Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Сибирский государственный медицинский университет»
Министерства здравоохранения Российской Федерации

О.В. Воробейчикова, И.С. Каверина

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ OBJECT PASCAL

Учебно-методическое пособие

Томск
Сибирский государственный медицинский университет
2017

УДК 004.42(075.8)
ББК 32.973.26-018.1я73
В 751

Воробейчикова О.В.

В 751 Программирование на языке Object Pascal в среде Borland Delphi 7.0: учебно-методическое пособие / О.В. Воробейчикова, И.С. Каверина. – Томск: Сибирский государственный медицинский университет, 2017. – 94 с.

Пособие содержит методические указания для создания консольных приложений на языке Object Pascal в среде Borland Delphi 7.0. Структура пособия включает в себя теоретический материал, практические задания, контрольные вопросы и тестовые задания. Набор практических заданий по теме содержит подробный разбор их выполнения, приведен пример выполнения задания, подобного заданию для самостоятельной проработки.

Пособие соответствует программе курса «Информатика, медицинская информатика» разделам «Типы и основные операторы Object Pascal» и «Подпрограммы и файлы Object Pascal» для специальности «медицинская кибернетика».

Пособие может быть использовано студентами других специальностей медицинского вуза, желающим освоить программирование в среде Borland Delphi 7.0.

УДК 004.42(075.8)
ББК 32.973.26-018.1я73

Рецензент:

Фокин В.А. – доктор физ.-мат. наук, профессор кафедры медицинской и биологической кибернетики с курсом мед. информатики ФГБОУ ВПО СибГМУ Минздрава России.

Утверждено и рекомендовано к печати учебно-методической комиссией ФГБОУ ВПО СибГМУ Минздрава России (протокол № 4 от 27 декабря 2017 г).

Введение

Пособие предназначено для студентов, желающих освоить приемы программирования. В основу положен курс «Информатика, медицинская информатика» разделы «Типы и основные операторы ObjectPascal» и «Подпрограммы и файлы ObjectPascal» для студентов СибГМУ, обучающихся по специальности «медицинская кибернетика». Пособие является дополнением к лекционному материалу, поэтому в нем представлена только информация, достаточная для решения практических задач рассматриваемой темы.

Материал пособия разбит на темы, содержащие теоретическую и практическую части. В теоретической части представлены инструкции по рассматриваемым в практической части заданиям, и примеры типовых задач. В практической части предлагаются задачи для самостоятельной работы студентов.

В пособии принято следующее форматирование:

- *курсивом:* обозначены команды ОС Windows, команды меню оболочки Delphi, названия диалоговых окон (ДО); если нужно выполнить последовательность команд меню, то они перечисляются с использованием прямого слеша – «/»;
- шрифтом Arial обозначены объекты в диалоговых окнах – строки ввода, флажки, названия групп и т.п.;
- `шрифтом Courier New` – примеры конструкций языка Object Pascal и листинг программ, который, кроме этого для наглядности, выделен рамкой.

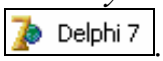
ТЕМА 1. ОПИСАНИЕ СРЕДЫ DELPHI И СОЗДАНИЕ КОНСОЛЬНОГО ПРИЛОЖЕНИЯ

Теоретическая часть

1. Запуск программы и настройка среды

Основу Delphi составляет не только сам язык, но и **RAD (Rapid Application Development)** – среда быстрой разработки программ.

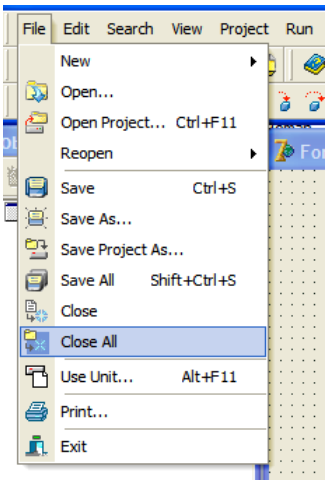
Запуск среды Delphi осуществляется либо с помощью соответствующей пиктограммы  на *Рабочем столе*, либо через *Главное меню Windows*:

Пуск/ Программы/ Borland Delphi/ Delphi7 пункт меню имеет вид: 

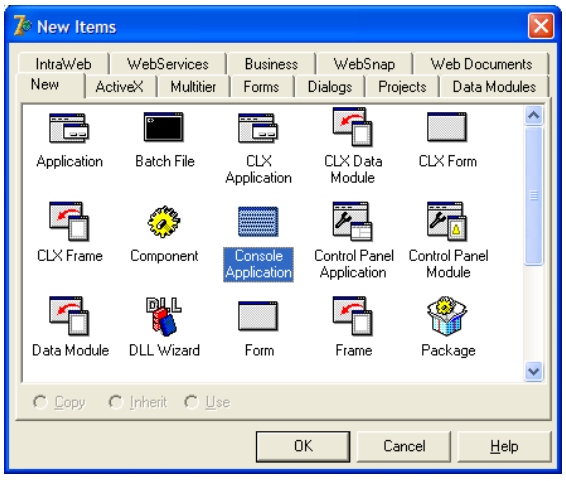
Интегрированная среда (ИС) Borland Delphi предназначена как для создания приложений под Windows, так и для создания **консольных приложений** (консольное приложение – это приложение, работающее с монитором в текстовом режиме). Начнем наше знакомство с ИС Borland Delphi с создания именно консольного приложения.

Алгоритм создания окна консольного приложения:

1. Запустить Delphi: *Пуск/ Программы/ Borland Delphi/ Delphi7*
2. Выполнить пункт меню (рис. 1а) *File / Close All.*
3. Выполнить пункт меню *File / New / Other*. При этом появляется диалоговое окно *New Items* (*Новые Приложения*) (рис. 1б). Выбрать *Console Application* (*консольное приложение*).



а)



б)

Рис. 1. Этапы создания консольного приложения в среде Delphi:

а) подпункт *Close All* (закрыть все**)**

б) выбор в диалоге *Console Application* (Консольное приложение)

Рассмотрим оболочку Delphi, настроенную на консольное приложение.

2. Описание интегрированной среды

Интегрированная среда Delphi состоит из главного окна, которое включает в себя несколько окон (рис. 2).

- окно Обозревателя объектов (Object TreeView) – в него помещаются все объекты, используемые при создании приложения под Windows, поэтому вначале окно – пустое.
- окно Инспектора объектов (Object Inspector) позволяет определять свойства и события используемых объектов и компонент. Для консольного приложения окно – пустое.
- окно Редактора кода (Project1) в него помещается программный код приложения.
- окно Проводника кода (Exploring Project1.dpr). Как правило, состыковано с окном Редактора кода, левее него. В это окно помещаются пиктограммы всех объектов, задействованных в программе: переменных, констант и т.п.

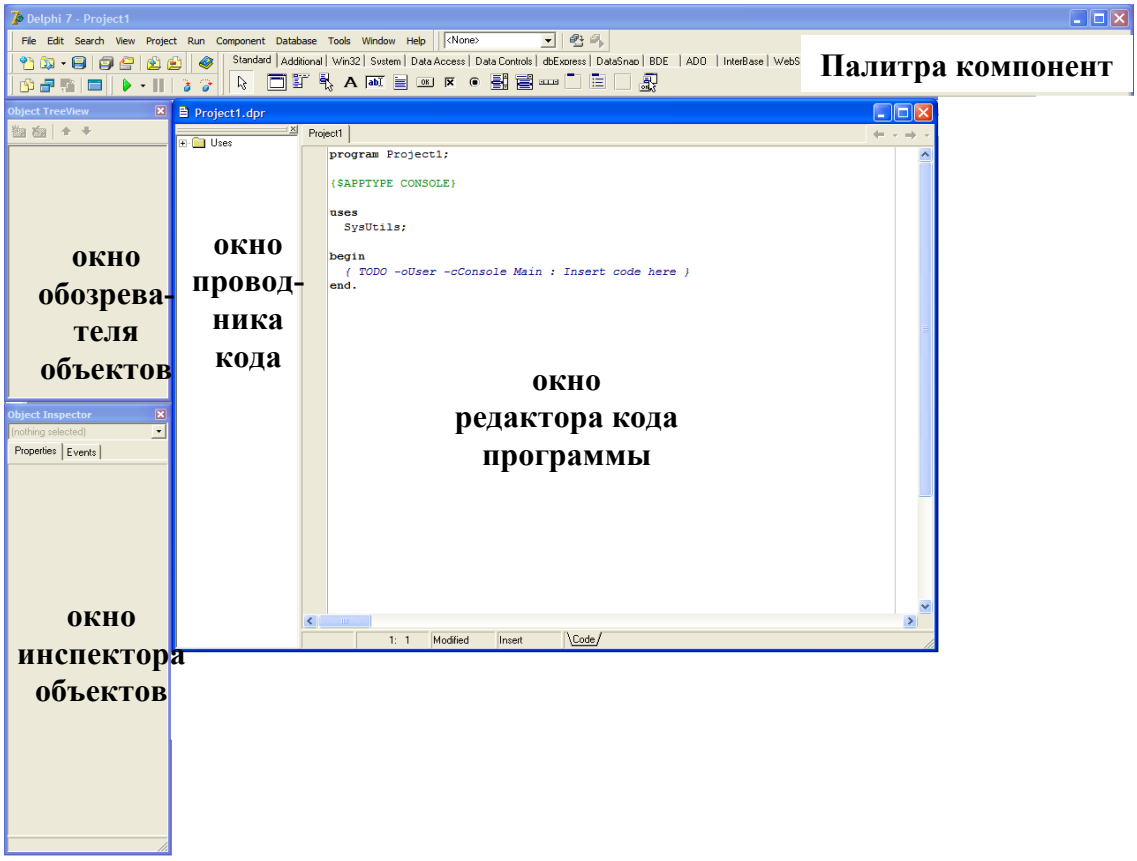


Рис. 2. Внешний вид интегрированной среды Delphi

Окна можно добавлять, убирать, состыковывать между собой по желанию пользователя. Например, для создания консольного приложения не нужны окна *Обозревателя объектов* и *Инспектора объектов*, поэтому их можно закрыть.

Примечание: Delphi одновременно может работать только с одним проектом. Если Вы хотите запустить несколько проектов одновременно, то нужно запустить соответственно несколько оболочек Delphi через *Главное меню* Windows.

2.1. Главное окно Delphi

Главное окно Delphi занимает при открытии оболочки весь экран, содержит заголовок приложения (Delphi 7) и проекта (Project1). При минимизации *Главного окна* сворачиваются все окна оболочки, при закрытии – закрываются все окна и само приложение Delphi. *Главное окно*, кроме перечисленных выше окон, содержит следующие элементы (рис. 2):

- *Строку главного меню* – набор команд для доступа к функциям Delphi.
- *Панель горячих кнопок* расположена ниже меню в левой части экрана и содержит кнопки для вызова наиболее часто используемых команд меню, например, *File/Open*  , *File/Save*  или *Run/Run*  .
- *Палитра компонент* расположена ниже меню и справа от панели «горячих» кнопок. Содержит библиотеку компонент, которые можно использовать при разработке пользовательского проекта под Windows.

Главное меню содержит следующие пункты:

- *File* – команды открытия и сохранения проектов и файлов;
- *Edit* – команды работы с буфером памяти и компонентами формы;
- *Search* – команды работы с редактором кода: поиск, изменение строк, и т.п.;
- *View* – команды переключения между окнами приложения и файлами проекта;
- *Project* – команды компиляции и работы с файлом проекта;
- *Run* – команды запуска проекта на выполнение, отладка и трассировка программы;
- *Component* – команды для работы с компонентами проекта;
- *DataBase* – команды, использующиеся при работе с запросами баз данных;
- *Tools* – команды настройки среды Delphi, вызов DataBase Desktop;
- *Window* – расположение окон и переключение между окнами оболочки;
- *Help* – подсказка среды.

2.2. Окно редактора кода программы

В окне редактора кода записывается текст программы – консольного приложения.

Первоначально это окно содержит только одну страницу, хранящую исходный текст нового проекта, который находится в разработке. В это же окно

помещаются новые страницы, если проект содержит какие-либо модули. Каждый модуль – в свое окно.

2.3. Окно проводника кода программы

Окно выполняет функцию визуального отображения данных, используемых в проекте. Каждый вид данных (переменные, константы, функции и т.п.) или объект, участвующий в проекте, отображается своей пиктограммой (таблица 1).

Таблица 1

Обозначения объектов проекта в проводнике кода

Пиктограмма	Название объекта проекта
	процедура
	функция
	переменная или константа
	модуль
	тип

Все объекты сгруппированы в иерархическую структуру по типу приложения *Проводник* Windows. Окном пользоваться удобно, если нужно быстро посмотреть описание какого-либо объекта в программе.

3. Раскрытие термина «проект» и файлы проекта

Система программирования Delphi – это объектно-ориентированный язык Object Pascal и программные модули. Поэтому программирование в Delphi – это модульное программирование. В модулях содержатся все используемые объекты: программы, описания типов, данных, ресурсы и т.п. так как модули являются отдельными программными единицами, то с программой, составленной в системе Delphi, всегда связывают понятие проекта.

Проект – это совокупность исходных программ, библиотек и файлов для всего приложения в целом.

Проект содержит следующие основные файлы:

- Файл с расширением **DPR** всегда содержит программный код основной части проекта.
- Файлы с расширением **PAS** являются модулями на языке Object Pascal. Эти модули содержат всю алгоритмическую часть и процедуры обработки событий. При разработке формы из набора типовых объектов генерируются пустые процедуры и все необходимые объявления переменных и типов.
- Файлы **DPR** и **PAS** - обычные текстовые файлы.

Кроме перечисленных файлов в проект входят файлы, перечисленные в таблице 2.

Таблица 2

Перечень основных файлов проекта

EXE	Откомпилированный исполнимый файл, содержащий машинные инструкции для выполнения пользовательского приложения.
DOF	Данные файлы содержат текущие установки для опций проекта (которые настраиваются в окне *Project - Options*)., как например, настройки компилятора и компоновщика, каталоги, условные директивы и параметры командной строки. Данные установки могут быть изменены пользователем путем изменений настроек проекта. При отсутствии этих файлов будут использоваться стандартные параметры.
CFG	

При разработке консольного приложения все файлы формируются с именем, указываемым при сохранении файла **dpr**, содержащего основной код.

Рекомендуется отводить под проект отдельную папку, где будут храниться все его файлы.

Если в один каталог сохранить несколько проектов, то файлы из разных проектов могут перемешаться, и можно будет отправлять всё в корзину.

4. Сохранение проекта

Для сохранения проекта применяется соответствующая группа команд пункта *File* главного меню Delphi (рис. 3).

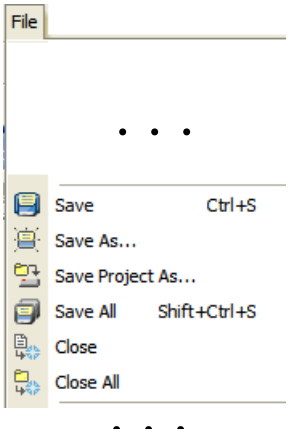


Рис. 3. Группа команд меню для сохранения проекта

Как принято в Windows пункт *Save As…* отличается от пункта *Save* тем, что позволяет управлять сохранением проекта в нужную папку и под нужным

именем. И если при первом сохранении проекта выбирается пункт *Save*, то автоматически срабатывает *Save As*.

Остальные пункты (*Save Project As* и *Save All*), при работе с консольным приложением, не различаются между собой.

При выборе пунктов меню *Close* и *Close all* закрывается окно с текущим проектом. Этим пунктом можно воспользоваться, если есть необходимость открыть новый проект, не закрывая среду Delphi. Тогда вначале следует закрыть текущий проект (для консольных приложений пункты работают одинаково), а затем создавать новый проект.

Главное, что следует помнить – в проекте участвуют несколько файлов. Поэтому следует под каждый проект создавать отдельную папку, причем это можно сделать прямо при сохранении проекта (рис. 4), затем сохранить проект под нужным именем.

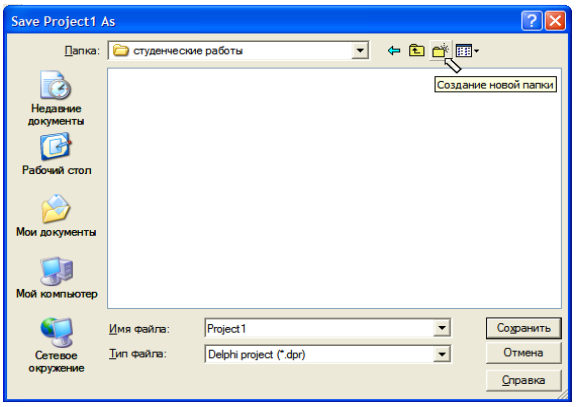


Рис. 4. Диалоговое окно Сохранение проекта

Примечание: Для удобства работы активизируйте режим *Автосохранение*, который срабатывает каждый раз при запуске проекта на выполнение. Для этого в пункте меню *Tools/Environment Options* в группе Autosave options включить флажок Editor files (рис. 5).

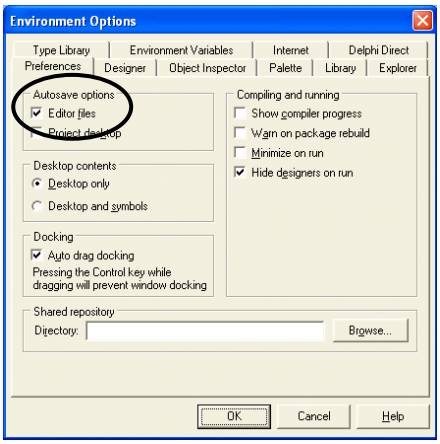


Рис. 5. Установление режима Автосохранение

После сохранения и компиляции программы создается загрузочный файл с расширением *exe*, который можно запускать без оболочки Delphi, например, из *Проводника* Windows.

5. Запуск проекта на выполнение

Запуск программы на выполнение из среды Delphi осуществляется с помощью пункта меню *Run/Run* или соответствующей быстрой кнопки на панели инструментов оболочки  или функциональной клавишей <*F9*>.

При «зависании» программы прервать ее выполнение из среды Delphi можно при помощи пункта меню *Run/Program Reset* или комбинацией клавиш <*Ctrl*>+<*F2*>.

«Быстрая кнопка» *Пауза*  выполняет приостановление выполнения программы и вызывает окно ассемблеровского отладчика, с помощью которого можно определить, в какой строке произошла остановка выполнения. Как правило, ее используют при отладке программ.

Контрольные вопросы

1. Что такое проект Delphi?
2. Почему под проект рекомендуют отводить отдельную папку?
3. Какие окна содержит оболочка Delphi?
4. Для чего предназначено окно Редактора кода?
5. Для чего предназначено окно Проводника кода?

Тестовые задания

Выберите один или несколько правильных ответов.

1. ПРОЕКТ DELPHI СОДЕРЖИТ ФАЙЛЫ С РАСШИРЕНИЕМ
   1) *.pas
   2) *.exe
   3) *.dof
   4) *.cfg
   5) *.dpr
   6) *.txt

2. ДЛЯ СОХРАНЕНИЯ ПРОЕКТА НУЖНО ВЫПОЛНИТЬ ПУНКТ МЕНЮ
   1) File/ Save all
   2) File/ Save
   3) File/ Close all
   4) File/ Close
   5) File/ New

3. ДЛЯ ЗАПУСКА ПРОЕКТА НА ВЫПОЛНЕНИЕ НУЖНО ВЫПОЛНИТЬ ДЕЙСТВИЕ

1) выбрать пункт меню Run/ Run
2) выбрать пункт меню Project/ Run
3) нажать «быструю кнопку» 
4) нажать «быструю кнопку» 

4. ПРЕРВАТЬ ВЫПОЛНЕНИЕ ПРОЕКТА ИЗ СРЕДЫ DELPHI МОЖНО

1) нажав «быструю кнопку» 
2) нажав «быструю кнопку» 
3) выполнив пункт меню Run/Program Reset
4) нажав комбинацию клавиш <Ctrl>+<F2>

Практическая часть

Задание 1. Создание окна приложения

Постановка задачи

Создать проект, содержащий код простейшей консольной программы, файлы проекта должны иметь стандартные имена, которые предлагаются по умолчанию.

Порядок выполнения

1. Создать в своей рабочей папке новую папку под именем **тема 1_1** при помощи соответствующего приложения Windows (*Мой компьютер* или *Проводник*).

2. Открыть оболочку Delphi. При этом автоматически создастся новый проект под Windows, включающий пустую форму – стандартное окно Windows.

3. Закрыть проект, выбрав пункт меню *File / Close All*.

4. Создать консольное приложение, выполнив пункт меню *File / New / Other* . При этом появляется Диалоговое окно *New Items* (Новые Приложения) (рис. 1б). Выбрать *Console Application* (Консольное Приложение).

Для нового проекта автоматически создается шаблон – текст головной программы. Он помещается в *Окно кода программы* как файл с расширением **`dpr`** и выглядит следующим образом (рис. 6):

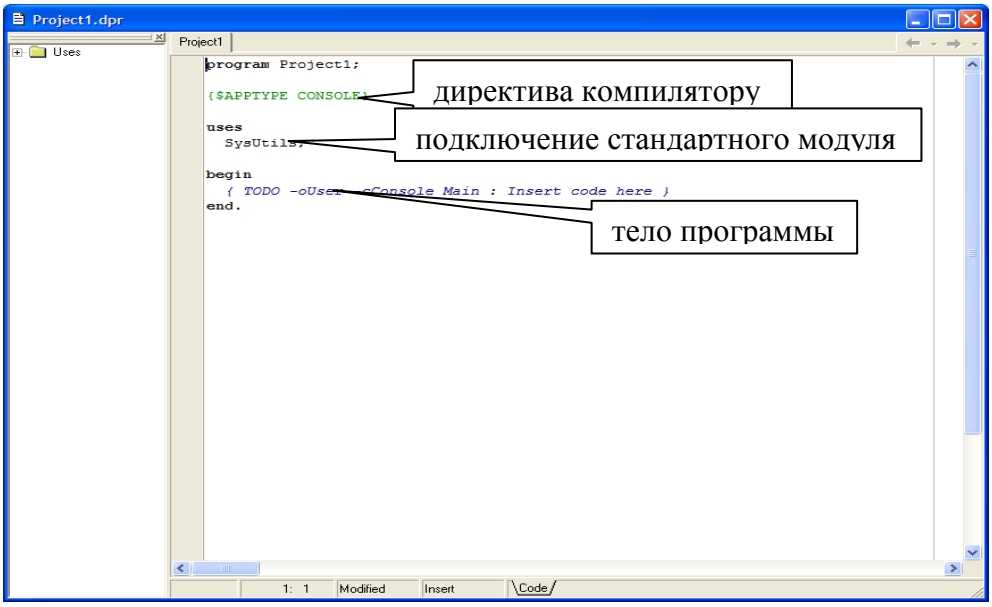


Рис. 6. Окно кода программы с шаблоном консольного приложения

Текст головной программы начинается служебным словом **`program`**, далее следует название программы, по умолчанию – `Project1`. Название можно менять по желанию. Данное название дает имя всему проекту, по нему будет сформирован и `exe`-файл.

Далее следует текст `{$APPTYPE CONSOLE}` – директива компилятору о том, что головная программа содержит консольное приложение. В редакторе Delphi директивы по умолчанию всегда выделены зеленым цветом.

В разделе **`uses`** указан программный модуль `SysUtils`, который система Delphi должна скомпоновать с данной программой при создании выполняемого файла.

Модуль `SysUtils` содержит библиотеку встроенных функций языка Object Pascal, например, таких как функции перевода строки в число и наоборот.

Между служебными словами **`begin`** и **`end`** помещен комментарий на английском языке, подсказывающий, что именно в этом месте нужно помещать операторы Object Pascal. Комментарии в редакторе Delphi выделяются синим цветом. В данном случае комментарий можно удалить.

Примечание: Целая строка удаляется, если поместить в нее курсор ввода и нажать комбинацию клавиш <*Ctrl*>+<*Y*>.

5. Сохранить файлы проекта, выбрав пункт меню *File/ Save All*. Delphi покажет диалоговое окно (рис. 4). Открыть в окне вновь созданную папку **тема 1_1** и сохранить в ней проект под именем, предлагаемым по умолчанию – **`project1.dpr`**.

Все необходимые файлы проекта сохранятся автоматически с этим именем и соответствующими расширениями.

Сделав это однажды, затем можно просто фиксировать все изменения, нажимая на горячую кнопку *Save* –  на панели кнопок.

6. Переключиться в Windows, открыть папку проекта **тема 1_1** с помощью приложения *Мой компьютер*, определить, какие файлы с какими именами составляют проект.

7. Вернуться в оболочку Delphi. В тело программы поместить один оператор – пустой оператор ввода: `Readln`. Посмотрите, как будет выглядеть текст Вашей первой программы (рис. 7):

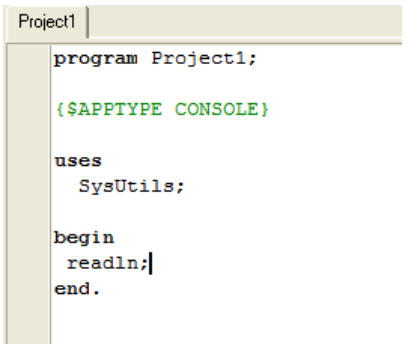


Рис. 7. Окно кода с текстом первой программы

Запустить приложение на выполнение – клавиша <*F9*>. Посмотреть, как выглядит первый проект, написанный в Delphi. Он представляет собой обычное окно Windows с черным экраном.

Примечание: На *Панели Задач* Windows появляется кнопка с названием программы – *Project1*.

Программа пока ничего не выполняет, а ожидает нажатия клавиши <Enter>.

8. Нажать клавишу <Enter>, вернуться в окно редактора кода Delphi.

9. Перечислить файлы проекта в своем отчете (можно поместить образ экрана, взятый клавишей <PrtScr>). Поместить в отчет внешний вид окна `Проводника` при выполнении проекта.

10. Не закрывая оболочку Delphi, закрыть текущий проект, выбрав пункт меню *File/ Close all* в главном меню окна Delphi.

Задание 2. Сохранение проекта с заданным именем

Постановка задачи

В новой папке создать проект, выводящий в консольное окно приветствие с Вашей фамилией, файлы проекта должны иметь имя, заданное пользователем – **myproject**.

Порядок выполнения

1. Создать новый проект, выполнив пункт меню *File/ New/ Other/ Console Application*.
2. Если все выполнено правильно, в окно кода помещается заголовок - `Project1`. Проверьте это!

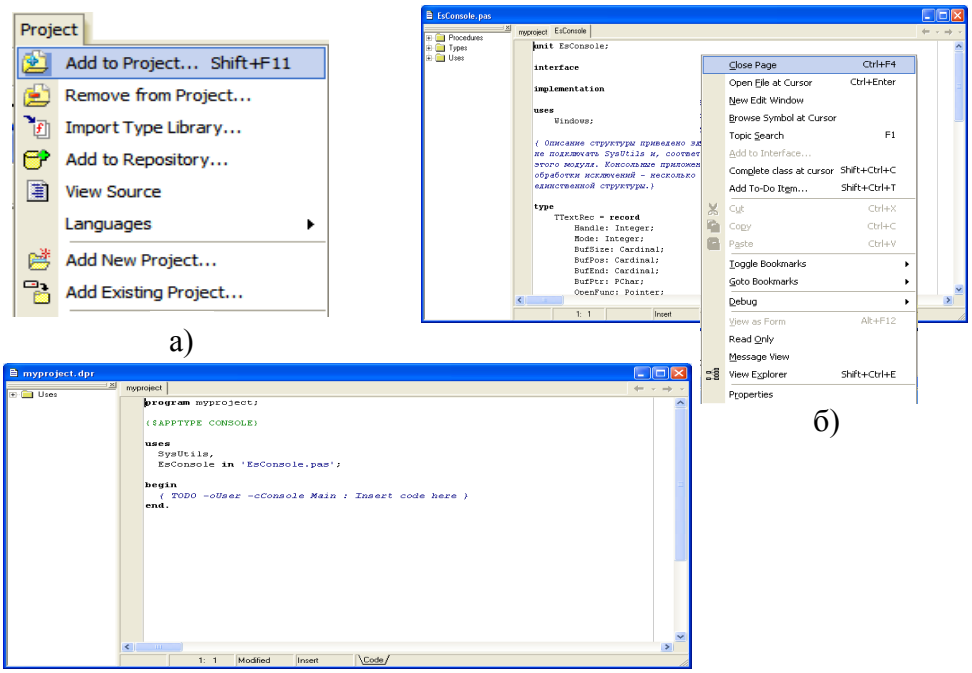
3. Сохранить текущий проект, выполнив пункт меню *File/ Save all*. Так как папки под проект еще нет, в открывшемся диалоге выйти на рабочую папку и создать в ней подпапку с названием **тема1_2**.

4. Открыть ее и сохранить под предложенным именем (**myproject.dpr**) разрабатываемую программу.

5. Обратить внимание на то, как изменились имена других файлов, входящих в проект: для этого открыть папку через приложение *Мой компьютер*.

6. Delphi не поддерживает вывод кириллицы на экран, поэтому подключим модуль, предназначенный для этого. **Скопировать модуль EsConsole.pas (располагается на сервере!) в папку с проектом.**

7. В оболочке Delphi выполнить пункт меню *Project/ Add to Project* (рис. 8а).



а) б)

в)

Рис. 8. Подключение нового модуля к проекту:

а) выбор пункта меню *Add to Project* (Добавить к проекту)

б) окно редактора кода с новым окном, содержащим текст подключаемого модуля

в) окно редактора кода, содержащее текст программы с подключенным модулем.

8. В окне Редактора кода появится новое окно (рис. 8б), содержащее текст подключаемого модуля, его можно закрыть. Щелкнуть правой кнопкой мышки по данному окну для вызова всплывающего меню (рис. 8б), и выберать пункт *Close Page* (Закрыть страницу).

9. Теперь текст программы выглядит так, как это изображено на рисунке 8в. В предложении `uses` через запятую перечисляются подключаемые модули и указывается файл, где находится подключаемый модуль.

10. Убрать комментарий в теле программы. Вместо него ввести оператор вывода на экран:

```
Writeln('Привет! Меня зовут <Ваша фамилия и имя>!');
```

И пустой оператор `Readln`; задерживающий возврат в редактор Delphi.

Проверить, как выглядит текст программы (рис. 9):

myproject

```
program myproject;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  EsConsole in 'EsConsole.pas';

begin
  writeln('Привет! Я - Иванов Иван!');
  readln;
end.
```

Рис. 9. Текст программы с приветствием

11. Запустить программу на выполнение.

12. Перечислить все файлы проекта в своем отчете, можно просто вставить образ папки проекта. А также вставить образ экрана при выполнении программы (окно с черно-белым выводом).

Задание 3. Операторы вывода Object Pascal

Постановка задачи

Организовать вывод констант (значения придумать самим) в соответствующем формате: целая константа должна быть выведена в формате :5, вещественная константа – в формате 10:3, символьная – в формате :2, логическая – в формате :6.

Методические указания

1. Для вывода информации в консольном программировании используются две формы процедуры вывода:

```
WRITE (<список вывода>);
WRITELN (<список вывода>);
```

<список вывода> может включать в себя имена переменных (или констант), разделенных запятой, а также выражения. В таком случае вначале вычисляется значение выражения, которое затем выводится на экран.

<список вывода> может отсутствовать, в этом случае говорят о **пустом** операторе вывода.

При использовании первой формы <список вывода> выводится на экран и курсор вывода остается на этой же строке; при использовании второй формы на экран выводится <список вывода> и происходит перевод курсора вывода на новую строку.

•Булевский тип данных

При выводе логических данных или выражений на экране появляется логическая константа.

Пример 1. Вывод значения булевской переменной:

```
. . .
Var b: Boolean;
begin
  b := False;
  writeln(b);
  . . .
End.
```

Результат выполнения (на экране):

```
FALSE
```

Пример 2. Проверка значения целой переменной на равенство 0:

```
. . .
Var x: integer;
begin
  x := 0;
  writeln(x=0);
  . . .
End.
```

Результат выполнения (на экране):

```
TRUE
```

•Символьный (строковый) тип данных

Для вывода символьных констант (или строк) используются апострофы. Выводимый текст помещается между двумя апострофами, может содержать только латинские символы. Для вывода кириллицы подключить соответствующий модуль русификации. На экране отображаются только символы между апострофами, сами апострофы не выводятся.

Примечание 1: В тексте программы символы между апострофами отображаются синим цветом.

Пример 1. Вывод на экран заголовка:

```
. . .
Writeln('Hello, World!');
. . .
```

Результат выполнения (на экране):

```
Hello, World!
```

•Целый тип

Значения целых переменных (выражений или констант) выводятся обычным образом.

Пример 1. Вывод на экран значений двух целых переменных:

```
. . .
Var I, J : integer;
Begin
  I := 1;
  J := 10;
  Writeln(I);
  Writeln(J);
  Readln;
End.
```

Результат выполнения (на экране):

```
1
10
```

Пример 2. Вывод значений двух целых переменных с заголовками:

```
. . .
Var I, J : integer;
Begin
  I := 1;
  J := 10;
  Writeln('i=',I,' j=',J);
  Readln;
End.
```

Результат выполнения (на экране):

```
i=1 j=10
```

•Вещественный тип данных

Значения вещественного типа на экран выводятся в форме с плавающей точкой.

Пример 1. Вывод значения вещественной переменной:

```
. . .
Var r: Real;
Begin
  R:=0.3;
  Writeln(R);
  Readln;
End.
```

Результат выполнения (на экране):

```
3.00000000000000E-0001
```

Для вывода вещественных переменных в форме с фиксированной точкой используют форматный вывод. Форматный вывод имеет две части, разделенные двоеточием: первая часть – целое число, задающее ширину поля вывода, вторая часть – целое число, задающее точность отображения (количество знаков после запятой). При выводе число «прижимается» к правой границе поля вывода.

Пример 2. Форматный вывод значения вещественной переменной в поле вывода шириной 10 позиций, из которых 2 отведены под дробную часть.

```
. . .
Var r: Real;
Begin
  R:=0.3;
  Writeln(R:10:2);
  Readln;
End.
```

Результат выполнения (на экране):

```
      0.30
```

Примечание: форматный вывод можно использовать для вывода данных других типов. В этом случае формат задает общую ширину поля вывода под значение.

Пример 3. Форматный вывод данных разных типов:

```
. . .
Var r: Real;
    i: integer;
    b: Boolean;
    c: Char;
Begin
  R:=0.31; i:=120; b:= True; c:='w';
  Writeln('r=',R:5:2,' i=',i:3,' b=',b,' c=',c:2);
  Readln;
End.
```

Результат выполнения (на экране):

```
 0.31 i=120 b=TRUE c=w
```

2. Структура программы на языке Object Pascal требует соблюдения следующих правил:

Начало программы – со служебного слова PROGRAM, за которым следует имя программы (по умолчанию PROJECT1).

В разделе описания констант помещаются **константы** – те данные, которые не могут менять своего значения в ходе выполнения программы; сразу с указанием их значения по формату:

<имя константы> = <значение константы>;

Пример:

```
Const c=10;
```

Если в программе используются несколько констант, то их описания размещаются через точку с запятой.

Пример:

```
Const c=10;
      k=5;
```

В разделе описания переменных **VAR** помещаются описания всех переменных программы с указанием их типа по формату:

<имя переменной> : <тип переменной>;

Пример:

```
Var a: integer;
```

Если несколько переменных имеют одинаковый тип, их можно указывать списком через запятую:

```
Var a,n: integer;
```

Описание переменных разных типов отделяются друг от друга точкой с запятой.

Пример:

```
Var a: integer;
    h: real;
```

После описания всех переменных, между служебными словами BEGIN и END помещаются операторы, обозначающие действия над переменными.

Пример программы на языке Object Pascal

В примере задаются две константы целого и вещественного типов, которые затем выводятся в формате: константа целого типа в формате :2, т.е. в поле шириной в 2 позиции, вещественная константа в формате :4:1, т.е. в поле шириной в 4 позиции из которых одна отведена под вывод дробной части.

```
Program Project1;

{$APPTYPE CONSOLE}

Uses EsConsole;

Const con1=3; con2=90.9;
```

```
Begin
  Writeln('const1=', con1:2);
  Writeln('const2=, con2:4:1);
  ReadLn;
End.
```

Задание 4. Организация ввода-вывода стандартных типов данных

Постановка задачи

Организовать ввод и вывод данных стандартных типов, используя диалог с пользователем. Вывод данных осуществить в соответствующем формате.

Методические указания

1. При вводе с терминала числа и символы можно набирать как на одной строке, так и на различных строках. При этом следует помнить, что ввод со следующей строки осуществляется в том случае, когда предыдущим оператором является `ReadLn`.
2. При работе в диалоговом режиме следует перед операторами ввода использовать оператор вывода на экран приглашения-подсказки о вводе информации.
3. Последним выполняемым оператором в программе поставить “пустой” оператор `ReadLn` для задержки возврата в оболочку.

Пример программы на языке Object Pascal

В приводимом примере описаны две константы, равные 3 и 90,9; две переменные целого типа; три переменные – вещественного типа и одна переменная символьного типа. Программа осуществляет ввод их значений с клавиатуры, и затем – вывод значений:

- значений констант;
- значение первой переменной увеличено вдвое;
- проверяется, равна ли 2 целая переменная числу 2;
- вывод вещественных чисел – без изменений;
- выведено значение символьной переменной без изменений.

```
Program MyTest1;

{$APPTYPE CONSOLE}

Uses EsConsole;

Const con1=3; con2=90.9;
Var Var1,Var2:Integer;{2 переменные целого типа}
    A,b,c:Real;{3 переменные вещественного типа}
    Ch1:Char;{1 переменная символьного типа}
Begin
  Writeln('Значение константы №1:', con1:2);
  Writeln('Значение константы №2:', con2:5:1);
  Writeln('Введите 2 числа целого типа');
  Readln(Var1,Var2);
```

```
  Writeln('Вы ввели числа:',Var1:4,Var2:4);
  Writeln('Результат увеличения первого числа вдвое:',Var1*2:5);
  Writeln('Второе число = 2?',Var2=2);
  Writeln('Введите 3 дробных числа');
  Readln(a,b,c);
  Writeln('Вы ввели:',a:8:2,b:8:2,c:8:2);
  Writeln('Введите любой символ');
  Readln(Ch1);
  Writeln('Вы ввели символ:',Ch1:2);
  ReadLn;
End. {MyTest1}
```

Результат работы программы:

```
Значение константы №1: 3
Значение константы №2: 90.9
Введите 2 числа целого типа:
3 9
Вы ввели числа:   3   9
Результат увеличения первого числа вдвое:    6
Второе число = 2? FALSE
Введите 3 дробных числа
2.3 4.2 1.41
Вы ввели:    2.30    4.20    1.41
Введите любой символ
S
Вы ввели символ S
```

ТЕМА 2. РЕДАКТИРОВАНИЕ И ОТЛАДКА ПРОГРАММ

Теоретическая часть

1. Редактирование программ

Пункт меню Edit

Данный пункт меню содержит основные команды работы с *Буфером Памяти* Windows для копирования и перемещения элементов текста программы (рис. 10).

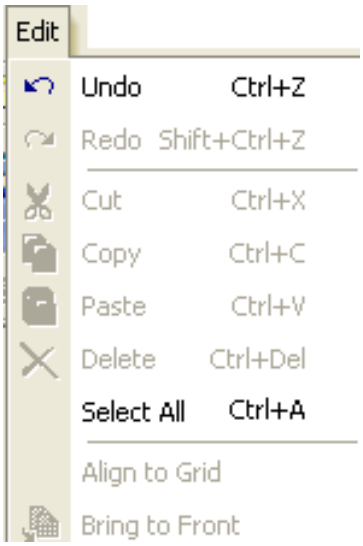


Рис. 10. Пункт основного меню Edit

- *Undo* – отменить последнюю выполненную команду;
- *Redo* – повторить последнюю выполненную команду;
- *Cut* – вырезать выделенный фрагмент текста в *Буфер Памяти*;
- *Copy* – скопировать выделенный фрагмент текста в *Буфер Памяти*;
- *Paste* – вставить фрагмент, находящийся в *Буфере Памяти*;
- *Delete* – удалить фрагмент текста;
- *Select All* – выделить весь текст.

Выделить фрагмент кода программы можно как принято в Windows – мышкой или клавишами управления курсора при нажатой клавише <*Shift*>.

Пункт меню Search

В этот пункт меню вынесены некоторые часто использующиеся команды редактора, а именно, команды поиска и замены (рис. 11).

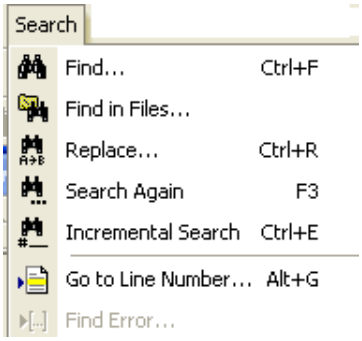


Рис. 11. Пункт меню Search

Подпункт Find

Служит для поиска информации в тексте программы. При выборе этойкоманда появляется диалоговое окно, в котором следует задать искомую последовательность символов и определить условия и область поиска. Условия поиска задаются с помощью флажков (рис. 12):

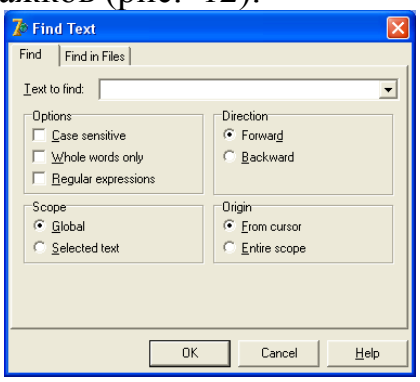


Рис. 12. ДО Find (Поиск)

- Case sensitive (различать прописные и строчные буквы);
- Whole words only (анализировать только целое слово или часть слова);
- Regular expression (распознавать включаемые в искомую строку спецификаторы формата).

Кроме того, с помощью кнопок-переключателей определяется область и направление поиска: областью поиска может быть весь текст (Global) или выделенный фрагмент (Selected text); начало или конец области может также помечаться курсором (From cursor); направление поиска может быть либо прямым, т.е. от начала области к концу (Forward), либо обратным (Backward).

Подпункт Replace

Подпункт служит для поиска и замены найденного фрагмента текста на новый.

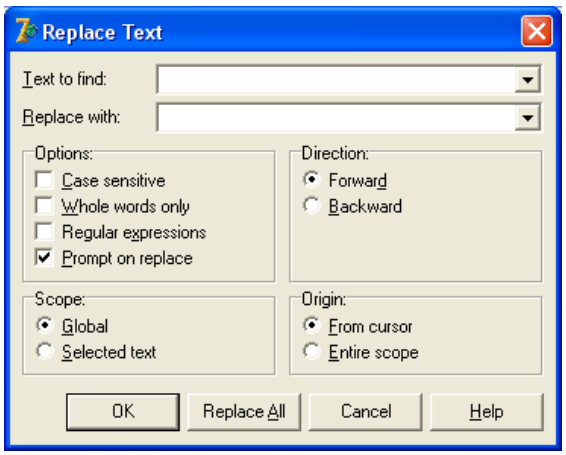


Рис. 13. ДО Replace (Замена)

Диалоговое окно (рис. 13), которое появляется при этом на экране, очень похоже на соответствующее окно опции Find; исключением является дополнительное поле (Replace with), в которое следует поместить строку замены. Если искомая строка найдена, система спрашивает, следует ли заменить только ее первое вхождение либо все вхождения сразу. Как и в случае опции Find, текст для поиска может быть взят из окна редактирования. Для этого курсор в окне редактирования должен находиться на необходимой строке.

Подпункт Search Again

Позволяет повторить последнюю команду опции Find, либо опции Replace с установленными условиями.

Подпункт Goto line number

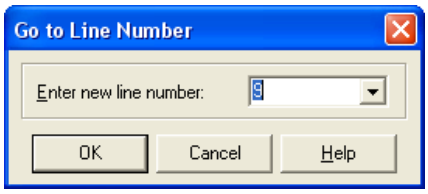
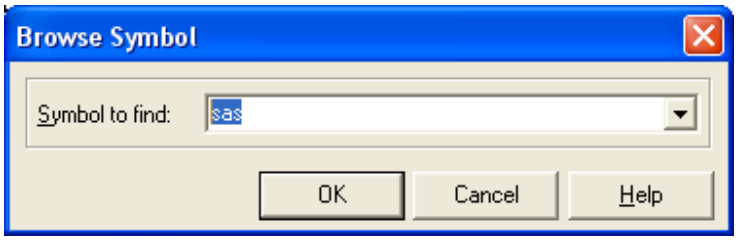


Рис. 14. Диалоговое окно Goto line number (перейти к строке с номером)

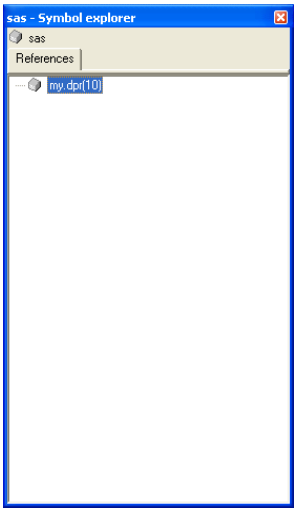
При выборе пункта меню появляется ДО (рис. 14), в котором в строку ввода вводится номер искомой строки. При нажатии кнопки ОК в тексте программы ищется участок текста, содержащий строку с указанным номером. Найденный участок появляется в окне редактирования.

Подпункт Browse Symbol

Позволяет быстро найти описание переменной или константы по ее имени и показывает, в какой строке кода какого проекта находится это описание (рис. 15).



а)



б)

Рис. 15. Диалоговые окна при работе с подпунктом Browse Symbol:
а) первый диалог – ввести имя искомой переменной
б) второй диалог – результат поиска

2. Отладка программы

Как известно, программ без ошибок не существует. Ошибки допускают как профессиональные программисты, так и начинающие. ИС Delphi имеет много различных средств, помогающих эффективной разработке приложений: обозреватель объектов, отладчик программ, разветвленная справочная система и т.д.

Ошибки можно разделить на синтаксические и логические.

Синтаксические ошибки

Синтаксические ошибки выявляются системой в процессе работы над программой. Если, например, ввести имя переменной русскими буквами и запустить программу на выполнение клавишей <F9>, в окне *Редактора кода* появится сообщение системы Delphi (рис. 16) об ошибке.

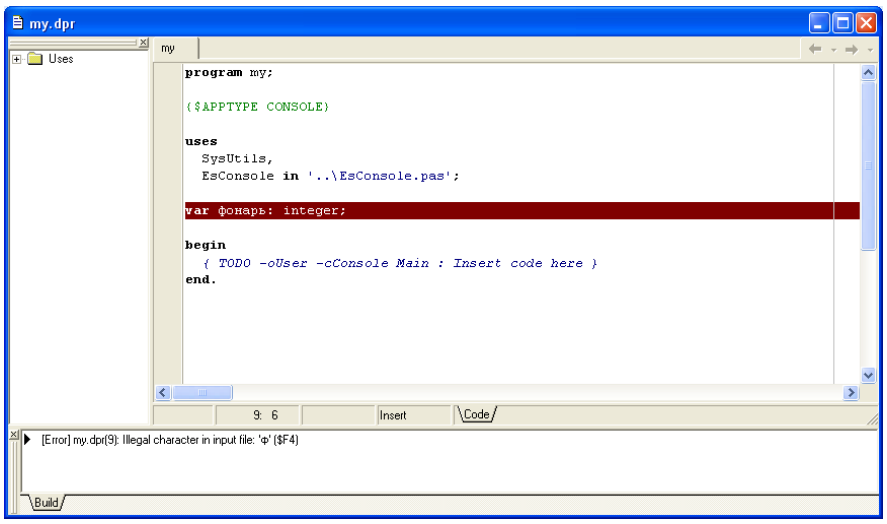


Рис. 16. Вывод сообщения об ошибке в окне Редактора кода

При этом система подсвечивает красной полосой строку, в которой обнаружена ошибка, и выдает расшифровку ошибки в нижней части окна редактора кода.

Логические ошибки

Логические ошибки возникают, когда код программы неправильно реализует алгоритм решения: наиболее часто встречаются такие ошибки, как выход за границы массива, переполнение при выполнении арифметической операции или ошибка, которая возникает при использовании некоторых вариантов исходных данных. Некоторые из таких ошибок можно отследить с помощью системы Delphi. Для этого в настройках проекта – соответствующее диалоговое окно вызывается пунктом меню *Project/Options* на вкладке *Compiler* (рис. 17) надо сделать **установки**:

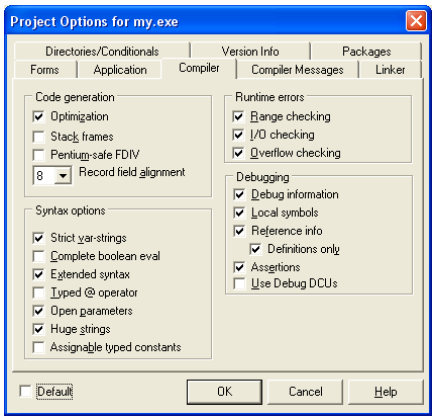


Рис. 17. Окно настроек отладки

- в группе *Code generation (Генерация машинного кода)* сбросить флажок *Optimization (Оптимизация)*. Когда компилятор создает оптимизированный код, он нередко вносит существенные улучшения в детали алгоритма.

Например, если ввести следующий код:

```
var x: integer;

begin
  x:=3+4;
end.
```

то после оптимизации этого кода строка вычисления значения переменной *X* становится невыполняемой (исполняемые операторы в редакторе кода помечаются слева голубой точкой). В данном случае нельзя узнать значения переменной *X* во время работы программы, потому что реально для нее не будет отводиться место в оперативной памяти.

А вот если добавить оператор, где используется эта переменная, например, оператор вывода, то в этом случае оператор вычисления значения переменой *x* становится выполняемым.

```
var x: integer;

begin
  x:=3+4;
  writeln(x);
end.
```

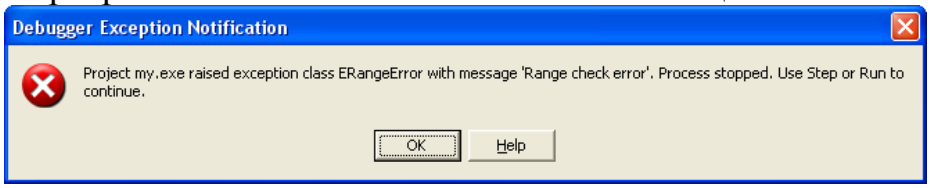
- в группе Runtime errors *(Ошибки времени выполнения)* должны быть установлены флажки Range checking *(Контроль выхода индекса за границу массива*), I/O Checking *(Контроль ошибок ввода/вывода)* и Overflow checking *(Контроль переполнения при целочисленных операциях)*. Так как все типы Object Pascal имеют строго ограниченные диапазоны типов, то, например, при сложении двух чисел, близких к максимально допустимому значению, получится число, которое не укладывается в этот диапазон. Последний флажок позволяет обнаруживать такие ошибки.

Например, если ввести следующий код:

```
var i,j: byte;

begin
  i:=255;
  j:=i*i;
  writeln(j);
  readln;
end.
```

то при запуске программы на выполнение появится сообщение об ошибке:



Сообщение можно перевести примерно следующим образом: «Проект my.exe вызвал прерывание класса ERangeError с сообщением «ошибка выхода за границы». Выполнение остановлено. Для продолжения работы использовать пункты меню *Run/ Step* или *Run/ Run*.»

При этом в окне редактора кода появляется синяя полоска, показывающая, в какой именно строке произошла остановка выполнения программы (рис. 18).

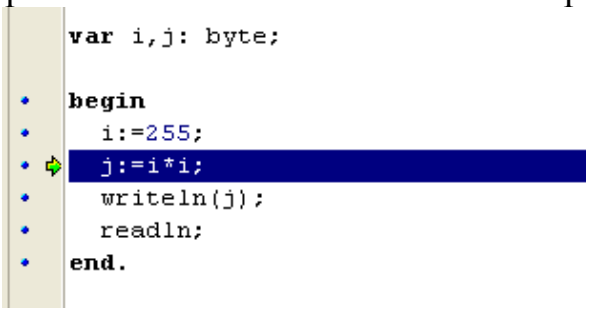


Рис. 18. Переход в пошаговый режим работы программы

Далее для отладки программы обычно переходят в пошаговый режим выполнения *Run/ Step* (режим трассировки, см. далее), или, если ошибка понятна, то прервать выполнение программы можно комбинацией клавиш *<Ctrl>+<F2>* или командой меню *Run/ Program Reset* и исправить ошибку.

- в группе Debugging *(Отладка)* установить флажки Debug information *(Добавление отладочной информации),* Local symbols *(Просмотр значений локальных переменных),* Reference info*(Просмотр структуры кода),* Assertions *(Включение процедуры Assert в машинный код)* и Use Debug DCUs *(Использование отладочных версий стандартных модулей библиотеки компонентов VCL)*.

Без отладочной информации отладка программы в системе Delphi вообще невозможна. Процедура *Assert* выполняет отладочные функции. В заключительной версии программы она, как правило, не нужна, а удалять ее вызовы из исходного текста неудобно – таких вызовов могут насчитываться сотни. Отключить генерацию машинного кода для этой процедуры можно с помощью флажка *Assertions*. Отладочные версии стандартных модулей содержат дополнительные режимы проверки корректности работы с компонентами Delphi.

Теперь, если в программе, запущенной из среды Delphi, встретится ошибка, например, выход индекса массива за границы, то выполнение программы прервется, а строка, в которой встретилась ошибка, будет подсвечена.

При этом система позволит быстро определить, в чем причина ошибки. Для этого нужно навести курсор мышки на какую-либо переменную, ее значение отобразится во всплывающей подсказке.

3. Трассировка программы

Режим трассировки или выполнение программы по шагам можно включить, выбрав пункт меню *Run/ Trace Into* или нажав клавишу *<F7>*. В этом слу-

чае операторы в коде программы будут отрабатывать лишь при нажатии этой клавиши.

В режиме трассировки обычно отлаживают работу программ, ищут логические ошибки. Поэтому при использовании данного режима применяют отладочное окно, в котором можно увидеть промежуточные результаты работы программы.

Вызвать отладочное окно можно, выполнив пункт меню *View/Debug Windows*. Работу с отладчиком разберем на практическом примере.

Пример:

Создать новый проект и вставить код:

```
program my;

{$APPTYPE CONSOLE}

uses
  SysUtils;

const n=5;
var a: array [1..n] of byte;
    i: Integer;

begin
 for i:=1 to n+1 do
  a[i]:=Random(100);
 readln;
end.
```

Примечание: в программе описывается массив из 5 целых чисел, и задаются его элементы с помощью датчика случайных чисел.

Чтобы установить в строке, где расположен первый выполняемый оператор кода, **точку прерывания** необходимо использовать клавиша <*F5*> или щелкнуть левой клавишей мышки по левому серому полю в окне редактора рядом с оператором. Визуально она отмечается красным цветом (рис. 19).

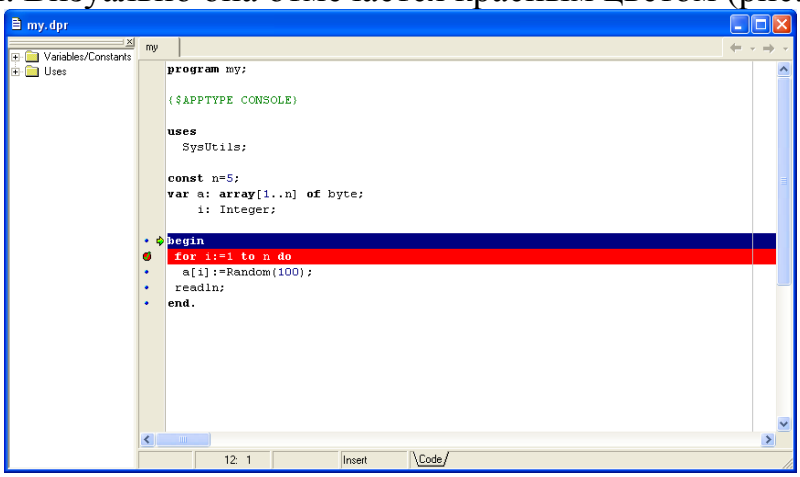


Рис. 19. Работа с программой в режиме трассировки

Если запустить программу на выполнение, то программа остановится, и управление будет передано отладочной системе Delphi, при этом на экране появится окно редактора кода, в котором синей полоской показана выполняемая строка кода (рис. 19).

При этом в заголовке главного окна Delphi появится информационное сообщение *my [Stopped] (Выполнение проекта my приостановлено)*.

Для дальнейшей **работы с программой в режиме трассировки** используют пункт меню *Run/Trace Into (Запуск/Войти внутрь*) (клавиша <*F7*>) или *Run/Step Over (Запуск/Перешагнуть) (*клавиша <*F8*>).

В данном режиме можно подвести курсор мышки к идентификатору `i` и увидеть его значение на каждой итерации цикла, а также посмотреть значение элемента массива (весь массив в целом можно увидеть, если подвести курсор мышки к имени массива.)

Чтобы **выйти из режима трассировки**, необходимо выполнить пункт меню *Run/Program Reset (Запуск/Сброс программы)* или использовать комбинацию клавиш <*Ctrl*>+<*F2*>.

Снять точку прерывания можно с помощью клавиши <*F5*> или щелчком мыши по левому полю оператора в окне *Редактора кода*.

Точек прерывания в тексте программы может быть установлено несколько. Просмотреть их все можно, выбрав пункт меню *View/Debug Windows/Breackpoints* (рис. 20).

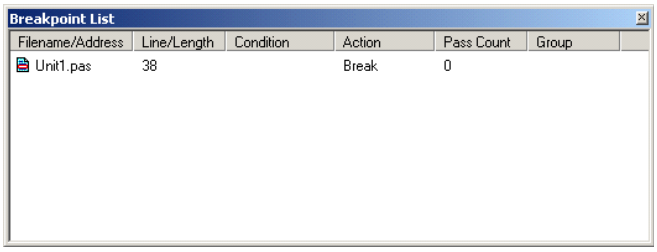
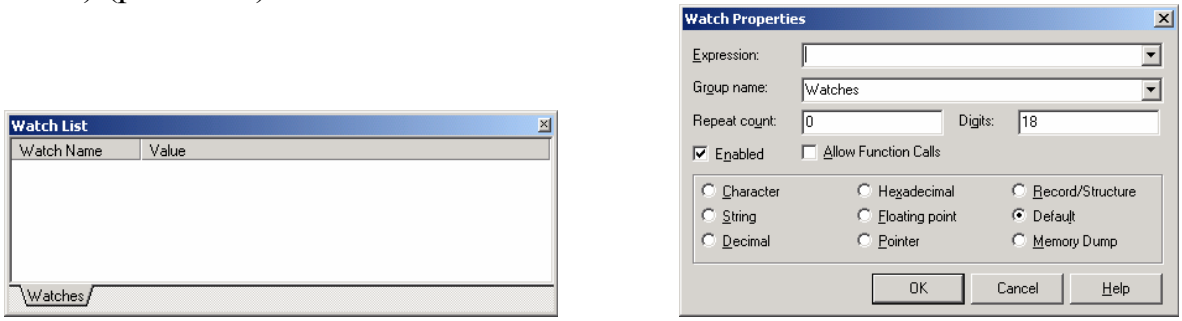


Рис. 20. Список точек прерываний программы

Если во время работы с программой возникает необходимость просмотра нескольких значений переменных, то можно воспользоваться окном *Watch*, которое открывается с помощью пункта меню *Run/Add Watch* (*Запуск/Добавить слежение*) (рис. 21а).



а) б)

Рис. 21. Окно отладки *Watch*:

а) список переменных с их значениями;

б) диалог добавления переменной в окно *Watch*

В диалоге (рис. 21б) нужно задать:

•в поле Expression *(Выражение)* – выражение или имя переменной, значения которой нужно отслеживать в ходе выполнения программы;

•с помощью переключателей в нижней части окна задать тип значения, если не требуется преобразования типа, то можно оставить значок Default *(По умолчанию)*;

•если исследуется массив, в поле Repeat count *(Число элементов)* указывается, количество выводимых элементов массива; если в поле Expression указан не первый элемент массива, то при выполнении программы в окне отладки отобразятся значения count элементов массив, начиная с указанного.

•в поле Digits *(Разрядность)* задается разрядность выводимых значений;

•флажок Enabled *(Включено)* временно отключает контроль значений некоторых переменных списка;

•флажок Allow Function Call *(Разрешить вызов функций)* разрешает использование в выражениях вызовов функций.

После нажатия кнопки *OK* на экране появляется окно *Watch List (Список отладки)* (Рис. 21а), в котором помещается список отслеживаемых величин. Внести дополнительные переменные в него можно, нажав клавишу <*Insert*> на клавиатуре (снова появится окно *Watch Properties* (Рис. 21б)).

4. Динамическая подсказка

При выполнении программы в пошаговом режиме можно получить так называемую «динамическую» подсказку о величине интересующей вас переменной. Для этого достаточно подвести мышку к этой величине и в всплывающем окне можно увидеть текущее значение переменной (рис. 22).

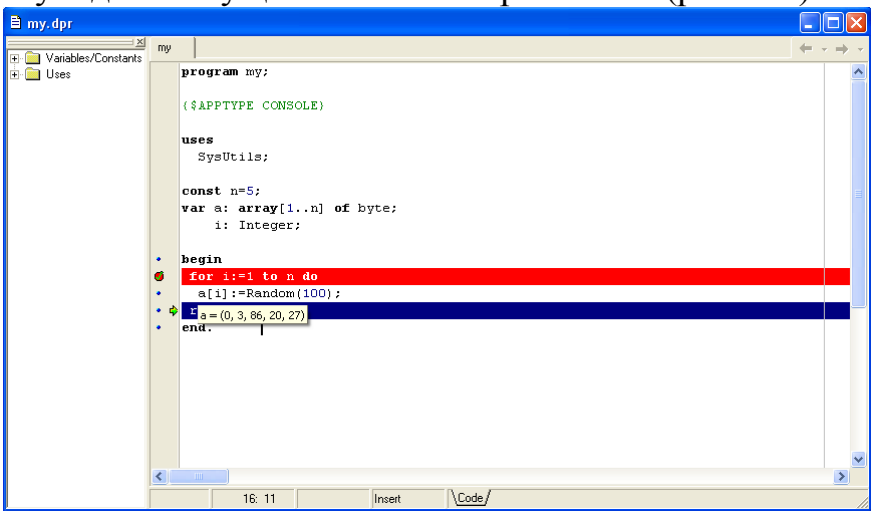


Рис. 22. Вывод «динамической» подсказки при работе над проектом в режиме отладки

Контрольные вопросы

1. Какие существуют средства для отладки программы в оболочке Delphi 7.0?
2. Как вывести окно отладки на экран?
3. Как включить пошаговый режим отладки?
4. В каких случаях используют динамическую подсказку?

Тестовые задания

Выберите один или несколько правильных ответов.

1. ПРИ НАПИСАНИИ КОДА ПРОГРАММЫ РАЗЛИЧАЮТ ОШИБКИ…
   1) логического типа
   2) синтаксического типа
   3) стилистического типа
   4) динамического типа

2. ПРИ ОТЛАДКЕ ПРОЕКТА DELPHI В ОКНО WATCH МОЖНО ВЫВЕСТИ…
   1) только одну переменную
   2) только одно из значений массива
   3) все значения массива
   4) значения нескольких переменных

3. ДЛЯ РАБОТЫ С ПРОЕКТОМ В ПОШАГОВОМ РЕЖИМЕ ИСПОЛЬЗУЕТСЯ КЛАВИША…
   1) <F1>
   2) <F3>
   3) <F5>
   4) <F7>
   5) <F9>

Практическая часть

Задание 1.

Запустить проект, рассмотренный ранее в примере, в режиме трассировки вывести в окно отладки значения:

- первых трех элементов массива;
- последних трех элементов массива.

Запишите их в файл отчета.

Задание 2.

Запустить предыдущий проект, в режиме трассировки вывести значения массива, используя «динамическую» подсказку. Запишите их в файл отчета.

Задание 3. Отладка готовой программы.

Постановка задачи

В приведенной программе найти ошибки и исправить их.

Методические указания

1. Создать новый проект. Скопировать программу, приведенную в индивидуальном варианте, в окно редактора кода. Сохранить проект.
2. Найти ошибки. Ошибки описать в отчете.
3. Определить результат и вывести найденное значение в фиксированном формате.
4. Окно результата работы программы привести в отчете.

Задание 4. программирование линейных алгоритмов с данными стандартных типов с использованием стандартной функции SizeOf.

Постановка задачи

С помощью стандартной функцией SizeOf определить размер стандартного типа, указанного в задании. Над данными стандартных типов осуществить действия, указанные в индивидуальном задании.

Методические указания

1. Ввод данных стандартных типов осуществить с клавиатуры.
2. Провести с этими данными указанные действия.
3. Вывод не предполагает использования модуля EsConsol. Заголовки результатов можно выводить латинскими символами.
4. Отчет включает в себя текст программы и окно результата работы.
5. При выполнении задания код символа определяется стандартной функцией Ord, которая имеет формат:

Формат функции:

```
function Ord ( Variable : Any type ) : Integer;
```

Справка по стандартной функции SizeOf:

Формат функции:

1 вариант: `function SizeOf ( Variable : Any type ) : Integer;`

2 вариант: `function SizeOf ( Type ) : Integer;`

Стандартная функция `SizeOf` возвращает размер (в байтах) своего аргумента, заданного в виде переменной (`Variable`) или типа (`Type`).

Пример:

```
var
  intNumber : Integer;
begin
```

```
  // Показ размеров численных типов данных
  writeln('Size of Integer=',SizeOf(Integer));
  writeln('Size Of intNumber=',SizeOf(intNumber));
 // Определение кода символа «d»
  IntNumber:=Ord('d');
  Writeln(intNumber);
end.
```

На экране появится:

```
Size of Integer=4
Size Of intNumber=4
100
```

Задание 5. Вычисление выражения

Постановка задачи

Запрограммировать формулу, используя стандартные функции и правила записи формул на языке Object Pascal.

Методические указания

1. Исходные данные задать оператором присваивания.
2. Если в формуле встречается функция, отсутствующая в Object Pascal, то ее выразить через функции, имеющиеся в языке, например, функция возведения в степень может быть выражена через функции экспоненты и логарифма:

y^5 как `exp(5*ln(y))`

3. Если в задании не указано, каким символом подчеркивать ответ, то можно использовать символ «–».

Пример

Найти значение функции:

$$y = \sqrt{\left|\frac{\cos kx - b}{a^2 + b^2}\right|} - e^{|a-b|} + \frac{tgk^2 x}{a - \sin kx}$$

Исходные данные: $a = -1.7;\ b = 2.32;\ k = 5;\ x = 5.7;$

Ответ задачи выдать на экран в следующем виде:

a) отступить 10 позиций и напечатать слова “ ИСХОДНЫЕ ДАННЫЕ”;
b) подчеркнуть эти слова символом “ *”;
c) под заголовком отпечатать значения исходных данных в виде:

A=……. B=…….
K=……. X=…….

d) пропустить две строки и напечатать в центре ответ в виде:

Y=…….

Порядок выполнения

1. Определяем количество необходимых данных: заданы значения четырех переменных *a, b, k, x*; для результата заведем отдельную новую переменную *y*.

2. По виду приведенных значений для исходных данных определяем их тип: *a, b, x* – вещественные, результат *y* – тоже вещественный, так как для вычисления значения *y* используются вещественные переменные и, кроме того, в формуле участвуют стандартные функции *cos,* exp и т.п., всегда дающие вещественный результат.

3. Можно приступить к кодированию:

```
program Formula_1;
{$APPTYPE CONSOLE}
Uses EsConsole;

var a,b,x,y:Real;
    k:integer;
Begin
 a:=-1.7;b:=2.32;k:=5;x:=5.7;
 y:=sqrt(abs((cos(k*x)-b)/(a*a+b*b)))-
   exp(abs(a-b))+sin(k*k*x)/cos(k*k*x)/(a-sin(k*x));
 writeln('          Исходные данные');
 writeln('          ***************');
 writeln('a=',a:4:1,' b=',4:2);
 writeln('k=',k:2,' x=',x:3:1);
 writeln;writeln;
 writeln('              y=',y:8:3);
 readln
end.
```

4. Результат работы программы:

```
         Исходные данные
           ***************
a=-1.7 b= 4
k= 5 x=5.7

               y= -56.500
```

Задание 6. Использование стандартных функций TRUNC и ROUND

Постановка задачи

1. Найти значение функции Y(X) при заданном X. Используя стандартные функции Паскаля, вычислить Y1=[Y], где [ ] означает целую часть числа Y, и значение Y2[Y ± 0.5].
2. Записать выражение, зависящее от координат точки X1 и Y1 и принимающее значение TRUE, если точка принадлежит заштрихованной области,

и FALSE, если не принадлежит. Для заданной точки вычислить это выражение и результат выдать на экран.

Методические указания

1. Организовать ввод значения переменной *x* с клавиатуры.

2. Область значений, обозначенную значком ∩ кодировать с помощью логической операции **and**

3. В отчет поместить листинг программы, а также внешний вид работающей программы.

4. Справка по функциям.

Формат функций:

```
function Trunc ( const Number : Extended ) : Integer;
function Round ( const Number : Extended ) : Int64;
```

Функция `Trunc` возвращает целочисленную часть числа с плавающей запятой. Она возвращает эту часть как целочисленное значение.

Функция `Round` округляет число с плавающей запятой (`Number`) до целого значения. Округление использует банковские правила, где точная половина значения вызывает округление к четному числу:

Пример 1

12.4 округляется до 12
12.5 округляется до 12
12.6 округляется до 13

13.4 округляется до 13
13.5 округляется до 14
13.6 округляется до 14

Пример 2

```
Begin
  writeln('Round(12.75) = ',Round(12.75));
  writeln('Trunc(12.75) = ',Trunc(12.75));
end;
```

На экране:

```
Round(12.75) = 13
Trunc(12.75) = 12
```

Пример 2

1) $y = 3^{1-x}\sin(x)$ при X= -1,5

2) координаты исследуемой точки (0,5; 1,2), область значений описывается выражением: $(x \geq 0) \cap (y \geq 0) \cap ((\sqrt{x} + \sqrt{y}) < 1)$

```
Program Example3;
{$APPTYPE CONSOLE}
Uses EsConsole;

Var x,x1,y,y1:Real;bool:boolean;
Begin
 Writeln('Ввести X'); Readln(x);
 y:=exp((1-x)*ln(3))*sin(x);
 Writeln('для x=',x:8:3,' y=',y:8:3);
 writeln('y1=',Trunc(y):4,' y2=',round(y):4);
 writeln('Ввести координаты точки плоскости');Readln(x1,y1);
 bool:=(x1>=0)and(y1>=0)and((sqr(x)+sqr(y))<=1);
 writeln(bool:20);
 writeln(' ':8,
   'Программа составлена студентом Ивановым И.И., группа 0001');
 readln
end.
```

Результат работы программы:

```
Ввести X
-1.5
для x=   -1.50 y= -15.549
y1= -15 y2= -16
Ввести координаты точки плоскости
0.5 1.2
               FALSE
         Программа составлена студентом Ивановым И.И., группа 0001
```

ТЕМА 3. ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ

Теоретическая часть

Для организации развилки в программах Object Pascal используется условный оператор, имеющий две формы:

1. формат **краткой формы**:

```
IF <условие> then <оператор>;
```

Изобразить работу этого оператора можно в виде блок-схемы, приведенной на рис. 23.

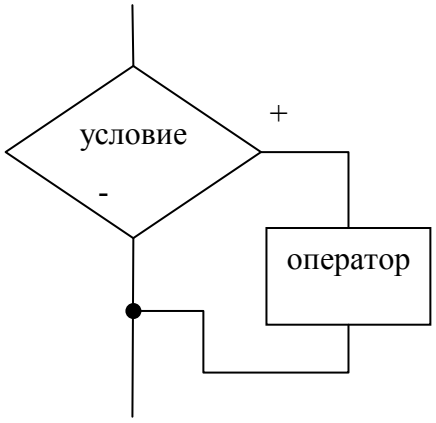


Рис. 23. Блок-схема неполного условного оператора

Если <условие> – истинно, то выполняется <оператор>, записанный за словом **then**, или говорят «расположенный на веточке THEN». Если <условие> – ложно, то управление программой передается на оператор, следующий за условным в программе.

2. формат **полной формы**:

```
IF <условие> then <оператор1> else <оператор2>;
```

Изобразить работу этого оператора можно блок-схемой (рис. 24).

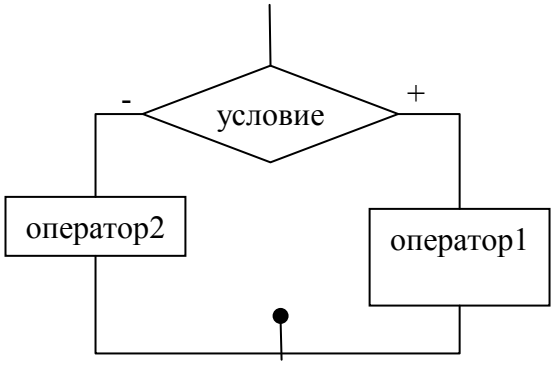


Рис. 24. Блок-схема условного оператора полной формы

Если <условие> истинно, то выполняется <оператор1>, записанный за словом **then**, если <условие> ложно, то выполняется оператор <оператор2>,

расположенный на веточке **else**. И только потом программа передает управление на оператор, следующий за условным.

Условие – это логическое выражение языка, получающееся при сравнении разных величин или выражений или при применении логических операций.

Примеры логических выражений:

```
S<0
('a'<dist) and (c>=9)
no (g=j)
```

Пример составления программы с разветвленной структурой

На практике алгоритмы линейной структуры встречаются редко, чаще всего приходится иметь дело с разветвляющимися алгоритмами. Такие алгоритмы реализуются с помощью условных операторов. Например, для нахождения значения y:

$$y = \begin{cases} x^2 + 1, & \text{если } x < 0; \\ x - 2{,}1, & \text{если } x > \pi/2; \\ \sin x, & \text{если } 0 \le x \le \pi/2; \end{cases}$$

придется воспользоваться условным оператором.

1 этап:

Очевидно, в программе будут три оператора присваивания:

```
y := sqr(x)+1 ;
y := x - 2.1 ;
y := sin (x) ;
```

2 этап:

Но эти операторы должны выполниться не последовательно друг за другом, а только в зависимости от значения переменной x, значит, перед вычислением значения y, необходимо проверить значение x. Проверка осуществляется с помощью условного оператора: if-then.

В данном случае удобно осуществить проверку в трех условных операторах:

```
if x < 0 then y := sqr(x)+1;
if x > 0 then y := x - 2.1;
if (x >= 0) and (x<=Pi/2) then  y := sin(x);
```

3 этап:

Для ввода исходных значений и вывода результата используются операторы:

```
Writeln('Input x:');
ReadLn(x);
{Здесь расположить операторы обработки значений x и y из 2 этапа}
Writeln('Answer:', y:10:3);
```

4 этап:

Описание типа переменных, используемых в программе, расположить до тела программы:

```
Var x,y:Real;
```

5 этап:

Собирая все воедино, получается следующий код:

```
Program MyTest2;
{$APPTYPE CONSOLE}
Var x, y: Real;
Begin
   Writeln('Input x:');
   ReadLn(x);
   if x < 0
     then y:=sqr(x)+1;
   if x > 0
     then y:=x - 2.1;
   if (x >= 0) and (x<=Pi/2)
     then  y:=sin(x);
   Writeln('Answer:', y:10:3);
   ReadLn;
End.
```

Результат работы программы:

```
Input x:
1.57
Answer:     1.000
```

Контрольные вопросы

1. Какие операторы используют для организации разветвления в программе?
2. Есть ли разница между операторами, приведенными в пунктах a и b?

a.
```
If a=b then x:=7 else y:=8;
```
b.
```
If a=b then x:=7;
If a<>b then y:=8;
```

Тестовые задания

Выберите один или несколько верных ответов.

1. **ВСЕГДА** В ОБРАЗОВАНИИ УСЛОВНОГО ОПЕРАТОРА УЧАСТВУЮТ КОНСТРУКЦИИ…:
   1) <условное выражение>
   2) If
   3) Then
   4) <оператор присваивания>
   5) Else
   6) <оператор вывода>
   7) <оператор ввода>
2. ВЕРНЫМИ ЗАПИСЯМИ УСЛОВНЫХ ВЫРАЖЕНИЙ ЯВЛЯЮТСЯ…

1) S=0
2) R:=0
3) D<0
4) F=<0
5) H>=D

3. СРЕДИ ПЕРЕЧИСЛЕННЫХ НИЖЕ ОПЕРАТОРОВ УСЛОВНЫЕ – ЭТО ОПЕРАТОРЫ В ПУНКТАХ…
1) Readln(a);
2) **if** a=0 **then** s:=0;
3) **if** a=0 **then if** b=0 **then** s:=0;
4) s:=0;
5) writeln(b);

4. СИНТАКСИЧЕСКИ ВЕРНЫЕ ЗАПИСИ УСЛОВНЫХ ОПЕРАТОРОВ – ЭТО …
1) **If** a:=0 **then** b:=0 **else** b:=1;
2) S:=0; **then** a=0;
3) **If** c=0 **then** s:=0;
4) **If** s=0 **else** c:=1;
5) **If** a=0 **then** d:=1;

Практическая часть

Задание 1. Нахождение значения функции по условию

Постановка задачи

Составить программу для нахождения значения функции в зависимости от значений исходных данных:

$$f(x,y,z)=\begin{cases} x+z, \text{если } x<y \\ y+z, \text{если } x>y \\ z, \text{если } x=0 \end{cases}$$

a) *x=1; y=2; z=3;*
b) *x=3; y=1; z=7;*
c) *x=0; y=8; z=2;*

Методические указания

1. При наборе текста программы строго соблюдать структуру условного оператора, как в рассмотренном примере программы `MyTest2`.

2. Проверить работу программы для всех заданных вариантов исходных значений, используя окно отладки Watches. Для этого:

- открыть окно отладки;
- занести в него имена исходных переменных;

- запустить в пошаговом режиме программу и ввести с клавиатуры значения переменных каждого из вариантов, указанных в индивидуальном задании, проверить, как работает программа.

3. В отчет включить результат работы программы при всех вариантах исходных данных.

Задание 2. Нахождение значения переменной по условию

Постановка задачи

Даны два вещественных числа. Если второе число больше первого на 4 единицы, разделить первое на 3, второе оставить без изменения, иначе – увеличить втрое первое, а второе оставить без изменения.

Методические указания

1. Исходные значения считываются с клавиатуры.

2. Полученные значения вывести на экран с заголовками.

ТЕМА 4. ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

Теоретическая часть

Цикл – это повторное выполнение каких-либо операторов в программе. Для реализации этого механизма в Object Pascal существует три формы операторов цикла. Соответственно, алгоритм, который содержит такие операторы, называется циклическим.

1 оператор цикла имеет формат:

```
FOR <параметр>:=<нач.знач-е> TO <кон.знач-е>DO <тело цикла>;
```

Называется **цикл с параметром**, используется в тех случаях, когда известно точное количество повторений в цикле, или говорят, **итераций** цикла.

Количество повторений задается в заголовке цикла и определяется как:

```
<кон.знач-е> - <нач.знач-е> + 1
```

При этом параметр цикла пробегает значения от первого до последнего с единичным шагом, поэтому *обязательным условием* является использование в операторе в качестве параметра цикла переменной дискретного типа.

Пример

```
Var i: integer;
. . .
For i:=1 to 10 do write(i:2);
. . .
```

Оператор выполнится 10 раз, параметр изменит свое значение от 1 до 10 с шагом 1, в результате на экране появятся числа от 1 до 10.

Оператор цикла с параметром имеет еще одну форму:

```
FOR <параметр>:=<нач.знач-е> DOWNTO <кон.знач-е> DO <тело цикла>;
```

В этом случае параметр цикла пробегает значения от начального до конечного с шагом -1.

Пример

```
Var ch: char;
. . .
For ch:='z' downto 'a' do write (ch:2);
. . .
```

В результате выполнения программы на экране появится английский алфавит в обратном порядке.

2 оператор цикла имеет формат:

```
WHILE <условие на вход> DO <тело цикла>;
```

Называется **цикл с предусловием,** используется в случаях:

- когда неизвестно точное количество повторений в цикле,
- или когда шаг цикла не дискретный,

- или когда шаг цикла не равен 1.

Основные условия использования цикла с предусловием:

- условие выполнения цикла должно быть определено до начала цикла;

- в цикле обязательно должен быть оператор, изменяющий условие выполнения цикла так, чтобы он не был бесконечным.

Пример

```
Var i: integer;
. . .
I:=1;
While i<=11 do begin
  Write(i:3); i:=i+2;
End;
. . .
```

В данном случае шаг цикла имеет дискретный тип и известно количество повторений цикла (6 итераций), но шаг цикла равен двум. Поэтому для организации цикла был использован оператор с предусловием.

3 оператор цикла имеет формат:

```
REPEAT <тело цикла> UNTIL <условие на выход>;
```

Называется **цикл с постусловием,** используется в случаях:

- когда неизвестно точное количество повторений в цикле,

- или когда шаг цикла не дискретный,

- или когда шаг цикла не равен 1.

Основные условия использования цикла с предусловием:

- в цикле обязательно должен быть оператор, изменяющий условие выхода из цикла так, чтобы он не был бесконечным.

Основное отличие от цикла с предусловием состоит в том, что вначале выполняется итерация цикла, а затем проверяется условие. И условие формируется таким образом, чтобы оно принимало истинное значение для остановки выполнения цикла.

Пример

```
Var r: real;
. . .
R:=0;
Repeat
  R:=r+0.2;
  Writeln(R:6:2);
Until r>10;
. . .
```

Применен оператор с постусловием, так как шаг цикла – недискретный.

Пример составления программы с циклом

Постановка задачи

Протабулировать (то есть получить таблицу значений) функцию $y = \sin x + \cos x$ на отрезке (-π; π) для 20 значений аргумента.

Решение

Для решения задачи определим переменные:

- x – значение аргумента;
- y – значение функции;
- h – значение шага изменения аргумента;
- n – количество точек на отрезке;
- i – параметр цикла

Шаг изменения аргумента функции вычисляется по формуле: $h = \frac{(b-a)}{(n-1)}$.

Для автоматического задания изменения значений аргумента используем цикл **For-to**, так как точно известно количество итераций цикла (20).

Блок-схема алгоритма приведена на рис. 25.

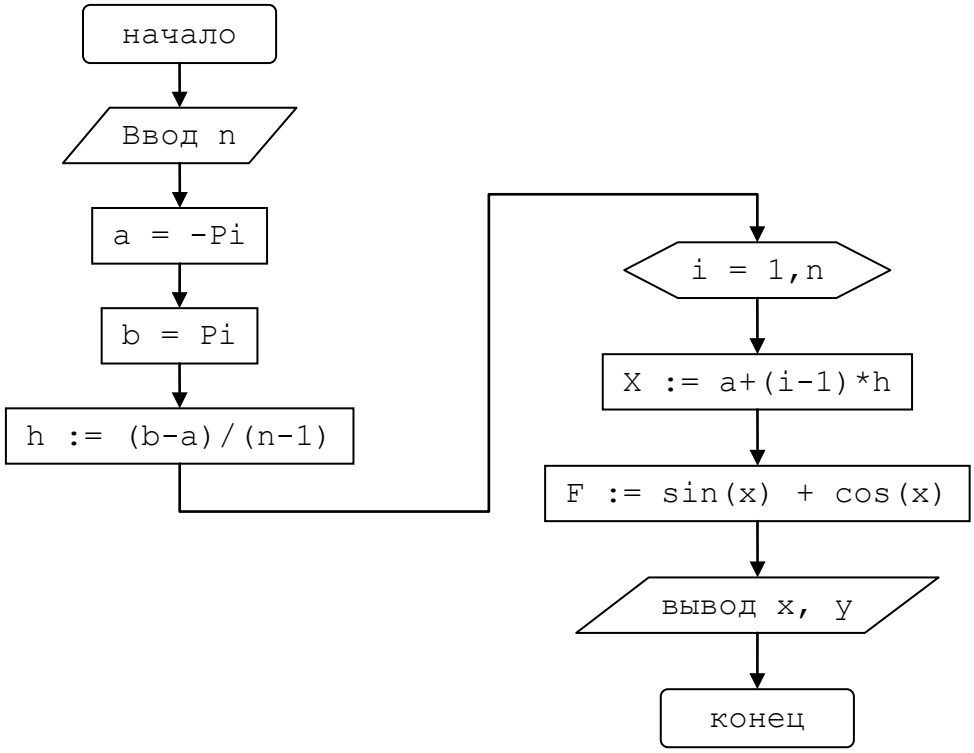


Рис. 25. Блок-схема алгоритма табулирования функции

Листинг программы

```
program For1;
{$APPTYPE CONSOLE}
const a=-Pi;b=Pi;
var x,f,h:real;
    n,i:integer;
begin
 writeln('Введите количество точек');readln(n);
 h:=(b-a)/(n-1);x:=a;
 for i:=1 to n do
 begin
  x:=a+(i-1)*h;
  f:=sin(x)+cos(x);
  writeln('x=',x:10:3,' f=',f:10:3);
 end;{for}
```

```
 readln;
end.
```

Результатом работы является табличка значений аргумента и соответствующие им значения функции (для экономии места не приводится).

В языке программирования Object Pascal принято понятие «вложенности». Оно означает, что некоторый составной оператор **содержит еще один такой же** оператор, или еще говорят «оператор вложен в другой оператор».

Примеры

1. Вложенный условный оператор:

```
If a<>f then
              If a<s then f:=s;
```

2. Вложенный оператор цикла

```
While s<10 do begin
   Sum:=0;  i:=1;
   While i<=3 do begin
      Sum:=sum+i;
      i:= i+1;
   end;{While i<=3}
end;{While s<10}
```

Пример программы с вложенным циклом FOR

Постановка задачи

Вывести на экран таблицу чисел следующего вида:

1 2 3 4 5

1 2 3 4 5

1 2 3 4 5

Для решения задачи воспользуемся вложенными циклами.

Решение

1. Для вывода используем оператор `Writeln`, для организации циклов переменные `I` и `j`. Внешний цикл будет отвечать за перебор строк, внутренний цикл – за перебор столбцов.

Примечание: Строки пронумерованы для пояснений, приводимых ниже.

```
 1 program forfor1;
 2 {$APPTYPE CONSOLE}
 3 const n=3;m=5;
 4 var i,j:integer;
 5 begin
 6  for i:=1 to n do begin
 7   for j:=1 to m do write(j:2);
 8   writeln;
 9  end;{for-i}
10  readln;
11 end.
```

Результатом работы программы будет искомая табличка цифр.

2. Если теперь изменить в строке 7 оператор вывода на следующий:

```
write(i:2);,
```

то на экране увидим строчки из одинаковых цифр:

```
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
```

3. Делая значения границ цикла динамическими, можно получить различные варианты таблиц вывода, например, если изменить границы внутреннего цикла по параметру `j` (строка 7):

```
for j:=1 to m-i do write(j:2);
```

получим следующую табличку:

```
1 2 3 4
1 2 3
1 2
1
```

Контрольные вопросы

1. Какой из операторов цикла следует применить, если заранее известно количество повторов и шаг цикла – дискретный?
2. Какой из операторов цикла следует применить, если шаг цикла не дискретный, заранее неизвестно, сколько раз должен выполниться этот цикл и должен ли вообще выполниться?

Тестовые задания

Выберите один или несколько верных ответов.

1. ДЛЯ ОБРАЗОВАНИЯ ОПЕРАТОРА ЦИКЛА С ПАРАМЕТРОМ ИСПОЛЬЗУЮТСЯ КОНСТРУКЦИИ…

1) for
2) to
3) while
4) repeat
5) until
6) <условие>
7) Do

2. ДЛЯ ОБРАЗОВАНИЯ ОПЕРАТОРА ЦИКЛА С ПРЕДУСЛОВИЕМ ИСПОЛЬЗУЮТСЯ КОНСТРУКЦИИ…

1) for
2) to
3) while
4) repeat
5) until
6) <условие>
7) Do

3. ДЛЯ ОБРАЗОВАНИЯ ОПЕРАТОРА ЦИКЛА С ПОСТУСЛОВИЕМ ИСПОЛЬЗУЮТСЯ КОНСТРУКЦИИ…

1) for
2) to
3) while
4) repeat
5) until
6) <условие>
7) do

Практическая часть

Задание 1. Вывод таблицы символов и их кодов

Постановка задачи

Напечатать таблицу соответствия символов и их кодов.

Методические указания

1. Для организации цикла использовать цикл For-to и стандартную функцию перевода кода в символ (тема 1 по лекциям). Вывод организовать в виде таблицы из двух столбцов (символ и его код) примерно посередине экрана.

2. Скопировать проект и изменить в проекте цикл с параметром на цикл с предусловием `While`. Вывод организовать в виде таблицы из двух столбцов (символ и его код) примерно посередине экрана.

3. Скопировать проект и изменить цикл с параметром на цикл с постусловием `Repeat`. Вывод организовать в виде таблицы из двух столбцов (символ и его код) примерно посередине экрана.

ТЕМА 5. СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ: ОДНОМЕРНЫЕ МАССИВЫ

Теоретическая часть

Одномерный массив в математике называется «последовательностью чисел» или «вектором». В Object Pascal тип массив используют для объединения пронумерованных элементов одного типа. Способ нумерации и тип элементов массива задается при его описании.

Пример (1 вариант описания):

```
Type TMass = array [1..20] of integer;
     Typearr= array [byte] of real;
     MyArrayOfChar: array [1..10] of char;
Var mass: TMass;
    arr: typearr;
    array_char: MyArrayOfChar;
```

Пример (2 вариант описания):

```
Var mass: array [1..20] of integer;
Var arr: array [byte] of real;
Var array_char: array [1..10] of char;
```

То есть в Object Pascal переменная типа массив – это ряд пронумерованных ячеек памяти, поэтому обращение к элементу массива производится по имени массива и по порядковому номеру элемента в массиве.

Пример

`Arr[2]` – 2-й элемент в массиве `arr`

`Array_char[i]` – `i`-й элемент в массиве `Array_char`

В программе все действия с массивами осуществляются поэлементно: ввод элементов с клавиатуры, задание значений в программе, вывод значений элементов на экран – для всех этих действий нужно перебрать все значения массива. Поэтому в программах для организации для этих действий используют цикл.

Так как мы всегда знаем количество элементов в массиве (задаем при описании) и нумеруем элементы дискретно, то для обработки элементов массива используется цикл с параметром, причем, в качестве параметра цикла берут индекс элементов массива.

Пример (задание элементов массива вещественного типа в интервале (0,1), используя датчик случайных чисел и вывод их на экран)

```
Program Project1;
{$APPTYPE CONSOLE}
Const n=30;
Var i: integer;
    Mas: array [1..n] of real;
Begin
  For i:=1 to n do
    Mas[i]:=random;
```

```
  For i:=1 to n do
    Write(mas[i]:5:2);
  Writeln;
End.
```

Примеры программ с использованием массива

Пример 1

Дан массив из 10 целых чисел со значениями, заданными в интервале от 0 до 100. Определить первый максимальный элемент массива и сравнить его по величине со значением, введенным с клавиатуры. Вывести сообщение о результате анализа.

Порядок выполнения задания

1. В программе используются обозначения:

`a` – массив;

`x` – значение, считываемое с клавиатуры;

`max` – максимальный элемент;

`i` – вспомогательная переменная для обозначения номера элемента (индекс элемента в массиве).

2. Алгоритм нахождения максимального состоит в следующем: предполагается, что максимальным является первый элемент массива. Затем в цикле просматриваются все элементы массива по очереди, начиная со второго, при этом сравнивается каждый элемент с предполагаемым максимумом. Если окажется что предполагаемый максимум меньше, чем очередной элемент массива, то максимуму присваивается значение этого элемента.

3. Затем, по условию задачи, необходимо сравнить найденный максимум с переменной `x`, введенной с клавиатуры. Вывод результата анализа на экран можно осуществить с помощью условных операторов.

4. Блок-схема описанного алгоритма приведена на рис. 26.

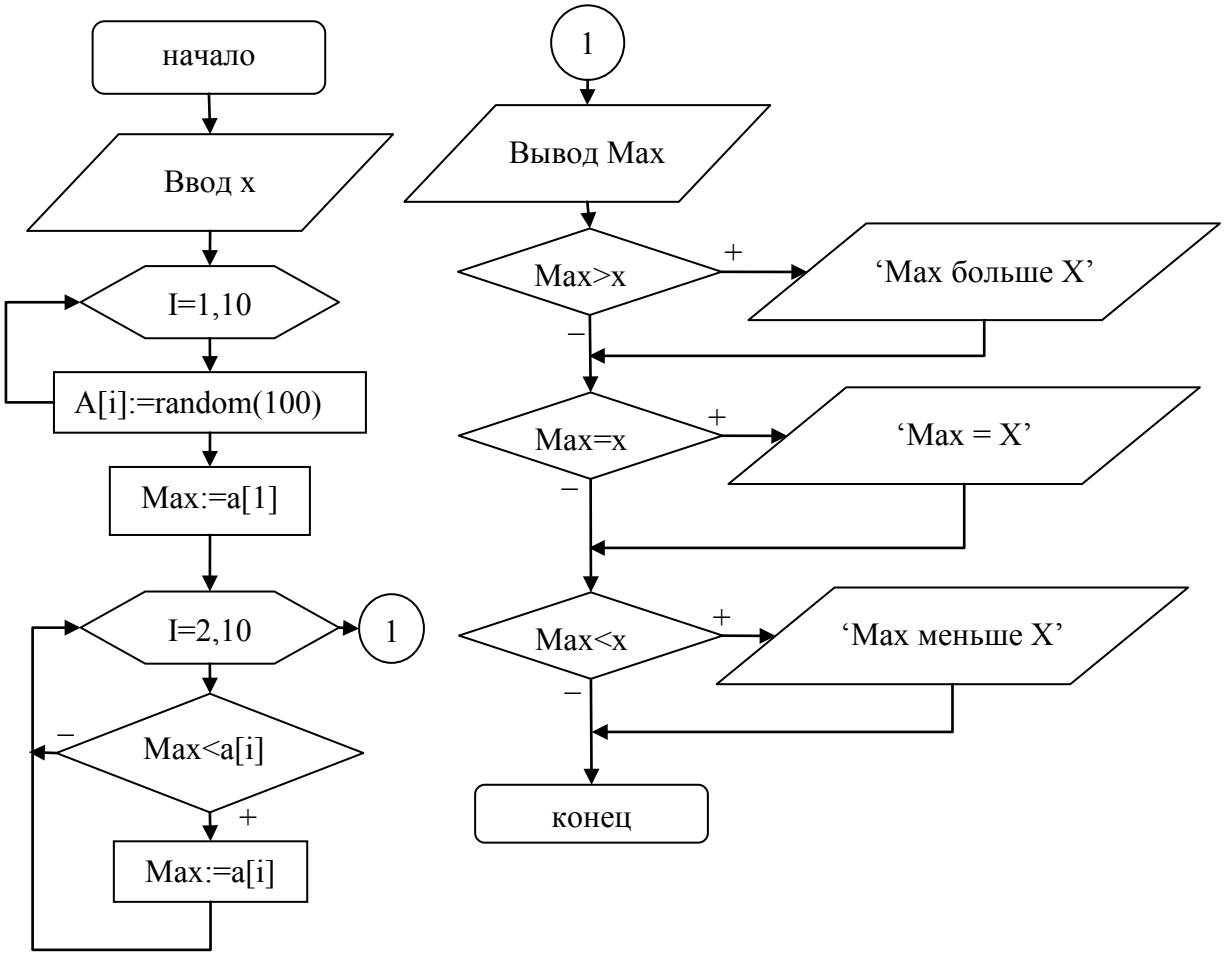


Рис. 26. Блок-схема алгоритма

5. Код программы:

```
. . .
Var a: array [1..10] of integer;
    Max, I, x: integer;
Begin
{Задаем массив датчиком случайных чисел}
 For i:=1 to 10 do a[i]:=random(100);
{выводим массив на дисплей}
 Writeln('Ischod massiv');
 For i:=1 to 10 do write(a[i],' '); writeln;
{считываем искомое значение с клавиатуры}
 Writeln('Vvod x'); Readln(x);
{Ищем максимум}
 Max:=a[1];
 For i:=2 to 10 do
  If max< a[i] then max:= a[i];
 Writeln('Max=',max);
{анализируем максимум и искомое значение}
 If max>x then writeln('Max>x');
 If max=x then writeln('Max=x');
 If max<x then writeln('Max<x');
 Readln
```

```
End.
```

Результат работы программы:

```
Ischod massiv
0 3 86 20 27 67 31 16 37 42
Vvod x
100
Max=86
Max<x
```

Пример 2

В массиве вещественных чисел, заданных случайным образом в интервале (-100, +100), подсчитать количество отрицательных элементов и сумму положительных.

Код программы:

```
.  . .
const n=20;
var mas:array[1..n] of real;
    i,count:integer;
    summa:real;
Begin
  writeln('Исходный массив');
 {Задание пол. и отр. элементов массива случайным образом}
 for i:=1 to n do mas[i]:=random*100-random*100;
 {Вывод массива на экран}
 for i:=1 to n do write(mas[i]:8:2);writeln;
 {Подсчет отрицательных элементов}
 count:=0;
 for i:=1 to n do
  if mas[i]<0 then inc(count);
 writeln('Колич-во отрицат. элементов=',count:10);
 {подсчет суммы положительных}
 for i:=1 to n do
  if mas[i]>0 then summa:=summa+mas[i];
 writeln('Сумма полож. элементов=',summa:10:2);
 readln;
end.
```

Результат работы программы:

```
    Исходный массив
        3.14  -65.85   39.87  -15.69    5.34   39.28   77.03   23.36  -54.94  -
44.67
       14.65  -41.13   16.69  -21.94  -54.66  -33.26  -58.70   20.37   39.54  -
80.70
    Колич-во отрицат. элементов=        10
    Сумма полож. элементов=    279.27
```

Контрольные вопросы

1. Какой из представленных наборов элементов можно назвать массивом, исходя из определения массива:

a) (‘q’, ‘w’, ‘e’, ’r’, ‘t’, ‘y’)

b) (2, 3, 4.5, 6, 3.6, 2)

c) (6, 7, 10, 2, 11, 4)

2. Как осуществляется обращение к элементам массива в программе?

Тестовые задания

Выберите один или несколько верных ответов.

1. ДЛЯ ОПИСАНИЯ МАССИВОВ ИСПОЛЬЗУЮТСЯ КОНСТРУКЦИИ…
    1) Array
    2) Massiv
    3) []
    4) {}
    5) ()
    6) Of
    7) For

2. ВЫБЕРИТЕ ВЕРНЫЕ ОПИСАНИЯ ПЕРЕМЕННОЙ A ТИПА МАССИВ ИЗ 20 ЦЕЛЫХ ЧИСЕЛ
    1) var A: integer = 20;
    2) var A: array of integer;
    3) var A: array[1..20] of integer;
    4) var A: array [integer] of integer = 20;
    5) var A: array[integer]

3. РЕЗУЛЬТАТ ВЫЧИСЛЕНИЯ ВЫРАЖЕНИЯ D[D[1]] – D[D[5]], ПРИ СЛЕДУЮЩИХ ЗНАЧЕНИЯХ МАССИВА D=(2, 4, 3, 1, 1) СОСТАВИТ…
    1) 1
    2) 2
    3) 3
    4) 4
    5) 5

Практические задания

Задание 1. Нахождение первого максимального значения в одномерном массиве

Постановка задачи

Задать массив из 53 целых чисел со случайными значениями из интервала (-10, +10). Найти первое максимальное значение среди элементов массива и сравнить его по величине с серединным элементом массива.

Методические указания

1. Данные задать, используя датчик случайных чисел (Random).

2. Оформить вывод, как исходного массива, так и полученных результатов с соответствующими заголовками.

3. Получение последнего или первого искомого элемента зависит от использования нестрогого или строгого сравнения.

Задание 2. Нахождение последнего минимального значения в одномерном массиве

Постановка задачи

Задать массив из 53 целых чисел со случайными значениями из интервала (-10, +10). Найти последнее минимальное значение среди элементов массива и сравнить его по величине с серединным элементом массива.

Методические указания

1. Данные задать, используя датчик случайных чисел (Random).

2. Оформить вывод, как исходного массива, так и полученных результатов с соответствующими заголовками.

3. Получение последнего или первого искомого элемента зависит от использования нестрогого или строгого сравнения.

ТЕМА 6. СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ: ДВУМЕРНЫЕ МАССИВЫ

Теоретическая часть

Количество размерностей массивов в языке Object Pascal – не ограничено. В практических расчетных задачах чаще всего используют двумерный массив, представленный в виде таблицы значений, который в математике называют **матрица**. В языке Object Pascal при описании двумерных массивов вначале указывают количество строк, затем количество столбцов матрицы.

Пример

```
//матрица размерности 20 x 10 из вещественных элементов
Var matr: array [1..20,1..10] of real;
//матрица размерности 3 x 4 из целых элементов
Var matr: array [1..3,1..4] of integer;
```

При обращении к элементу матрицы необходимо указать имя переменной, номер строки и номер столбца.

Пример

```
{присвоение значения элементу матрицы, расположенному в 5 строке и
2 столбце}
matr[5,2]:= 3.45 / 2;
```

Как и в случае с одномерными массивами, для обработки элементов матриц используются циклы. Так как нам заранее известна размерность матрицы, то при работе с ее элементами используют вложенные циклы **FOR**, один – внешний для изменения номера строки, второй – внутренний – для изменения номера столбца. Данный порядок соответствует расположению двумерного массива в памяти ЭВМ, поэтому и обработка его осуществляется быстрее именно в таком порядке.

Пример программы с использованием матриц

Задание

Задать случайным образом матрицу размера 5×10, элементы которой не превышают значения 10. Вывести ее на экран.

```
Program MyTestMatr;
{$APPTYPE CONSOLE}
Uses SysUtils, EsConsole in 'EsConsole.pas';
Const granI=5;
      granJ=10;
Type IndexI=1..granI;
     IndexJ= 1..granJ;
     Matr=array[indexI,indexJ] of Integer;
Var x:Matr;
    i:indexI;
    j:indexJ;
```

```
begin
  For i:=1 to granI do
   For j:=1 to granJ do x[i,j]:=Random(10);
  Writeln ('Исходный массив ');
  For i:=1 to granI do
    begin
      For j:=1 to granJ do
       write(x[i,j]:5);
      writeln
     end; {for-i}
end. {MyTestMatr}
```

Результат работы программы:

```
    Исходный массив
        0    0    8    2    2    6    3    1    3    4
        0    4    0    8    0    2    9    3    7    3
        6    8    7    3    1    3    4    2    8    2
        4    1    8    2    7    9    4    8    8    0
        1    1    5    0    5    0    7    6    7    7
```

Контрольные вопросы

1. Как обращаться к элементу матрицы в программе?
2. Какие операторы цикла можно использовать для действий с матрицей в программе?
3. Почему при кодировании алгоритма вначале располагают цикл изменения строк матрицы, а затем – столбцов?

Тестовые задания

Выберите один или несколько верных ответов.

1. ДЛЯ ОПИСАНИЯ ДВУМЕРНЫХ МАССИВОВ ИСПОЛЬЗУЮТСЯ КОНСТРУКЦИИ…

1) Array
2) Massiv
3) []
4) {}
5) ()
6) Of
7) For

2. ВЕРНЫМИ ОПИСАНИЯМИ ПЕРЕМЕННОЙ A ТИПА ЦЕЛОЧИСЛЕННЫЙ ДВУМЕРНЫЙ МАССИВ ИЗ 5 СТРОК И 3 СТОЛБЦОВ ЯВЛЯЮТСЯ…

1) var A: integer = 5;
2) var A: array of integer;
3) var A: array[1..15] of integer;
4) var A: array [1..5,1..3] of integer;

5) var A: array[1..5,1..3]of real

3. ПРИ ОБРАЩЕНИИ К НЕКОТОРОМУ ЗНАЧЕНИЮ ПЕРЕМЕННОЙ ТИПА ДВУМЕРНЫЙ МАССИВ …

1) достаточно указать только индекс строки
2) достаточно указать только индекс столбца
3) вначале указывается индекс строки, затем столбца
4) вначале указывается индекс столбца, затем строки
5) можно вообще не указывать индексы, достаточно обратиться к массиву по имени

Практическая часть

Задание 1. Вывод матрицы и одной строки

Постановка задачи

Задать матрицу размером 10 х 10, содержащую вещественные значения в интервале (0, 1). Вывести матрицу в формате с 2 знаками после запятой. Отступить одну строку. Вывести отдельно первую строку матрицы в том же формате.

Методические указания

1. Матрицу задавать счетчиком случайных чисел.
2. Вывод оформить соответствующими заголовками

Задание 2. Вывод матрицы и одного столбца

Постановка задачи

Задать матрицу размером 10 х 10, содержащую вещественные значения в интервале (0, 1). Вывести матрицу в формате с 2 знаками после запятой. Правее матрицы вывести первый столбец матрицы в том же формате.

ТЕМА 7. СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ: СТРОКИ

Теоретическая часть

Строкой в Object Pascal называется набор символов, пронумерованных с единицы. Максимально возможная длина строки – 2 Гб. Для описания строк используется зарезервированное слово **STRING**.

Пример

```
Var s: string;       // описание строки максимальной длины
    H: string[40];   // описание строки из 40 символов
    X: string[10];   // описание строки из 10 символов
```

Длина строки – это ограничение на максимально возможное количество символов в строке. Она может содержать и меньшее количество. Текущую длину строки можно определить с помощью функции:

```
function Length(St: string) : integer;
```

Обращение к символу в строке – по имени строки и номеру символа. Строки похожи на массивы, но, в отличие от массивов, их можно считывать с клавиатуры и выводить на экран по имени, а не посимвольно, одним оператором ввода или вывода. Посимвольная обработка строк осуществляется в цикле, чаще всего – **FOR**.

Примеры программ с использованием строк

Пример 1: Определить последний символ в строке, заданной с клавиатуры.

```
program str1;
{$APPTYPE CONSOLE}
Uses SysUtils, EsConsole in 'EsConsole.pas';
var MyStr:string[80];{строка будет содержать не более 80 символов}
    n,S:integer;
    ch:char;
begin
 writeln('Введите строку');
 readln(MyStr);{Ввод строки с клавиатуры}
 n:=Length(MyStr);{Определение номера последнего символа в строке.
Функция Length возвращает количество символов в строке, фактиче-
ски, это номер последнего символа}
 ch:=MyStr[n];{присваиваем последний символ переменной ch}
 writeln('Последний символ=',ch);{Вывод символа на экран}
readln
end.
```

Результат работы программы:

```
    Введите строку
    jjfldjaljkljfdand
    Последний символ=d
```

Пример 2: Подсчитать количество слов в строке, заданной с клавиатуры.

Примечание: словом считается последовательность символов до пробела.

```
program str2;
{$APPTYPE CONSOLE}
Uses SysUtils, EsConsole in 'EsConsole.pas';
var MyStr:string[80];
    i,S:integer;
begin
 writeln('Введите строку');readln(MyStr);
 s:=0;
 for i:=1 to Length(MyStr) do
  if MyStr[i]=' ' then s:=s+1;
 writeln('Всего слов в строке=',s);
 readln
end.
```

Результат работы программы:

```
    Введите строку
    jjfld jaljkljf danda
    Всего слов в строке=2
```

Ошибка алгоритма в том, что слова отслеживаются по символу пробела, но это верно пока символы пробела ищутся внутри строки, а в конце строки (за последним словом) идет не пробел. Поэтому нужно внести следующие изменения в алгоритм:

```
program str3;
{$APPTYPE CONSOLE}
Uses SysUtils, EsConsole in 'EsConsole.pas';
var MyStr:string[80];
    i,S:integer;
    ch:char;
begin
 writeln('Введите строку');readln(MyStr);
 s:=0;
 for i:=1 to Length(MyStr) do
  if MyStr[i]=' ' then s:=s+1;
 {здесь разместить операторы, определяющие последний символ в
строке}
     if ch<>' ' then s:=s+1;
 writeln(' Всего слов в строке=',s);
 readln
end.
```

Результат работы программы:

```
    Введите строку
    jjfldjaljkljf danda jjkhk lkjlkh
    Всего слов в строке=4
```

Пример 3. С клавиатуры считывается строка символов. Вывести коды, соответствующие этим символам.

```
program s3;
{$APPTYPE CONSOLE}
```

```
Uses SysUtils, EsConsole in 'EsConsole.pas';
var s:string;
    i:integer;
begin
 writeln('Введите строку');readln(s);
 writeln('Коды символов этой строки');

 for i:=1 to length(s) do
  write(Ord(s[i]):4);
 writeln;
 readln
end.
```

Результат работы программы:

```
    Введите строку
    Red blue green
    Коды символов этой строки
    114 101 100 32 98 108 117 101 32 103 114 101 101 110
```

Контрольные вопросы

1. Что такое строка?
2. Чем строки отличаются от массивов?

Тестовые задания

Выберите один или несколько верных ответов.

1. ДЛЯ ОПИСАНИЯ СТРОК ИСПОЛЬЗУЮТСЯ КОНСТРУКЦИИ…
    1) String
    2) []
    3) Array
    4) ()
    5) {}
    6) For

2. ВЕРНЫЕ ОПИСАНИЯ СТРОК – …
    1) Var t: string(30);
    2) Var d: array[1..30] of char;
    3) Var f: char=30;
    4) Var m: string;
    5) Var w: string[40] of char;
    6) Var q: string[80]

3. ВЕРНЫЕ ИСПОЛЬЗОВАНИЯ СТРОКИ A, СОСТОЯЩЕЙ ИЗ 20 СИМВОЛОВ (I – ПЕРЕМЕННАЯ ЦЕЛОГО ТИПА) – ЭТО …
    1) A[i] :=’r’;
    2) A[i]:=3;
    3) A[i]:=’3’;
    4) A[i]:=a[i]+a[i+1];
    5) A[21]:=’f’;

6) A[i]:=#123;

Практическая часть

Задание 1. Замена символов в строке (контрольное)

Постановка задачи

С клавиатуры вводится строка символов S и один символ C. Заменить в строке третий символ на символ, введенный с клавиатуры. Если в строке третий символ является пробелом, то вывести об этом сообщение.

Методические указания

1. Длину строки задать из 80 символов.
2. Вывод оформить поясняющими заголовками .

Задание 2. Работа с несколькими строками

Постановка задачи

С клавиатуры считываются две строки. Определить, на какой символ начинается первая строка, и на какой символ заканчивается вторая строка.

Методические указания

1. Строки считываются с клавиатуры, длину строк задать из 80 символов.
2. При решении использовать стандартные процедуры и функции, предусмотренные для работы со строками (материал лекции).
3. Вывод оформить поясняющими заголовками.

Задание 3. Кодирование строки

Постановка задачи

Закодировать строку, введенную с клавиатуры, используя «код Цезаря». Каждый символ заменить на символ, отстоящий от первого на заданное число.

пример: Задаем число: 4 Строка: абвгде Заменится на: гдежзи	**пример**: строка mama moet ramu при сдвиге на 6 заменится на строку sgsg&sukz&xgs{

Методические указания

1. Строка считывается с клавиатуры, длину строки задать из 80 символов.
2. Величина сдвига также считывается с клавиатуры.
3. При решении использовать примеры, приведенные выше.
4. Вывод оформить соответствующими заголовками.

ТЕМА 8. СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ: МНОЖЕСТВА

Теоретическая часть

Множество – это неупорядоченный набор однотипных неповторяющихся данных. Поэтому при работе с множествами необходимо постоянно перебирать все множество элементов, которые могут содержаться в данном множестве. Например, чтобы определить содержится ли элемент во множестве, нужно в цикле перебрать все возможные элементы множества с помощью стандартной функции **`in`**. Для вывода всего множества в цикле проверяют вхождение элемента во множество.

Для множеств введено понятие **мощность** – это количество элементов, содержащихся в данный момент времени во множестве.

Для множеств в языке Object Pascal введены следующие операции над множествами в целом:

1. **Объединение множеств** обозначается символом «+». В результате объединения двух множеств получается новое множество, содержащее элементы, принадлежащие или первому, или второму множеству:

Пример:

Даны два множества [‘a’,’f’,’g’] и [‘f’,’b’,’t’]. Объединением этих множеств будет новое множество [‘a’,’f’,’g’,’b’,’t’].

2. **Пересечение множеств** обозначается символом «*». В результате пересечения двух множеств получается новое множество, содержащее только те элементы, которые принадлежат и первому и второму множеству одновременно.

Пример:

Даны два множества [‘a’,’f’,’g’] и [‘f’,’b’,’t’]. Пересечением этих множеств будет новое множество, содержащее всего один элемент [’f’].

3. **Разность двух множеств** обозначается символом «-». Разностью двух множеств называется новое множество, содержащее элементы, принадлежащие первому, но не принадлежащие второму множеству.

Пример:

Даны два множества [‘a’,’f’,’g’] и [‘f’,’b’,’t’]. Разностью этих множеств будет новое множество [‘a’,’g’].

4. Проверка вхождения одного множества в другое обозначается двойным символом: «<=», слева от которого ставится множество, вхождение которого проверяется, справа множество – вхождение в которое проверяется. Результат операции – логический. Множество считается полностью содержащимся во втором множестве, если второе содержит все элементы первого.

Пример:

Даны два множества [‘a’,’f’,’g’] и [‘f’,’b’,’t’]. Результат проверки вхождения первого множества во второе будет FALSE, так как в первом множестве есть элементы, отличные от элементов второго.

Пример программы с использованием множества

Задание: Даны два множества X1, X2, содержащие целые числа из диапазона 0 ... 100 и множество X3, содержащее целые числа из диапазона 5…120. Известно, что мощность каждого множества равна 10. Следует сформировать новое множество Y = (X1∩ X2) ∪(X1\X2) и вывести его на печать, проверить выполнение условия: X3⊆Y и сформировать еще одно множество Y2, в которое поместить только четные элементы множества Y.

```
Program MyTest6;
{$APPTYPE CONSOLE}
Uses SysUtils, EsConsole in 'EsConsole.pas';
Const iMax1=100; m1=10; iMax2=120;
Type Tint1=0..iMax1; Tint2=5..iMax2 ;
Var X1,X2:set of Tint1;
    X3: set of Tint2;
    Y,Y2: set of byte;
    i:byte;
    m1,m2,m3:byte;
begin
 {--В начале выполнения программы множества пустые--}
 X1:=[];X2:=[];X3:=[];
 {--их мощность =0 ---}
 m1:=0;m2:=0;m3:=0;
 {--Задание исходных множеств с помощью датчика случайных чисел--}
 while m1<m do begin
  i:=random(100);
  if not(I in X1) then begin X1:=X1+[i];m1:=m1+1; end;
 end; {while}
 while m2<m do begin
  i:=random(100);
  if not(I in X2) then begin X2:=X2+[i];m2:=m2+1; end;
 end; {while}
 while m3<m do begin
  i:=random(120-5)+5;
  if not(I in X3) then begin X3:=X3+[i];m3:=m3+1; end;
 end; {while}
 {--Распечатка исходных множеств, используя операцию проверки принадлежности--}
 WriteLn('Исходные множества :');
 for i:=low(byte) to High(byte) do
   if i in X1 then Write(i:4);
 WriteLn;
 for i:=low(byte) to High(byte) do
   if i in X2 then Write(i:4);
 WriteLn;
```

```
 for i:=low(byte) to High(byte) do
   if i in X3 then Write(i:4);
 WriteLn;
 {--Формирование нового множества--}
 Y:=(X1*X2)+(X1-X2);
 WriteLn ('Полученное множество:');
 for i:=low(byte) to High(byte) do
if i in Y then Write(i:4);
 WriteLn;
 {--формирование множества четных чисел--}
 for i:=low(byte) to High(byte) do
if (i in Y) and ((I mod 2)=0) then Y2:=Y2+[i];
 WriteLn ('Выделяем четные числа из множества:');
 for i:=low(byte) to High(byte) do
   if i in Y2 then Write(i:4);
 {--Проверка вхождения множества X3 во множество Y--}
 if X3<=Y then WriteLn ('X3 принадлежит Y')
       else WriteLn ('X3 не принадлежит Y');
end.
```

Контрольные вопросы

1. Дайте определение множеству.
2. Какие из приведенных множеств являются эквивалентными
   a) {4, 5, 4, 3, 6, 9}
   b) {4, 4, 5, 3}
   c) {4, 5, 3, 6, 9}
   d) {5, 3, 4}

Тестовые задания

Выберите один или несколько верных ответов.

1. КОНСТРУКЦИИ, ИСПОЛЬЗУЕМЫЕ ПРИ ОПИСАНИИ МНОЖЕСТВ…
   1) SET
   2) OF
   3) []
   4) {}
   5) ()
   6) FOR
2. МОЩНОСТЬ МНОЖЕСТВА, ПОЛУЧЕННОГО В РЕЗУЛЬТАТЕ ВЫПОЛНЕНИЯ ОПЕРАЦИИ

   [1, 2, 3] + [2, 3, 4]

   РАВНА
   1) 1
   2) 2
   3) 3
   4) 4
   5) 5

6) 6

3. МОЩНОСТЬ МНОЖЕСТВА, ПОЛУЧЕННОГО В РЕЗУЛЬТАТЕ ВЫПОЛНЕНИЯ ОПЕРАЦИИ

[1, 2, 3] - [2, 3, 4]

РАВНА

1) 1
2) 2
3) 3
4) 4
5) 5
6) 6

4. МОЩНОСТЬ МНОЖЕСТВА, ПОЛУЧЕННОГО В РЕЗУЛЬТАТЕ ВЫПОЛНЕНИЯ ОПЕРАЦИИ

[1, 2, 3] * [2, 3, 4]

РАВНА

1) 1
2) 2
3) 3
4) 4
5) 5
6) 6

Практическая часть

Задание 1. Обработка числовых множеств

Постановка задачи

Задать случайным образом два множества A и B. Множество A содержит 10 элементов из интервала (20..60), множество B содержит 20 четных элементов из интервала (0..100). Определить новые множества: A+B, A–B, B–A, A*B.

Методические указания

При работе использовать приведенный выше пример.

Задание 2: Работа с символьными множествами

Постановка задачи

Составить программу ввода значений символьного множества с клавиатуры. Определить мощность множества. Вывести все прописные символы, встречающиеся только один раз.

Методические указания

1. Программа должна правильно работать для произвольного набора символов латинского алфавита.

2. Входная строка символов может быть длиннее строки экрана дисплея.

3. Латинскими гласными будем считать буквы, составляющие множество: (a, e, o, u, i), согласными – все остальные.

4. Строчные – это буквы нижнего регистра (маленькие), прописные – верхнего (большие).

ТЕМА 9. ТЕКСТОВЫЕ ФАЙЛЫ

Теоретическая часть

Текстовые файлы – это файлы, состоящие из строк различной длины. Для обозначения текстовых файлов используют зарезервированное слово **`TEXTFILE`**.

Для создания текстовых файлов можно использовать любой редактор текста, например, стандартную программу Windows – *Блокнот*. Также файлы можно создавать программно.

При работе с файлами **всегда** вначале связывают файловую переменную с местом хранения на диске. Для этого применяется процедура:

```
AssignFile (<файловая переменная>,<имя файла>) ;
```

В качестве `<имя файла>` указывают либо полное имя файла, включая путь к нему, либо, если файл хранится в папке проекта, только имя с расширением.

Далее файл **обязательно** нужно открыть для чтения из него данных, либо для записи в новый файл, либо для дополнения информации в существующий файл (только в конец). Схема работы с текстовыми файлами приведена на рис. 27.

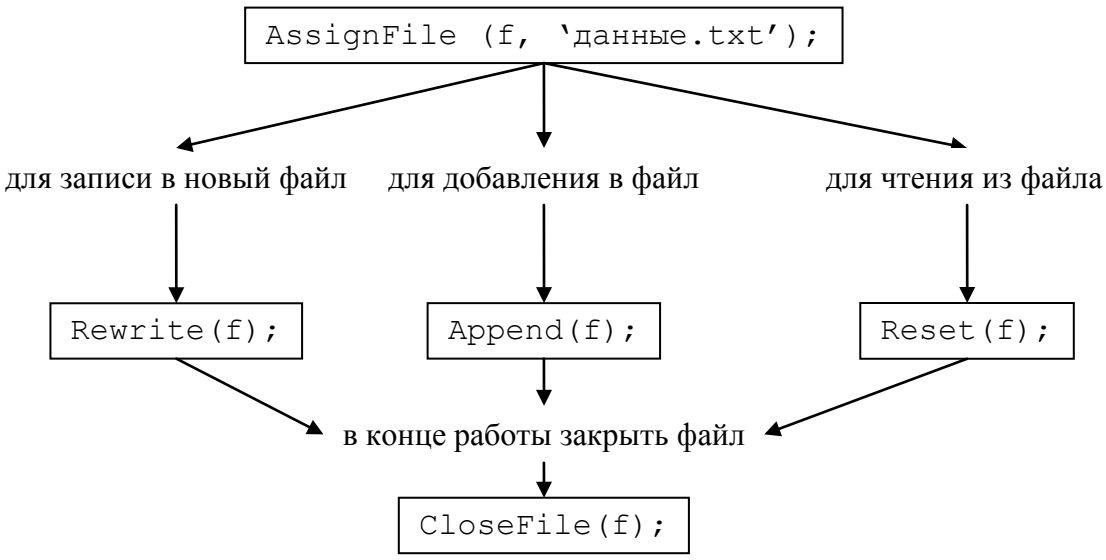


Рис. 27. Схема работы с текстовыми файлами

Для чтения данных из текстового файла применяют процедуры `READ` и `READLN`. Разница между ними состоит в том, что при использовании первой процедуры считываются значения, указанные в его списке, и курсор остается на этой же строке файла. При использовании второй процедуры считывается список значений и курсор в файле переходит на новую строку.

Для записи данных в текстовой файл – процедуры `WRITE` и `WRITELN`.

При завершении работы с файлами их необходимо закрыть (рис. 27). Особенно это важно при работе с новыми файлами, так как запись в файл происходит медленнее, чем обработка кода программы, поэтому есть возможность прерывания записи в файл.

В текстовых файлах можно хранить разнообразные данные: числа, символы. Поэтому запись в файл и чтение из файла осуществляется переменной необходимого типа.

Так как файл может содержать разное количество строк, то, как правило, при работе с файлами применяют операторы цикла с пост- или предусловием. При этом для определения конца файла используется стандартная функция **EOF,** для конца строки используется стандартная логическая функция **EOLN**. Функции булевского типа возвращают значение **ИСТИНА (TRUE)**, когда при работе с файлом достигнут конец файла или конец строки.

Примеры работы с текстовыми файлами

Пример 1. Дан текстовый файл с именем **str.txt**. В него занесено несколько строк. Программа считывает строки из файла и распечатывает их на экран.

```
Program MyTest10_1;
{$APPTYPE CONSOLE}
Uses SysUtils, EsConsole in 'EsConsole.pas';
Var F: TextFile ;
    Str1: String[80] ;
begin
 AssignFile (f,'str.txt') ;
 Reset (f);
{---Распечатка строк из файла на экран---}
  while not eof (f) do
   begin
     readln(f,str) ;
     writeln(str);
   end;
 CloseFile(f);
end.
```

Пример 2. Дан текстовый файл с именем input.txt, имеющий следующие данные. В первой строке указана размерность массива. Во второй строке содержится число k. Начиная с третьей строки, заданы целочисленные элементы одномерного массива. Вывести массив из файла на экран. Вычислить сумму элементов массива от первого до элемента с индексом k: $\sum_{i=1}^{k} a_i$ и среднее арифметическое значение оставшихся. Создать новый текстовый файл output.txt, в который с соответствующими заголовками поместить исходный массив и результат работы. Перед каждым заголовком вставить по две пустой строке.

Пример исходного и полученного файлов приведен на рис. 28.

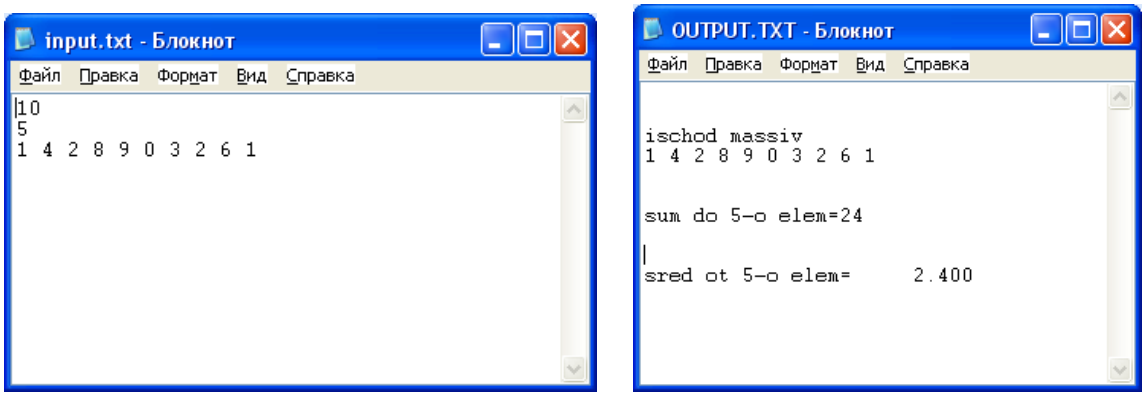


Рис. 28. Пример входного и выходного файла задачи

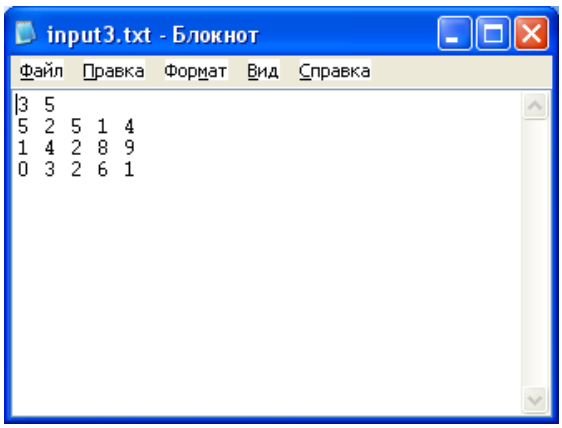
Листинг программы:

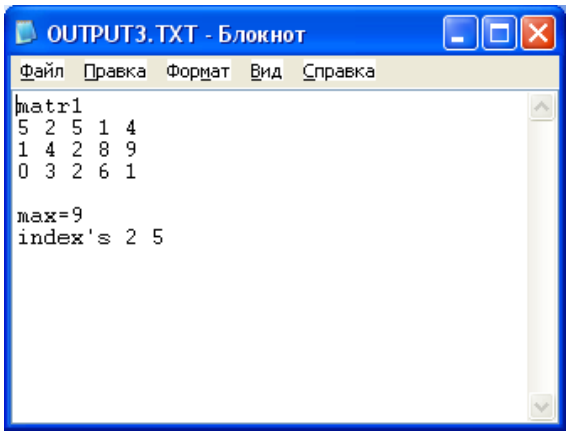
```
program ex2;
{$APPTYPE CONSOLE}
Uses SysUtils, EsConsole in 'EsConsole.pas';
var f1,f2: textFile;
    n,k,i,sumk: integer;
    sred:real;
    a: array[1..100] of integer;
begin
 assignFile(f1,'input.txt');
 reset(f1);{открываем файл для чтения}
 {считываем размер массива и номер элемента}
 readln(f1,n);readln(f1,k);
 for i:=1 to n do read(f1,a[i]); {считываем элементы массива}
 closeFile(f1);{закрываем исходный файл, он больше не нужен}
 assignFile(f2,'output.txt');
 rewrite(f2); {открываем новый файл для записи}
 {выводим в файл две пустые строки и заголовок}
 writeln(f2);writeln(f2);writeln(f2,'ischod massiv');
 {заносим в файл элементы массива}
 for i:=1 to n do write(f2,a[i],' ');writeln(f2);
 {рассчитываем сумму до k-го элемента}
 sumk:=0;
 for i:=1 to k do sumk:=sumk+a[i];
 {выводим сумму в файл, вставив перед ней две пустые строки}
 writeln(f2);writeln(f2);writeln(f2,'sum do ',k,'-o elem=',sumk);
 {рассчитываем среднее от k-го элемента}
 sred:=0;
 for i:=k+1 to n do sred:=sred+a[i];
 sred:=sred/(n-k);
 {выводим среднее в файл, вставив перед две пустые строки}
 writeln(f2);writeln(f2);
 writeln(f2,'sred ot ',k,'-o elem=',sred:10:3);
 closeFile(f2);
end.
```

Пример 3. Дан текстовый файл с именем **dat.dat**, в который занесен двумерный массив целых чисел. Найти среди них максимум. Размер матрицы задан в первой строке файла, элементы, начиная со второй строки.

Создать новый текстовый файл, в который вывести исходную матрицу, отступив одну строку, вывести максимум и его индексы.

Пример исходного и полученного файлов:





Листинг программы:

```
program ex_file;
{$APPTYPE CONSOLE}
Uses SysUtils, EsConsole in 'EsConsole.pas';
var f:textFile;
    n,m,i,j,max,iMax,jMax: integer;
    matr: array[1..100,1..100] of integer;
begin
 assignFile(f,'input3.txt');reset(f);
 readln(f,n,m);
 for i:=1 to n do for j:=1 to m do read(f,matr[i,j]);closeFile(f);
 assignFile(f,'output3.txt');rewrite(f);
 writeln(f,'matr1');
 for i:=1 to n do begin
  for j:=1 to m do write(f,matr[i,j],' ');
  writeln(f);
 end;
 max:=matr[1,1]; iMax:=i;jMax:=j;
 for i:=1 to n do for j:=1 to m do
  if max<matr[i,j] then begin max:=matr[i,j];iMax:=i;jMax:=j end;
 writeln(f);writeln(f,'max=',max);
 writeln(f,'index''s ',iMax,' ',jMax);
 closeFile(f);
end.
```

Контрольные вопросы

1. Чем отличается текстовый файл от типизированного файла?
2. Какие операторы служат для чтения информации из текстового файла?
3. Какие операторы служат для записи информации в текстовый файл?

Тестовые задания

Выберите один или несколько верных ответов.

1. КОНСТРУКЦИИ, КОТОРЫЕ ИСПОЛЬЗУЮТСЯ ПРИ ОПИСАНИИ ТЕКСТОВЫХ ФАЙЛОВ:

1) File
2) Filetext
3) Textfile
4) Of
5) <тип элементов>
6) string

2. ВЕРНОЕ ОПИСАНИЕ ТЕКСТОВОГО ФАЙЛА – ЭТО…

1) Var t: textfile;
2) Var t: file of string;
3) Var t: filetext;
4) Var t: text of file;

3. ПРИ ЧТЕНИИ ИНФОРМАЦИИ ИЗ ТЕКСТОВОГО ФАЙЛА ЖЕЛАТЕЛЬНО ИСПОЛЬЗОВАТЬ ОПЕРАТОРЫ ЦИКЛА –

1) For-to-do
2) For-downto-do
3) While-do
4) Repeat-until

Практическая часть

Задание 1. Работа с текстовым файлом, содержащем строки

Постановка задачи

Задать текстовый файл **`input1.txt`**, содержащий несколько строк. Определить, сколько строк содержится в этом файле, сколько символов «v» содержится во второй строке. Результаты заносятся в отдельный текстовый файл **`output1.txt`** по следующему формату:

Исходный текст
<все исходные строки>
<пропустить 1 строку>
Кол-во строк=<количество строк>
<пропустить 2 строки>
Символов v=<количество символов v>

Методические указания

Исходный файл данных создать в редакторе *Блокнот*.

Задание 2. Обработка текстового файла, содержащего одномерный массив

Постановка задачи

Определить, сколько всего элементов содержится в текстовом файле **`input3.txt`**. Результат вывести в новый файл **`output3.txt`**.

Методические указания

1. Исходные данные задать как текстовый файл в экранном редакторе *Блокнот*. Данные целого типа находятся в первой строке файла через пробел.

2. Файл отчета должен содержать исходный массив с заголовком и результат решения:

massiv:
<элементы исходного массива в столбик>
<пустая строка>
<5 пробелов>itog=<количество элементов>

Задание 3. Обработка текстового файла, содержащего матрицу

Постановка задачи

1. Написать программу, которая создает текстовый файл, содержащий матрицу размера 10×10. Первая строка файла содержит размерность. Элементы матрицы – целые числа в интервале $(-50,+50)$, располагаются по строкам, начиная со второй строки файла.

2. Написать программу, которая определяет минимум первой строки матрицы и максимум последней строки.

Методические указания

1. Первая программа создает матрицу 10×10, элементы которой задаются случайным образом. В текстовый файл заносится размерность матрицы, затем элементы найденной матрицы (см. пример 3).

2. Вторая программа считывает данные из файла, используя переменную типа двумерный массив с элементами целого типа. Результат выводится на экран.

ТЕМА 10. ПОЛЬЗОВАТЕЛЬСКИЕ ПРОЦЕДУРЫ И ФУНКЦИИ

Теоретическая часть

Механизм подпрограмм используют в тех случаях, когда необходимо выполнять часть кода программы неоднократно. Например, при работе с несколькими массивами постоянно требуется вывод элементов массива на экран. В Object Pascal различают два вида подпрограмм: процедуры и функции. Различаются они своим функциональным назначением. Функции используют в случаях получения (или, говорят, возврата) конкретного значения. Процедуры используют в случаях, когда предполагается возврат не конкретного результата, а некоторого набора действий. Мы уже применяли механизм подпрограмм в своих программах, например, функция возврата значения синуса – `SIN`, или процедура связывания физического файла и переменной типа файл – `ASSIGNFILE`. Рассмотрим правила написания пользовательских подпрограмм.

Итак, программа, содержащая вызов подпрограммы, называется **основной**. **Процедура (функция)** – это подпрограмма, которая выполняет какие-то действия, и которую можно вызвать из любого места программы. После выполнения процедуры (функции) выполнение программы продолжается с того места, откуда она была вызвана.

Объявление (описание) процедуры (функции) помещается в блоке описаний до **`begin`**-а основной программы.

Синтаксис **объявления процедуры** такой:

```
procedure <ИмяПроцедуры>(<список формальных параметров>);
<блок описания локальных данных, действующих в пределах процедуры>
begin
<тело процедуры – блоки, описывающие действия процедуры>
end;
```

Синтаксис **объявления функции** такой:

```
function <ИмяФункции>(<список формальных параметров>) : Тип;
<блок описания локальных данных, действующих в пределах функции>
begin
{ инструкции функции }
Result:= результат вычислений;
end;
```

Здесь следует обратить внимание на два момента: после имени функции и параметров в круглых скобках, после двоеточия, указывается тип возвращаемого значения. Кроме того, в каждой функции по умолчанию имеется переменная *Result*, которая имеет тот же тип, что и тип возвращаемого значения. Эту переменную специально объявлять не нужно, она уже готова к работе. В отличие от других языков, в Delphi этой переменной можно присваивать значение неоднократно. Результатом будет последнее присвоенное значение.

Есть еще один способ вернуть из функции результат вычислений: использовать переменную с таким же именем, как и имя функции. Эту переменную тоже объявлять не нужно. В нашем примере, строка

```
Result := результат вычислений;
```

будет полностью идентичной строке

```
<ИмяФункции> := результат вычислений;
```

Какой из способов использовать – решайте сами, оба способа правильны.

<ИмяПроцедурыФункции> – это идентификатор. Имя процедуры или функции должно быть уникальным в пределах программы.

Result – это зарезервированная переменная внутри функции, с помощью которой функция возвращает искомое значение. Переменную объявлять не надо!

Параметры – это входные данные для подпрограмм. В принципе, список формальных параметров может и отсутствовать.

Различают несколько типов формальных параметров (см. лекцию «Процедуры и функции»). Имеются три вида формальных параметров: параметры-значения, параметры-переменные, параметры-константы.

1) формальный параметр-значение

Синтаксис описания параметра-значения:

```
Имя_параметра: Тип;
```

2) формальный параметр-переменная

Синтаксис описания параметра-значения:

```
var Имя_параметра: Тип;
```

3) формальный параметр-константа

```
const Имя_параметра: Тип;
```

При вызове подпрограммы передача данных для этих видов осуществляется по-разному. **Параметры-значения** копируются, и подпрограмма работает с их копией, что требует дополнительных затрат памяти. Поэтому рекомендуется использовать параметры-константы или параметры-переменные. При использовании **параметров-переменных** и **параметров-констант** в подпрограмму передаются адреса (указатели фактических параметров) и она работает непосредственно с фактическими параметрами. Благодаря этому экономится память, а также организуется передача результата работы подпрограммы вызывающей программе через параметры-переменные. Через параметры-константы этого делать нельзя, т.к. их нельзя менять внутри подпрограммы.

Вызов процедуры осуществляется в теле основной программы по ее имени и списку фактических параметров без указания их типа. Между фактическими и формальными параметрами должно быть соответствие по типу, порядку следования и по количеству. Если список формальных параметров отсутствует, список фактических параметров также отсутствует и скобки за именем процедуры при ее вызове опускаются.

В качестве фактических параметров для параметров-переменных могут использоваться только переменные. Для формальных параметров – констант или параметров-значений в качестве фактических параметров могут использоваться как переменные, так и константы или даже арифметические выражения.

Пример процедуры с параметрами:

```
procedure Primer(a,b : Integer);
begin
  a := a * b;
end;
```

Обратите внимание, что в описании процедуры обязательно нужно указывать тип параметров. Теперь мы можем вызвать эту процедуру, указав ей, какие числа нужно перемножить. Примеры вызова процедуры:

```
Primer(10, 20); //передаем целые числа
Primer(a, 100); {передаем переменную a целого типа, и целое число}
Primer(c, d); //передаем две переменных целого типа
```

Сразу следует сказать об области видимости переменных. Бывают переменные *глобальные* и *локальные*. Глобальные переменные видны во всей программе, их мы будем использовать позже. А локальные переменные создаются внутри процедуры, в разделе **var**, и видны только в этой процедуре. Локальные переменные создаются в памяти в то время, когда процедура начинает работу, и уничтожаются, когда процедура закончила работу. Таким образом, в основной программе можно сделать две или более процедур, и указать в них переменные с одинаковым именем. Это будут разные переменные, и они не будут мешать друг другу.

Примеры использования подпрограмм

Пример1.

Вычислить значение переменной *f*:

$$f = \frac{y + a}{2} + 2a$$

где $a = \begin{cases} y / z \cdot x, при\ \text{x} \neq 0 \\ y / z, при\ \text{x} = 0 \end{cases}$

Примечание: для вычисления параметра *a* использовать механизм подпрограмм-функций.

Решение

1. В программе определены переменные:

f - результат наших вычислений, имеет вещественный тип, так как формула содержит операцию деления.

a - для вычисления параметра *a* предлагается использовать механизм функций, так как по определению функция – это подпрограмма, возвра-

щающая одно конкретное значение, и у нас параметр *a* имеет вполне конкретное значение, но определяющееся по условию. Определяем как функцию вещественного типа. В вычислениях внутри функции участвуют переменные *z* и *x* – для общности опишем и их как переменные вещественного типа. По определению – это локальные переменные.

y - переменная, от которой зависит значение *f* и *a*. Ее опишем как глобальную переменную.

z, x – эти переменные нужны только для вычисления функции *a*, поэтому опишем их как локальные внутри нашей функции. Следовательно и ввод их значений также нужно поместить внутри функции.

1 листинг программы:

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  EsConsole in 'EsConsole.pas';

var y,f:real;
function a:real;
var x,z: real;
begin
 repeat
  writeln('Введите значения x,z:');
  readln(x,z);
  if z=0 then writeln('Ошибка при вводе z (z<>0)!');
 until z<>0;
 if x<>0 then Result:=y/x else Result:=y/z
end;{a}

begin

end.
```

2. В основной программе следует предусмотреть ввод значения глобальной переменной *y* с клавиатуры.

3. При вычислении *a* примем во внимание, что возможно прерывание работы программы, если окажется, что $z = 0$, поэтому в примере организован цикл для ввода исходных значений: если вводится $z = 0$, то на экране появляется сообщение об ошибке и пользователя просят ввести исходные данные еще раз.

2 листинг программы:

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  EsConsole in 'EsConsole.pas';

var x,y,z:real;
    f:real;
```

```
function a:real;
var x,z: real;
begin
 repeat
  writeln('Введите значения x,z:');
  readln(x,z);
  if z=0 then writeln('Ошибка при вводе z (z<>0)!');
 until z<>0;
 if x<>0 then Result:=y/x else Result:=y/z
end;{a}

BEGIN
 writeln('Введите значение y:');
 readln(y);
 f:=(y+a)/2+2*a;
 writeln('          f=',f:10:3);
 readln;
END.
```

Результат работы программы:

```
Введите значение y:
1
Введите значения x,z:
2 0
Ошибка при вводе z (z<>0)!
Введите значения x,z:
2 3
Введите значения x,z:
2 3
          f=     1.750
```

Что здесь не учтено? Механизм работы функции – следующий: при встрече ее имени в основной программе происходит вызов тела функции, после отработки функции – возврат на действие, следующее за именем функции в основной программе. В нашем примере вызов функции встречается два раза при вычислении переменной *f*. Поэтому нам пришлось два раза вводить значения переменных *x, z*. Можно перенести ввод значений этих переменных в основную программу, следовательно их описание – так же в основную программу и передавать их как параметры-константы в функцию. Перепишем код программы с учетом этих замечаний:

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  EsConsole in 'EsConsole.pas';

var y:real;
    f:real;
    x,z: real;

function a(const x,y,z: real):real;
begin
 if x<>0 then Result:=y/x else Result:=y/z
end;{a}

BEGIN
```

```
 Writeln('Введите значение y');
 readln(y);
 repeat
  writeln('Введите значения x,z:');
  readln(x,z);
  if z=0 then writeln('Ошибка при вводе z (z<>0)!');
 until z<>0;
 f:=(y+a(x,y,z))/2+2*a(x,y,z);
 writeln('          f=',f:10:3);
 readln;
END.
```

Результат работы программы:

```
Введите значение y
1
Введите значения x,z:
2 0
Ошибка при вводе z (z<>0)!
Введите значения x,z:
2 3
          f=     1.750
```

Пример 2.

Задание. Даны три целочисленных множества, мощностью 10, 8, 7 элементов, находящихся в интервалах (3…100), (10…150), (10…210) соответственно. Определить новое множество по формуле: D=E\(A∪B∪C), где E – это множество всех возможных натуральных чисел.

Решение

Создание и вывод множества будут организованы в виде процедур. Множества передаются в процедуры как параметры-переменные (это единственная возможность получения множества из процедуры). Кроме этого, в процедуру создания множества передается количество элементов (мощность) как параметр-константу и границы интервала значений.

```
program Project;
{$APPTYPE CONSOLE}
uses
  SysUtils;

type setByte=set of byte;
var a,b,c,d,e:setbyte;

procedure ProcSet(const power,gran1,gran2:byte; var x:setByte);
{процедура задания элементов множества случайным образом}
var m,i:integer;
begin
  x:=[];
  m:=0;
 while m<power do begin
  i:=gran1+random(gran1-gran2);
  if not(I in X) then begin X:=X+[i];m:=m+1; end;
```

```
 end; {while}
end;{ProcSet}

procedure WriteSet(var x:setByte);
{процедура вывода на экран элементов множества}
var i:integer;
begin
 for i:=low(byte) to High(byte) do
   if i in X then Write(i:4);
 writeln;
end;{WriteSet}
{основная программа}
BEGIN
 ProcSet(10,3,100,a);
 writeln('1 set');
 WriteSet(a);
 ProcSet(8,10,150,b);
 writeln('2 set');
 WriteSet(b);
 ProcSet(7,10,210,c);
 writeln('3 set');
 WriteSet(c);
 ProcSet(High(byte),0,255,e);
 d:=e-(a+b+c);
 writeln('Result');
 WriteSet(d);
 readln;
end.
```

Результат работы программы:

```
1 set
   3   7  28  34  45  52  60  68 107 136
2 set
  16  17  18  40  48  59  98 106
3 set
  17  23  24  41  42  44  47
Result
   0   1   2   4   5   6   8   9  10  11  12  13  14  15  19  20  21  22  25  26
  27  29  30  31  32  33  35  36  37  38  39  43  46  49  50  51  53  54  55  56
  57  58  61  62  63  64  65  66  67  69  70  71  72  73  74  75  76  77  78  79
  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  99 100
 101 102 103 104 105 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
 123 124 125 126 127 128 129 130 131 132 133 134 135 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163
 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183
 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203
 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223
 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243
 244 245 246 247 248 249 250 251 252 253 254
```

Пример 3.

Программа, содержащая две процедуры: одна для задания значений массива, вторая – для вывода его на экран. Все массивы в основной программе имеют одинаковый тип элементов, но разную размерность, поэтому в подпрограммы массивы передаются как открытые массивы.

```
Program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
Const
    n=10;
    m=20;
    l=15;
Var i, j, k : integer ;
    A: Array[1..n] of Real;
    B: array[1..m]of real;
    C: array[1..l]of real;
    s: real;

 Procedure zadan(Var Matr: array of real);
 var i: integer;
 begin
  for i:=low(Matr) to High(Matr) do Matr[i]:=Random(100);
 end; {zadan}

 Procedure vivod(Var Matr: array of real);
 var i: integer;
 begin
  for i:=low(Matr) to High(Matr) do write(Matr[i]:8:2);
  writeln;
 end; {vivod}

Begin
 Zadan(a);writeln('Первый массив');vivod(a);
 Zadan(b);writeln('Второй массив');vivod(b);
 Zadan(c);writeln('Третий массив');vivod(c);
 Readln
end.{MyTest8}
```

Контрольные вопросы

1. Дайте определение понятию «процедура».
2. Чем процедуры отличаются от функций?
3. Что такое основная программа?
4. Какие типы параметров Вы знаете?
5. Чем параметры-константы отличаются от параметров-переменных?

Тестовые задания

Выберите один или несколько верных ответов.

1. ВЕРНЫЕ ОПИСАНИЯ ПРОЦЕДУР – …

```
1)  Procedure Proc1(var a: real): real;
2)  Function Func1(a: real): real;
3)  Procedure Proc2;
4)  Function Func;
```

```
5)  Procedure Proc3(a:real);
6)  Function Func3(var a: real);
```

2. ИЗ ПЕРЕЧИСЛЕННЫХ ФУНКЦИЙ ВСЕГДА БУДЕТ ВОЗВРАЩАТЬ НУЛЕВОЙ РЕЗУЛЬТАТ ФУНКЦИЯ (ФУНКЦИИ)…

```
1)  Function Func1(a: byte): byte;
    Begin
      Result:= a*10;
    End;
2)  Function Func1: byte;
    Var a: byte;
    Begin
      Result:=a*10;
    End;
3)  Function Func1(var a: byte): byte;
    Begin
      Result:=a*10;
    End;
```

3. ИМЕЕТСЯ СЛЕДУЮЩЕЕ ОПИСАНИЕ ПРОЦЕДУРЫ.

```
Procedure MyProc(var s: real; const a: integer; b: integer);
Var d: real;
begin
  d:=0;
  if a>b then s:=s*10 else s:=(a+b)/2;
end;
```

ВЕРНЫМИ ЯВЛЯЮТСЯ УТВЕРЖДЕНИЯ:

1) s – параметр-переменная,
   a – параметр-переменная,
   b – параметр-константа
2) s – параметр-переменная,
   a – параметр-константа,
   b – параметр-константа
3) s – параметр-значение,
   a – параметр-переменная,
   b – параметр-значение
4) s – параметр-значение,
   a – параметр- константа,
   b – параметр- переменная
5) s – параметр-переменная,
   a – параметр-константа,
   b – параметр-значение

Практическая часть

Задание 1. Использование функций

Постановка задачи

Вычислить значение переменной *f*:

$$f = \frac{x-a}{y+a} + 2(z-a) + \frac{2a}{8+z}$$

где $a = \begin{cases} \dfrac{x+y}{z}, при\ z \neq 0 \\ \dfrac{2x}{y}, при\ \mathrm{z} = 0 \end{cases}$

при следующих значениях:

1. x=1, y=2, z=0

2. x=1, y=0, z=0 и при *z=1*

3. x=1, y=1, z=1

Методические указания

1. Задание исходных величин осуществляется с клавиатуры.

2. Проверить входные данные на корректность. Если какое-то значение задано некорректно, организовать вывод сообщения об этом и запустить алгоритм снова (см. пример).

3. При программировании использовать механизм передачи исходных данных как параметров-констант.

Задание 2. Процедуры и функции с параметрами открытые массивы

Постановка задачи

Заданы три массива: A размерность 10 элементов; B размерность 15 элементов и C размерность 13 элементов. Значения элементов заданы случайным образом. Написать процедуру задания и вывода массивов на экран. Также написать функцию подсчета суммы элементов каждого массива в отдельности.

Методические указания

1. Задание и вывод на экран элементов массивов оформить в виде процедур.

2. При задании элементов массивов использовать датчик случайных чисел в диапазоне (-20,20).

3. При программировании использовать механизм передачи массивов как открытых параметров.

ПРИЛОЖЕНИЕ

ПРИМЕРЫ ТИПОВЫХ АЛГОРИТМОВ

Задание элементов одномерного массива

1. ввод с клавиатуры

```
. . .
const n=10;{размерность массива}
var a:array[1..n] of integer;{массив из 10 целых чисел}
    i:integer;{индекс элементов}
. . .
begin
. . .
 for i:=1 to N do read(a[i]);
. . .
```

2. с использованием датчика случайных чисел (для вещественных значений)

```
. . .
const n=10;{размерность массива}
var a:array[1..n] of real;{массив из 10 вещественных чисел}
    i:integer;{индекс элементов}
. . .
begin
. . .
 for i:=1 to N do a[i]:=random*100;
. . .
```

3. с использованием датчика случайных чисел в заданном диапазоне (для вещественных значений)

```
. . .
const n=10;{размерность массива}
      d1=-2; d2=5; {границы диапазона (-2,5)}
var a:array[1..n] of real;{массив из 10 вещественных чисел}
    i:integer;{индекс элементов}
. . .
begin
. . .
 for i:=1 to N do a[i]:=random*(-2)+random(5);
. . .
```

4. с использованием датчика случайных чисел (для целых значений)

```
. . .
const n=10;{размерность массива}
var a:array[1..n] of real;{массив из 10 вещественных чисел}
    i:integer;{индекс элементов}
```

```
. . .
begin
. . .
 for i:=1 to N do a[i]:=random(100);
. . .
```

5. из типизированного файла

```
. . .
var a:array[1..100] of integer;{размерность берем максимум}
    i,n:integer;{n – размерность массива}
    f:file of integer;{типизированный файл}
. . .
begin
. . .
 Assign(f,'1.my');reset(f);{файл должен уже существовать}
 i:=0;
 while not eof(f) do begin
  inc(i);read(f,a[i]);
 end;
 n:=i;
. . .
```

6. из текстового файла

```
. . .
{числа расположены в файле в строку через пробел}
var a:array[1..100] of integer;{размерность берем максимум}
    i,n:integer; {n – размерность массива}
    f:text;{текстовый файл}
. . .
begin
. . .
 Assign(f,'5_1.txt');reset(f);{открываем файл для чтения}
 i:=0;
 while not eof(f) do begin
  inc(i);read(f,a[i]);
 end;
 n:=i;
. . .
```

Вывод элементов одномерного массива

1. на экран

```
. . .
const n=10;{размерность массива}
var a:array[1..n] of integer;
    i:integer;
begin
 {здесь располагается алгоритм задания элементов массива и его обработка}
. . .

 writeln('Массив');
```

```
 for i:=1 to N do write(a[i]:4);
 writeln;
. . .
```

2. в типизированный файл

```
. . .
var a:array[1..10] of integer;
    i:integer;
    f:file of integer;
begin
 {здесь располагается алгоритм задания элементов массива и его обработка}
. . .
 Assign(f,'1.my');rewrite(f);
 for i:=1 to 10 do write(f,a[i]);
 Close(f);
. . .
```

3. в текстовый файл (массив с элементами целого типа)

```
. . .
var a:array[1..10] of integer;
    i:integer;
    f:text;
begin
 {здесь располагается алгоритм задания элементов массива и его обработка}
. . .
 Assign(f,'2_3.txt');rewrite(f);
 for i:=1 to 10 do write(f,a[i],' ');
. . .
```

4. в текстовый файл (массив с элементами вещественного типа)

```
. . .
var a:array[1..n] of real;{n-размер массива}
    i:integer;
    f:text;
begin
 {здесь располагается алгоритм задания элементов массива и его обработка}
. . .
 Assign(f,'2_4.txt');rewrite(f);
 for i:=1 to 10 do write(f,a[i]:8:2);
. . .
```

Обработка элементов одномерного массива

1. Нахождение суммы элементов одномерного массива

```
. . .
const n=10;{размерность массива}
var a:array[1..n] of real;{массив чисел}
    i:integer;{индекс элементов}
    s:real;{сумма значений}
begin
 {здесь располагается алгоритм задания элементов массива}
```

```
. . .
 s:=0;
 for i:=1 to N do s:=s+a[i];
 writeln('сумма значений=',s:8:2);
. . .
```

2. Нахождение количества элементов одномерного массива, удовлетворяющих заданному критерию

```
{алгоритм нахождения элементов массива равных 0}
const n=10;{размерность массива}
var a:array[1..n] of real;{массив чисел}
    i:integer;{индекс элементов}
    kol_0:integer;{количество нулей}
begin
 {здесь располагается алгоритм задания элементов массива}
. . .
 Kol_0:=0;
 for i:=1 to N do if a[i]=0 then Kol_0:=kol_0+1;
. . .
```

3. Нахождение максимального значения в одномерном массиве

```
const n=10;{размерность массива}
var a:array[1..n] of real;{массив чисел}
    i:integer;{индекс элементов}
    max:real;{максимум}
begin
 {здесь располагается алгоритм задания элементов массива и его обработка}
. . .
 max:=a[1];
 for i:=2 to N do
  if max<a[i] then max:=a[i];
. . .
```

4. Нахождение индекса минимального значения в одномерном массиве

```
. . .
const n=10;{размерность массива}
var a:array[1..n] of real;{массив чисел}
    i,k_min:integer;{индекс элементов и индекс минимума}
    min:real;{минимум}
begin
 {здесь располагается алгоритм задания элементов массива и его обработка}
 min:=a[1];
 for i:=2 to N do
  if min>a[i] then begin min:=a[i];k_min:=i end;
 writeln('Минимум под номером =',k_min:8);
. . .
```

Задание элементов двумерного массива

1. с использованием датчика случайных чисел

```
. . .
const n=10,m=5;{размерность матрицы}
var a:array[1..n,1..m] of real;{матрица N x M}
    i,j:integer;{индексы элементов}
begin
 for i:=1 to N do
   for j:=1 to M do a[i,j]:=Random*100;
. . .
```

2. считывание из текстового файла

```
. . .
const n=10,m=5;{размерность матрицы}
var a:array[1..n,1..m] of real;{матрица N x M}
    i,j:integer;{индексы элементов}
    f:text;
begin
 Assign(f,'matr.dat');reset(f);
 for i:=1 to N do begin
   for j:=1 to M do read(f,a[i,j]);
   readln(f);
 end;
. . .
```

Вывод элементов двумерного массива

1. на экран

```
. . .
const n=10,m=5;{размерность матрицы}
var a:array[1..n,1..m] of real;{матрица N x M}
    i,j:integer;{индексы элементов}
begin
. . .
 for i:=1 to N do begin
   for j:=1 to M do write(a[i,j]:5:2);
   writeln;
 end;
. . .
```

2. в текстовый файл

```
. . .
const n=10,m=5;{размерность матрицы}
var a:array[1..n,1..m] of real;{матрица N x M}
    i,j:integer;{индексы элементов}
    f:text;
begin
. . .
  Assign(f,'mymatr.txt');rewrite(f);
 for i:=1 to N do begin
   for j:=1 to M do write(f,a[i,j]:5:2);
```

```
        writeln(f);
      end;
    . . .
```

ОТВЕТЫ НА ТЕСТОВЫЕ ЗАДАНИЯ

Тема 1. Среда Delphi и создание консольных приложений.

1. 1), 2), 3), 4), 5)
2. 1), 2)
3. 1), 3)
4. 2), 3)

Тема 2. Редактирование и отладка программ

1. 1), 2)
2. 3), 4)
3. 4)

Тема 3. Программирование разветвляющихся алгоритмов

1. 1), 2), 3)
2. 1), 3), 5)
3. 2), 3)
4. 1) 3) 5)

Тема 4. Программирование циклических алгоритмов

1. 1), 2), 7)
2. 3), 6), 7)
3. 4), 5), 6)
4. 4)
5. 5)
6. 3)

Тема 5. Структурированные типы данных: одномерные массивы

1. 1), 3), 6)
2. 3)
3. 2)

Тема 6. Структурированные типы данных: двумерные массивы

1. 1), 3), 6)
2. 4)
3. 3)

Тема 7. Структурированные типы данных: строки

1. 1), 2)
2. 4), 6)
3. 1), 3), 6)

Тема 8. Структурированные типы данных: множества

1. 1), 2), 3) 5)
2. 4)
3. 1)
4. 2)

Тема 9. Текстовые файлы

1. 3)
2. 1)

3. 3), 4)

Тема 10. Пользовательские процедуры и функции

1. 2), 3), 4), 5)
2. 2)
3. 5)

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Бобровский С. Delphi 5: учебный курс. – СПб: Издательство «Питер», 2000. – 640 с.

2. Хомоненко А.Д. и др. Delphi 7 / Под общей редакцией А.Д. Хомоненко. – СПб.: БХВ-Петербург, 2005. – 1216 с.

3. Форум программистов PASCAL. [Электронный ресурс]. Режим доступа: http://www.cyberforum.ru/pascal/thread16051.html.

СОДЕРЖАНИЕ

Учебное издание

Ольга Владимировна Воробейчикова
Ирина Сергеевна Каверина

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ OBJECT PASCAL

Учебно-методическое пособие

Издано в авторской редакции

Издательство СибГМУ
634050, г. Томск, пр. Ленина, 107
тел. 8(3822) 51–41–53
E-mail: otd.redaktor@ssmu.ru

Издано в электронном виде