группа параметров,перед которыми стоит ключевое слово VAR и за которыми не следует тип, является списком нетипизованных параметров-переменных (рассматриваются в следующей лабораторной работе).

Параметры-значения. Формальный параметр-значение обрабатывается как локальная по отношению к процедуре или функции переменная, за исключением того, что он получает свое начальное значение из соответствующего фактического параметра при активизации процедуры или функции. Изменения, которые претерпевает формальный параметр-значение, не влияют на значение фактического параметра.

Параметры-переменные. Параметр-переменная используется в тех случаях, когда значение должно передаваться из процедуры или функции вызывающей программе. Соответствующий фактический параметр в операторе вызова процедуры или функции должен быть ссылкой на переменную. При активизации процедуры или функции формальный параметр-переменная заменяется фактической переменной, любые изменения в значении формального параметра-переменной отражаются на фактическом параметре.

В этом случае компилятор рассматривает параметр как адрес переменной указанного типа. Так как параметр является не самим значением, а его адресом, то результат любого действия над параметром внутри процедуры будет записан в память по адресу фактического параметра и, таким образом, станет доступен вызывающей программе.

Нельзя вызывать процедуру с объявленным параметром-переменной, указывая вместо этого параметра явное вычисляемое выражение, так как при вызове должен передаваться именно адрес переменной.

Документирование подпрограммы. Необходимым элементом разработки подпрограммы является ее документирование, которое заключается в подробном описании с помощью операторов комментария после заголовка PROCEDURE или FUNCTION. В них, в частности, необходимо указать:

1. Назначение
2. Описание параметров
3. Используемый метод расчета
4. Используемые внешние модули и процедуры
5. Примечания

Описание каждой процедуры и функции в программе необходимо по той простой причине, что детально разобраться в совместной работе большого количества подрограмм можно только при хорошем понимании деталей их функционирования.

Вопросы для самоконтроля

1. Назначение подрограмм
1. Подпрограмма FUNCTION. Правила оформления и обращения.
2. Подпрограмма PROCEDURE. Правила оформления и обращения.
3. Формальные и фактические параметры функций и процедур.
4. Глобальные и локальные переменные.
5. Параметры-значения и параметры-переменные
6. В чем отличие в правилах оформления и вызова процедур и функций ?

СОДЕРЖАНИЕ ЗАДАНИЯ

ЗАДАЧА 1. Составить программу, в которой содержится функция, выполняющая указанные действия. В самой программе выполняется ввод исходных данных, вызов функции для реализации указанных действий и вывод результатов на экран.

1. Вычисление кинетической энергии движущегося объекта. Произвести вычисления для значений а) m=1.2 кг; v=120 м/с; б) m=1e-6 кг; v=1200 м/с.

2. Перевод углов из градусной меры в радианную. Произвести вычисления для значений 15'; 30'; 45'.

3. Вычисление вещественной степени положительного числа. Вычислить: 1.2^5; 3.5^4; 2^10;

4. Вычисление площади треугольника по формуле Герона: p=(a+b+c)/2; S=SQRT(p(p-a)(p-b)(p-c)). Вычислить площадь для сторон а) a=12.4; b=23.8; c=19.1; б) a=2.3; b=3.5; c=5.1;

5. Вычисления модуля вектора в 3-мерном пространстве. Вычислить модуль векторов w=(1.2,2.5,12.1); s=(5.8,11.3,7.5); q=(6.4,9.2,0.8).

ЗАДАЧА 2. Составить программу вычисления и печати таблицы значений двух заданных функций при указанных равноотстоящих значения аргумента. Вычисление первой функции оформить с помощью процедуры, вычисление второй функции - с помощью функции.

$S(x)=6+4\cos(x)-x^2$; $W(x)=3x^2+2x*\sin(x)-8$; [-3;3] h=0.1

На основании построенной таблицы указать интервалы, внутри которых функции имеют корни, имеют минимальное и максимальное значения.

ТЕМА 10. ОБРАБОТКА СИМВОЛОВ И СТРОК В ЯЗЫКЕТУРБО-ПАСКАЛЬ

Цель работы: изучение и программная реализация алогоритмов обработки символьных и строковых данных в языке Турбо-Паскаль.

10.1 Символьные константы и переменные

Символьные константы. Константа типа Char - это непустой символ из кодовой таблицы ПК, заключенный в апострофах, например: 'D', '*' и т.д.

Символы можно задавать и с помощью кодового номера с префиксом #, например: #7 - звуковой сигнал, #254 - символ ¦, и т.д. Для неклавишных символов это - один из основных способов задания.

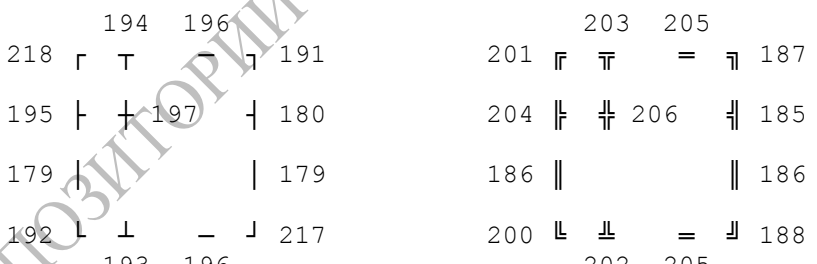
Символы с кодами 0-31 являются управляющими, это означает, что вывод их на экран или в файл эквивалентен подаче команды, управляющей работой устройства.

Группы символов кодовой таблицы ПК:

```
 0..127 - стандартные символы ASCII
 0...31 - управляющие символы;
 32 - пробел
 33..47, 58..64, 91..96 - математические и другие
знаки;
 48-57 – цифры (цифра 0 – это символ #48, и так далее)
 65 - A .. 90 - Z -прописные латинские буквы (A – #65,
B - #66,и т.д.)
 97 - a .. 122 - z - строчные латинские буквы
 128-А..159-Я - прописные русские буквы
 160-а..175-п, 224-р..239-я - строчные русские буквы
 176..178 - символы заполнения фона
 179..222 - символы псевдографики
```

Символы, используемые при оформлении программ:

Символы, используемые для построения таблиц и рамок (символы псевдографики):



Символьные переменные. Описание и присваивание значения символьной переменной:

```
Var ch:char;
. . .
ch:='a';        {или ch:=#97; }
Write(ch);
```

Для набора в тексте программы нужного символа необходимо при нажатой клавише [Alt] набрать код символа на малой клавиатуре.

10.2 Операции и функции для символьных величин

Стандартные функции обработки символьных величин. Встроенные функции реализуют посимвольную обработку строки и совместимы с типом char.

`CHR(k:byte):char` - получение символа по его коду;

`ORD(c:char):byte` - получение кода заданного символа;

`UPCASE(c:char):char` - перевод строчных латинских символов 'a'..'z' в прописные 'A'..'Z'; остальные символы не изменяются;

`PRED(c:char):char` - получение символа, предшествующего заданному; (но нет Pred(#0));

`SUCC(c:char):char` - получение символа, следующего за заданным; (но нет Succ(#255));

Операции над символьными переменными. Над символьными данными можно выполнять следующие операции:

Присваивание. Три способа присвоить значение символьной переменной:

```
c:='A';        c:=#65;          c:=Chr(65);
```

Сравнение (по коду). Большим считается тот символ, у которого числовой код больше:

```
'R'='R'    'Q'<'R'    'R'<'r'
```

Ввод и вывод значений символьных переменных осуществляется без апострофов.

10.3 Строковые константы и переменные

Строковые константы. Данные типа string(строка символов), как и числовые данные, подразделяются на константы и переменные.

Строковая константа - это последовательность символов, заключенных в апострофы. Например, 'abcdefgh'.

Строковые константы явно указываются в разделе констант:

```
const Str='Строка';
```

Строковые переменные. Турбо-Паскаль реализует также переменные типа string, которые являются расширением стандарта

языка Паскаль. Строка - это последовательность символов кодовой таблицы ЭВМ.

Описание строковых переменных имеет вид

```
type имя типа=string[N];
var имя переменной:имя типа;
```

или

```
var имя переменной:string[N];
```

Здесь N-целая константа,указывающая максимальную длину строки (количество символов в строке). В Турбо-Паскале 1<=N<=255.

Возможно, а зачастую и более удобно, также описание вида

```
var имя переменной:string;
```

которое определяет строковую переменную максимально возможной длины(в 255 символов).

Например:

```
Type  FIO=string[30];
Const F1='Иванов';
Var   s1:FIO;
```

По умолчанию (т.е. без указания конкретной длины) строка может иметь 256 символов, т.е. изменяться от 0 до 255.

Размер строки можно указать заранее, определив в квадратных скобках максимальное количество символов.

Пример**:**

```
Var  FIO:String[50];
```

В Турбо-Паскале переменные типа string[N] занимают N+1 байт. Строковые переменные аналогичны массивам типа char. Их отличием является то,что число символов (или текущая длина строки) может динамически меняться в интервале от нуля до заданного верхнего значения N. Как и в массивах, к отдельным символам строки можно обратиться с помощью индексов в квадратных скобках, например:

```
Stroka[3]:='W';
simvol:=Stroka[4];
```

Отдельные символы строковых переменных можно присваивать символьным переменным (переменным типа char).

Нулевой символ строки содержит информацию о текущей длине строки, - при действиях над строкой этот символ автоматически устанавливается таким, что его код равен числу символов в строке.

10.4 Выполнение действий по обработке строковых величин

Стандартные операции над строками. Ввод и вывод значений строковых переменных осуществляется без апострофов. Например, для выполнения оператора Readln(Str2); во входном файле необходимо набрать текст, начиная с 1-ой позиции. Для исключения ошибок ввода строковых переменных всегда используйте оператор Readln(вместо Read).

Строковые данные могут участвовать в строковых выражениях, состоящих из строковых констант, переменых типа char, знаков операций и встроенных функций. При этом над строковыми данными допустимы следующие операции: присваивание, объединение, сравнение.

Общий вид операции присваивания:

Имя строковой переменной:=Строковое выражение;

Например, STR1:='AD';

Если длина строкового выражения превышает максимальную длину строковой переменной, то все лишние символы справа отбрасываются.

Операция объединения применяется для сцепления нескольких строк в одну результирующую строку. Для обозначения этой операции в Турбо-Паскале используется знак "+", например:

```
STR1:='Результат вычислений:'+str3;
```

Строковую переменную можно рассматривать как массив символов (массив элементов типа char). Так, к отдельному символу строки можно обратиться так же, как и к элементу массива:

```
St[5]:='w';

Var st:string;
X:char;
Begin
```

```
St:='abcd';
X:=st[3];            {x='c'}
```

Все символы в строке автоматически нумеруются и строку поэтому можно воспринимать как одномерный массив символьных переменных.

Отдельный элемент строки является символом, а потому он совместим с переменной типа CHAR.

Операции сравнения (=,<>,>,< и т.д.) двух строковых операндов имеют более низкий приоритет, чем операция сцепления. Сравнение строк производится слева направо до первого несовпадающего символа, и та строка считается большей, в которой первый несовпадающий символ имеет больший числовой код. Результат выполнения операций отношения над строковыми операндами всегда имеет логический тип и принимает значение True(истинно) или False (ложно).

Сравнение строк производится слева направо до первого несовпадающего символа, и та строка считается большей, в которой первый несовпадающий символ имеет больший код.

Результатом является логическое выражение True (правда), False (неправда).

Строки считаются равными, если они полностью совпадают по текущей (а не по объявленной) длине и содержат одни и те же символы.

Если строки имеют различную длину, но в общей части символы совпадают, то считается, что более короткая строка меньше чем более длинная.

Встроенные функции для обработки строк. Встроенные функции позволяют обрабатывать часть строки.

Length(Str):integer - функция, которая вычисляет текущую длину (в символах) строки Str. Результат имеет целочисленный тип. Текущая длина строки содержится также в коде символа Str[0].

Пример:

```
program X;
var
x:integer;
s:string;
```

```
begin
s:='123456789';
x:=Length(s);                {x=9}
end.
```

Copy(St,K,J):string - функция,которая выделяет из St подстроку из J символов, начиная с позиции K. Если K>Length(St), то результатом будет пустая строка. Например:

```
Str:='abcdefgh';
Z:=Copy(str,3,2);        {Z='cd'}
```

Pos(S,St,I):integer - функция, которая определяет наименьший номер символа в St, начиная с которого S входит в St как подстрока. Если в St не найдено S, то результат равен нулю. Результат имеет тип byte.

Пример:

```
ST1:='abcde';
x:=Pos('de',ST1);   {x=4}
```

Concat(str1,str2,..):string - функция, выполняющая объединение в одну строку всех строк, указанных в качестве аргументов. Количество аргументов может быть любым.

Пример:

```
s:=ConCat('AA','XX','YY');          {S='AAXXYY'}
```

Copy(s:string,N,L:integer):string - Функция позволяет получить фрагмент заданной строки, начинающейся с позиции N длиной L символов.

Пример:

```
. . .
S:='ABCDEFG';
S1:=Copy(S,2,3);                              {S1='BCD'}
```

Delete(s:string,n,l:integer):string - Функция удаляет из строки S, начиная с позиции n

l-количество символов.

Пример:

```
. . .
S:='Река Волга';
S1:=Delete(S,1,5);          {S1='Волга'}
```

Insert(s1,s2:string,n:integer):string - Функция производит вставку строки s1 в строку s2, начиная с позиции n.

Пример:

```
. . .
s1:='Мерседес';
s2:='190';
s3:=Insert(s1,s2,1);      {s3='Мерседес190'}
```

Str(x:integer,s:string):string - Функция преобразовывает число x и помещает его в строку s.

Пример:

```
. . .
x:=1500;
str(x:6,s);               {s:='__1500'}
```

Val(s:string,x:real,cod:byte):real - Функция преобразовывает строковое значение s в число вещественного типа с выдачей кода ошибки. Если переменная cod=0, то преобразование прошло нормально.

Пример:

```
. . .
s:='1450';
Val(st,x,cod);                {x=1450, cod=0}
```

Встроенные процедуры для обработки строк. Встроенные процедуры позволяют обработать часть строки.

Delete(Str,K,J) - процедура, которая удаляет из строки Str J символов, начиная с K-го. Результатом является строка Str без удаленной подстроки. Если K больше размера строки, символы не удаляются.

Insert(Str1,Str2,K) - процедура, которая вставляет строку Str1 в строку Str2, начиная с позиции K. Если полученная в результате строка Str2 превышает 255 символов, то все лишние символы (начиная с 256) отбрасываются.

Str(x,st) - процедура, которая преобразует число X вещественного или целого типа в последовательность символов St. Число X может быть указано с форматом, аналогичным формату вывода в процедуре Write, например: Str(x:8:3,st).

Val(st,x,Code) - процедура, которая преобразует символьное представление числа в числовое значение, соответствующее типу переменной x (вещественному или целому). Переменная Code:integer

служит для контроля правильности выполнения преобразования. Если Code=0, значит, преобразование выполнено правильно. Если Code<>0, произошла ошибка, причем значение Code указывает номер ошибочного символа.

Val(‘123’,x,code);

X=123;

Вопросы для самоконтроля

1. Символьные константы и переменные
2. Группы символов в кодовой таблице
3. Строковые константы и переменные
4. Операции над строками
5. Встроеные функции для обработки строк
6. Встроенные процедуры для обработки строк

СОДЕРЖАНИЕ ЗАДАНИЯ

Составить программу для выполнения указанных действий над строкой текста, вводимой с клавиатуры. Исходную строку и результат обработки вывести на экран с поясняющим текстом, предварительно очистив экран.

1. Проверить, имеется ли в заданном предложении баланс открывающих и закрывающих скобок.

2. Составить процедуру посимвольного вывода заданной строки на экран, каждый символ в новой строке.

3. Отредактировать предложение, удаляя из него лишние пробелы, оставляя только по одному пробелу между словами.

4. Составить программу, подсчитывающую количество слов в предложении.

5. Составить программу, которая осуществляет кодирование предложения, заменяя все символы на символы с кодом, на 10 большим.

6. Составить программу, которая осуществляет послоговый анализ слов предложения, вставляя внутрь слов после гласных символ переноса.

7. Составить программу, которая запрашивает начальный и конечный коды и выводит на экран все символы этого интервала в 5

столбцов в форме код-символ. При вводе выполнить проверку корректности вводимых параметров 31<k1<255, k2<255 с помощью оператора repeat..until.

8. Составить программу, которая осуществляет ввод строки с клавиатуры и преобразует все большие буквы русского алфавита в малые, и наоборот.

9. Составить программу, которая осуществляет ввод строки с клавиатуры и вывод ее на экран в одинарной рамке из символов псевдографики.

10. Составить программу, которая осуществляет ввод строки с клавиатуры и вывод ее на экран в двойной рамке из символов псевдографики.

11. В заданном предложении найти самое короткое и самое длинное слова.

12. Составить программу, которая по заданному предложению строит новое, располагая все слова в обратном порядке.

ТЕМА 11. ИЗУЧЕНИЕ И ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ СОРТИРОВКИ

Цель работы: изучение и программная реализация простейших алгоритмов внутренней сортировки одномерных массивов данных

11.1 Сортировка данных. Основные понятия

Многие задачи, связанные с обработкой и поиском информации, решаются эффективнее, если данные хранятся в памяти ЭВМ в определенном порядке. Можно представить, насколько трудно было бы пользоваться словарем или телефонным справочником, если бы записи в них не располагались по алфавиту. Точно так же от порядка, в котором хранятся элементы в памяти ЭВМ, во многом зависит скорость и простота алгоритмов, предназначенных для их обработки.

Хотя в словарях слово "сортировка" определяется как "распределение, отбор по сортам; деление на категории", программисты традиционно используют это слово в гораздо более

узком смысле, обозначая им расстановку элементов в возрастающем или убывающем порядке. Можно дать следующее определение: сортировкой называется упорядочение элементов массива (записей файла) в соответствии с заданным условием следования.

По способу размещения данных выделяют сортировку внутренюю (все данные помещаются в ОП) и внешнюю (данные находятся на магнитном диске).

По типу обрабатываемых данных различают:

а) простую (скалярную) сортировку: элементы числового или строкового массива расставляются по убыванию или возрастанию их значений;

б) сортировку по ключу: обрабатываются массивы или файлы записей (таблицы). Каждая запись состоит из полей, содержащих информацию определенного типа. При сортировке одно из полей рассматривается как главное (ключ), и записи расставляются таким образом, чтобы это главное поле было упорядоченным. Например, если ключом является фамилия, то файл может сортироваться с целью упорядочить фамилии по алфавиту. Если ключ - период полураспада изотопа, то - по убыванию этой величины, и т.п.

Существует множество различных методов сортировки, причем для разных типов задач подходящими являются разные методы. Практически каждый метод сортировки содержит такие этапы:

- сравнение, определяющее упорядоченность пары элементов;
- перестановку, меняющую местами пару элементов;
- способ организации операций сравнения и перестановки до тех пор, пока все элементы не будут упорядочены.

В данной работе рассмотрены алгоритмы сортировки, которые наиболее часто излагаются в учебной литературе и достаточно просты для программирования.

11.2 Простейшие методы сортировки

Метод обмена соседних элементов (метод "пузырька"). При сортировке числового массива из N элементов методом "пузырька" обрабатываемый массив просматривается несколько раз, от элемента к элементу, причем текущий элемент сравнивается с соседним. Если числа в паре расположены в правильном порядке, то переходят к следующему элементу, в противном случае меняют элементы

местами и также переходят к следующему элементу. За один просмотр самый большой из обрабатываемых элементов переставляется на самое последнее (N-е) место, так что на следующем этапе обрабатывать нужно на один элемент меньше. Минимальный элемент при этом перемещается на одну позицию вверх ("всплывает"). На каждом следующем просмотре максимальные из оставшихся элементов занимают места в конце массива. Сортировка считается оконченной, если в ходе просмотра не была произведена ни одна перестановка.

Рассмотрим пример массива из 6 чисел:

```
исходный массив:          1 7 6 4 2 5
1-й просмотр:             . 6 7 . . .     (за один проход
                          . . 4 7 . .      самый большой элемент
                          . . . 2 7     "тонет" на дно, а минимальный
                          . . . 5 7      "всплывает" на одну позицию)
после 1-го просмотра:     1 6 4 2 5 7
                          └─────────┘      (———— - область просмотра)
2-й просмотр:             . 4 6 . . .     ( для 2-го прохода)
                          . . 2 6 . .
                          . . . 5 6 .
после 2-го просмотра:     1 4 2 5 6 7
                          └───────┘        (———— - область просмотра)
                            . . .          ( для 3-го прохода)
после 3-го просмотра:     1 2 4 5 6 7
```

Сортировка окончена, так как во время четвертого просмотра не было совершено ни одной перестановки. Описание алгоритма:

```
Заполнение массива (ввод данных)
просмотр=0
Повторять
| перестановка=0
| просмотр=просмотр+1
| Для i=1 до N-просмотр выполнить
| | Если A[i] > A[i+1] то
| | | Переставить(A[i],A[i+1])
| | | перестановка=1
до выполнения условия перестановка=0
```

Вспомогательный алгоритм Переставить(a,b) осуществляет обмен значениями переменных a и b.

Метод "четных и нечетных транспозиций". Этот метод является разновидностью метода обмена. Сортировка выполняется следующим образом. Массив просматривается многократно, причем на первом просмотре сравниваются A[i] с A[i+1] для всех нечетных i. На втором просмотре сравниваются A[i] с A[i+1] для всех четных i. Каждый раз, когда A[i]>A[i+1], выполняется обмен этих элементов. Просмотры продолжаются до тех пор, пока массив не будет упорядочен. Очевидно, в таком случае не будет выполнен ни один обмен элементов. Описание алгоритма:

```
Заполнение массива
Повторять
| перестановка=0
| i=1
| Пока i<N выполнять
| | Если A[i]>A[i+1] то
| | | Переставить(A[i],A[i+1])
| | | перестановка=1
| | i=i+2
| i=2
| Пока i<N выполнять
| | Если A[i]>A[i+1] то
| | | Переставить(A[i],A[i+1]);
| | | перестановка=1;
| | i=i+2;
до получения перестановка=0;
```

Метод выбора (метод поиска минимума). Последовательными сравнениями ищется наименьший элемент массива, причем в процессе поиска запоминается индекс текущего наименьшего числа.

После того как просмотрен последний элемент, можно поменять местами 1-й элемент с наименьшим, т.к. его индекс известен. Затем эти действия повторяются, начиная со 2-го элемента (1-й уже стоит на правильном месте), затем с 3-го элемента, и т.д.

В случае массива из 5 чисел получим, например:

```
исходный массив:                    50 40 30 10 20
индекс наименьшего элемента: [4]      └──────────┘  область просмотра
массив после перестановки:          10 40 30 50 20
индекс наименьшего элемента: [5]         └───────┘  область просмотра
массив после перестановки:          10 20 30 50 40
индекс наименьшего элемента: [3]            └────┘  область просмотра
(в данном случае перестановка не нужна, т.к. индекс совпадает
с началом области просмотра)
```

```
                                10 20 30 50 40
индекс наименьшего числа:  [5]              └──  область просмотра
массив после перестановки:      10 20 30 40 50
```

Таким образом, перестановка массива завершена. Описание алгоритма:

```
Заполнение массива
Для i=1 до N-1 выполнять
|  Amin=A(i)
|  nom=i
|  Для j=i+1 до N выполнять
|  |  Если A[j]<Amin то
|  |  |  Amin=A[j]
|  |  |  nom=j
|  Переставить(A[i];A[nom])
```

Оптимизированный метод выбора. Формулировка метода выбора является наиболее естественной, так как содержит стандартный фрагмент поиска минимального элемента. Но его можно упростить, если заметить, что для последующей перестановке элементов важна не величина минимального элемента, а его номер в массиве. Поэтому величину Amin можно не учитывать. Тогда получаем следующий вариант алгоритма выбора:

```
Заполнение массива
Для i=1 до N-1 выполнять
| nom=i
| Для j=i+1 до N выполнять
|  | Если A[j]<A[nom] то nom=j
| Обмен(A[i];A[nom]);
```

По сравнению с методом пузырька, в котором перестановки призводятся при каждом сравнении, оптимизированный метод выбора дает значительный выигрыш времени, так как удается обойтись (да и то не при каждом сравнении) только одним присваиванием вместо трех.

Метод сдвига. В этом методе выполняется упорядочение массива, начиная с его самых первых элементов (A[1] и A[2]). Затем в область

упорядочения включается A[3], A[4] и так далее, до полного исчерпания массива.

Пусть элементы A[1]..A[k] уже упорядочены. Тогда элемент A[k+1] ставится на надлежащее место последовательными перестановками с элементами A[k], A[k-1].., т.е., все элементы, большие A[k+1] сдвигаются вниз на одну позицию. Описание алгоритма:

```
Заполнение массива
Для i=2 до N выполнять
| t=A[i]
| j=i-1
| Пока A[j]>t и j>0 выполнять
|   | A[j+1]=A[j]
|   | j=j-1
| A[j+1]=t
```

Метод подсчета (вспомогательного массива). В этом методе используется, помимо обрабатываемого массива, еще один массив такого же размера. Кроме того, предполагается, что в исходном массиве нет одинаковых элементов. Затраты оперативной памяти оправдываются простотой выполняемых действий. Алгоритм основан на использовании очевидного факта: элемент, стоящий на k-м месте в уже упорядоченном массиве, по величине превосходит (k-1) остальных элементов.

Описание алгоритма:

```
Заполнение массива A
Для i=1 до N выполнять
| k=0
| Для j=1 до N выполнять
|   |  Если A[i]>A[j] то k=k+1
| B[k+1]=A[i]
```

Примечание: Если по сути задачи требуется сохранение результата в массиве A, после выполнения сортировки достаточно выполнить присваивание A:=B; (в Турбо-Паскале допускается присваивание массивов одинакового типа).

Метод вставки. В этом методе объединены черты метода сдвига и метода вспомогательного массива. Элементы исходного массива

берутся по очереди и записываются во вспомогательный массив, но так, чтобы получился упорядоченный набор данных. Вначале элемент A[1] записывают первым в массиве B, т. е. B[1]=A[1]. Далее элемент A[2] сравнивается с B[1]. Если в результате сравнения оказалось, что A[2]<B[1], то элемент B[1] передвигается на одну позицию дальше, а элемент A[2] занимает его место в массиве B. Теперь в массиве B размещены два элемента B[1] и B[2], образующих последовательность, упорядоченную по возрастанию значений.

На каждом i-м просмотре процесса сортировки очередной элемент A[i] сравнивается поочередно со всеми элементами, записанными ранее в массив B (их i-1 штук), начиная с B[1]. При обнаружении B[j], большего чем A[i], элементы B[j], B[j+1], B[j+1], ... , B[i-1] передвигаются на одну позицию, освобождая место для элемента A[i], который занимает j-ю позицию (это означает, что выполняется присваивание B[j]=A[i]).

Отметим, что, в отличие от метода сдвига, в методе вставок просмотр массива B начинается от его начала.

В отличие от метода подсчета, метод вставки позволяет упорядочить массив A и при наличии в нем совпадающих элементов. Описание алгоритма:

```
Заполнение массива A
B[1]=A[1]
Для i=2 до N выполнять
| j=1
| Пока B[j]<A[i] и j<i выполнять j=j+1
| Если j=i то
|   | B[i]=A[i]
| иначе
|   | Для k=i-1 до j (шаг -1) выполнять B[k+1]=B[k]
|   | B[j]=A[i]
```

Примечание: Если по сути задачи требуется хранение результата в массиве A, после выполнения сортировки достаточно выполнить присваивание A:=B (в Турбо-Паскале допускается присваивание массивов одинакового типа).

СОДЕРЖАНИЕ ЗАДАНИЯ

Составить процедуры: а) заполнения массива, содержащего заданное количество N элементов типа byte, случайными числами в указанном диапазоне [k1..k2]; б) сортировки массива заданным методом; в) вывода элементов массива на экран.

Составить программу, в которой выполняется заполнение массива, сордировка массива и вывод результатов. Исходную расстановку чисел, промежуточные и окончательный результаты вывести на экран.

ТЕМА 12. ПРИМЕНЕНИЕ ТЕКСТОВЫХ ФАЙЛОВ ДЛЯ ВВОДА ДАННЫХ И ВЫВОДА РЕЗУЛЬТАТОВ ВЫЧИСЛЕНИЙ

Цель работы: изучение и применение программных элементов использования текстовых файлов для ввода и вывода числовых значений

Текстовые файлы и примеры их использования. Текстовые файлы содержат набор строк переменной длины, состоящих из обычных символов (букв алфавита, цифр). В программе описание текстового файла имет вид

```
var имя_файла:text;
```

Здесь text - стандартный идентификатор, точно такой же, как real, char и т.д. Для работы с текстовыми файлами используются следующие стандартные процедуры:

Append(var ft:text); - открытие уже существующего текстового файла, связанного с файловой переменно ft, для добавления данных в конец файла;

Writeln(var ft:file); - завершение текущей строки текстового файла, связанного с файловой переменной ft (при его записи);

Readln(var ft:file); - переход к началу следующей строки текстового файла ft (при его чтении).

Writeln(var ft:file; X1,X2,..,XN); - запись в текстовый файл ft значений переменных X1,X2,...,XN с завершением текущей строки;

Readln(var ft:file; X1,X2,..,XN); - чтение N символов файла ft с переходом к новой строке.

ПРИМЕР 1. Работа с текстовым файлом для записи результатов вычислений

```
Program USE_TXFL;
Uses Crt,Dos;
Const   h:real=0.1;
Var
  tf1:text;
  i:byte;
  x,y,ys:real;
  st1,st2,st3:string;
BEGIN
    {запись результатов в текстовый файл
     для просмотра и печати данных}
  Assign(tf1,'txfile.dat');
  Rewrite(tf1);
  Writeln(tf1,’Результаты вычислений про программе
USE_TXFL’);
  Writeln(tf1,’ программу составил Байтов А.Б., группа
Ф-27’);
  writeln(‘--------------------------------‘);
  For i:=0 to 20 do begin
    x:=h*i;
    y:=sin(x);
    writeln('x = ',x:6:3,' ':8,'y = ',y:6:3);
    writeln(tf1,'x = ',x:6:3,' ':8,'y = ',y:6:3);
  end;
  writeln(‘--------------------------------‘);
  Close(tf1);
  Readln;
End.
```

ПРИМЕР 2. Чтение данных из текстового файла и их обработка

```
Program USE_TXF2;
Uses  Crt, Dos;
Var
  ftext:text;
```

```
  n:integer;
  x,y,ys:real;
BEGIN
  Assign(ftext,'txfile1.dat');
  Reset(ftext);
  n:=0;
  ys:=0;
  while not Eof(ftext) do begin
    readln(ftext,x,y);
    writeln('x = ',x:6:3,' ':8,'y = ',y:6:3);
    n:=n+1;
    ys:=ys+y;
  end;
  Close(ftext);
  If n>0 then ys:=ys/n;
  writeln('Прочитано ',n,' строк из файла');
  writeln('Среднее значение величины Y = ',ys:8:4);
  Readln;
End.
```

СОДЕРЖАНИЕ ЗАДАНИЯ

ЗАДАЧА 1. Составить программу, в которой решается Задача 2 из предыдущей лабораторной работы, но результаты выводятся и на экран, и в текстовый файл. Предусмотреть вывод в начале текстового файла информации о задаче и авторе программы. Содержимое текстового файла включить в отчет по работе.

ЗАДАЧА 2. В текстовом файле с именем ‘pmm_NN.dat’ (где NN-номер варианта) содержатся числовые данные о результатах экспериментального измерения значений величины y для фиксированных значений величины x. В каждой строке файла расположены два числа (x_i, y_i). Составить программу, которая читает данные из файла и выводит на экран и в другой текстовый файл сами данные, количество измерений и среднее значение величины y.

ЛИТЕРАТУРА

Алексеев В.Е., и др. Вычислительная техника и программирование. Практикум по программированию. Под ред. А.В.Петрова. М., 1991.

Вычислительная техника и программирование. (Под ред. А.В.Петрова). М,1990.

Гурин Н.И. Работа на персональном компьютере. Справочное пособие. – Мн.: Беларусь, 1994. – 224 с.

Енохович А.С. Справочник по физике и технике. М.: Просвещение. 1989.

Епанешников А.М., Епанешников В.А. Программирование в среде Турбо Паскаль 7.0. – 3-е изд. –М.: Диалог-МИФИ, 1996. – 288 с.

Кошкин Н.И., Ширкевич М.Г. Справочник по элементарной физике. М.: Наука, 1980.

Немнюгин С.А. Turbo Pascal. Программирование на языке высокого уровня: Учебник для вузов. – СПб.: Питер, 2005. – 544 с.

Офицеров Д.В. и др. Программирование на персональных ЭВМ: Практикум: Учеб. пособие/ Д.В.Офицеров, А.Б.Долгий, В.А.Старых; Под общ.ред. Д.В.Офицерова. — Мн.: Выш.шк., 1993. — 256 с.

Рапаков Г.Г., Ржеуцкая С.Ю. Turbo Pascal для студентов и школьников. – СПб.: БХВ-Петербург, 2005. – 352 с.

Турбо Паскаль 7.0. Самоучитель. – СПб.: Питер; К.: Изд. группа BHV, 2002.– 416 с.

Фаронов В.В. Турбо Паскаль 7.0. Начальный курс. Уч. пособие. Изд. 7-е. – М.: Изд-во «Нолидж», 2001. – 576 с.

Фаронов В.В. Турбо Паскаль 7.0. Практика программирования. Уч. пособие. Изд. 7-е. – М.: Изд-во «Нолидж», 2001. – 416 с.

Фигурнов В.Э. IBM PC для пользователя. Изд. 7-е. – М.: ИНФРА-М, 2000. – 640 с.

Дей Евгений Александрович

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ТУРБО-ПАСКАЛЬ И ЧИСЛЕННОЕ РЕШЕНИЕ ФИЗИЧЕСКИХ ЗАДАЧ

ПРАКТИЧЕСКОЕ ПОСОБИЕ
для студентов 1 курса
специальности 1- 31 04 01 – 02 – «Физика»

В авторской редакции

Подписано в печать_____. Формат 60x84 1/16. Бумага писчая №1. Гарнитура «Таймс». Усл.п.л._____. Уч.-изд.л._____ . Тираж 50 экз. Заказ №_____.

Отпечатано с оригинала-макета на ризографе
учреждения образования
«Гомельский государственный университет
имени Франциска Скорины»
Лицензия ЛП № 02330/0056611 от 16.02.04.
246019, г. Гомель, ул. Советская, 104