Министерство образования и науки Российской Федерации

Государственное образовательное учреждение
высшего профессионального образования

«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

М.И. ОЗЕРОВА

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ
Delphi

Практикум

Владимир 2011

УДК 004.43
ББК 21.183.49
О-46

Рецензенты:
Доктор технических наук, профессор кафедры
специальной техники и информационных технологий
Владимирского юридического института Федеральной
службы исполнения наказаний
Б.Ю. Житников

Кандидат экономических наук, доцент кафедры управления
и информатики в технических и экономических системах
Владимирского государственного университета
Д.А. Градусов

Печатается по решению редакционного совета
Владимирского государственного университета

Озерова, М. И.

О-46 Информационные технологии Delphi : практикум / М. И. Озерова ; Владим. гос. ун-т. – Владимир : Изд-во Владим. гос. ун-та, 2011. – 107 с.
ISBN 978-5-9984-0178-7

В виде практических работ представлен систематизированный материал по основам программирования на языке Delphi. лабораторные работы содержат теоретический материал, практическое задание и контрольные вопросы.

Предназначен для студентов, изучающих среду программирования Delphi в рамках дисциплин «Информационные технологии» и «Программирование на ЯВУ», обучающихся по направлению 230400 «Информационные системы и технологии». Может быть полезен студентам других специальностей, а также широкому кругу читателей, самостоятельно осваивающих среду программирования Delphi.

Рекомендован для формирования профессиональных компетенций в соответствии с ФГОС 3-го поколения.

Библиогр.: 2 назв.

УДК 004.43
ББК 21.183.49

ISBN 978-5-9984-0178-7

ПРЕДИСЛОВИЕ

Учебный практикум «Информационные технологии Delphi» предназначен для студентов, изучающих среду программирования Delphi в рамках дисциплин «Информационные технологии» специальности 230201 – информационные системы и технологии специализации 23020104 – системы компьютерной графики и мультимедиатехнологии и «Программирование на ЯВУ», направления бакалавриата 230400 «Информационные системы и технологии» профиль «Информационные технологии в дизайне».

В процессе изучения этих дисциплин студенты знакомятся с основами теории алгоритмов, принципами программирования на языках высокого уровня, методами построения алгоритмов и структур данных, способствующими развитию конструктивно-логического мышления студента. В результате изучения у студентов формируются следующие профессиональные компетенции: способность на основе информационной модели задачи (словесной постановки) формировать цель решения задачи (ПК-1); способность проводить анализ характера известных и искомых данных, анализировать область их существования, определять условия, при которых задача может быть решена (ПК-4); способность построить математическую модель: формализовать описание задачи с использованием математических, логических и других методов (ПК-5); способность разрабатывать и записывать алгоритмы, обоснованно выбирать язык программирования и реализовывать на нем типовые алгоритмические конструкции (ПК-12); способность правильно выбрать структуры данных для эффективного решения алгоритмических задач (ПК-4); способность осуществлять проверку правильности программы (ПК-22).

Бурное развитие вычислительной техники, потребность в эффективных средствах разработки программного обеспечения привели к появлению систем программирования, ориентированных на так называемую "быструю разработку", среди которых можно выделить Borland Delphi и Microsoft Visual Basic. Delphi – это среда быстрой разработки, в которой в качестве языка программирования используется язык Delphi – строго типизированный объектно-ориентированный язык, в основе которого лежит хорошо знакомый программистам Object Pascal.

С помощью Delphi создаются приложения для операционной системы Windows, но помимо этого с помощью дополнительных средств можно написать, например, программы и для Linux. Среда Delphi легко расширяется установкой дополнительных модулей. Пользовательский интерфейс также хорошо настраиваемый.

В России Borland Delphi появляется в конце 1993 г. и сразу же завоевывает широкую популярность. Новые версии выходят практически каждый год. В них реализуются все новые компоненты и технологии программирования. Например, OpenGl можно использовать с разными языками программирования, поддерживающими работу с **DLL**. Одним из таких языков программирования является **Object Pascal**, использующийся в среде **Delphi,** что позволяет создавать не только базы данных, но и мультимедийные приложения (игры, демо-программки).

Delphi оказал огромное влияние на создание концепции языка C# для платформы .NET. Многие его элементы и концептуальные решения вошли в состав C#.

Практикум состоит из лабораторных работ, которые содержат теоретический материал, практическое задание и контрольные вопросы. Рассмотрены материал по структуре Delphi, основные элементы языка Object Pascal.

Лабораторная работа № 1

ОСНОВНЫЕ ПОНЯТИЯ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ. СИСТЕМА ВИЗУАЛЬНОГО ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОЕКТИРОВАНИЯ Delphi

Цель работы: изучить основные понятия объектно-ориентированного программирования, познакомиться с системой визуального объектно-ориентированного проектирования Delphi.

Теоретическая часть

В объектно-ориентированном программировании главной отправной точкой при проектировании программы является не процедура, не действие, а *объект*. Такой подход достаточно естественен, поскольку в реальном мире нас окружают именно объекты, взаимодействующие друг с другом.

Объектно-ориентированное программирование базируется на трех основных принципах: *наследование, инкапсуляция и полиморфизм*. Программа, построенная по этим принципам, - это не последовательность операторов, не некий жесткий алгоритм, а совокупность объектов и способов их взаимодействия. Обмен информацией между объектами происходит посредством *сообщений*.

Объекты

Назовем объектом понятие, абстракцию или любой предмет с четко очерченными границами, имеющий смысл в контексте рассматриваемой прикладной проблемы. Все объекты можно идентифицировать, между ними можно установить отношение тождества (два Сидоровых Ивана – это разные люди, но они студенты одной группы).

Классы объектов

Классом называют особую структуру, которая может иметь в своем составе поля, методы и свойства. Класс выступает в качестве объектного типа данных, а объект – это конкретный *экземпляр* класса. Например, два Сидоровых Ивана принадлежат одному и тому же классу объектов, они

студенты группы ТКС-206. Именно с этим связана их одинаковость (одинаковый шифр группы, одно расписание занятий и т.д.).

Каждый конкретный класс имеет свои особенности поведения и характеристики, определяющие этот класс. Например,

<table>
<tr><th colspan="4">Геометрический объект</th></tr>
<tr><th>Объемный</th><th colspan="3">Плоский</th></tr>
<tr><td></td><td>С вершинами</td><td colspan="2">Без вершин</td></tr>
<tr><td></td><td></td><td>Окружность</td><td>Эллипс</td></tr>
</table>

Наивысший уровень – самый общий и самый простой, каждый последующий уровень более специфический и менее общий. На самом последнем уровне можно определить цвет, стиль заполнения, величину радиуса окружности и т.п.

Если характеристика уже однажды определена для более высокого уровня, то все уровни, расположенные ниже, имеют ту же характеристику (если определена окружность, понятно, что вершин у нее нет).

Таким образом, классы-наследники могут наследовать характеристики классов-родителей.

Свойства

Свойства – перечень параметров объекта, которые определяют внешний вид и поведение объекта, выделяют уникальные особенности каждого экземпляра. К свойствам относятся имя, тип, значение, цвет, размер и др. *Состояние* – совокупность всех свойств данного объекта.

Методы

Метод **–** это некоторое действие (операция), которое можно выполнять над данным объектом. В результате этого действия в объекте что-нибудь меняется (например, местоположение, цвет и др.). Другими словами можно еще сказать, методом называется команда, которую может выполнять объект. Для каждого класса объектов имеется свой перечень методов, которые можно к нему применить или которые он может выполнить. Например, объект можно удалить с экрана, переместить в другое место.

События

Каждый объект способен реагировать на определенные события. Это разновидность свойства объекта. При возникновении события производится его обработка.

События – сигналы, формируемые внешней средой, на которые объект должен отреагировать соответствующим образом.

Средой взаимодействия объектов являются *сообщения*, генерируемые в результате наступления различных *событий*.

События наступают в результате действий пользователя – перемещения курсора пользователя, нажатия кнопок мыши или клавиш на клавиатуре, а также в результате работы самих объектов. Для каждого объекта определено множество событий, на которые он может реагировать. Для конкретных экземпляров объекта могут быть определены *обработчики* каких-либо из этих событий, которые и определяют реакцию данного экземпляра объекта.

Объект можно определить как совокупность свойств и методов, а также событий, на которые он может реагировать.

Внешнее управление объектом осуществляется через обработчиков событий, которые обращаются к методам и свойствам объекта.

Инкапсуляция

С одной стороны объект, обладает определенными свойствами, которые характеризуют его состояние в данный момент. С другой стороны, над объектами возможны операции, которые приводят к изменению этих свойств. Доступ к изменению свойств осуществляется только с помощью методов, присущих данному классу объектов. Есть метод, данное свойство данного объекта можно изменить, нет метода – нельзя. Методы как бы «окружают» свойства объекта, говорят, что свойства «инкапсулированы» в объект. Для обеспечения инкапсуляции класс не должен позволять прямого доступа к своим данным. *Инкапсуляция* - механизм скрытия всех внутренних деталей объекта, не влияющих на его поведение.

Наследование

Классы-наследники могут наследовать характеристики классов-родителей, т.е. один объект приобретает свойства другого объекта, добавляя к ним свойства, характерные только для него.

Наследование определяет отношение между классами: объекты класса-наследника обладают всеми свойствами и методами объектов класса-родителя и не должны их повторно реализовывать.

Класс «Точка» (родитель)		Класс «Окружность» (наследник)	
Свойства	**Методы**	**Свойства**	**Методы**
Координаты (x,y)	**Перемещение**	**Координаты центра (x, y)**	**Перемещение**
Цвет	**Изменение цвета**	**Цвет**	**Изменение цвета**
		Радиус	**Изменение радиуса**

Полиморфизм (имеющий много форм)

К объектам разных классов можно применять один и тот же метод, вот только действовать этот метод будет по-разному. Например, к большинству объектов в Windows&Office можно применять одни и те же методы: копирование, перемещение, переименование, удаление и т.п. Однако механизмы реализации этих методов для разных классов (файл в Windows и документ Word) неодинаковы.

Полиморфизм – возможность использования одних и тех же методов для объектов разных классов, только реализация этих методов будет *индивидуальной* для каждого класса.

Система визуального объектно-ориентированного проектирования Delphi

Delphi – это среда разработки программ, ориентированных на работу в Windows. В основе идеологии Delphi лежат методология объектно-ориентированного программирования и технология визуального проектирования.

Работа производится в интегрированной среде разработки (ИСР) Delphi, которая предоставляет пользователю формы, в которых размещаются с помощью мыши необходимые компоненты, имеющиеся в библиотеке Delphi. С помощью простых манипуляций мышью можно изменять размеры и расположение этих компонент. При этом в процессе проектирования можно постоянно видеть результат – изображение формы и расположенных на ней компонентов, а самое главное заключается в том, что во время проектирования формы редактор кода Delphi автоматически генерирует код программы, включая в нее соответствующие фрагменты, описывающие данный компонент. В соответствующих диалоговых окнах можно изменить заданные по умолчанию свойства компонентов и при необходимости написать обработчики событий.

Компоненты могут быть визуальными, видимыми при работе приложения и невизуальными, выполняющими некоторые служебные функции, они отображаются в виде значка в процессе проектирования и не видны при работе приложения.

Система визуального объектно-ориентированного проектирования Delphi позволяет создавать:

- законченные приложения для Windows самой различной направленности, от чисто вычислительных и логических до использующих графику и мультимедиа;
- профессионально выглядящий оконный интерфейс для любых приложений, написанных на разных языках; интерфейс удовлетворяет всем требованиям Windows и автоматически настраивается на ту систему, которая установлена на компьютере пользователя, поскольку использует функции, процедуры и библиотеки Windows;
- динамически присоединяемые библиотеки (Dll) компонентов, форм, функций, которые затем можно использовать из других языков программирования;
- мощные системы работы с локальными и удаленными базами данных любых типов;
- формировать и печатать сложные отчеты, включающие таблицы, графики и т.п.;
- справочные системы (файлы.hlp) как для своих приложений, так и для любых других, с которыми можно работать не только из приложений, но и просто из Windows;
- профессиональные программы установки для приложений Windows, учитывающие всю специфику и все требования операционной системы.

Включать объекты в свою программу пользователь может вручную, используя соответствующие операторы, или путем визуального программирования, используя заготовки-компоненты.

Контрольные вопросы

1. В чем заключается основное различие процедурного и объектно-ориентированного программирования?
2. Назовите основные принципы объектно-ориентированного программирования.
3. Что такое объект?
4. Что такое класс?
5. В чем заключается наследование?

6. Что такое инкапсуляция и для чего она применяется?
7. В чем заключается полиморфизм операций?
8. Что такое метод?
9. Что такое свойство объекта?
10. Что такое событие?
11. Каково назначение сообщений?
12. В чем заключаются преимущества визуального программирования интерфейса?

Содержание отчета

1. Название, цели лабораторной работы.
2. Формулировка практической части работы.
3. Выводы по работе.
4. Ответы на контрольные вопросы.

Лабораторная работа № 2

ИЗУЧЕНИЕ СРЕДЫ Delphi И РАБОТА В НЕЙ. ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ АЛГОРИТМОВ

Цель работы: изучить структуру модуля приложения Delphi, научиться создавать проект в Delphi. Программирование алгоритмов линейной структуры.

Теоретическая часть

Общая организация программы в Delphi

Программа, создаваемая в среде Delphi в процессе проектирования приложения, основана на *модульном принципе*. Головная программа состоит из объявления списка используемых модулей и нескольких операторов, создающих объекты для необходимых форм и запускающих приложение на выполнение.

Все объекты компонентов размещаются в формах. Для каждой формы, проектируемой в приложении, Delphi автоматически создает отдельный модуль, в который пользователь может ввести собственный код, создавая обработчики различных событий. Именно в модулях и осуществляется программирование задачи.

Структура головной программы приложения Delphi

Код головной программы создается Delphi автоматически и, как правило, не требует модификации. Пример такой головной программы приведен ниже. Не задумывайтесь пока над ее содержанием, посмотрите только общую структуру модуля.

program Project1;	Программа начинается с ключевого слова program, после которого указывается имя программы. Оно совпадает с именем файла, в котором был сохранен проект. Это же имя присваивается исполняемому файлу приложения. По умолчанию используется имя Project1
uses	В разделе uses перечисляются все модули, загружаемые программой.

Forms, Unit1 in 'Unit1.pas' {Form1};	Первый модуль Forms является системным, а следующие – модулями разработанных пользователем форм Данный проект состоит из одной формы с именем Form1, которая содержится в модуле Unit1 После ключевого слова in указывается имя файла, в котором содержится модуль Unit1
{$R *.res}	Эта строка представляет собой директиву компилятора, которая связывает с исполняемым модулем файлы ресурсов Windows (.Dfm, .Res). По умолчанию для файлов ресурсов используется расширение .Res.
begin Application.Initialize; Application.CreateForm(TForm1, Form1); Application.Run; end.	Оператор Application.Initialize инициализирует приложение. Application.CreateForm создает объекты формы. Application.Run начинает выполнение приложения.

Структуры модуля приложения Delphi

Текст программ хранится в модулях, названия которых должны совпадать с именами файлов. Модуль состоит из трех разделов: *интерфейса*, *реализации* и *инициализации*. Структура модуля приложения Delphi:

Unit Unitl;	Название модуля (Это название используется в предложении Uses при подключении модуля к программе).
Interface	Раздел интерфейса
…	
Implementation	Раздел реализации (исполняемая часть)
…	
begin	Раздел инициализации
…	
end.	

Раздел интерфейса начинается ключевым словом Interface, он сообщает компилятору, какая часть модуля является доступной для других модулей программы.

Здесь могут размещаться списки подключаемых модулей, объявления типов, констант, переменных, заголовки функций и процедур, к которым будет доступ из других модулей. Иными словами, в этом разделе перечисляется все то, что должно быть видимым из программы, которая его использует.

Раздел реализации начинается ключевым словом Implementation и содержит объявления локальных переменных, процедур и функций, поддерживающих работу формы.

Начинается раздел директивой {$R *.dfm}, указывающей компилятору, что в раздел реализации нужно вставить инструкции установки значений свойств формы, которые находятся в файле с расширением .dfm , имя которого совпадает с именем модуля.

Далее в разделе реализации могут помещаться предложения uses, объявления типов, констант, переменных, к которым не будет доступа из других модулей. Здесь же располагаются все тексты процедур и функций, объявленных в разделе interface. Заголовки процедур и функций могут полностью совпадать с заголовками из интерфейсной части или отличаться от них полным отсутствием параметров. Если в этой части набран текст функции или процедуры, не представленной в Interface, то данная функция или процедура будет локальной.

Раздел инициализации позволяет выполнить инициализацию переменных модуля. Располагается после раздела реализации между Begin и End.

В этой части помещают операторы, которые должны выполняться один раз при первом обращении к модулю (после ключевого слова Initialization), или операторы, выполняемые при любом завершении модуля (после ключевого слова Finalization). Эта часть является необязательной. На практике данный подход используется редко, поэтому обычно после Implementation сразу ставится End с точкой (без Begin).

Структура событийной процедуры

Procedure <название процедуры>	Заголовок процедуры. Название процедуры состоит из двух частей: *названия объекта + название события*

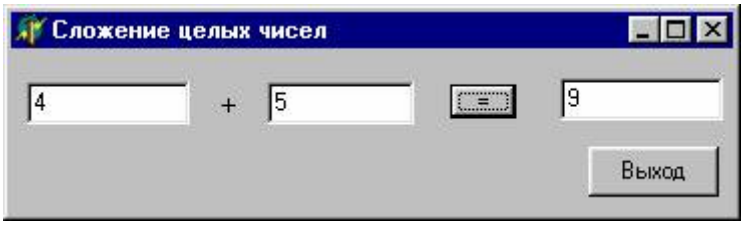
Const <имя константы> = <значение константы>; …… <имя константы> = <значение константы>;	Раздел описания констант
Type <имя типа> = <тип>; …… <имя типа> = <тип>;	Раздел типов
Var <имя переменной>:<тип>; …… <имя переменной>:<тип>;	Раздел описания переменных
<Тексты локальных процедур и функций с заголовками>	Раздел процедур и функций

Begin
<инструкции>
End;
	Раздел, в котором пишутся инструкции процедуры обработки событий

Пример структуры модуля приложения

Создадим следующее приложение: ввести два числа, найти их сумму и показать результат.

Форма данного приложения будет содержать следующие компоненты: три текстовых окна Tedit для ввода данных и вывода результата, одну метку Tlabel для изображения на экране знака «+» и две кнопки Tbutton, одна – для реализации задачи, другая – для ее завершения.



Модуль данного приложения практически весь генерируется автоматически Delphi при проектировании приложения. Пользователь лишь вво-

дит свой код для обработки событий, наступающих при нажатии кнопок (в тексте модуля он выделен жирным шрифтом).

Структура модуля данного приложения:

unit Unit1;	Модуль начинается с ключевого слова unit, после которого указывается имя модуля, совпадающее с именем файла, где сохранен модуль.
interface	**Открытый интерфейс модуля.**

uses
  Windows, Messages, SysUtils, Variants,
  Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;
	Список подключаемых библиотечных модулей.

type	

TForm1 = class(TForm)
  Edit1: TEdit;
  Edit2: TEdit;
  Edit3: TEdit;
  Label1: TLabel;
  Button1: TButton;
  Button2: TButton;
	Объявление класса формы, а также типов всех компонентов, включаемых в класс формы.

procedure Button1Click(Sender: Tobject); procedure Button2Click(Sender: Tobject);	Объявление процедур, описывающих действия, которые должны происходить при нажатии кнопок.
private { Private declarations }	**Закрытый раздел класса**. Объявление переменных, функций и процедур, включаемых в класс формы, но *недоступных для других модулей*.
public { Public declarations } end;	**Открытый раздел класса**. Объявление переменных, функций и процедур, включаемых в класс формы, *доступных для других модулей.*

var Form1: TForm1;	Объявление типов, констант, переменных, функций и процедур, к которым будет доступ из других модулей, но которые не включаются в класс формы.
implementation	**Реализация модуля**. Сюда могут помещаться предложения uses, объявления типов, констант, переменных, к которым не будет доступа из других модулей. Здесь же располагаются все реализации процедур и функций, объявленных в разделе interface.
{$R *.dfm}	Директива компилятора, обеспечивающая компоновку файлов ресурсов формы. Удалять из текста модуля нельзя, так как загрузочный модуль не будет создан.

procedure TForm1.Button1Click(Sender: TObject);
var
a,b,c :integer;
begin
a:=strtoint(edit1.Text);
b:=strtoint(edit2.Text);
c:=a+b;
Edit3.Text:=inttostr(c);
end;
	Реализация событийной процедуры, реализующей обработки события *Click* на кнопке ***Button1***, в результате выполнения которой складывается два введенных числа и полученная сумма выводится на экран.

procedure TForm1.Button2Click(Sender: TObject);
begin
Close;
end;
end.
	Реализация событийной процедуры, реализующей обработки события *Click* на кнопке ***Button2***, в результате выполнения которой заканчивается работа приложения.

Структура проекта в Delphi

Проект Delphi представляет собой набор программных единиц – модулей, которые хранятся в отдельных файлах.

Примечание

В Delphi существуют файл проекта и файлы проекта. Это разные вещи. Файл проекта – это файл головной программы с расширением .Dpr, файлы проекта – это набор всех файлов приложения.

Файл с расширением .Dpr содержит основную информацию о проекте. По умолчанию этот файл называется Project1.dpr. Новичкам редактировать его не рекомендуется.

Файл с расширением .Pas – это файл модуля. В нем хранится текст программы на языке Object Pascal. По умолчанию этот файл называется Unit1.pas. Это единственный файл, который можно редактировать начинающим.

Файл с расширением .Dfm содержит информацию о внешнем виде формы. Информация закодирована.

Файл с расширением .Res – это ресурсовый файл, в котором хранится информация о курсорах, иконках и т.п.

Файл с расширением .Exe – исполняемый файл приложения.

Файлы с расширениями .~Df, .~Pa – файлы со старыми версиями приложения.

У файлов с расширениями Pas, Dfm, ~Df, ~Pa всегда одинаковое имя (по умолчанию Unit1).

У файлов с расширениями Dpr, Exe, Res – также одинаковое имя (по умолчанию Project1).

Примечания

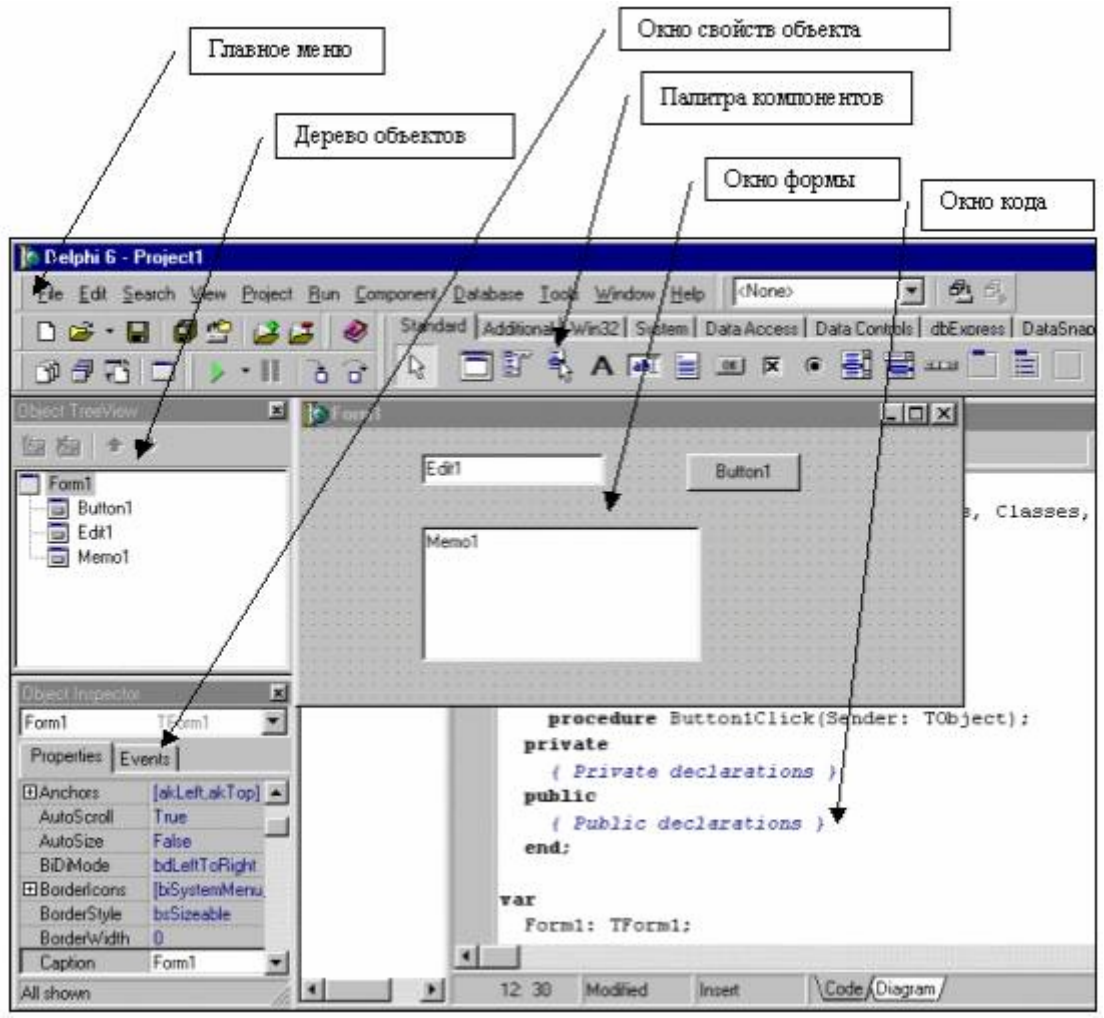
1. При сохранении нового проекта приложения рекомендуется создавать отдельную папку.
2. Названия Project1 и Unit1 рекомендуется не изменять.

3. Копирование файлов проекта на дискету или в другую папку выполняется только с помощью файлового менеджера Windows (например, проводника). В среде Delphi это делать запрещается.

4. Файлы с расширением Exe, ~Df, ~Pa, Dsm можно не копировать.

Общее описание среды Delphi

После запуска Delphi на экране компьютера появляется основное окно интегрированной среды разработки (ИСР).



В верхней части окна ИСР отображается полоса главного меню. Ниже две инструментальные панели:

 Левая панель содержит два ряда кнопок, дублирующих некоторые наиболее часто используемые команды меню.

 Правая панель содержит панель библиотеки визуальных компонентов (Visual Component Library - VCL), в дальнейшем просто *палитра компонентов*.

Палитра компонентов позволяет выбрать с помощью иконок визуальные и другие компоненты, из которых, как из «строительных блоков», собирается разрабатываемое Delphi-приложение.

Палитра содержит ряд страниц, закладки которых видны в ее верхней части. Всего страниц тринадцать. Наиболее употребляемые из них Standard (стандартные компоненты) и Additional (дополнительные компоненты).

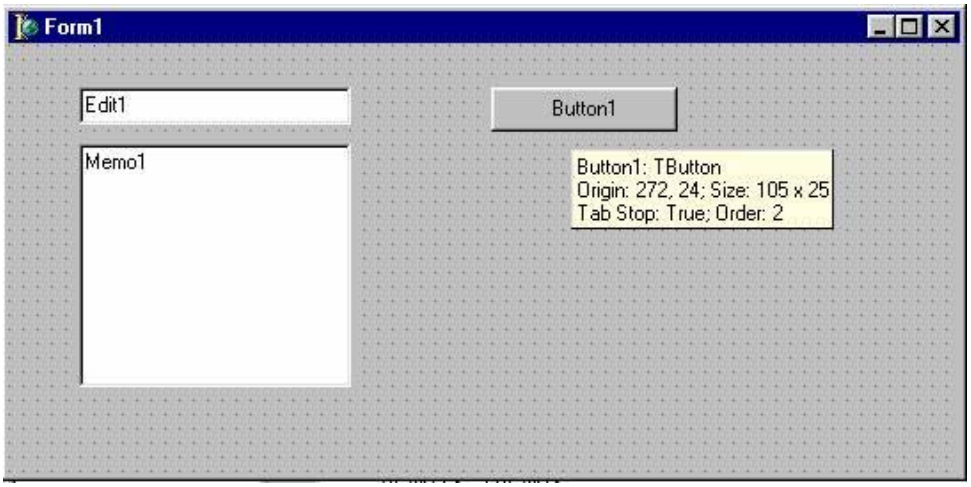
Стандартные компоненты. Это классы, реализующие интерфейсные элементы среды Windows. Среди них главное меню (MainMenu), всплывающее меню (PopupMenu), этикетка или метка (Label), текстовое поле (Edit) и другие.

Дополнительные компоненты. Эти классы представляют собой различные дополнительные интерфейсные элементы – графические кнопки (BitBtn), редактор с вводом по шаблону (MaskEdit) и другие.

Правее полосы главного меню располагается небольшая инструментальная панель, которая служит для сохранения и выбора различных конфигураций окна ИСР.

На основном окне интегрированной среды разработки расположены еще три окна:

Окно формы Form1 представляет собой заготовку (макет) окна разрабатываемого приложения.



Окно инспектора объектов Object Inspector позволяет изменять свойства (характеристики) объектов: формы, командных кнопок, полей ввода и т.д.

Инспектор объектов состоит из двух страниц: Properties (свойства) и Events (события).

Каждая страница разделена на две части. На странице свойств в левой части находится название свойства, а в правой - его значение.

"+" слева от названия указывает на то, что свойство состоит из нескольких значений. Значениями свойств могут быть слова, числа, значения из раскрывающегося списка.

Нажав на "▼" справа от поля значения, можно получить раскрывающийся список значений.

При нажатии на "..." вызывается специальное диалоговое окно.

Примечание: если значением является число или текст, то после его набора лучше нажать Enter, иначе оно может быть не зафиксировано. При нажатии на "Esc" ввод отменяется

Страница Events используется для задания реакции на событие. Страница также состоит из двух частей. В первой - название события, а во второй - название процедур, обрабатывающих данное событие. Если в правой части ничего не написано, то программа на данное событие не реагирует. Название процедуры состоит из двух частей: *названия объекта + название события*.

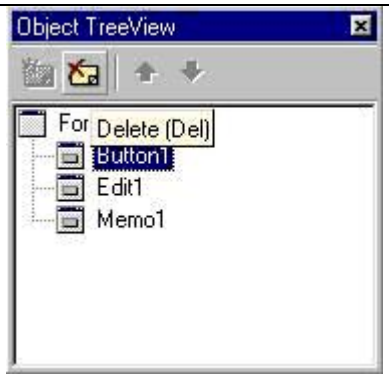
Для создания новой реакции на событие необходимо дважды щелкнуть в правой половине напротив нужного события. В этом месте появится название новой процедуры, а на экране - окно редактора кода.

Например:

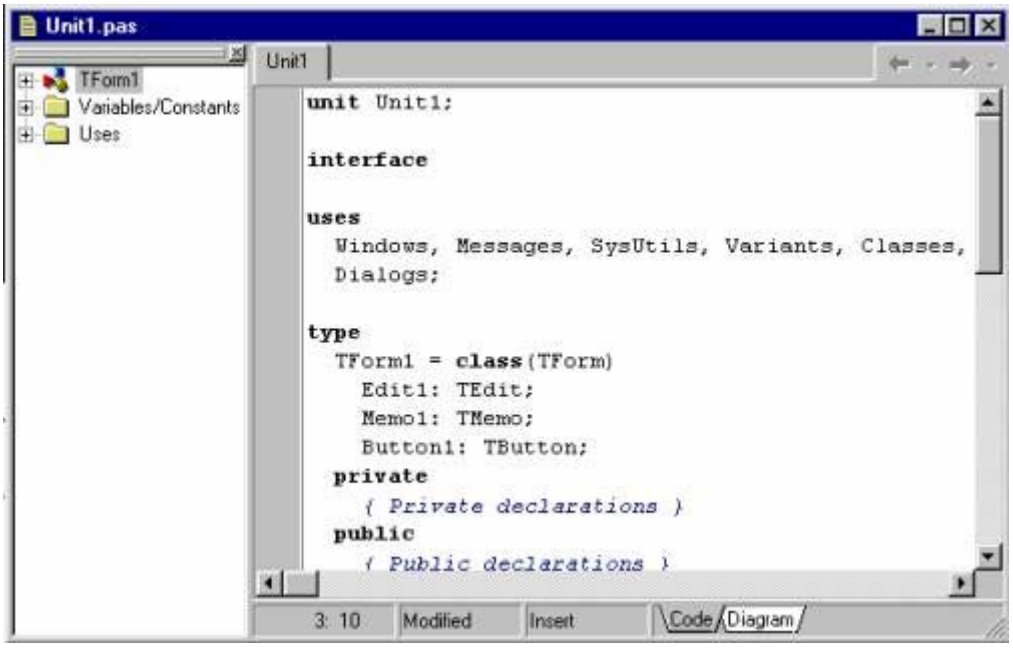
```
procedure TForm1.Button2Click(Sender: TObject);
begin
Close;
end;
```

	Примечания 1) если Вы случайно создали ненужную реакцию на событие, то не стоит обращать на это внимание: при сохранении или выполнении программы данная ошибка исправляется автоматически средой Delphi; 2) запрещается исправлять текст программы, написанный самой Delphi, иначе вся программа может быть уничтожена.

Окно Object TreeView (дерево объектов) отображает иерархию компонентов приложения с точки зрения их принадлежности друг другу.

	В дереве объектов можно осуществлять операции щелчка и перетаскивания, перемещая дочерние компоненты в другие контейнеры, при этом изменения синхронно отображаются в редакторе форм

Окно Code Editor (редактор кода), в котором между Begin и End можно печатать инструкции Object Pascal, реализующие процедуру обработки событий.



Краткая характеристика некоторых компонентов

Ниже перечислены компоненты, их основные свойства и события, при совершении которых будут выполняться запрограммированные действия.

Компонент **Form** (экранная форма). Форма представляет не только внешний вид окна приложения, но и сама является полноценным компонентом с собственными свойствами и событиями, хотя на палитре компонентов ее нет.

Основные свойства компонента Form

Align	Задает режим выравнивания объектов внутри формы
BorderStyle	Задает стиль обрамления формы, а также поведение формы (возможность менять размеры окна)
Caption	Задает заголовок окна формы
Color	Задает цвет формы
Font	Задает атрибуты шрифта формы

Значения свойств можно задать либо в окне свойств объекта, либо в программе.

Пример использования в программе

```
Form1.Color:=clRed;      {задание цвета формы}
```

Основное событие компоненты Form

Load	Происходит при загрузке формы.

Компонент **Label** (надпись или метка)  . Назначение – нести на себе надпись. Можно использовать для вывода ответа или пояснения вводимых данных. Относится к группе Standard.

Основные свойства компоненты Label

Caption	Задает заголовок надписи, выводимой на экран
Alignment	Задает режим выравнивания текста метки
AutoSize	Позволяет автоматически менять размеры метки, чтобы соответствовать размерам надписи (значение True)
Font	Задает шрифт, используемый для отображения текста
Visible	Задает видимость надписи на экране. Имеет два значения. Если значение True, то надпись видна, False – нет
WordWrap	Разрешает разбивку и перенос непомещающихся строк, следует согласовывать значение этого свойства со свойством AutoSize

Пример использования в программе

```
Label1.Caption:='Сумма = '+IntToStr(s);

{Оформление вывода результата }
```

Компонент ***Edit*** (поле редактирования)  . Используется для ввода/вывода чисел и текста в программу. Относится к группе Standard.

Основные свойства компонента Edit

AutoSize	Задает необходимость изменения размера компонента при изменении размера шрифта (если True)
BorderStyle	Задает стиль обрамления поля
Text	Задает содержимое строки редактирования
MaxLength	Ограничивает число вводимых в поле символов
ReadOnly	Запрещает редактировать отображаемый текст (если True)

Пример использования в программе

```
edit3.text:=FloatToStrF(c,ffFixed,10,4)

{Оформление вывода результата }
```

Основное событие компоненты Label

Change	Происходит, когда пользователь изменяет текст

Компонент **Button** (командная кнопка)  . Используется для задания реакции на событие. Относится к группе Standard.

Основные свойства Button

Caption	Задает название кнопки
Height	Задает высоту кнопки
Width	Задает ширину кнопки
Left	Задает расстояние от левой границы кнопки до левой границы формы
Top	Задает расстояние от верхней границы кнопки до верхней границы формы

Основное событие компонента Button

OnClick	Происходит, когда пользователь щелкает основной (левой) кнопкой мыши на объекте

Пример использования в программе

```
procedure TForm1.Button2Click(Sender: TObject);
begin
Close        {Заканчивает выполнение программы}
end;
```

Пример создания, сохранения, компиляции и отладки простого приложения

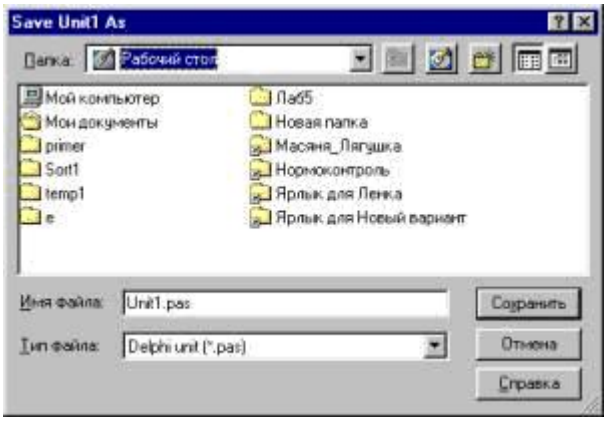
Создадим приложение, которое обеспечивает ввод двух целых чисел, вычисляет их сумму и выводит значение результата.

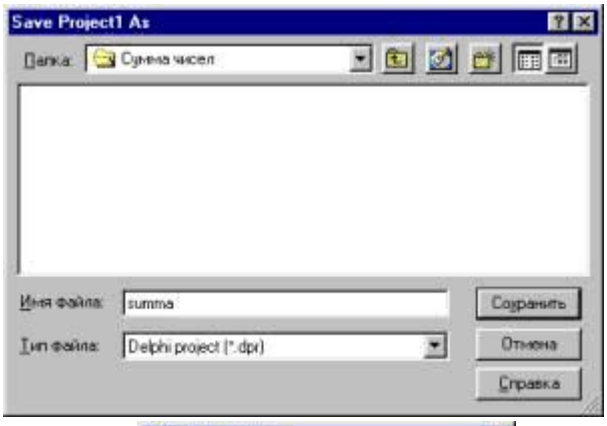
Для этого выполним следующие действия:

1. Запустите ИСР Delphi с помощью команды Главного меню Windows Пуск → Программы → Borland Delphi 7 → Delphi 7.

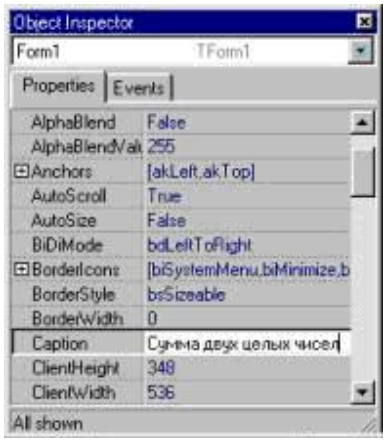
2. Создайте новый проект при помощи команды File → New → Application. В раскрытом окне формы можно размещать визуальные компоненты для реализации проекта приложения.

3. Сохраните новый проект командой меню File → Save Project As. В появившемся окне Save Unit1 As с помощью кнопки создайте новую папку для файлов создаваемого проекта с названием «Сумма чисел» на Рабочем столе. Откройте созданную папку и нажмите кнопку Сохранить. После сохранения файла модуля Unit1.pas откроется окно Save Project As. Задайте имя файла проекта «Summa»и нажмите кнопку Сохранить.

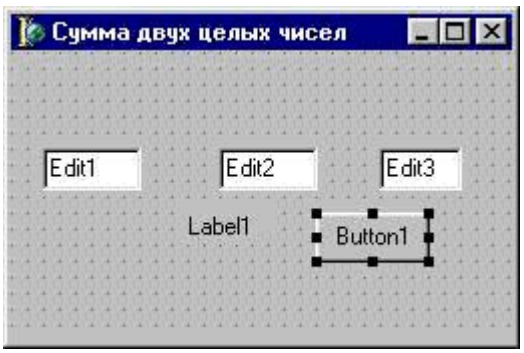




4. Измените заголовок формы Form1. Для этого в окне Инспектора объектов откройте страницу Свойства (*Properties*), выберите свойство *Caption* (Заголовок) и задайте его новое значение «Сумма двух целых чисел».



5. Разместите на форме компоненты Edit1, Edit2, Edit3, Label1, Button1, используя соответствующие пиктограммы    группы Standard.



Для этого щелкните на вкладке Standard палитры компонентов, затем выберите пиктограмму требуемой компоненты и щелкните в окне формы в том месте, где хотите ее расположить.

Примечание. Выделенный компонент формы окружен восемью маркерами. Именно его свойства отображаются в окне Инспектора объектов. Выделенный компонент можно перемещать по форме, меняя его местоположение. Можно также менять его размеры, потянув за один из маркеров.

6. Измените свойства компонента *Label1*: задайте свойство *Caption* (заголовок) как «+», в списке свойств *Font* (Шрифт) свойству *Size* (Размер) присвойте значение 20.

7. Измените свойства компонента *Button1*: задайте свойство *Caption* (заголовок) как «=», в списке свойств *Font* (Шрифт) свойству *Size* (Размер) присвойте значение 20.

8. Выровняйте компоненты на форме. Для этого выделите их все при нажатой клавише Shift, затем в контекстном меню выберите команду Position → Align.

9. Измените свойства компонентов *Edit1, Edit2, Edit3*: удалите текст из свойства *Text*.

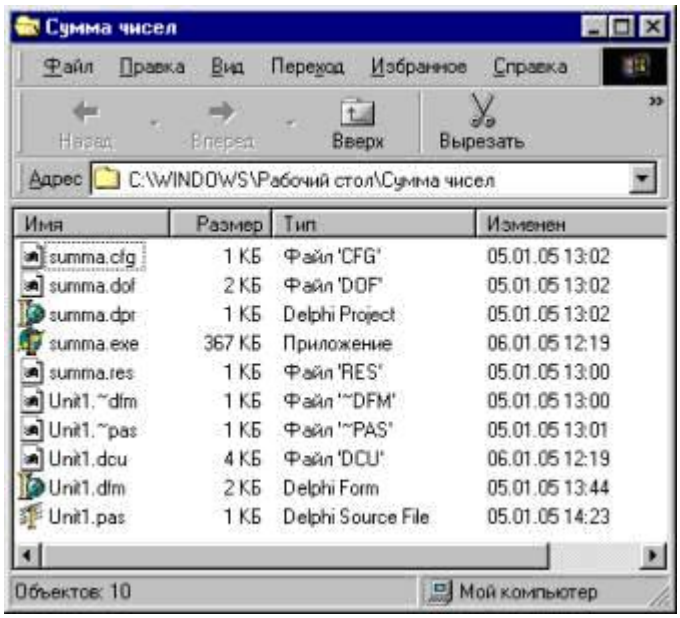
10. Добавьте на форму еще три объекта *Label*, расположите их над объектами *Edit1* - *Edit3* и задайте их свойствам *Caption* значения «Слагаемое», «Слагаемое» и «Сумма».

11. Активизируйте окно Редактора кода, нажав F12. Просмотрите сгенерированный Delphi модуль описания формы и размещенных на ней компонентов.

12. Сохраните изменения, внесенные в проект, командой меню File → Save All.

13. Откомпилируйте созданный проект командой меню Project → Compile summa (summa – это имя проекта).

14. С помощью файлового менеджера просмотрите папку проекта.



summa.cfg – файл конфигурации проекта

summa.dof – файл параметров проекта.

summa.dpr – файл проекта Delphi. Главная программа приложения.

summa.exe – откомпилированный проект. Исполняемая Windows-программа.

summa.res – файл ресурсов Windows.

Unit1.~dfm, unit1.~pas – файлы резервных копий.

Unit1.dcu – откомпилированный модуль.

Unit1.dfm – файл формы (двоичный файл) содержит начальные данные для компонент.

Unit1.pas – исходный код модуля формы.

15. Закройте окно файлового менеджера, активизируйте окно Delphi и запустите проект на выполнение командой меню Run → Run.

16. Рассмотрите окно созданной формы. Обратите внимание, что оно имеет стандартные атрибуты окна Windows.

17. Завершите работу приложения любым стандартным методом.

18. Создайте код обработчика событий: при нажатии на кнопку «=» два введенных символа должны складываться.

Для этого выделите кнопку *Button1*, в окне Инспектора объектов перейдите на вкладку *Events* (События), выберите событие *OnClick* и дважды щелкните левой кнопкой мыши на пустом поле списка. В открывшемся окне Редактора кода между операторами begin и end разместите необходимые операторы.

Окончательно процедура обработки события щелчка на кнопке *Button1* должна выглядеть следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  a,b,c: integer;     {слагаемые и сумма целые числа}
begin
a:=StrToInt(Edit1.Text); {преобразование текстовой строки в целое число}
b:=StrToInt(Edit2.Text);
c:=a+b;
Edit3.Text:=IntToStr(c); {преобразование целого числа в текстовую строку}
end;
```

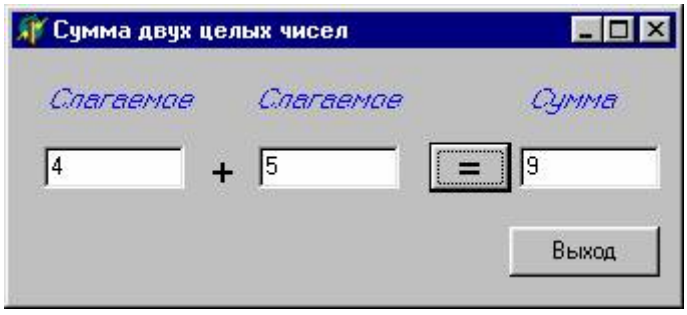
19. Сохраните изменения в проекте.

20. Запустите приложение на выполнение.

21. После проверки работы приложения закройте его.

22. Запустите приложение из Windows, используя исполняемый файл.

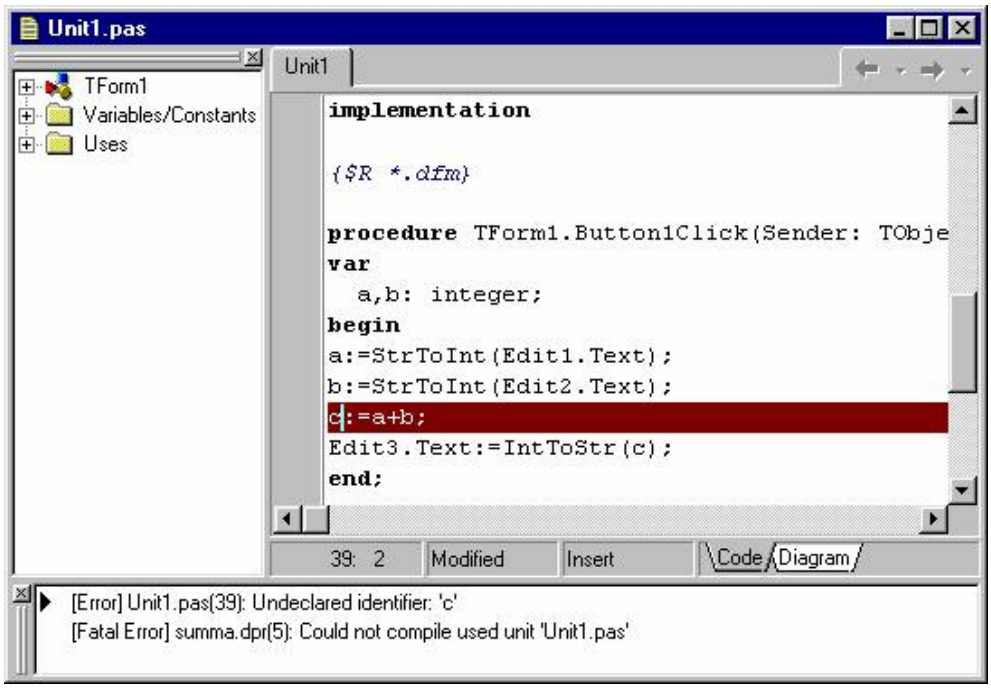
Самостоятельно в созданном проекте измените свойства компонентов *Label2, Label3*, *Label4*, задав стиль шрифта – курсив, цвет шрифта – синий, размер символов – 10 пунктов, и добавьте кнопку на закрытие приложения. В редакторе кода используйте оператор *Close*.



Теперь преобразуйте программу так, чтобы она могла оперировать с вещественными значениями, используя функции *StrToFloat* и *FloatToStr*.

Ошибки выполнения приложения

Во время компиляции текст программы проверяется на отсутствие синтаксических ошибок. Компилятор просматривает программу от начала. Если обнаруживается ошибка, то процесс компиляции приостанавливается и в окне редактора кода выделяется строка, которая, по мнению компилятора, содержит ошибочную конструкцию.



В нижнюю часть окна редактора кода компилятор выводит сообщения об ошибках. Первая ошибка – это первая от начала текста программы синтаксическая ошибка, обнаруженная компилятором. Наличие в тексте даже одной синтаксической ошибки приводит к возникновению второй, фатальной, ошибки (Fatal Error) – невозможности генерации исполняемой программы.

Примечание

Строка, выделенная компилятором, не всегда содержит ошибку. Довольно часто ошибочной является инструкция, находящаяся в предыдущей строке.

Наиболее типичные ошибки компиляции

Сообщения компилятора	Вероятная причина
Undeclared identifier (Необъявленный идентификатор)	Используется переменная, не объявленная в разделе var программы. Ошибка при написании имени переменной. Ошибка при написании имени инструкции (оператора)
Unterminated string (Незавершенная строка)	При записи строковой константы не поставлена завершающая кавычка
Incompaible types … and … (Несовместимые типы)	В операторе присваивания тип выражения не соответствует переменной, получающей значение выражения, или не может быть приведен к её типу
Missing operator or semicolon (Отсутствует оператор или точка с запятой)	Не поставлена точка с запятой после инструкции программы

Если синтаксических ошибок в программе нет, компилятор создаст исполняемый файл программы, который позже можно будет запустить из Windows. Имя исполняемого файла такое же, как и у файла проекта, а расширение - .exe. Delphi помещает исполняемый файл в ту же папку, где находится файл проекта.

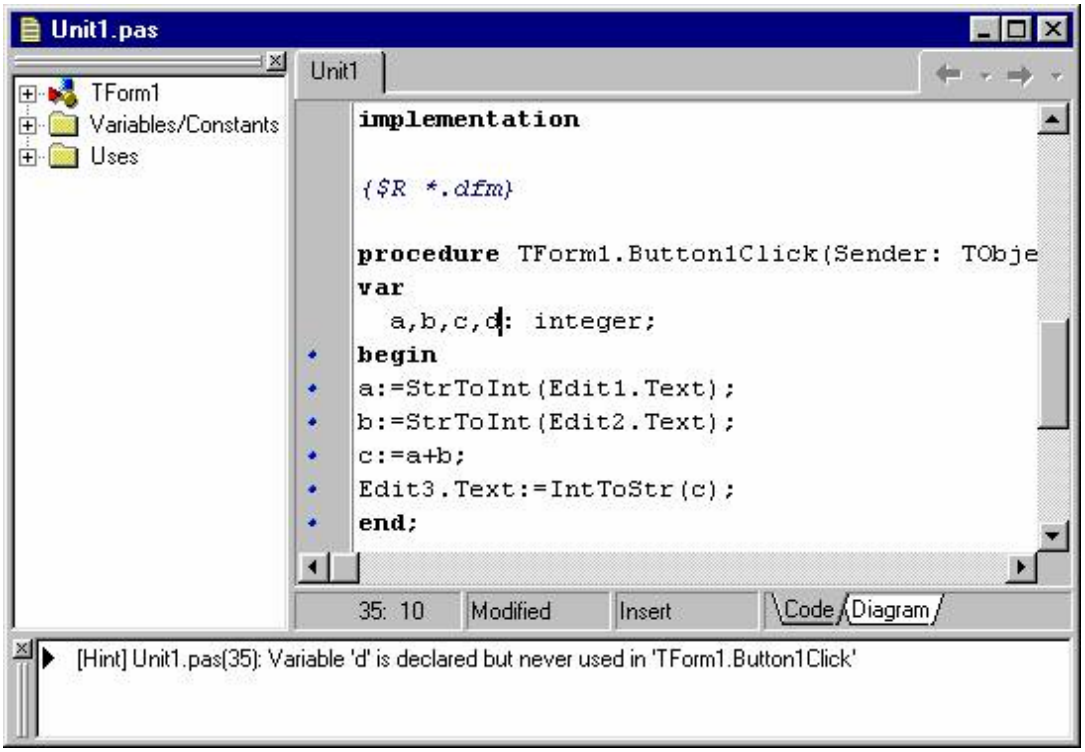
Предупреждения и подсказки компилятора

При обнаружении в программе неточностей, которые не являются ошибками, компилятор выводит подсказки и предупреждения.

Наиболее часто выводимые подсказки и предупреждения:

сообщение об объявленной, но не используемой в программе переменной:

 [Hint] Unit1.pas(35): Variable 'd' is declared but never used in 'TForm1.Button1Click' (сообщение об объявленной переменной, не используемой в программе):



 [Warning] Unit1.pas(39): Variable 'a' might not have been initialized (переменной, вероятно, не присвоено начальное значение).

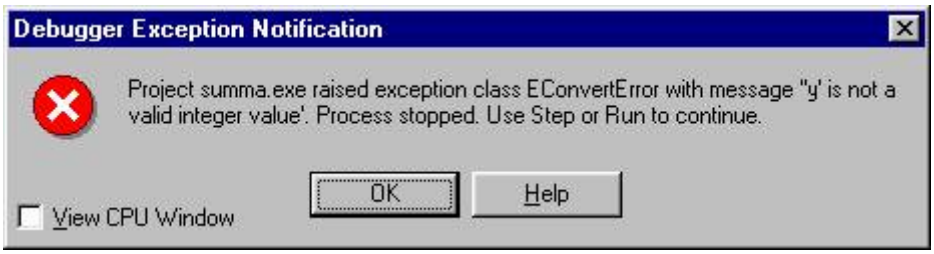
Ошибки времени выполнения

Во время работы приложения могут возникать ошибки, которые называются ошибками времени выполнения. В большинстве случаев причинами являются неверные исходные данные.

Созданное приложение можно запустить на выполнение в ОС Windows и из среды Delphi, при этом сообщение об одной и той же ошибке времени выполнения будет выводиться на экран в разного вида окна.

Например, если одно из целых чисел ввести не как число, а как букву или вещественное число, то на экран будет выведено следующее окно с сообщением об ошибке и выполнение программы приостанавливается.

Если программа запускается из среды Delphi с помощью встроенного отладчика, то при возникновении ошибки также появляется окно с сообщением об ошибке и выполнение программы приостанавливается, но в этом окне помимо сообщения об ошибке указывается тип ошибки. Например, в этом окне сообщается о невозможности преобразования строки «y» в число.



Чтобы остановить работу программы, в которой возникла ошибка, надо выполнить команду меню Run → Program Reset.

При разработке проекта программист должен предусмотреть все возможные варианты некорректных действий пользователя, которые могут привести к возникновению ошибок времени выполнения, и обеспечить способы защиты от них.

Программирование алгоритмов линейной структуры

Оператор присваивания

В результате выполнения оператора присваивания переменная получает значение. Формат оператора присваивания:

<Имя Переменной> **:=** <Выражение>, где двоеточие и следующий за ним знак равенства – это символ оператора присваивания.

После каждой инструкции программы ставится символ “точка с запятой”.

Примеры

```
Counter:=0;
d:=b*b-4*a*c;
pi:=3.1415926;
z:=(rl+r2)/(rl*r2);
```

Выражение

Выражение состоит из операндов и операций, которые выполняются над операндами.

При вычислении значений выражений соблюдается приоритет операций. Операции с более высоким приоритетом выполняются раньше операций, приоритет которых ниже. Если приоритет операций одинаков, то опе-

рация, стоящая левее, выполняется раньше. Для задания любого нужного порядка выполнения операций в выражении можно использовать скобки.

Выражение, заключенное в скобки, трактуется как один операнд. Это значит, что операции, стоящие в скобках, будут выполняться в обычном порядке, но раньше, чем операции, находящиеся за скобками. При записи выражений, содержащих скобки, должна соблюдаться парность скобок, т. е. число открывающих скобок должно быть равно числу закрывающих скобок. Нарушение парности скобок – наиболее распространенная ошибка при записи выражений.

Арифметические выражения

Операндами арифметических выражений могут быть числа (константы) и переменные, функции и арифметические выражения, заключенные в круглые скобки. Операции обозначают действия, выполняемые над операндами.

В простейшем случае арифметическое выражение – это константа или переменная.

Тип результата зависит от типа операндов. Тип константы, входящей в выражение, определяется видом самой константы. Например, константы 0, 1 и -512 целого типа (INTEGER), а константы 1.0, 0.0, 3.2e-05 – вещественного типа (REAL).

Арифметические операции языка Pascal представлены в следующей таблице.

Операция	**Действие**	**Выражения, использующие арифметические операции**	**Тип операндов**	**Тип результата**
+	Сложение	A+B, A-B, A*B	REAL	REAL
-	Вычитание		INTEGER	INTEGER
*	Умножение		INTEGER, REAL	REAL
/	Деление	A/B	REAL	REAL
			INTEGER	REAL
			INTEGER, REAL	REAL
DIV	Деление нацело (отбрасывание остатка)	A **DIV** B	INTEGER	INTEGER
MOD	Вычисление остатка от деления	A **MOD** B	INTEGER	INTEGER

Примеры арифметических выражений:

123
0.001
i
A+B/C
Summa*0.75
(B1+B3+B3)/100
Cena MOD 1000
(Cena DIV 1000)+Cena
(rl+r2+r3)/(rl*r2*r3)

В арифметических выражениях приняты следующие правила выполнения операций: ***, /, DIV, MOD** имеют более высокий приоритет, чем **+** и **-**.

Примеры:

Выражение	**Программная запись**
$ab+cd$	a*b + c*d
$\frac{a+b}{c+d}$	(a+b) / (c+d)
$a+\frac{cd}{b}+\frac{e}{fg}$	a+c*d/b+e/(f*g) или a+c*d/b+e/f/g

Строковые выражения

Операндами строковых выражений могут быть символы или последовательность символов, заключенная в апострофы (строки). Подробнее строковые выражения и операции над ними будут разобраны в лабораторной работе «Работа со строками».

Логические выражения

Результатом выполнения логических выражений является значение True или False. Для написания простых логических выражений используются знаки отношений (**=, <, >, <=, >=, <>**), а для составления сложных – логические операции (**Not, And, Or**). Подробнее логические операции и логические выражения будут разобраны в лабораторной работе «Разветвляющие процессы».

Выполнение оператора присваивания

Выполнение инструкции присваивания заключается в следующем: сначала вычисляется результат выражения, находящегося справа от символа

присваивания, затем вычисленное значение записывается в область памяти, предназначенной для хранения переменной, имя которой стоит слева от символа присваивания (другими словами, вычисленное значение присваивается этой переменной).

Например, в результате выполнения инструкций

i=0	Значение переменной i становится равным нулю
a=b+c;	Значением переменной a будет число, равное сумме значений переменных b и c
j=j+1;	Значение переменной j увеличивается на единицу
Label1.Caption=’Пример’;	Задается заголовок надписи
Edit1.Text=FloatToStr(a);	Задается значение текстового поля

Оператор присваивания считается верным, если тип выражения соответствует переменной или может быть приведен к её типу. Переменной типа REAL можно присвоить значение выражения типа REAL или INTEGER. Переменной типа INTEGER можно присвоить значение выражения только типа INTEGER.

Во время перевода исходной программы в исполняемую компилятор проверяет соответствие типов выражений и переменных. Если тип выражения не соответствует типу переменной, то компилятор выдает сообщение об ошибке.

Комментарии

Для облегчения понимания текста программы в него могут включаться комментарии – любой текст, заключенный в фигурные скобки.

{Этот текст является комментарием}

Если необходимо закомментировать отдельную строку, то используют следующий вид комментария:

Здесь программный код // здесь текст, являющийся комментарием

Комментарии обычно располагают на отдельной строке или в конце строки текста программы после инструкции. Комментарий может занимать несколько строк текста программы.

Практическая часть

Создать приложение, вычисляющее значения переменных по заданным расчетным формулам и наборам исходных данных. На экран вывести значения вводимых исходных данных и результаты вычислений, сопровождая ввод и вывод поясняющими комментариями.

Вариант задания	Расчетные формулы	Значения исходных данных
1	$a = \frac{2\cos(x-\pi/6)}{1/2+\sin^2 y}$ $b = 1+\frac{z^2}{3+z^2/5}$	$x = 1.426$ $y = -1.22$ $z = 3.5$
2	$\gamma = \left\lvert x^{y/x} - \sqrt[3]{y/x}\right\rvert$ $\psi = (y-x)\frac{y-z/(y-x)}{1+(y-x)^2}$	$x = 1.825$ $y = 18.225$ $z = -3.298$
3	$s = 1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}$ $\psi = x(\sin x^3 + \cos^2 y)$	$x = 0.335$ $y = 0.025$
4	$y = e^{-bt}\sin(at+b) - \sqrt{\lvert bt+a\rvert}$ $s = b\sin(at^2\cos 2t) - 1$	$a = -0.5$ $b = 1.7$ $t = 0.44$
5	$\omega = \sqrt{x^2+b} - b^2\sin^3(x+a)/x$ $y = \cos^2 x^3 - x/\sqrt{a^2+b^2}$	$a = 1.5$ $b = 15.5$ $x = -2.9$
6	$s = x^3 tg^2(x+b)^2 + a/\sqrt{x+b}$ $Q = \frac{bx^2-a}{e^{ax}-1}$	$a = 16.5$ $b = 3.4$ $x = 0.61$
7	$R = x^2(x+1)/b - \sin^2(x+a)$ $s = \sqrt{xb/a} + \cos^2(x+b)^2$	$a = 0.7$ $b = 0.05$ $x = 0.5$
8	$y = \sin^3(x^2+a)^2 - \sqrt{x/b}$ $z = \frac{x^2}{a} + \cos(x+b)^3$	$a = 1.1$ $b = 0.004$ $x = 0.2$
9	$f = \sqrt[3]{mctb + \lvert c\sin t\rvert}$ $z = m\cos(bt\sin t) + c$	$m = 2$ $c = -1$ $t = 1.2$ $b = 0.7$

10	$y = abx^2 - \frac{a}{\sin^2(x/a)}$ $d = ae^{-\sqrt{a}}\cos(bx/a)$	$a = 3.2$ $b = 17.5$ $x = -4.8$
11	$f = \ln(a + x^2) + \sin^2(x/b)$ $z = e^{-ax}\frac{x + \sqrt{x+a}}{x - \sqrt{\lvert x-b \rvert}}$	$a = 10.2$ $b = 9.2$ $c = 0.5$ $x = 2.2$
12	$y = \frac{a^{2x} + b^{-x}\cos(a+b)x}{x+1}$ $R = \sqrt{x^2 + b} - b^2\sin^3(x+a)/x$	$a = 0.3$ $b = 0.9$ $x = 0.61$
13	$z = \sqrt{ax\sin 2x + e^{-2x}(x+b)}$ $\omega = \cos^2 x^3 - x/\sqrt{a^2 + b^2}$	$a = 0.5$ $b = 3.1$ $x = 1.4$
14	$U = \frac{a^2 x + e^{-x}\cos bx}{bx - e^{-x}\sin bx + 1}$ $f = e^{2x}\ln(a+x) - b^{3x}\ln(b-x)$	$a = 0.5$ $b = 2.9$ $x = 0.3$
15	$z = \frac{\sin x}{\sqrt{m^2 + \sin^2 x}} - cm\ln mx$ $s = e^{-ax}\sqrt{x+1} + e^{-bx}\sqrt{x+1.5}$	$m = 0.7$ $c = 2.1$ $x = 1.7$ $a = 0.5$ $b = 1.08$

Контрольные вопросы

1. Что такое свойство объекта, каким образом его можно изменять?
2. Опишите структуру и назначение отдельных элементов головной программы приложения Delphi.
3. Каково назначение модуля в проекте приложения Delphi? Опишите назначение отдельных разделов модуля.
4. Как различается доступность объектов, описанных в разделах interface и implementation?
5. Как указать ссылку на свойства и методы объекта в тексте программы?
6. Какие компоненты входят в интегрированную среду разработки приложений Delphi?
7. Перечислите основные компоненты окна среды Delphi и укажите их назначение.
8. Как получить подсказку Delphi по элементам окна и компонентам проектируемого графического интерфейса?
9. Каково назначение страниц Properties и Events в окне Object Inspector?

10. Как разместить компонент на форме?
11. Какими способами можно изменять свойства компонента? Приведите примеры.
12. Из чего состоит проект Delphi? Опишите назначение различных файлов.
13. Каково назначение обработчиков событий? Каким образом можно инициировать создание процедуры-обработчика событий?
14. Опишите порядок получения справки по синтаксису языка Object Pascal в ИСР Delphi.
15. Изучите и опишите информацию о компонентах палитры Standard по справке Delphi.
16. Изучите и опишите информацию о числовых типах данных языка Object Pascal по справке Delphi.
17. Изучите и опишите информацию об арифметических операциях языка Object Pascal по справке Delphi.
18. Изучите и опишите информацию о функции Random языка Object Pascal по справке Delphi.
19. Что хранится в файле проекта с расширением .Dfm?
20. Что хранится в файле проекта с расширением .Pas?
21. Что хранится в файле проекта с расширением .Dpr?
22. Что хранится в файле проекта с расширением .Res?

Содержание отчета

1. Название, цели лабораторной работы.
2. Формулировка практической части работы.
3. Фрагменты программного кода.
4. Выводы по работе.
5. Ответы на контрольные вопросы.

Лабораторная работа № 3

СТРУКТУРНЫЕ ОПЕРАТОРЫ. ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ РАЗВЕТВЛЯЮЩЕЙСЯ СТРУКТУРЫ

Цель работы: изучить основные логические операции, условные операторы, оператор выбора Case.

Теоретическая часть

Операторы в программе-обработчике событий выполняются в той последовательности, в которой они записаны. Однако достаточно часто требуется изменить порядок выполнения операторов в зависимости от выполнения (или невыполнения) определенного условия. Существуют управляющие конструкции, предназначенные для управления порядком выполнения операторов. Основанием для принятия решений в управляющих операторах является истинность или ложность условного (логического) выражения.

Условные – это такие выражения, которые возвращают одно из двух значений True (Истина) или False (Ложь). Простые логические выражения содержат операции отношения (операции сравнения): = (равно), > (больше), < (меньше), <> (не равно), >= (больше или равно), <= (меньше или равно). Сложные логические выражения строятся из простых логических выражений и логических операций, примененных к ним.

Основные логические операции

В приведенной таблице A и B – логические выражения

№ п/п	Операция	Обозначение	Истолкование
1	Отрицание (инверсия)	not A	Не A; Неверно, что A
2	Конъюнкция (логическое произведение, логическое И)	A and B	A и B; A, но B; как A, так и B; A вместе с B; A в то время как B

3	Дизъюнкция (логическое сложение, логическое ИЛИ)	A or B	A или B; A или B, или оба
4	Дизъюнкция (исключающее ИЛИ)	A xor B	A либо B; A или B, но не оба

Приоритеты выполнения логических операций в логических выражениях:

1. Отрицание (not).
2. Логическое произведение (and).
3. Логическое сложение (or), исключающее или (xor).

Скобки меняют порядок выполнения операций.

Таблица истинности для основных логических операций

A	**B**	**Not A**	**A and B**	**A or B**	**A xor B**
False	False	True	False	False	False
False	True	True	False	True	True
True	False	False	False	True	True
True	True	False	True	True	False

Условные операторы. Оператор условия If

Условные операторы предназначены для выбора на исполнение одного из возможных действий (операторов) в зависимости от некоторого условия, при этом одно действие может отсутствовать.

Выбор действия в зависимости от выполнения условия может быть реализован при помощи оператора **IF.**

Оператор условия If

Различают два типа условных операторов: **If…Then** и **If…Then…Else**.

Конструкция If…Then применяется, когда необходимо выполнить определенные действия только в том случае, если значение некоторого условия равно «истина».

Синтаксис оператора If…Then

If <условие> Then < Инструкция для обработки истинного условия >

Вначале вычисляется значение условия (выражения логического типа) и, если значение условия равно «истина», выполняется инструкция, следующая за словом Then.

Если в результате выполнения условного оператора должны выполняться 2, 3 инструкции или более, то необходимо использовать составной оператор.

Синтаксис составного оператора

```
Begin
    Оператор1;
    Оператор2;
    …
End;
```

Пару Begin…End часто называют операторными скобками и в программе для наглядности записывают на одном уровне (End под соответствующим ему Begin). Внутри одного составного оператора может находиться другой составной оператор.

Синтаксис оператора If…Then…Else:

```
If условие Then
  <Инструкция для обработки истинного условия >
Else
  < Инструкция для обработки ложного условия>
```

Оператор If…Then…Else выполняется следующим образом:

1. Вычисляется значение условия (выражения логического типа).

2. Если значение условия равно «истина», то выполняется инструкция, следующая за словом Then. Если значение условия равно «ложь», то выполняется инструкция, следующая за словом Else.

Примеры

1. Программа поиска наибольшего из трех действительных чисел:

```
var
Max, a, b, c: real;
begin
a:=strtofloat(edit1.Text);
b:=strtofloat(edit2.Text);
c:=strtofloat(edit3.Text);
if a >= b
then if a >= c then Max:=a else Max:=c
else if b >= c then Max:=b else Max:=c;
edit4.text:=floattostr(Max);
end;
```

2. Программа проверки на равенство нулю делителя при выполнении операции вещественного деления:

```
if b<> 0 then   {проверка отличия делителя от 0}
begin
c:=a / b; {выполнение операции вещественного деления}
Edit3.text:=FloatToStrF(c,ffGeneral,7,4);
{преобразование вещественного числа в форматированную текстовую строку}
end
else
 begin
   Edit3.Font.Color:=clRed;     {изменение цвета шрифта сообщения на красный}
   Edit3.Width:=130;            {изменение ширины текстового окна}
   Edit3.text:='На ноль делить нельзя'  {вывод сообщения}
 end;
```

Условные операторы. Оператор выбора Case

Оператор Case позволяет реализовать множественный выбор. Переход организуется на одну их ветвей в зависимости от значения заданного выражения (селектора выбора).

Оператор Case существует также в двух вариантах:

Case k of
A1: <инструкция1>;
A2: <инструкция2>;
…
AN: <инструкцияN>;
End;

и

Case k of
A1: < инструкция 1>;
A2: < инструкция 2>;
…
AN: < инструкция N>
Else
< инструкция, выполняемая в случае, если значение выражения не попало ни в один из списков констант A1, A2, …,AN>
End;

Здесь:

 k – выражение-селектор, от значения которого зависит дальнейший ход программы, может иметь только простой порядковый тип (целый, символьный, логический).

 Список констант (A1,…,AN) – константы того же типа, что и селектор, выполняющие роль меток ветвей. Если константы представляют диапазон чисел, то вместо списка можно указать первую и последнюю константу диапазона, разделив их двумя точками.

Исполнение оператора начинается с вычисления выражения k, полученное значение сравнивается с константами (метками) и выполняется соответствующий оператор.

Например

Для нахождения наибольшего из двух неравных действительных чисел используется оператор

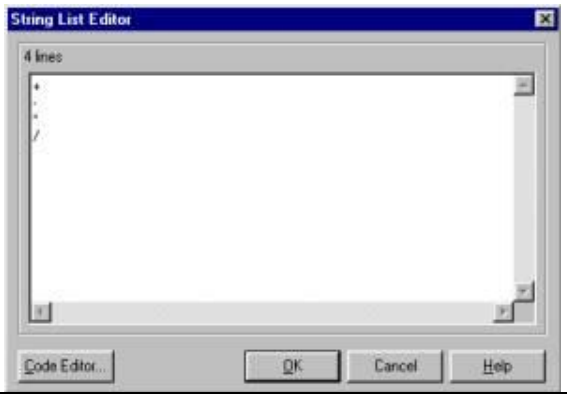
```
var
Max, a, b : real;
begin
a:=strtofloat(edit1.Text);
b:=strtofloat(edit2.Text);
case a > b of
   true: Max:=a;
   false: Max:=b;
end;
edit3.text:=floattostr(Max);
```

Краткая характеристика некоторых компонентов, наиболее часто используемых в примерах для построения управляющих конструкций.

RadioGroup

Компонент **RadioGroup** (группа переключателей)  . Позволяет отображать поля с ограниченным множеством значений. Относится к группе Standard.

Основные свойства компонента RadioGroup

Caption	Задает название группы переключателей
Items	Определяется количество переключателей в группе и надписи около них. Надписи задаются в окне String List Editor 

ItemsIndex	Задает номер кнопки, выбранной по умолчанию. 0 – первая, -1 – ни одна кнопка не выбрана.

Пример использования в программе

(*выбор арифметического действия по номеру отмеченной кнопки*)

```
case Radiogroup1.ItemIndex of
 0 : c:=a+b;
 1 : c:=a-b;
 2 : c:=a*b;
end;
```

CheckBox

Компонент **CheckBox** (кнопка с независимой фиксацией - флажок Windows)  . Позволяет пользователю выбрать/отменить определенную опцию. Состояние кнопки содержится в свойстве Checked. Относится к группе Standard.

Основные свойства компонента CheckBox

Caption	Задает текст, сопровождающий кнопку
AllowGrayed	Задает наличие у кнопки третьего состояния. Если значение этого свойства False, то кнопка может находиться в двух состояниях – включенном или выключенном. Если значение свойства равно True, то добавляется третье состояние, когда кнопка неактивна
Checked	Возвращает или задает, наличие галочки флажка. True – кнопка включена, False - выключена

Пример использования в программе

(проверка наличия галочки у флажка, если есть – вывод форматный, нет – бесформатный)

```
if checkbox1.checked then
 edit3.text:=FloatToStrF(c,ffFixed,10,4)
else
  edit3.text:=FloatToStr(c)
```

RadioButton

Компонент **RadioButton** (радиокнопка)  . Позволяет пользователю выбрать определенную опцию. Состояние кнопки содержится в свойстве Checked. Относится к группе Standard. Может быть активна только одна

радиокнопка в конкретной группе (например, на форме или на компонентах Panel, GroupBox и т.д.)

Основные свойства компонента RadioButton

Caption	Задает текст, сопровождающий кнопку
Checked	Возвращает или задает, наличие галочки флажка. True – кнопка включена, False - выключена

Пример использования в программе

(*проверка выбранной радио кнопки, если выбрана первая – вывод форматный, если вторая – бесформатный*)

```
if RadioButton1.checked then
 edit3.text:=FloatToStrF(c,ffFixed,10,4)
if RadioButton2.checked then
  edit3.text:=FloatToStr(c)
```

Практическая часть

1. В следующих задачах организовать выполнение алгоритма следующим образом: выбор вычисления по той или иной формуле с помощью элемента управления переключатель, форматный или бесформатный вывод с помощью элемента управления флажок.

Вариант задания	**Формулировка задачи**
1	Найти количество отрицательных, положительных чисел среди *a, b, c* и абсолютное значение суммы этих чисел
2	Напечатать числа *a, b, c* в порядке возрастания и убывания
3	Переменной A присвоить ее значение, увеличенное в N раз, 2N раз, 3N раз
4	Для двух целых чисел A и B определить сумму S, разность R и среднеарифметическое SR
5	По известному радиусу вычислить объем и площадь поверхности шара. $V = \frac{4}{3}\pi r^3, s = 4\pi r^2$
6	Даны два числа. Вычислить их сумму, разность, произведение и частное
7	Вычислить объем и площадь полной поверхности цилиндра, если известны высота и радиус основания. $V = \pi r^2 h, s = 2\pi r(r + h)$
8	Вычислить квадратный корень, квадрат числа A

9	Вывести наибольшее, наименьшее, среднее значения из переменных *a, b, c* и среднеарифметическое этих чисел
10	Вычислить периметр и площадь прямоугольного треугольника по заданным длинам двух катетов a и b
11	Вычислить площади геометрических фигур: прямоугольника и треугольника по заданным сторонам
12	Даны два числа. Найти среднеарифметическое кубов этих чисел и среднегеометрическое модулей этих чисел
13	Дана сторона равностороннего треугольника. Найти площадь этого треугольника, его высоту, радиусы вписанной и описанной окружностей $S=\frac{\sqrt{3}}{4}a^2, h=\frac{\sqrt{3}}{2}a, R=\frac{2}{\sqrt{3}}a, r=\frac{a}{2\sqrt{3}}$
14	Заданы стороны прямоугольника. Определить его периметр, площадь и длину диагонали
15	Вычислить площади геометрических фигур: трапеции и круга. $S_T=(a+b)h/2$

2. Вывести число пар на выбранный день недели, при выборе субботы и воскресенья вывести «Выходной». Учесть выбор числителя, знаменателя.

3. Написать программу, определяющую, можно ли коробку размерами $a \times b \times c$ упаковать в посылку размерами $r \times s \times t$. «Углом» укладывать нельзя.

Контрольные вопросы

1. С помощью какого оператора реализуется алгоритмическая структура «Развилка»? Нарисуйте ее блок-схему.
2. С помощью какого оператора реализуется алгоритмическая структура «Выбор»? Нарисуйте ее блок-схему.
3. Перечислите знаки отношений.
4. Перечислите основные логические операции.
5. Когда применяется условный оператор?
6. Назовите два вида условного оператора.
7. Что позволяет выполнить составной оператор?
8. Что позволяет делать оператор выбора?
9. Что такое селектор?
10. Приведите пример использования оператора условия.
11. Приведите пример использования оператора выбора.
12. Для чего служит кнопка с независимой фиксацией?

13. Опишите свойство Checked.
14. Изучите и опишите информацию о компоненте RadioButton палитры Standard по справке Delphi.
15. В чем различие объектов CheckBox и RadioButton?
16. Каково назначение объекта RadioGroup?
17. Как задать список элементов-переключателей в панели RadioGroup?
18. По какому свойству RadioGroup определяется выбранный переключатель? Как задать этому свойству значение?
19. Какое свойство объекта Label можно использовать для вывода ответа?
20. Какое свойство объекта Edit можно использовать для вывода ответа?

Содержание отчета

1. Название, цели лабораторной работы.
2. Формулировка практической части работы.
3. Фрагменты программного кода.
4. Выводы по работе.
5. Ответы на контрольные вопросы.

Лабораторная работа № 4

СТРУКТУРНЫЕ ОПЕРАТОРЫ. ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ЦИКЛИЧЕСКОЙ СТРУКТУРЫ

Цель работы: изучить элементарные циклические структуры, процедуры и функции.

Теоретическая часть

Циклические структуры

Циклические структуры обеспечивают многократное выполнение одной и той же последовательности инструкций, которая называется *телом цикла*. Существуют два вида элементарных циклических структур:

- *циклы с параметром;*
- *итерационные циклы, или циклы с условием.*

Циклы с параметром используют тогда, когда количество повторов тела цикла заранее известно. В языке *Pascal* циклы с параметром реализуются с помощью оператора For.

Итерационные циклы используются тогда, когда число повторений заранее неизвестно, но задано условие окончания цикла, причем если условие окончания цикла проверяется перед выполнением тела цикла, то такие циклические структуры называют итерационными циклами *с предусловием* (“Выполнять пока”), а если проверка условия происходит после выполнения тела цикла, – итерационными циклами *с постусловием* (“Выполнять до тех пор пока не”).

На практике циклы с условием чаще всего используют в двух случаях:

- Число повторений заранее неизвестно (например, цикл до достижения требуемой точности результата).
- Число повторений заранее известно, но шаг параметра цикла не равен 1 (или –1).

В языке Pascal итерационные циклы *с предусловием* реализуются с помощью оператора While, а итерационные циклы *с постусловием* - с помощью оператора Repeat … Until.

Циклы с параметром. Оператор FOR

Синтаксис оператора For:

Если шаг параметра цикла равен 1

```
For счетчик_цикла:=нач_зн_счетч_цик to кон_зн_счетч_цик do
Begin
     {последовательность операторов}
end;
```

Если шаг параметра цикла равен -1

```
For счетчик_цикла:=нач_зн_счетч_цик downto кон_зн_счетч_цик do
Begin
     {последовательность операторов}
end;
```

Последовательность операторов, находящаяся между begin и end, будет выполнена (кон_зн_счетчика - нач_зн_счетчика_цикла + 1) раз. Если начальное значение счетчика превышает конечное при шаге цикла равном 1, то последовательность операторов, находящаяся между begin и end, ни разу не будет выполнена.

Пример

Протабулировать функцию (найти значения функции) $y=\sin x$ на отрезке *[a, b]* с шагом *h*.

Программа, реализующая данный алгоритм, имеет вид:

```
var
a,b,h,x,y: real;
n, i: integer;
begin
a:=strtofloat(edit1.Text);      {Начало отезка}
b:=strtofloat(edit2.Text);      {Конец отрезка}
h:=strtofloat(edit3.Text);      {Шаг табуляции}
n:=ceil((b-a)/h)+1;             {Расчет количества точек}
x:=a;
for i:=1 to n do
begin
  y:=sin(x);                     {Вычисление следующего значения функции}
  x:=x+h;                        {Вычисление следующей точки табуляции}
end;
end;
```

Вычислить сумму конечного ряда $y=\sum_{1}^{n}\frac{k+0.3}{3k^2+5}$, т.е. сумму первых *n* членов последовательности $\frac{k+0.3}{3k^2+5}$ (*k=1,2,3..,n*).

Программа, реализующая данный алгоритм, будет иметь вид:

```
var
 k,n: integer;
 y,s: real;
begin
n:=strtoint(edit1.Text);
s:=0;
{Организация цикла по числу членов последовательности}
for k:=1 to n do
 begin
  y:=(k+0.3)/(3*k*k+5);    {Вычисление очередного члена последовательности}
  s:=s+y;                  {Вычисление суммы последовательноти}
 end;
edit2.Text:=floattostr(s);{Вывод результирующей суммы}
end;
```

Замечание. В том случае если тело цикла состоит только из одной инструкции, нет смысла использовать составной оператор (операторные скобки Begin…End).

Итерационные циклы. Операторы циклов с условием

Синтаксис цикла с предусловием While.

```
While условие do
begin
{ последовательность инструкций }
end;
```

Работа оператора цикла:

 Инструкции, находящиеся между begin и end, выполняются до тех пор, пока условие истинно. Если условие неверно при входе в цикл, инструкции ни разу не исполняются.

 Чтобы цикл завершился, необходимо, чтобы последовательность инструкций между begin и end изменяла значения переменных, входящих в выражение условия.

Пример. Подсчитать количество цифр в заданном целом числе.

Программа, реализующая данный алгоритм, будет иметь вид:

```
var
a, k: integer;
begin
a:=strtoint(edit1.Text);
k:=0;
while a<>0 do
begin
k:=k+1;
a:=a div 10
end;
edit2.text:=inttostr(k);
end;
```

Замечание. В том случае если тело цикла состоит только из одной инструкции, нет смысла использовать составной оператор (операторные скобки Begin…End).

Синтаксис цикла с постусловием Repeat … Until

```
Repeat
{ последовательность инструкций }
Until условие;
```

Работа оператора:

 выполняются инструкции, следующие после ключевого слова Repeat;

 вычисляется значение условия. Если условие ложно, то повторно выполняются инструкции цикла. Если условие истинно, то выполнение цикла прекращается.

Замечание. При использовании цикла с постусловием нет необходимости использовать операторные скобки.

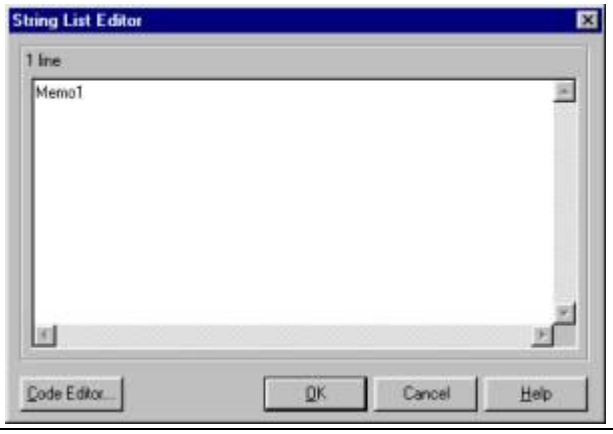
Пример. Вычислить сумму бесконечного ряда $y = \sum_{1}^{\infty} \frac{k+0.3}{3k^2+5}$ с заданной точностью Eps, (т.е. вычислить сумму всех членов последовательности $\frac{k+0.3}{3k^2+5}$, не меньших заданного числа Eps).

Программа, реализующая данный алгоритм, будет иметь вид:

```
var
 k: integer;
 e,y,s: real;
begin
e:=strtofloat(edit1.Text);
k:=1;
s:=0;
repeat
 y:=(k+0.3)/(3*k*k+5);   {Вычисление очередного члена последовательности}
 s:=s+y;                 {Вычисление суммы последовательности}
 k:=k+1                  {Вычисление количества членов последовательности}
until y<=e;
{Вывод полученной суммы}
edit2.Text:=floattostr(s)+'   '+inttostr(k-1);
end;
```

Краткая характеристика некоторых компонентов, наиболее часто используемых в примерах при построении циклических конструкций

Компонент **Memo** (многострочное окно редактирования)  . Используется для ввода, отображения и редактирования многострочных текстов. Относится к группе Standard.

Alignment	Задает режим выравнивания текста внутри Memo
AutoSize	Задает необходимость изменения размера компонента при изменении размера шрифта
BorderStyle	Задается стиль обрамления Memo
Color	Задает цвет, которым изображается элемент Memo на экране
Lines	Определяет текст, который будет выведен построчно в окне Memo при запуске программы. Текст задается в окне String List Editor 
MaxLength	Позволяет ограничивать число вводимых пользователем символов
ScrollBars	Задает наличие полос прокрутки
Text	Используется, чтобы получить текст компонента Memo как одну строку. Значение этого свойства не отображается в окне Object Inspector, к нему можно обратиться только во время выполнения программы

Пример использования в программе

1. Заполнение Memo с использованием свойства Text на примере табулирования функции.

```
var
a,b,h,x,y: real;
n, i: integer;
begin
a:=strtofloat(edit1.Text);      {Начало отезка}
b:=strtofloat(edit2.Text);      {Конец отрезка}
h:=strtofloat(edit3.Text);      {Шаг табуляции}
n:=ceil((b-a)/h)+1;             {Расчет количества точек}
x:=a;
{Формирования заголовка таблицы табуляции, которая выводится в Memo.
Добавление в конец строки chr(13)+chr(10) - кодов 13 - возврат каретки
 и 10 - переход на одну строку; позволяет разбить таблицу на строки}
```

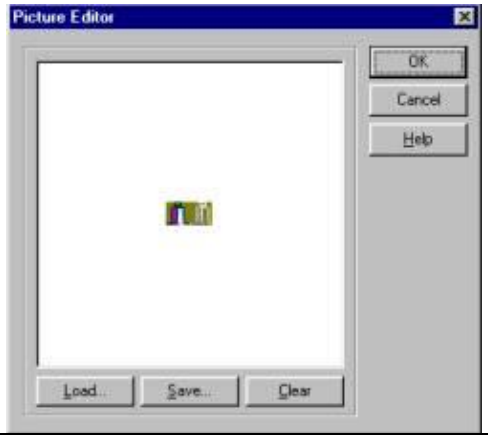
```
memo1.Text:='    x      |     y  '+chr(13)+chr(10);
memo1.Text:=memo1.Text+'---------------------' +chr(13)+chr(10);
for i:=1 to n do
begin
  y:=sin(x);                        {Вычисление следующего значения функции}
{Формирование очередной строки таблицы с результатами табулирования}
  memo1.Text:=memo1.Text+floattostrf(x,fffixed,5,2)+'  |  '
              + floattostrf(y,fffixed,5,2)+chr(13)+chr(10);
  x:=x+h;                           {Вычисление следующей точки табуляции}
end;
```

2. Тот же пример, но Memo заполняется с использованием свойства Lines. Метод Add, примененный к Lines, позволяет добавить строку в Memo. Оператор memo1.Lienes[0]:=’ x | y ’; задает первую строку в Memo .

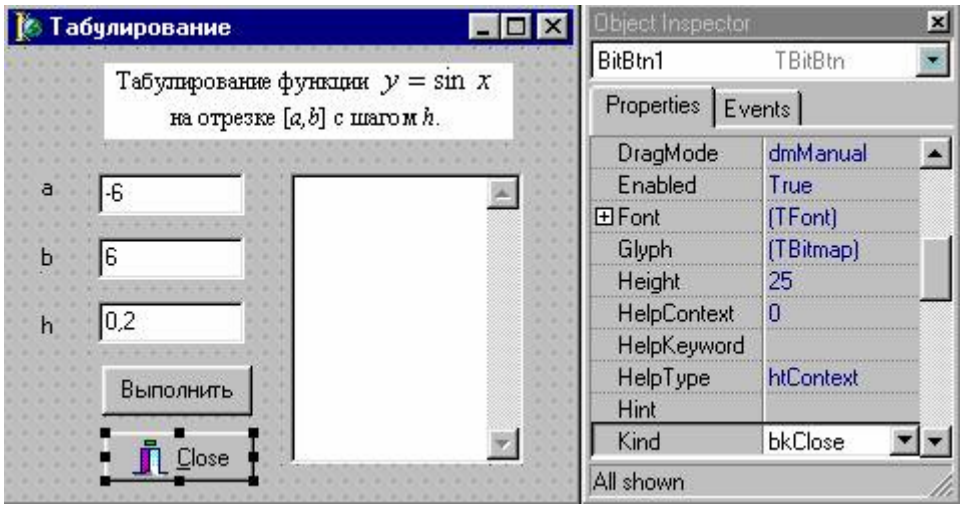
```
{Табулирование функции на отрезке}
var
a,b,h,x,y: real;
begin
a:=strtofloat(edit1.Text);      {Начало отезка}
b:=strtofloat(edit2.Text);      {Конец отрезка}
h:=strtofloat(edit3.Text);      {Шаг табуляции}
x:=a;
memo1.Clear;
{Формирования заголовка таблицы табуляции, которая выводится в Memo.}
memo1.Lines[0]:='    x      |     y  ';
memo1.Lines.Add('---------------------');
while x<=b do
begin
  y:=sin(x);                        {Вычисление следующего значения функции}
{Формирование очередной строки Memo с результатами табулирования}
  memo1.Lines.Add(floattostrf(x,fffixed,5,2)+'  |  '
              + floattostrf(y,fffixed,5,2));
  x:=x+h;                           {Вычисление следующей точки табуляции}
end;
end;
```

Компонент **BitBtn** (кнопка с изображением)  . Используется как обычная кнопка для инициирования некоторого события, но может содержать графическое изображение. Относится к группе Additional.

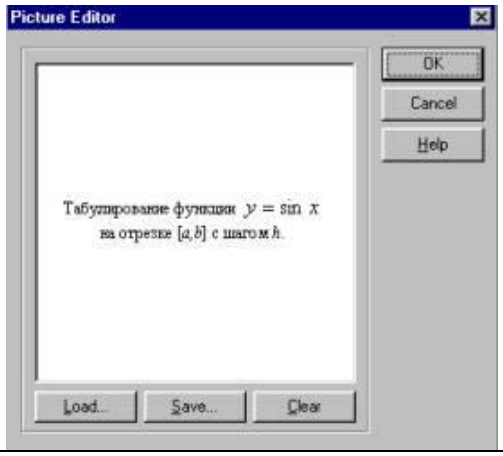
Caption	Задает текст надписи на кнопке
Kind	Предлагает на выбор десять предопределенных типов кнопок

Glith	Предлагает создание собственного типа кнопки с помощью окна Picture Editor 

Пример использования в программе

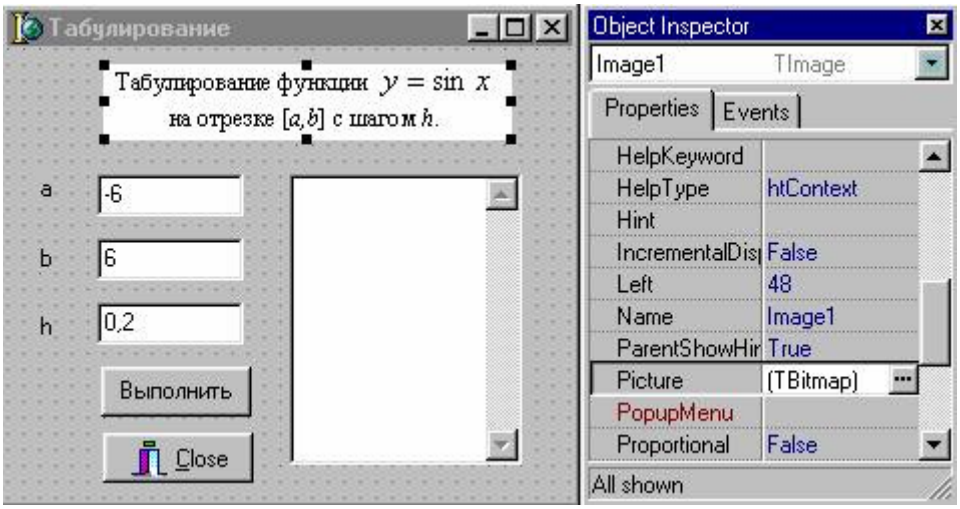


Компонент **Image** (графический образ)  . Позволяет отображать рисунок, загруженный из графического файла. Относится к группе Additional.

Align	Задает режим выравнивания расположения объекта Image внутри формы
Picture	Задает имя файла с рисунком с помощью окна Picture Editor 

Streth	Задает разрешение на автоматическое масштабирование рисунка относительно Image

Пример использования в программе



Процедуры и функции пользователя

Если в программе возникает необходимость частого обращения к некоторой группе операторов, то рационально сгруппировать такие операторы в самостоятельный блок, к которому можно обращаться, указывая его имя. Такие самостоятельные программные блоки называются *подпрограммами пользователя*.

Передача данных из главной программы в подпрограмму и возврат результата выполнения осуществляются с помощью параметров. Различают *формальные параметры*, определенные в заголовке подпрограммы, и *фактические параметры* – выражения, задающие конкретные значения при обращении к подпрограмме.

Реализуют подпрограммы в виде процедур или функций, которые определяются в разделе описания функций и процедур.

Главное отличие функции от процедуры заключается в том, что результат работы функции – единственное значение, а результат работы процедуры – одно значение или несколько, или ни одного (например, процедура, которая распечатывает полученный ранее результат в виде красивых таблиц). Кроме того, обращение к функции является разновидностью операнда, а вызов процедуры – разновидностью оператора.

Процедуры

Структура процедуры

Procedure <имя процедуры> (список формальных параметров);

<Раздел описаний программного кода процедуры>
Begin
  <Операторы тела процедуры>
End;

Список формальных параметров может включать:

 параметры-значения и входные параметры, значения которых должны быть установлены до начала работы данной процедуры (определяют исходные данные для работы процедуры);

 параметры-переменные, или выходные параметры, получающие свое конкретное значение в результате работы процедуры (определяют выходные данные процедуры). Перед перечислением параметров-переменных в списке формальных параметров должно стоять ключевое слово var.

 каждый параметр имеет имя и тип, указанный через «:». Параметры отделяются друг от друга «;»

Обращение к процедуре осуществляется в основной программе путем задания ее имени и списка фактических параметров того же типа и количества, что и формальные.

Функции

Это подпрограммы, в результате которых вычисляется только одно значение, которое присваивается имени функции.

Структура функции

Function <имя функции>(список формальных параметров):<тип результата>;
<Раздел описаний программного кода функции>
Begin
  <Операторы тела функции>
  <Имя функции>:=<Вычисленное значение>
End;

Список формальных параметров включает имена переменных со своими типами, с помощью которых определяются исходные данные, необходимые для работы функции.

В разделе операторов должен присутствовать, по крайней мере, один оператор, присваивающий вычисленное значение имени функции. В точку вызова возвращается результат последнего такого присваивания. Если такого оператора нет, то значение функции не определено.

Обращение к функции осуществляется в основной программе путем задания ее имени и списка фактических параметров того же типа и количества, что и формальные.

Рекурсивные процедуры и функции

Слово «рекурсия» происходит от латинского слова «*recursio*» - возвращение.

Если процедура (или функция) обращается сама к себе как к процедуре (*или функции) непосредственно или через цепочку подпрограмм, то это называется* рекурсией.

Для того чтобы подобного типа программы не зацикливались (что очень реально), в первую очередь необходимо обеспечить выход из рекурсии.

Пример. Вычислить факториал заданного числа с помощью подпрограммы-функции, использующей рекурсию.

Факториал заданного числа – это произведение натуральных чисел от 1 до заданного числа.

Например, *n*-факториал – это

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$$

```
var
 n: integer;
 F: longint;
 {Описание функции вычисления факториала
   n - формальный параметр-значение типа integer,
   результат выполнения функции типа longint}
function Fakt(n: integer): longint;
  begin

    if n=1 then Fakt:=1          {Проверка условия завершения рекурсии}
           else Fakt:=n*Fakt(n-1){Рекурсивное вычисление n!}
  end;
 {Начало главной программы}
begin
  n:=strtoint(edit1.Text);
  {Вызов функции для фактического параметра n}
  F:=Fakt(n);
  edit2.Text:=inttostr(F)
end;
```

Практическая часть

1. Табулирование функции

Вариант задания	Функция	Границы отрезка и шаг
1	$y = 3\sin\sqrt{x} + 0{,}35x - 3{,}8$	[2,3] 0,1
2	$y = 0{,}25x^3 + x - 1{,}2502$	[0,2] 0,2
3	$y = x + \sqrt{x} + \sqrt[3]{x} - 2{,}5$	[0,4;1] 0,05
4	$y = \sin(\ln x) - \cos(\ln x) + 2\ln x$	[1;3] 0,2
5	$y = \cos\frac{2}{x} - 2\sin\frac{1}{x} + \frac{1}{x}$	[1;2] 0,1
6	$y = 3x - 4\ln x - 5$	[2;4] 0,2
7	$y = \sqrt{1-x} - \cos\sqrt{1-x}$	[0;1] 0,1
8	$y = tgx - \frac{1}{3}tg^3x + \frac{1}{5}tg^5 - \frac{1}{3}$	[0;0,8] 0,05
9	$y = 0{,}1x^2 - x\ln x$	[1;2] 0,1
10	$y = x - 1/(3 + \sin 3{,}6x)$	[0;0,85] 0,05
11	$y = x + \cos(x^{0{,}52} + 2)$	[0,5;1] 0,05
12	$y = 3\ln^2 x + 6\ln x - 5$	[1;3] 0,2
13	$y = 3x - 14 + e^x - e^{-x}$	[1;3] 0,2
14	$y = \sqrt{1-x} - tgx$	[0;1] 0,1
15	$y = \cos x - e^{-\frac{x^2}{2}} + x - 1$	[1;2] 0,1

2. Вычисление суммы ряда

По заданной формуле члена последовательности с номером *k* составить две программы вычисления суммы:

- первых *n* членов последовательности (*k=1,2,3..,n*);
- всех членов последовательности, не меньших заданного числа *e*.

Вариант задания	Член последовательности	Вариант задания	Член последовательности	Вариант задания	Член последовательности
1	$\frac{1}{(2k-1)(2k+1)}$	6	$\frac{k+4}{(k^2+2)(k+8)}$	11	$\frac{k+2}{k^2+4}$
2	$\frac{k}{(k+1)^2+3}$	7	$\frac{3(k+1)}{7k^2+9}$	12	$\frac{2k}{3k+k^2+4}$
3	$\frac{2k}{(k^2+1)(k+2)}$	8	$\frac{1}{\sqrt{k}+15}$	13	$\frac{\sqrt{k}+1}{2\sqrt{k^2+1}}$
4	$\frac{k+1}{k-\sqrt{k+2}}$	9	$\frac{1}{k^2+3k+4}$	14	$\frac{4k}{5k^2+8k-1}$
5	$\frac{k}{(3k^2+7)(k^2+1)}$	10	$\frac{k+1}{k(k+2)(k+3)}$	15	$\frac{2k+1}{(2k^2+3)k}$

Контрольные вопросы

1. Каково назначение операторов цикла?
2. Какие операторы цикла вы знаете?
3. Когда используется цикл с параметром For?
4. Когда используются циклы с предусловием или постусловием?
5. Как изменяется управляющая переменная в цикле For?
6. Когда в цикле применяется составной оператор?
7. Когда выполняется тело оператора цикла с предусловием While?
8. Когда выполняется тело оператора цикла Repeat с постусловием?
9. Чем различаются циклы While и Repeat?
10. Для чего служит управляющий элемент Image?
11. Какие форматы поддерживает Image?
12. Каково назначение свойства Stretch?
13. Для чего используется объект Memo?
14. Как используется свойство Text компоненты Memo?
15. Для чего необходимы коды ASCII-таблицы 13 и 10?
16. Для чего используется свойство Lines компоненты Memo?
17. Что такое подпрограмма?
18. Опишите различия между функцией и процедурой.
19. Что такое формальные параметры?
20. Что такое фактические параметры?

Содержание отчета

1. Название, цели лабораторной работы.
2. Формулировка практической части работы.
3. Фрагменты программного кода.
4. Выводы по работе.
5. Ответы на контрольные вопросы.

Лабораторная работа № 5

РАБОТА С МАССИВАМИ

Цель работы: изучить массивы Delphi.

Теоретическая часть

Многие задачи, которые решаются с помощью компьютера, связаны с обработкой больших объемов информации, представляющей совокупность данных, объединенных единым математическим содержанием или связанных между собой по смыслу. Такие данные удобно представлять в виде линейных или прямоугольных таблиц.

В линейной таблице каждому ее элементу соответствует порядковый номер. Для элемента прямоугольной таблицы должны быть указаны два номера: номер по вертикали (номер строки) и номер по горизонтали (номер столбца). В высшей математике табличные величины называют соответственно векторами и матрицами.

В программе для представления таких данных используются массивы. *Массив* – это упорядоченная совокупность однотипных данных, с каждым из которых связан упорядоченный набор целых чисел, называемых *индексами.*

Массив характеризуется именем, размерностью и размером.

Имя массива образуется по общему правилу образования имен, т. е. представляет собой идентификатор, например `A, Bl, C8` и т. д. Однако оно не должно совпадать с именем ни одной простой переменной, используемой в той же программе.

Работа с массивом сводится к действиям над его элементами. Для того чтобы указать, какой элемент в данный момент используется, достаточно задать его порядковый номер, который приписывается к имени соответствующего массива. Таким образом, элементы массива обозначаются переменной с индексами. *Запись переменной с индексами* состоит из имени массива и следующего за ним в квадратных скобках списка индексов, например `A[1], A[I], B1[K], C8[I, J], C8[2, 1]`

Индексы определяют положение элемента в массиве. *Число индексов определяет размерность массива*, т.е. форму его компоновки: одномерный, двумерный и т. д. Одномерный массив соответствует линейной таб-

лице. Его элемент обозначается переменной с одним индексом: `A[1]`, `A[I]` – соответственно первый и *i*-й элементы одномерного массива A.

Двумерный массив описывает в программе прямоугольную таблицу. Его элементы обозначаются переменной с двумя индексами: `C8[I, J]`, `C8[2, 1]`, где первый индекс обозначает номер строки, а второй – номер столбца.

Таким образом, для обращения к конкретному элементу массива необходимо указать имя массива и значения индексов.

Для записи элементов массива в память компьютера нужно выделить для их хранения необходимое количество (массив) ячеек памяти, которое определяется размером массива. *Размеры массива* задаются границами изменения индексов по каждому измерению (минимальное и максимальное значение индекса).

По умолчанию применяется так называемая нумерация с нулевой базой, т.е. элементы массива нумеруются, начиная с 0.

В программе для каждого массива должны быть указаны его параметры: *имя, размерность и размеры*. Эта информация нужна для резервирования необходимого объема памяти для хранения числовых значений; она задается специальным *оператором описания массивов.*

Описание статического массива определяет имя, размер массива и тип данных, которые в нем хранятся. Формат описания в разделе переменных:

```
Var
    <имя_массива>: array <[тип_индекса]> of <тип_данных>;
```

Чаще всего в качестве типа индекса используется интервальный целый тип (тип-диапазон). Интервальный тип задается начальным и конечным значениями, которые разделяются двумя точками.

Например

```
Var
 A : array [1..10] of real;
```

Описывается одномерный массив вещественных чисел `A`, который максимально может состоять из 10 элементов. Нижняя граница индекса равна 1, верхняя – 10.

Начиная с версии Delphi 4, можно использовать также и динамические массивы, когда количество элементов может меняться по ходу выполнения программы.

Динамические массивы отличаются от обычных статических тем, что для них не объявляется заранее длина – число элементов. Объявление такого массива содержит только имя и тип элементов.

```
Var  <имя_массива>: array of <тип_данных>.
```

При объявлении динамического массива место под него не отводится. Прежде чем использовать такой массив, надо задать в программе его размер процедурой `SetLength`. Параметры данной процедуры – количество элементов по каждой размерности. Например, `SetLength(10,20)` – для двумерного массива.

Линейные (одномерные) массивы

Линейный (одномерный) массив – это просто список элементов данных.

Примеры описания одномерных массивов:

```
var B : array [0..5] of real
R : array [1..34] of char;
N : array ['A'..'Z'] of integer;
M : array of integer; {динамический массив}
```

Для доступа к элементу массива следует указать имя массива с последующим числом (индексом), заключенным в квадратные скобки.

Элементы массива можно использовать в любом выражении точно так же, как и значение константы или переменной.

Например

```
a[0]=11.2; a[1]=10.2; a[3]=22.1; a[4]=1.1;
Y = a[0] * 2 – a[1];
```

Основные действия по обработке линейного массива.

Сортировка массива

Существует множество алгоритмов для сортировки массивов. Ниже рассмотрены два из них: *сортировка выбором* и *методом пузырька.*

Сортировка выбором

Суть этого метода очень проста и может быть описана так:

1. В последовательности из *n* элементов выбирается наименьший (наибольший) элемент.

2. Меняется местом с первым.

3. Далее процесс повторяется с оставшимися *n*-1 элементами, затем с оставшимися *n*-2 элементами и так далее, до тех пор, пока не останется один самый большой (маленький) элемент.

Для реализации этого алгоритма необходимо использовать два вложенных цикла с параметром `For`. Внешний цикл (по `i`) предназначен для последовательного фиксирования элементов массива, внутренний (по `j`) - осуществляет поиск минимального (максимального) элемента и его позиции в неотсортированной части массива. После выхода из внутреннего цикла следует перестановка элементов. Последний элемент во внешнем цикле не рассматривается: он сам встанет на свое место.

Сортировка методом пузырька

Метод основан на сравнении соседних элементов. «Неправильно» расположенные по отношению друг к другу элементы меняются местами. Во вложенных циклах поочередно фиксируется пара соседних элементов массива. В результате первого прохода элемент с минимальным значением оказывается в первой позиции массива (всплывает).

Уплотнение массива

Уплотнение массива – это удаление из него элементов, отвечающих тем или иным условиям. Образующиеся пустоты заполняются за счет сдвига всех оставшихся элементов. Так как массив укорачивается, при его обработке необходимо использовать не цикл с параметром, а цикл с условием.

Вставка элемента в массив

Вставка элемента в массив – задача обратная предыдущей. Прием используется тот же – смещение группы элементов на одну позицию, только при уплотнении сдвиг производится влево, при вставке – вправо. При вставке возникает проблема: Что делать с последними элементами? Если в дальнейшей работе с массивом участвуют только заявленные элементы, то «хвост» придется вытеснить, последние значения при этом будут утрачены. Иначе, нужно создавать дополнительный массив, размерность которого будет больше исходного на количество вставленных элементов. Выбор типа цикла для работы с массивом зависит от конкретного случая.

Кольцевой сдвиг элементов массива

Кольцевой сдвиг – это смещение элементов массива вправо либо влево, причем вытесненные элементы занимают освободившиеся в результате смещения позиции в противоположном конце массива так, словно массив представляет собой кольцо (первый и последний элементы смыкаются). Порядок следования элементов при этом сохраняется.

Многомерные массивы

Часто бывает необходимо представить таблицы данных в формате строк и столбцов. Для этого используют многомерные массивы.

Доступ к элементам, например, двухмерного массива (матрицы) осуществляется с помощью двух индексов. Первый индекс отвечает за строку, второй – за столбец: `a[1,3]`, `a[0,4]` и т.д., например:

$$\begin{Vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{Vmatrix}$$

Формат описания двумерного массива в разделе объявления переменных:

```
Var
<имя_массива>:array<[тип_индекса,тип_индекса]>of
<тип_данных>
```

Например

```
Var
A : array [1..10, 1..10] of real;
```

Описывается двухмерный массив вещественных чисел `A`, который максимально может состоять из 10 строк и 10 столбцов. Нижняя граница индексов по обоим измерениям равна 1, верхняя – 10.

Краткая характеристика компонента StringGrid

В среде Delphi для ввода и вывода массивов используется компонент `StringGrid`, поэтому работа с массивами в Delphi сильно отличается от работы в Pascal.

Компонент `StringGrid`  группы `Additional` представляет собой таблицу, содержащую строки. Таблица может иметь полосы прокрутки, причем заданное число первых строк и столбцов может быть фиксированным и не подвергаться прокрутке. Таким образом, можно задать заголовки столбцов и строк, постоянно присутствующих в окне компонента. Каждой ячейке таблицы может быть поставлен в соответствие некоторый объект.

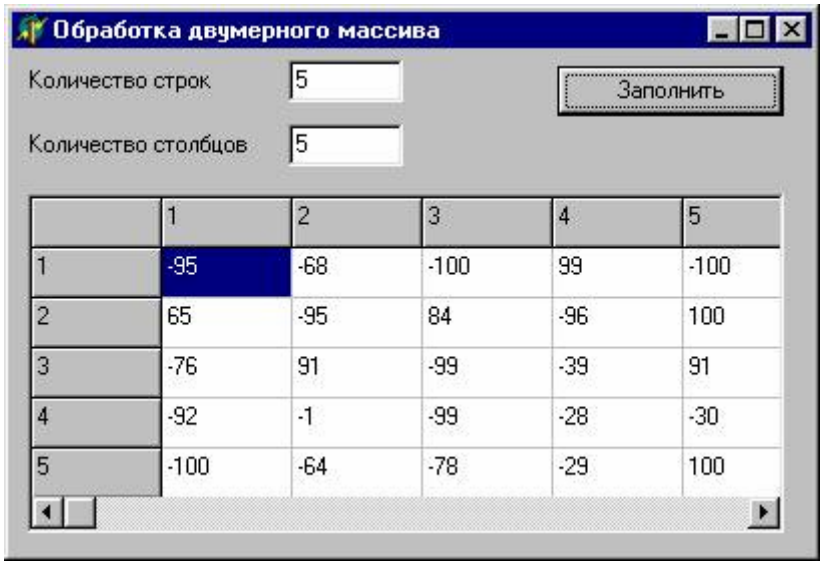
Свойства компонента StringGrid

`Cells`	В этом свойстве хранятся все элементы таблицы. Имеет тип `String`

FixedCols	Задает фиксированное количество столбцов в таблице
FixedRows	Задает фиксированное количество строк в таблице
Options → GoEditing	По умолчанию данные в таблицу вводить нельзя. Чтобы снять этот запрет в этом свойстве, надо задать True
ColCount	Задает общее количество столбцов таблицы
RowCount	Задает общее количество строк таблицы

Пример использования в программе компонента StringGrid для вывода двумерного массива

Окно формы приложения



Программный код приложения

```
procedure TForm1.Button1Click(Sender: TObject);
var
i,j,n,m : integer;
a : array [0..30,0..30] of integer;     {Описание массива}
begin
Randomize;      {Инициализация датчика случайных чисел}
n:=strtoint(Edit1.Text);    {Количество строк}
m:=strtoint(Edit2.Text);    {Количество столбцов}
for i:=0 to n-1 do
 for j:=0 to m-1 do
   a[i,j]:=Round(Sin(Random(100))*100); {Заполнение массива случайными
                                         числами из диапазона [-100,100]}
StringGrid1.RowCount:=n+1;     {Количество строк в заголовке таблицы}
StringGrid1.ColCount:=m+1;     {Количество столбцов в заголовке таблицы}
{Формирование заголовков строк и стобцов}
with StringGrid1 do
{Оператор with в данном случае позволяет не использовать имя объекта
при обращении к свойствам этого объекта}
```

```
begin
  i:=0;                          {столбец 0}
  for j:=1 to RowCount  do       {вывод номеров строк}
   Cells[i,j]:=IntToStr(j);
  j:=0;                          {строка 0}
  for i:=1 to ColCount  do       {вывод номеров столбцов}
   Cells[i,j]:=IntToStr(i);
end;
{Вывод элементов массива в таблицу}
with StringGrid1 do
  for i:=1 to n  do
   for j:=1 to m do
    Cells[j,i]:=IntToStr(a[i-1,j-1]);
end;

end.
```

Краткая характеристика компоненты GroupBox

Панель `GroupBox`  группы `Standard` – это контейнер с рамкой и надписью, объединяющий группу связанных органов управления, таких как переключатели `RadioButton` , флажки `CheckBox` и т.д.

Свойства панели `GroupBox`

`Caption`	Задает надпись для рамки, выделяющей группу объединенных компонент

Если компоненты, размещаемые на панели, оказываются под панелью и не отображаются, то следует выделить панель и выбрать в контекстном меню команду Control → Send to Back (Порядок → На задний план).

Практическая часть

1. Выполните задание, соответствующее вашему варианту

Номер варианта	Задание
1	1. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-10,10]. Найти сумму элементов, имеющих нечетное значение. 2. Вывести индексы тех элементов, значения которых больше заданного числа *A*. 3. Определить, есть ли в данном массиве положительные элементы, кратные заданному числу *K*.

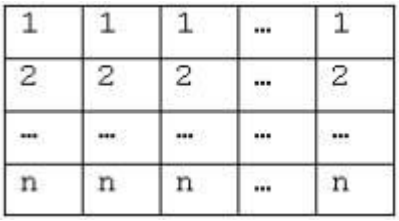
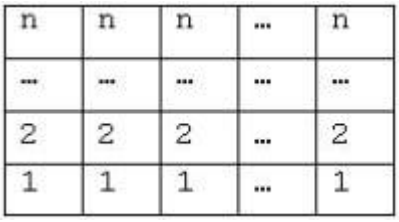
2	1. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-15,15]. Найти произведение элементов, имеющих четное значение. 2. Вывести индексы тех элементов, значения которых по модулю меньше заданного числа *A*. 3. Определить, есть ли в данном массиве положительные элементы, делящиеся на заданное число *k* с остатком 2.
3	1. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-10,20]. Найти сумму элементов, имеющих нечетные индексы. 2. Подсчитать количество элементов массива, значения которых больше заданного числа *A* и кратных 5. 3. Найти номер первого отрицательного элемента, делящегося на 5 с остатком 2.
4	1. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-1000,1000]. Найти сумму четных элементов. 2. Подсчитать количество элементов массива, значения которых состоят из двух цифр. 3. Найти номер первого положительного элемента, делящегося на 5 с остатком 2.
5	1. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-100,100]. Найти сумму положительных элементов, значения которых меньше 10. 2. Вывести индексы тех элементов, значения которых кратны 3 и 5. 3. Определить, есть ли пара соседних элементов с суммой, равной заданному числу.
6	1. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-1000,1000]. Найти сумму отрицательных элементов, значения которых кратны10.

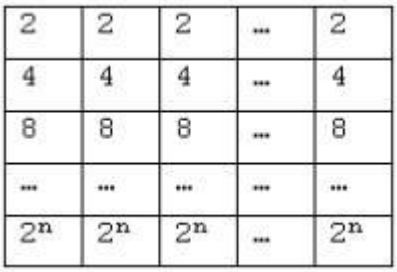
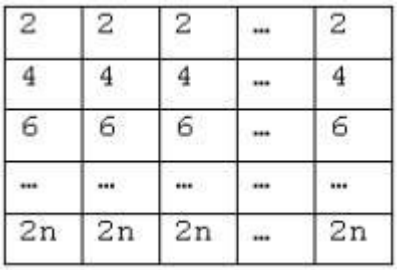
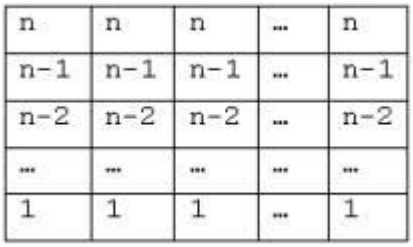
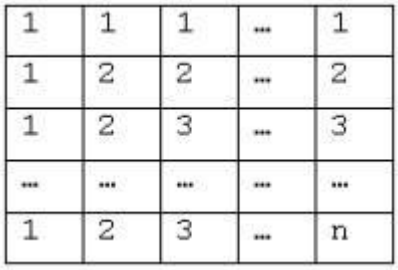
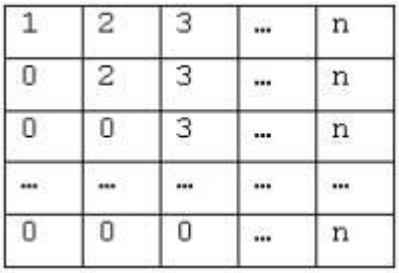
6	2. Вывести индексы тех элементов, значения которых кратны 5 и 10. 3. Определить, есть ли пара соседних элементов с произведением, равным заданному числу.
7	1. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-1000,1000]. Найти сумму четных отрицательных элементов. 2. Вывести индексы тех элементов, значения которых кратны 3 и 6. 3. Определить, есть ли пара соседних элементов с суммой, равной заданному числу.
8	1. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-20,40]. Найти удвоенную сумму положительных элементов. 2. Вывести индексы тех элементов, значения которых больше значения предыдущего элемента (начиная со второго). 3. Определить, есть ли две пары соседних элементов с одинаковыми знаками.
9	1. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-40,40]. Найти сумму элементов, значения которых по модулю меньше 10. 2. Вывести индексы тех элементов, значения которых больше значения последующего элемента. 3. Определить, есть ли две пары соседних элементов с разными знаками.
10	1. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-100,200]. Найти сумму отрицательных элементов. 2. Найти количество тех элементов, значения которых положительны и не превосходят заданного числа A. 3. Найти номер последней пары соседних элементов с разными знаками.

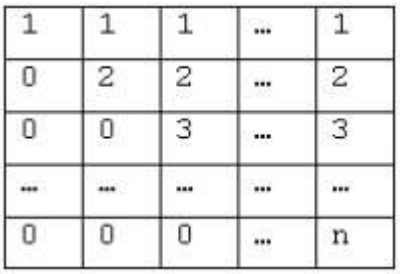
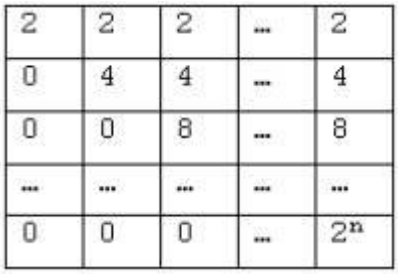
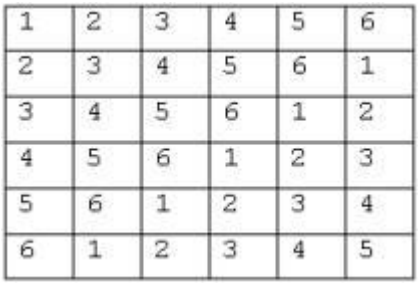
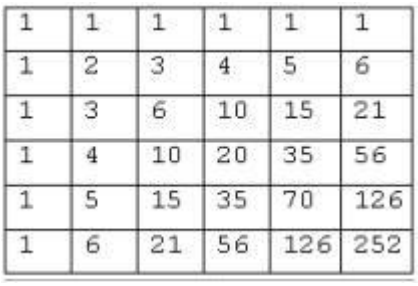
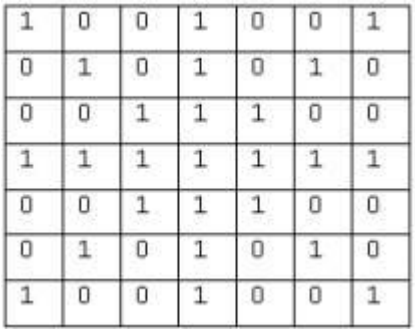
11	1. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-100,200]. Найти сумму четных элементов, значения которых больше заданного числа. 2. Найти количество тех элементов, значения которых отрицательны и по модулю не превосходят заданного числа A. 3. Найти номер первой пары соседних элементов с разными знаками.
12	1. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-10,20]. Найти произведение четных элементов, значения которых по модулю меньше 5. 2. Найти количество тех элементов, значения которых нечетны и по модулю превосходят заданное число A. 3. Найти номер последней пары соседних элементов, сумма которых больше заданного числа.
13	1. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-10,10]. Найти сумму элементов, значения которых кратны 3 и 5. 2. Найти количество тех элементов, значения которых положительны и по модулю не превосходят заданное число A. 3. Найти номер первой пары соседних элементов, сумма которых меньше заданного числа.
14	1. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-100,100]. Найти сумму элементов, значения которых состоят из одной цифры. 2. Найти количество тех элементов, значения которых положительны и кратны 3 и 5. 3. Найти номер последней пары соседних элементов с одинаковыми знаками, произведение которых меньше заданного числа.

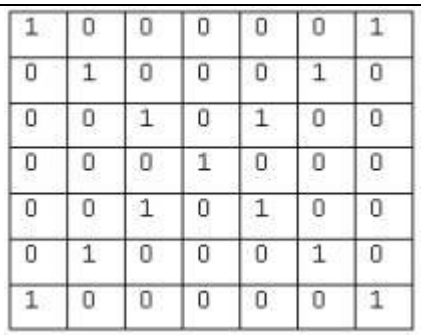
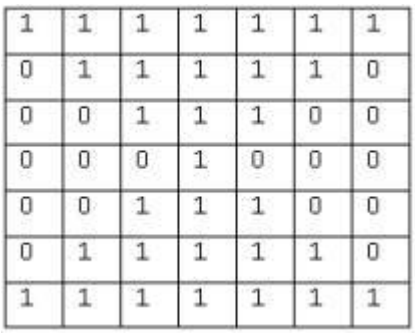
15	1. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-1000,1000]. Найти сумму положительных элементов, значения которых состоят из двух цифр. 2. Найти количество тех элементов, значения которых по модулю превосходят 100 и кратны 5 и 10. 3. Найти номер первой пары соседних элементов с разными знаками, сумма которых меньше заданного числа.

2. Выполните задание, соответствующее вашему варианту

Номер варианта	**Задание**
1	1. Дан двумерный массив размерами n*m, заполненный случайными числами. Определить, есть ли в данном массиве строка, в которой имеется два элемента массива, имеющие наибольшие значения. 2. Заполнить массив n*n по правилу: 
2	1. Дан двумерный массив размерами n*m, заполненный случайными числами. Определить в нем разность между средним арифметическим максимального и средним арифметическим минимального элементов. 2. Заполнить массив n*n по правилу: 
3	1. Дан двумерный массив размерами n*m, заполненный случайными числами. Найти строку с минимальной суммой и в ней – максимальный элемент. 2. Заполнить массив n*n по правилу: 

4	1. Дан двумерный массив размерами n*m, заполненный случайными числами. Определить, есть ли в данном массиве столбец, в котором имеются одинаковые элементы. Если имеются, вывести их номера. 2. Заполнить массив n*n по правилу: 
5	1. Дан двумерный массив размерами n*m, заполненный случайными числами. Определить, есть ли в данном массиве строка, в которой ровно два отрицательных элемента. 2. Заполнить массив n*n по правилу: 
6	1. Дан двумерный массив размерами n*m, заполненный случайными числами. Определить, есть ли в данном массиве строка, содержащая больше положительных элементов, чем отрицательных. 2. Заполнить массив n*n по правилу: 
7	1. Дан двумерный массив размерами n*m, заполненный случайными числами. Определить в нем столбец с максимальной суммой и в нем – минимальный по величине элемент. 2. Заполнить массив n*n по правилу: 
8	1. Дан двумерный массив размерами n*m, заполненный случайными числами. Определить, есть ли в данном массиве столбец, в котором равное количество положительных и отрицательных элементов. 2. Заполнить массив n*n по правилу: 

9	1. Дан двумерный массив размерами `n*m`, заполненный случайными числами. Определить номера строк массива, содержащих только положительные элементы и найти среди них наибольший. 2. Заполнить массив `n*n` по правилу: 
10	1. Дан двумерный массив размерами `n*m`, заполненный случайными числами. Найти среднеарифметическое элементов, принадлежащих первой и последней строке, первому и последнему столбцу. 2. Заполнить массив `n*n` по правилу: 
11	1. Дан двумерный массив размерами `n*m`, заполненный случайными числами. Изменить массив путем деления всех его элементов на максимальный по модулю элемент. 2. Заполнить массив 6*6 по правилу: 
12	1. Дан двумерный массив размерами `n*m`, заполненный случайными числами. Найти сумму его элементов, расположенных между максимальным и минимальным элементами (включая оба этих числа). 2. Заполнить массив 6*6 по правилу: 
13	1. Дан двумерный массив размером `n*m`, заполненный случайными натуральными числами в диапазоне от 1 до 100. Определить, сколько чисел в массиве равны произведению своих индексов `i*j`. 2. Заполнить массив 7*7 по правилу: 

14	1. Дан двумерный массив размерами n*m, заполненный случайными числами. Определить в нем строку с максимальной суммой элементов и столбец с минимальной. Задачу решить за один проход. 2. Заполнить массив 7*7 по правилу:	
15	1. Дан двумерный массив размерами n*m, заполненный случайными числами. Найти в каждой строке массива максимальный и минимальный элементы и поменять их с первым и последним элементом соответственно. 2. Заполнить массив 7*7 по правилу:	

Контрольные вопросы

1. Что такое массив данных?
2. Как и в каком разделе программного кода описываются массивы?
3. Как определить местоположение элемента в массиве?
4. Что такое индекс? Каким требованиям он должен удовлетворять?
5. Как осуществляется доступ к элементам массива?
6. Что такое динамический массив?
7. Опишите различия статических и динамических массивов. Каков порядок описания и применения динамических массивов?
8. Когда и для чего используется процедура `SetLength`?
9. Каково назначение компоненты `GroupBox`? Как задать надпись в этом компоненте?
10. Как в среде Delphi осуществляется ввод и вывод элементов массива?
11. Что следует сделать, чтобы в компоненте `StringGrid` можно было вводить данные?
12. Как используется свойство `Cells` компоненты `StringGrid`?
13. Для чего используются свойства `FixedRows` и `FixedCols` в `StringGrid`?

Содержание отчета

1. Название, цели лабораторной работы.
2. Формулировка практической части работы.
3. Фрагменты программного кода.
4. Выводы по работе.
5. Ответы на контрольные вопросы.

Лабораторная работа № 6

РАБОТА СО СТРОКАМИ

Цель работы: научиться работать со строками. Процедуры обработки строковых данных.

Теоретическая часть

В реальных задачах часто встречаются объекты символьного типа – строки. Строка в Pascal трактуется как последовательность символов. В состав строки могут входить буквы латинского алфавита и кириллицы, цифры, всевозможные знаки, скобки, пробел и др. Каждый символ строковой величины занимает 1 байт памяти (десятичный код от 0 до 255, зафиксированный в кодовой таблице ASCII).

Фрагмент таблицы символьной кодировки ASCII

32 -	48 - 0	64 - @	80 - P	96 - `	112 - p
33 - !	49 - 1	65 - A	81 - Q	97 - a	113 - q
34 - "	50 - 2	66 - B	82 - R	98 - b	114 - r
35 - #	51 - 3	67 - C	83 - S	99 - c	115 - s
36 - $	52 - 4	68 - D	84 - T	100 - d	116 - t
37 - %	53 - 5	69 - E	85 - U	101 - e	117 - u
38 - &	54 - 6	70 - F	86 - V	102 - f	118 - v
39 - '	55 - 7	71 - G	87 - W	103 - g	119 - w
40 - (	56 - 8	72 - H	88 - X	104 - h	120 - x
41 -)	57 - 9	73 - I	89 - Y	105 - i	121 - y
42 - *	58 - :	74 - J	90 - Z	106 - j	122 - z
43 - +	59 - ;	75 - K	91 - [	107 - k	123 - {
44 - ,	60 - <	76 - L	92 - \	108 - l	124 - |
45 - -	61 - =	77 - M	93 -]	109 - m	125 - }
46 - .	62 - >	78 - N	94 - ^	110 - n	126 - ~
47 - /	63 - ?	79 - O	95 - _	111 - o	127 - □

Количество символов в строке называется ее *длиной*. Длина строки может динамически изменяться от 0 до 255. Пустая строка имеет нулевую длину.

Строковая константа – последовательность символов, заключенных в апострофы. Например:

```
'Это строковая константа', '123'.
```

Таким образом, при использовании в выражениях строка обязательно заключается в кавычки.

Два следующих друг за другом апострофа (`''`) обозначают пустую строку, т.е. строку с нулевой длиной.

Строковая переменная описывается в разделе описания переменных:

```
Var <имя> : string [<максимальная длина строки>]
```

Например:

```
Var
 Name: string[20];
 St: string;
```

Если максимальная длина не указана, то она принимается равной 255 (по умолчанию).

Элементы строки идентифицируются именем строки с индексом, заключенным в квадратные скобки, т.е. к любому символу в строке можно обратиться точно так же, как и к элементу одномерного массива по его номеру.

Например: `N[5], S[i], slovo[k+1]`

Индекс может быть положительной константой, переменной, выражением целого типа. Значение индекса не должно выходить за границы описания.

Для хранения и обработки *отдельных* символов используют переменные типа `char`. Значением переменной такого типа может быть любой один символ.

Например: `var a: char; ch: char;`

Операции над строками

Выражения, в которых операндами служат строковые данные, называются *строковыми*. Над строковыми данными допустимы операции сцепления и отношения.

Операция сцепления (конкатенации) (+) применяется для соединения нескольких строк в одну результирующую строку. Сцеплять можно как строковые константы, так и строковые переменные.

Например:

'Бейсик' + 'Паскаль' + 'Си' + '–' + 'языки программирования высокого уровня'.

В результате получится строка:

'Бейсик Паскаль Си – языки программирования высокого уровня'.

Длина результирующей строки не должна превышать 255 символов.

Операции отношения =, <, >, <=, >=, <> позволяют произвести сравнение двух строк, в результате чего получается логическое значение (True или False). Операции отношения имеют более низкий приоритет, чем операции сцепления. Сравнение строк производится слева направо до первого несовпадающего символа, и та строка считается больше, в которой первый несовпадающий символ имеет больший номер в таблице символьной кодировки. Если строки имеют различную длину, но в общей части символы совпадают, считается, что более короткая строка меньше, чем более длинная. Строки равны, если они полностью совпадают по длине и содержат одни и те же символы.

Например:

Выражение	Результат
'True1' < 'True2'	True
'Student' > 'STUDENT'	True
'Студент' <> ' Студент'	True
'Группа' = 'Группа'	True

Типовые задачи на обработку строк

Анализ символьных строк

Пример. Ввести строку символов. Определить:

1. Сколько цифр в строке.
2. Есть ли в заданной строке одинаковые символы.
3. Составить перечень всех гласных латинских букв.

Программный код

```
procedure TForm1.Button1Click(Sender: TObject);
var S,S1: string;
i:integer;
```

```
begin
S:=Edit1.Text;       {Введенная строка}
edit2.Text:='';
{Цикл по всем символам строки для поиска цифр}
for i:=1 to length(s) do
 if (S[i]>='0') and (S[i]<='9') then edit2.Text:=edit2.Text+S[i]+' ';
edit4.text:='Нет';
{Цикл по всем символам строки для поиска одинаковых символов}
for i:=1 to length(s) do
 if Pos(Copy(S,i,1),copy(S,i+1,length(s)-i)) <>0 then
  begin
   edit4.text:='Да';
   break;  {Немедленный выход из цикла, если найден хотя бы
            один одинаковый символ}
  end;
edit3.Text:='';
{Цикл по всем символам строки для поиска гласных латинских букв}
for i:=1 to length(S) do
 case S[i] of
  'A','E','I','O','U','Y' : edit3.Text:=edit3.Text+S[i]+' ';
  'a','e','i','o','u','y' : edit3.Text:=edit3.Text+S[i]+' ';
 end;
if edit3.Text='' then edit3.Text:='Латинских гласных нет';
end;
end.
```

Изменение строк (замена, удаление, вставка символов, лексем)

Пример. Заменить во введенной строке строчные русские буквы на прописные.

Программный код

```
procedure TForm1.Button1Click(Sender: TObject);
var S,S1: string;
i:integer;
begin
{S-заданная строка, S1-итоговая строка
после преобразования}
S:=Edit1.Text;
S1:='';
{Цикл по всем символам строки}
for i:=1 to length(S) do
 case S[i] of
{Преобразование встреченной строчной
 буквы в прописную и добавление ее в
 итоговую строку}
```

```
  'а'..'я' : S1:=S1+chr(ord(S[i])-32);
{Добавление символа в итоговую строку
 без преобразования}
   else S1:=S1+S[i];
 end;
edit2.Text:=S1;
end;
end.
```

Пример. Ввести текст в виде строки, состоящей из слов, разделенных пробелами (одним или более). Преобразовать строку так, чтобы между словами был только один пробел. Вывести сообщение о количестве удаленных пробелов.

Фрагмент кода программы

```
S:=Trim(Edit1.Text);  {Введенная строка без лидирующих и
                       завершающих пробелов}
S1:='';               {Выходная строка}
edit2.Text:='';
{Цикл по всем символам строки для поиска пробелов}
for i:=1 to length(s) do
 if (S[i]<>' ') then S1:=S1+S[i] {Переписывание не-пробелов}
 else
 {Запись в выходную строку только одного пробела,
  независимо от их количества}
   if (S[i+1]<>' ') then S1:=S1+' ';
k:=length(s)-length(s1); {Подсчет количества удаленных пробелов}
{Вывод новой строки}
edit2.text:=s1;
```

Пример. Дана строка длиной до 254 символов. Удалить все знаки «+» перед символами, не являющимися цифрами.

Фрагмент программного кода

```
S:=Edit1.Text+' ';  {К исходной строке добавляется пробел -
                     для работы с последним символом}
S1:='';             {Результирующая строка}
edit2.Text:='';
{Цикл по всем символам строки для поиска + .
 Рассматривается пара символов - текущий и следующий (за
 исключением последнего: это тот пробел, который был добавлен
 к исходной строке)}
for i:=1 to length(S)-1 do
 if (S[i]='+') then
```

```
  begin
   if (S[i+1]>='0') and (S[i+1]<='9') then
   S1:=S1+S[i] {Переписывание + перед цифрой}
  end
 else
 {Переписывание любого символа, если он не +}
  S1:=S1+s[i];
{Вывод новой строки}
edit2.text:=s1;
```

Пример. В заданной строке заменить все слова «всегда» на «часто».

Фрагмент программного кода

```
S:=Edit1.Text;      {Исходная строка}
a:=' '+edit2.text; {Слово, которое меняем}
b:=' '+edit3.text; {Слово, на которое меняем}
n:=pos(a,s);        {Номер позиции первого появления нужного слова}
while n<>0 do       {Условие выполнения цикла: слово найдено, т.е.
                     номер позиции не равен 0}
begin
 delete(s,n,length(a));{Удаление из строки найденного слова}
 insert(b,s,n);          {Вставка нового слова на место старого}
 n:=pos(a,s);       {Номер позиции следующего появления нужного слова}
end;
```

Пример. Дана строка, содержащая слова, разделенные пробелами. Преобразовать строку в массив слов (лексем).

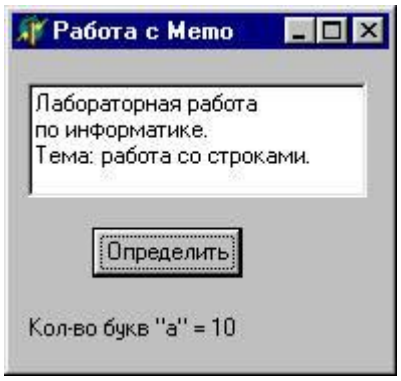
Фрагмент программного кода

```
S:=Edit1.Text;    {Исходная строка}
memo1.Text:='';
j:=0; {Задание индекса первого элемента массива}
{Цикл по всем символам строки для поиска очередного пробела}
for i:=1 to length(s) do
begin
if s[i]<>' ' then m[j]:=m[j]+s[i]   {Заполнение символами строки
                                      очередного элемента массива}
else
j:=j+1;  {Вычисление очередного индекса массива}
end;
{Вывод полученного массива строк в  объект Memo}
for i:=0 to j do
memo1.Text:=memo1.Text+m[i]+chr(13)+chr(10);
```

Использование компонента Memo для ввода данных символьного типа, состоящих из нескольких строк

Пример. Ввести с помощью компонента `Memo` текст из русских букв, состоящий из нескольких строк. Сосчитать, сколько в этом тексте букв `'a'`.

Окно формы проекта	Программный код

	`procedure TForm1.Button1Click(Sender: TObject);`
	`var`
	`  i, n, kol: integer;`
	`  s: string;`
	`begin`
	`kol:=0;`
	`{В свойстве lines компонента Memo содержатся`
	`строки текста, свойство lines.count содержит`
	`количество строк}`
	`with Memo1 do`
	`begin`
	`for n:=0 to lines.count-1 do`
	`begin`
	`s:=lines[n];`
	`for i:=1 to length(s) do`
	`if copy(s,i,1)='a' then kol:=kol+1;`
	`end;`
	`end;`
	`Label1.Caption:='Кол-во букв "a" = '`
	`+inttostr(kol);`
	`end;`
	`end.`

Практическая часть

1. Выполните задание, соответствующее вашему варианту.

Номер варианта	Задание
1	Дана строка, заканчивающаяся точкой. Подсчитать, сколько в ней слов.
2	Дана строка, содержащая английский текст. Найти количество слов, начинающихся с буквы `b`.
3	Дана строка. Определить, сколько в ней символов «*», «**;**», «**:**».
4	Дана строка символов, среди которых есть двоеточие (**:**). Определить, сколько символов ему предшествуют.

5	Дана строка. Преобразовать ее, удалив каждый символ «*».
6	Дана строка. Определить, сколько раз входит в нее группа букв ***abc***.
7	Дана строка. Подсчитать количество букв ***k*** в последнем ее слове.
8	Дана строка символов, среди которых есть одна открывающаяся скобка и одна закрывающаяся. Вывести на экран все символы, расположенные внутри этих скобок.
9	В строке заменить все двоеточия (**:**) точкой с запятой (**;**). Подсчитать количество замен.
10	В строке между словами вставить вместо пробела запятую и пробел.
11	Удалить часть символьной строки, заключенной в скобки (вместе со скобками).
12	Определить, сколько раз в строке встречается заданное слово.
13	Проверить, одинаковое ли количество открывающихся и закрывающихся скобок в данной строке.
14	Строка содержит произвольный текст на русском языке. Проверить, каких букв в нем больше: гласных или согласных.
15	В строке имеется одна точка с запятой (**;**). Подсчитать количество символов до точки с запятой и после нее.

2. Выполните задание, соответствующее вашему варианту

Номер варианта	**Задание**
1	Дан текст, записанный заглавными русскими буквами. Получить тот же текст, записанный строчными буквами.
2	Дан произвольный текст. Выяснить, чего в нем больше: русских букв или цифр.
3	Дан текст на русском языке. Выяснить, входит ли данное слово в указанный текст, и если да, то сколько раз.
4	Дан текст на русском языке. Определить сколько раз встречается в нем самое длинное слово.
5	Дан текст на русском языке. Выбрать из него только те символы, которые встречаются в нем только один раз, в том порядке, в котором они встречаются в тексте.
6	Дан текст на русском языке. Определить, сколько раз встречается в нем самое короткое слово.

7	Дан текст на русском языке и некоторая буква. Подсчитать, сколько слов начинается с указанной буквой.
8	Дан текст. Сколько слов в тексте? Сколько цифр в тексте?
9	Дан текст и некоторое слово. Вывести те предложения текста, которые содержат данное слово.
10	Дан текст, в котором встречаются арифметические выражения вида `a ⊕ b`, где ⊕ - один из знаков `+, -, *, /`. Выписать все арифметические выражения и вычислить их значения.
11	Дан текст на русском языке. Подсчитать количество слов, начинающихся и заканчивающихся на одну и ту же букву.
12	Дан текст на русском языке и некоторая буква. Найти слово, содержащее наибольшее количество указанных букв.
13	Дан произвольный текст. Проверить, правильно ли в нем расставлены круглые скобки.
14	Дан текст на русском языке и некоторые два слова. Определить, сколько раз они входят в текст и сколько раз они входят непосредственно друг за другом.
15	Дан зашифрованный текст на русском языке. Каждая буква заменяется на следующую за ней (буква *я* заменяется на *а*). Получить новый текст, содержащий расшифровку данного текста.

Контрольные вопросы

1. Как описываются переменные символьного типа?
2. Опишите стандартные функции для работы с символьными переменными.
3. Как описываются строковые переменные?
4. Как записывается строковая константа?
5. Какова максимальная длина строки?
6. Что общего между строкой и символьным массивом?
7. Какие операции применимы к строковым переменным и константам?
8. Опишите с примерами стандартные функции обработки строковых данных.
9. Опишите работу стандартных процедур обработки строковых данных.
10. Как можно обратиться к элементам строки?
11. Как в `Memo` можно обработать несколько строк?

Содержание отчета

1. Название, цели лабораторной работы.
2. Формулировка практической части работы.
3. Фрагменты программного кода.
4. Выводы по работе.
5. Ответы на контрольные вопросы.

Лабораторная работа № 7

РАБОТА С ФАЙЛАМИ

Цель работы: изучить типы файлов, процедуры обработки файлов.

Теоретическая часть

Чтобы сохранять входные данные и результаты неограниченно долго и иметь возможность воспользоваться ими в любой момент, используют файлы на магнитных носителях информации.

По способу доступа к информации, записанной в файл, различают файлы прямого и последовательного доступа.

Файлом *последовательного доступа* называется файл, к элементам которого обеспечивается доступ в такой же последовательности, в какой они записывались. Как правило, это текстовые файлы.

Файлом *прямого доступа* называется файл, доступ к элементам которого осуществляется по адресу элемента. Как правило, это файлы баз данных.

Turbo Pascal поддерживает три типа файлов:

- текстовые;
- типизированные;
- нетипизированные.

К *типизированным* относятся файлы строго определенного типа. Чаще всего это файлы, состоящие из записей. Они применяются для создания различных баз данных. Содержимое такого файла рассматривается как последовательность записей определенной структуры. Это файлы прямого доступа. Единицей измерения такого набора данных является сама запись.

Нетипизированные файлы не имеют строго определенного типа, их можно рассматривать как совокупность символов или байтов. Внутренняя реализация поддержки таких файлов наиболее близка к аппаратной поддержке работы с внешними носителями. За счет этого достигается максимальная скорость доступа к наборам данных. При работе с такими файлами не тратится время на преобразование типов и поиск управляющих последовательностей, достаточно считать содержимое файла в определенную область памяти. Это файлы прямого доступа, самым важным параметром служит длина записи в байтах.

Текстовые файлы

Текстовый файл можно рассматривать как последовательность символов, разбитую на строки длиной от 0 до 256 символов. Это файлы последовательного доступа. Структурной единицей текстовых файлов является строка. Данные в таких файлах хранятся в виде цепочки ASCII кодов и могут обрабатываться любым текстовым редактором. Каждая строка завершается маркером конца строки. На практике такой маркер представляет собой последовательность из двух символов: перевод строки `chr(10)` и возврат каретки `chr(13)`. Эти два символа задают стандартные действия по управлению текстовыми файлами.

Текстовые файлы описываются в разделе описания переменных:

```
Var
  <файловая переменная>: TextFile;
```

Файловая переменная – это имя переменной, которое используется в программном коде для работы с файлом.

Открытие текстового файла

Перед тем как записать данные в файл или прочитать данные из файла, необходимо сначала открыть этот файл. Открытие текстового файла на запись, чтение или дозапись осуществляются с помощью разных процедур. Но прежде чем их использовать, необходимо во всех случаях присвоить файлу на магнитном носителе имя, т.е. установить соответствие между *файловой переменной* в программе и *именем файла* на диске. Это делается с помощью процедуры **`AssignFile:`**

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

Здесь имя файла - любое выражение строкового типа, которое строится по правилам определения имен в операционной системе.

Например:

```
AssignFile(F,'c:\Student\Primer.Txt');
```

Процедуры для открытия текстовых файлов

В таблице F – имя файловой переменной.

Обращение к процедуре	Действие
`Rewrite` `(F)`	Открывает (создает) новый файл. Имя файла предварительно определяется в процедуре `AssignFile.` Если на диске уже был файл с таким именем, то он уничтожается.

Reset (F)	Открывает уже существующий файл. Файл считывается последовательно. Если эта процедура применена к несуществующему файлу, то возникает ошибка ввода-вывода.
Append (F)	Открывает уже существующий файл для дозаписи. Запись производится в конец файла.

Процедуры для обработки текстовых файлов

У текстовых файлов есть своя специфика. Специальные расширения стандартных процедур чтения (Read) и записи (Write), описанных ниже, разрешают работать со значениями несимвольного типа. Другими словами, последовательность символов автоматически преобразуется к значению того типа переменной, которая используется в файловых операциях.

Вызов Read(F,Ww), где Ww - переменная типа word, осуществляет чтение из файла F последовательности цифр, которая затем интерпретируется в число, значение которого и будет присвоено переменной Ww. В случае если вместо последовательности цифр идет любая другая последовательность символов, использование такого оператора приводит к ошибке выполнения программы.

В нижеследующей таблице F – имя файловой переменной, V1,V2,…,Vn – переменные разных типов.

Обращение к процедуре	Действие
Read (F,V1[,V2,…,Vn]);	Считывает из дискового файла строки символов в переменные V1, V2,…,Vn.
Readln (F,V1[,V2,…,Vn]);	Выполняет те же действия, что и **Read**, и дополнительно – чтение до маркера конца строки и переход к новой строке **Readln(F)** без списка переменных позволяет пропустить строку в файле и перейти на новую строку.
Write (F,V1[,V2,…,Vn]);	Записывает значения переменных V1,V2,…, Vn в файл на диске.
Writeln (F,V1[,V2,…,Vn]);	Выполняет те же действия, что и **Write,** но обеспечивает запись всех величин с обязательной установкой маркера конца строки в файл **Writeln(F)** без списка переменных записывает в файл пустую строку.

Закрытие файла

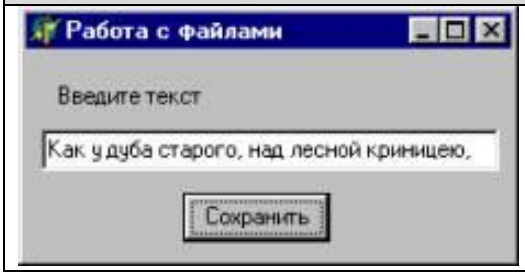
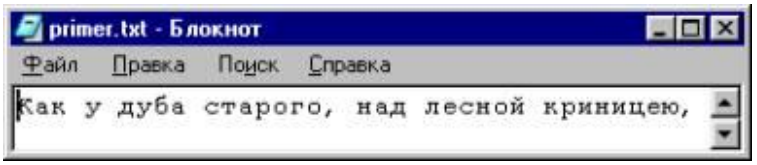
После работы с файлом его нужно обязательно закрыть, иначе информация в файле может быть потеряна. Это делается с помощью процедуры CloseFile(F).

Функции для работы с файлами

Функция	Действие
`Eoln(F)`	Возвращает булевское значение `True`, если текущая файловая позиция находится на маркере конца строки или вызов `Eof(F)` вернул значение `True`. Во всех остальных случаях значение функции будет `False`.
`Eof(F)`	Возвращает булевское значение `True`, если указатель конца файла находится сразу за последним компонентом, и `False` – в противном случае.
`SeekEoln(F)`	Возвращает булевское значение `True` при достижении маркера конца строки, причем указатель файла пропускает все пробелы и знаки табуляции, предшествующие маркеру. В противном случае возвращает значение `False`.
`SeekEof(F)`	Возвращает булевское значение `True`, если указатель файла находится на маркере конца файла. Эта функция также пропускает все пробелы и знаки табуляции, предшествующие маркеру, и выполняет автоматический пропуск маркера конца строки.

Примеры работы с файлами

Пример. Записать в файл текст, введенный в окне `Edit`.

Окно формы проекта	Полученный файл, открытый в блокноте
	

Программный код

```
var
F: TextFile;     {Описание файловой переменной}
begin
 AssignFile(F,'primer.txt');  {Связь файловой
                              переменной с файлом}
 Rewrite(F);                  {Создать новый файл}
 Writeln(F,Edit1.Text);          {Записать в файл}
 CloseFile(F);                      {Закрыть файл}
end;
```

Пример. Открыть текстовый файл для чтения, считать из него текст в окно Memo. Перед открытием файла проверить его наличие; в случае отсутствия выдать сообщение.

Фрагмент кода программы

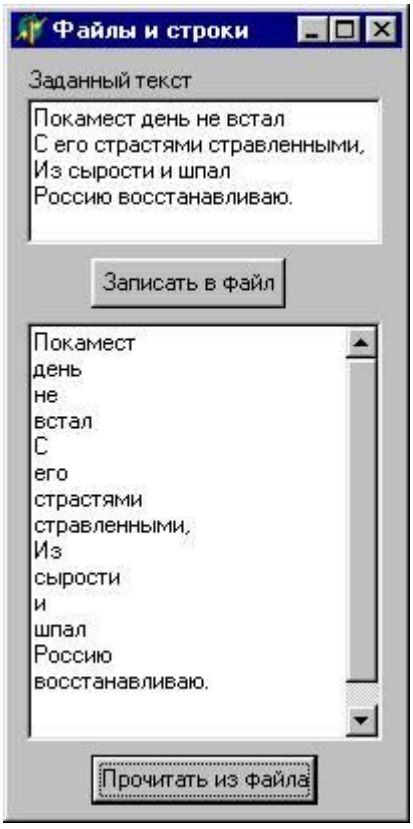
```
var
F: TextFile;      {Описание файловой переменной}
Ch: char;         {Описание переменной, в которую
                   будет считываться символ из файла}
begin
 AssignFile(F,'primer.txt');      {Связь файловой  переменной с
                                   файлом на диске}
 {$I-}                            {Директива компилятора: отключить
                                   проверку ошибок ввода вывода}
 Reset (F);                       {Открыть файл для чтения}
 {$I+}                            {Директива компилятора: включить
                                   проверку ошибок ввода вывода}

 if IOResult = 0 then             {Если операция открыть файл
                                   выполнена успешно}
  begin
   while not Eof(f) do            {Пока не конец файла}
    begin
     Read(F,Ch);                  {Прочитать из файла символ}
     Memo1.text:=memo1.text+Ch;   {Вывести символ в поле Memo}
    end;
    CloseFile(F);                 {Закрыть файл}
  end
  else   {IOResult <> 0 - операция открыть файл не выполнена}
    ShowMessage('Нет такого файла');
end;
```

Пример. Открыть текстовый файл для дополнения, добавить в него текст из окна Edit.

Фрагмент кода программы

```
var
F: TextFile;     {Описание файловой переменной}
begin
 AssignFile(F,'primer.txt');  {Связь файловой переменной с файлом}
 Append(F);                   {Открыть существующий файл для
                               добавления текста в его конец}
 Writeln(F,Edit1.Text);       {Записать в файл}
 CloseFile(F);                {Закрыть файл}
end;
```

Пример. Стихотворный текст (в строке не более 80 символов) записать «лесенкой» (по одному слову в строке).

Окно формы проекта	Программный код
	

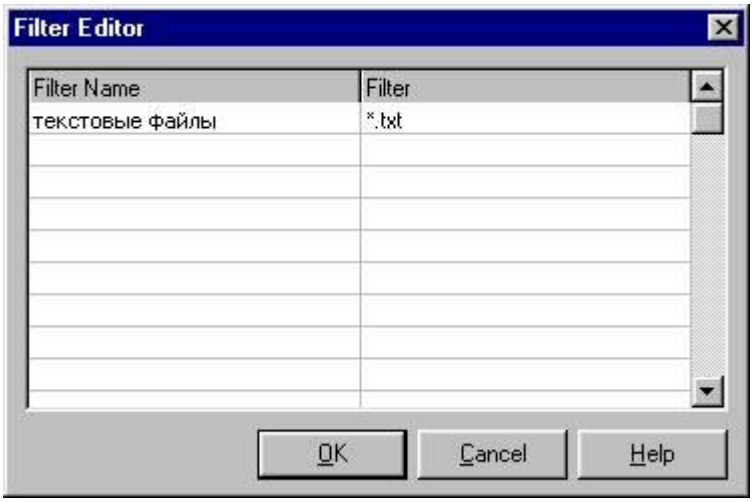

```
var
  Form1: TForm1;
  f: TextFile;  {Описание файловой переменной}
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var
 m: array[0..100] of string[80];
 i, j, n: integer;
 s: string;
begin
j:=0;
{Деление строк, записанных в Memo на слова и
 запись слов в строковый массив m}
with Memo1 do
 begin
  for n:=0 to lines.count-1 do
   begin
    s:= lines[n]+' ';
    for i:=1 to length (s) do
     begin
      m[j]:=m[j]+copy(s,i,1);
      if (copy(s,i,1)=' ') then j:=j+1;
     end;
   end;
 end;
{Запись полученного строкового массива в файл
 text.txt}
AssignFile(f,'text.txt');  {связь файловой
                           переменной с файлом}
Rewrite(f);                {Создать новый файл}
for i:=0 to j do
 Writeln(f,m[i]);          {Запись слов в файл}
closefile(f);                    {Закрыть файл}
end;

procedure TForm1.Button2Click(Sender: TObject);
var
 s: string;
begin
Reset(f);              {Открыть файл для чтения}
Readln(f,s);
Memo2.Lines[0]:=s;
while not eof(f) do        {Пока не конец файла}
 begin
  Readln(f,s);       {Прочитать из файла строку}
             {Вывести прочитанную строку в Memo}
  Memo2.lines.Add(s);
 end;
closefile(f);                    {Закрыть файл}
end;

end.
```

Краткая характеристика компонентов группы Dialogs

В состав среды Delphi входят десять компонентов группы `Dialogs`, реализующих работу со стандартными диалоговыми панелями Windows. Эти компоненты облегчают разработку стандартных фрагментов приложения, таких как выбор нужного файла, цвета, шрифта, работу с принтером. Для определения, какая кнопка диалоговой панели была нажата, используется метод `Execute`. Если нажата кнопка «ОК», то он возвращает значение *True*, если кнопка «Отмена», – то значение *False*.

Компонент `OpenDialog`  позволяет выбрать файл для открытия из стандартного диалогового окна Windows.

Свойства компонента OpenDialog

`DefaultExt`	Создает расширение файла автоматически.
`Filter`	Позволяет задать шаблон имен открываемых файлов в окне редактора фильтров. Можно одновременно определить несколько фильтров. 
`InitialDir`	В этом свойстве хранится название рабочего каталога.
`FileName`	В этом свойстве хранится название выбранного файла.
`Options →`	В этом свойстве находится ряд опций открываемой панели – свойств типа Boolean. Опция включена, если ее значение *True*, и отключена, если *False*.

Options →	OfAllowMultiSelect	Позволяет выбрать несколько файлов одновременно
	ofCreatePromt	Выдает запрос на создание файла, если его нет
	ofFileMustExist	Позволяет выбирать только существующие файлы
	ofNoChangeDir	Запрещает изменение каталога
	ofOwerwritePromt	Запрашивает подтверждение на сохранение существующего файла
	ofReadOnly	Помечает флажок Read Only (только для чтения)
	ofNoReadOnlyReturn	Запрещает выбор файла только для чтения
Title	Задает текст заголовка панели	

Пример использования в программе

```
{Открыть диалог и запомнить имя файла с диска в переменной Name}
if OpenDialog1.Execute Then Name:= OpenDialog1.FileName;
```

Компонент SaveDialog  позволяет выбрать файл для сохранения из стандартного диалогового окна *Windows*. По внешнему виду и порядку работы практически аналогичен компоненту OpenDialog. Различаются только надписи и некоторые свойства.

Компонент FontDialog  позволяет вызвать стандартную диалоговую панель выбора шрифтов и их характеристик.

Свойства компонента FontDialog

Font	Определяет, какой шрифт был выбран для последующей работы	
Device	Выбор списка используемых шрифтов: только для принтера, только для дисплея или для того и другого устройства	
Options	Задает опции диалоговой панели	
	FdAnsiOnly	Работа только со шрифтами, поддерживаемыми Windows
	fdEffects	Использование для шрифтов эффектов и цветов
	fdFixedPitchOnly	Работа только с непропорциональными шрифтами

Options	fdForceFontExist	Выдает сообщение при выборе несуществующего шрифта
	fdTrueTypeOnly	Работа только со шрифтами TrueType
	fdLimitSize -	Ограничивает размер выбираемого шрифта

Пример использования в программе

```
if fontdialog1.execute then
memo1.font:=fontdialog1.font;
```

Компонент ColorDialog  позволяет вызвать стандартную диалоговую панель настройки цветов.

Свойства компонента ColorDialog

Color	Сохраняет выбранный цвет	
Options	Свойство состоит из следующих опций:	
	CdFullOpen	Выполнить полный показ диалога
	cdPreventFullOpen	Запретить полный показ диалога

Пример использования в программе

```
if colordialog1.execute then
form1.color:=colordialog1.color;
```

Компонент FindDialog  позволяет вызвать стандартную диалоговую панель поиска текста, ввести текст для поиска. Сам поиск программируется самостоятельно.

Свойства компонента FindDialog

FindText	Хранит текст для последующего поиска
Options	В этом свойстве заданы опции панели. Это опции на включение в панель дополнительных сервисных кнопок

Компонент ReplaseDialog  позволяет вызвать стандартную диалоговую панель поиска и замены текста, ввести текст для поиска. Сам поиск программируется самостоятельно.

Свойства компонента ReplaseDialog

FindText	Хранит текст для поиска
ReplaceText	Хранит текст для замены

Пример. Ввести двумерную матрицу из файла. Для открытия нужного файла использовать компонент OpenDialog.

Программный код

```
var
i,j: integer;
F: TextFile;                {Описание файловой переменной}
Name: String;
begin
 name:='';
 {Открыть диалог и запомнить имя файла с диска в переменной Name}
 if OpenDialog1.Execute Then Name:= OpenDialog1.FileName;
 AssignFile(F,Name);        {Связать файловую переменную
                             с файлом на диске}
 {$I-}                      {Директива компилятора: отключить
                             проверку ошибок ввода вывода}
 Reset(F);                  {Открыть файл для чтения}
 {$I+}
 If IOResult=0 Then begin   {Если операция открытия файла
                             прошла успешно}
  Readln(F,n,m);            {Считывание из файла размерности матрицы}
  for i:=0 to n-1 do
   begin
    for j:=0 to m-1 do
     read(f,a[i,j]);        {Считывание из файла элементов
                             строки матрицы}
    readln(f);              {Перевод на новую строку файла}
   end;
  CloseFile(F);      end    {Закрытие файла}
 else
  ShowMessage('Нет такого файла');
end;
```

Практическая часть

1. При выполнении индивидуального задания необходимо текст считать из текстового файла. Результат выполнения программы записать в исходный файл.

Номер варианта	Задание
1	В тексте имеется одна точка с запятой (**;**). Подсчитать количество символов до точки с запятой и после нее.
2	Дан произвольный текст на русском языке. Проверить, каких букв в нем больше: гласных или согласных.
3	Проверить, одинаковое ли число открывающихся и закрывающихся скобок в данном тексте.

4	Определить, сколько раз в тексте встречается заданное слово.
5	Удалить часть текста, заключенного в скобки (вместе со скобками).
6	В тексте между словами вставить вместо пробела запятую и пробел.
7	В тексте заменить все двоеточия (**:**) точкой с запятой (**;**). Подсчитать количество замен.
8	Дан текст, в котором есть одна открывающаяся и одна закрывающаяся скобки. Вывести на экран все символы, расположенные внутри этих скобок.
9	Дан текст. Подсчитать количество букв ***k*** в последнем слове.
10	Дан текст. Определить, сколько раз входит в него группа букв ***abc***.
11	Дан текст. Преобразовать его, удалив каждый символ «*».
12	Дан текст, среди символов которого есть двоеточие (**:**). Определить, сколько символов ему предшествует.
13	Дан текст. Определить, сколько в нем символов «*», «**;**», «**:**».
14	Дан текст на английском языке. Определить количество слов, начинающихся с буквы `b`.
15	Дан текст, заканчивающийся точкой. Подсчитать, сколько в нем слов.

2. Сформировать массив `n*m` случайным образом и записать его в файл. При выполнении индивидуального задания сгенерированный массив и его размерность считать из файла. Результат выполнения программы вывести при помощи компонента StringGrid.

Номер варианта	**Задание**
1	Дан двумерный массив размерами `n*m`. 1. Заменить четный элемент каждой строки нулем. 2. Вставить после всех строк, содержащих минимальный элемент массива, строку 2, 4, 6,… 3. Удалить все столбцы, в которых встретится элемент, по модулю больший левого верхнего ($a_{1,1}$). 4. Поменять местами третий и последний столбцы.
2	Дан двумерный массив размерами `n*m`. 1. Заменить максимальный элемент каждой строки номером столбца, в которой он находится. 2. Вставить после всех столбцов, содержащих нулевой элемент, первый столбец.

	3. Удалить все строки, в которых встретится четный отрицательный элемент. 4. Поменять местами первый и предпоследний столбцы.
3	Дан двумерный массив размерами n*m. 1. Заменить нечетный элемент каждой строки нулем. 2. Вставить после всех строк, содержащих минимальное значение, строку 1,2,3,…. 3. Удалить все столбцы, в которых первый элемент четный. 4. Поменять местами первый и последний столбцы.
4	Дан двумерный массив размерами n*m. 1. Заменить четный элемент каждого столбца максимальным по модулю. 2. Вставить после столбцов, содержащих минимальное значение, другой столбец. 3. Удалить все строки, в которых второй элемент больше предпоследнего. 4. Поменять местами первый и средний столбцы.
5	Дан двумерный массив размерами n*m. 1. Заменить элемент, кратный трем, в каждом столбце нулем. 2. Вставить после каждого столбца, начиная со второго, первый столбец. 3. Удалить из массива каждый столбец, содержащий элемент, кратный пяти. 4. Поменять местами третий и последний столбцы.
6	Дан двумерный массив размерами n*m. 1. Заменить отрицательный элемент каждого столбца нулем. 2. Вставить после каждого столбца, содержащего максимальный по модулю элемент, строку из нулей. 3. Удалить из массива каждую строку, содержащую элемент, кратный трем. 4. Поменять местами первый и последний столбцы.
7	Дан двумерный массив размерами n*m. 1. Заменить нулевой элемент каждого столбца максимальным по модулю элементом массива. 2. Вставить после каждой строки, содержащей максимальный по модулю элемент, последнюю строку. 3. Удалить из массива каждую строку, содержащую нулевой элемент. 4. Поменять местами два средних столбца.

8	Дан двумерный массив размерами n*m. 1. Заменить максимальный элемент каждого столбца нулем. 2. Вставить после всех строк, содержащих максимальный по модулю элемент, первую строку. 3. Удалить из массива строку и столбец, на перекрестье которых находится максимальный по модулю элемент. 4. Поменять местами последний и предпоследний столбцы.
9	Дан двумерный массив размерами n*m. 1. Заменить максимальный элемент каждой строки нулем. 2. Вставить после каждого столбца, содержащего максимальный элемент массива, столбец из нулей. 3. Удалить все столбцы, в которых встретится нечетный положительный элемент. 4. Поменять местами первый и предпоследний столбцы.
10	Дан двумерный массив размерами n*m. 1. Заменить максимальный элемент каждой строки нулем. 2. Вставить перед всеми строками, первый элемент которых делится на 3, строку из нулей. 3. Удалить самый левый столбец, в котором встретится четный отрицательный элемент. 4. Поменять местами второй и предпоследний столбцы.
11	Дан двумерный массив размерами n*m. 1. Заменить максимальный элемент каждой строки на противоположный по знаку. 2. Вставить после всех столбцов, содержащих максимальный элемент, столбец из нулей. 3. Удалить все столбцы, в которых есть отрицательный элемент. 4. Поменять местами первый и последний столбцы.
12	Дан двумерный массив размерами n*m. 1. Заменить минимальный по модулю элемент каждого столбца на противоположный. 2. Вставить после каждого столбца, содержащего значение, равное нулю, столбец из нулей. 3. Удалить все строки, содержащие максимальные элементы. 4. Поменять местами первый и последний столбцы.
13	Дан двумерный массив размерами n*m. 1. Заменить все элементы первых трех столбцов на их квадраты. 2. Вставить после каждой нечетной строки первую строку.

	3. Удалить все столбцы, в которых первый элемент больше последнего. 4. Поменять местами средние строки с первой и последней.
14	Дан двумерный массив размерами `n*m`. 1. Заменить минимальный по модулю элемент каждого столбца нулем. 2. Вставить после каждой строки, содержащей минимальное значение, строку из нулей. 3. Удалить все столбцы, в которых первый элемент больше последнего. 4. Поменять местами первый и последний столбцы.
15	Дан двумерный массив размерами `n*m`. 1. Заменить максимальный по модулю элемент каждой строки на противоположный по знаку. 2. Вставить после каждой четной строки первую строку. 3. Удалить все строки, содержащие ноль. 4. Поменять местами средние столбцы.

Контрольные вопросы

1. Как объявить файловую переменную?
2. Как файловую переменную связать с физическим файлом?
3. Опишите назначение процедур и функций файлового ввода-вывода: `Append, AssignFile, CloseFile, Eof, IOResult, Read, Reset, Rewrite, Seek, Write`.
4. Каково назначение директивы компилятора `{$I-}`?
5. Опишите назначение компонентов `OpenDialog, SaveDialog, FontDialog` страницы Dialogs палитры компонентов. Каковы особенности их размещения на форме приложения?
6. Каково назначение свойства `OpenDialog1.Filter`? Какими способами можно изменять значение этого свойства?
7. Описать разницу между текстовыми, типизированными и нетипизированными файлами.

Содержание отчета

1. Название, цели лабораторной работы.
2. Формулировка практической части работы.
3. Фрагменты программного кода.
4. Выводы по работе.
5. Ответы на контрольные вопросы.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

Основные элементы языка Object Pascal

Переменные и константы

В программе каждый элемент данных является либо константой, либо переменной.

Константами называют простейшие объекты программы, значения которых заранее известны и в процессе выполнения программы не изменяются.

В разделе описания констант производится присваивание константам постоянных значений. Раздел начинается зарезервированным словом `const`, за которым следует ряд выражений, присваивающих именам констант постоянные числовые или строковые значения. Выражения присваивания заканчиваются точкой с запятой.

```
const имя константы = значение константы;
```

Примеры:

```
const
     MaxInd: integer = 100; {Типизированная константа}
Name = ‘Петя’;          {Строковая константа}
g = 9.8;          {Вещественная константа}
```

Переменные в программе – это простейшие объекты программы, предназначенные для хранения в памяти некоторых значений и обработки их по заданному алгоритму. Все переменные в программе перед использованием нужно объявлять (декларировать). При объявлении указывается имя переменной и ее тип. Имя переменной однозначно связывает переменную с некоторой физической областью памяти компьютера (ячейкой для хранения переменной), а тип переменной определяет способ хранения и представления (формат) в памяти компьютерной системы.

Значение переменной – это данные, которые хранятся и обрабатываются по заданному алгоритму в процессе выполнения программы. В разное время переменные могут иметь различные значения.

Каждая переменная в программе должна иметь имя. В качестве имени можно использовать буквы латинского алфавита, цифры и символ нижней

черты _; первым символом в имени не должна быть цифра; пробелы в имени недопустимы; прописные и строчные буквы воспринимаются программой как синонимы.

Основные типы данных

Компьютер обрабатывает данные различных типов: целые и дробные числа, символы. Создавая переменную, программист должен указать *тип переменной* и тем самым определить, для хранения каких данных она предназначена.

В Pascal существуют четыре основных типа данных:

INTEGER	Целый
REAL	Вещественный
CHAR	Символьный
BOOLEAN	Логический

Значением переменной типа INTEGER может быть положительное или отрицательное целое число, а также ноль. Для описания целочисленного типа помимо типа INTEGER можно использовать также типы BYTE, WORD, LONGINT. Для каждого типа определен свой диапазон значений.

Идентификатор типа	**Диапазон значений**
Shortint (короткое целое)	-128...127
Integer (целое)	32 768...32 676
Longint (длинное целое)	-2 147 483 648...2 147 483 647
Byte (байт)	0...255
Word (слово)	0...65 535

Переменные и константы типа REAL используются для хранения вещественных чисел. На физическом уровне такие числа хранятся в памяти компьютера в виде мантиссы и порядка. Вместо 10 в степени, употребляется буква E. Такая форма называется представлением чисел с плавающей запятой. Например, вещественное число $0.31415926*10^{1}$ в программе будет выглядеть как .31415926E01, $0.1*10^{-3}$ как .1 E–03, $0.100002*10^{6}$ как .100002E 6.

Для описания вещественного типа помимо типа REAL можно использовать также типы **Single** , **Double, Extended, Comp.** Для каждого типа определен свой диапазон значений.

Идентификатор типа	Диапазон значений	Число цифр	Количество байт
Real	$2.9e-^{39}...1.7e^{38}$	11 – 12	6
Single	$1.5e-^{45}$ ... $3.4e^{38}$	7 – 8	4
Double	$5.0e^{-324}...1.7e^{308}$	15 – 16	8
Extended	$3.4e^{-4932}...1.1e^{4932}$	19 – 20	10
Comp	$9.2e^{18}$... $9.2e^{18}$	20 – 21	8

Описание переменных

В программах на языке Pascal каждая переменная перед использованием должна быть описана. С помощью описания устанавливается не только факт существования переменной, но и задается ее тип, тем самым определяется диапазон допустимых значений.

В тексте программы описание переменных начинается с ключевого слова ***var***, каждая переменная, как правило, помещается на отдельной строке. После имени переменной через двоеточие указывается тип переменной, затем ставится символ “точка с запятой”.

Примеры:

```
var
  a: real;
b: real;
i: integer;
```

В приведенном примере объявлены две переменные типа REAL и одна переменная типа INTEGER.

Если в программе несколько переменных одного типа, то можно через запятую перечислить имена переменных, относящихся к одному типу, и после имени последней переменной через двоеточие указать тип.

Примеры:

```
var
a,b,c: real;
x1,x2: real;
```

Операторы

Оператор – это такая синтаксическая единица, которая используется в программе для выполнения отдельного предписания.

Для присваивания переменной вычисленного значения используется оператор присваивания.

Для организации последовательности выполняемых действий используются алгоритмические операторы (операторы безусловных переходов, условные операторы, операторы циклов).

Стандартные процедуры и функции

Библиотечные модули Delphi содержат большое количество стандартных подпрограмм (процедур и функций). Ниже приведены наиболее часто используемые.

В приведенных ниже таблицах `Real` означают любой вещественный тип, `Integer` – любой целый, `String` – символьный.

Математические функции

Обращение	Тип параметра	Тип результата	Описание	Библиотечный модуль
Тригонометрические				
`ArcCos(x)`	`Real`	`Real`	Арккосинус	`Math`
`ArcCot(x)`	`Real`	`Real`	Арккотангенс	`Math`
`ArcSin(x)`	`Real`	`Real`	Арксинус	`Math`
`ArcTan(x)`	`Real`	`Real`	Арктангенс (квадрант не учитывается)	`System`
`ArcTan2(y,x)`	`y-Real, x-Real`	`Real`	Вычисляет арктангенс `y/x` и возвращает угол в правильный квадрант	`Math`
`Cos(x)`	`Real`	`Real`	Косинус	`Math`
`Cosecant(x)`	`Real`	`Real`	Косеканс (`1/Sin(x)`)	`Math`
`Cotan(x)`	`Real`	`Real`	Котангенс	`Math`
`Sec(x)`	`Real`	`Real`	Секанс (`1/Cos(x)`)	`Math`
`Sin(x)`	`Real`	`Real`	Синус	`Math`
`Tan(x)`	`Real`	`Real`	Тангенс	`Math`
Логарифмические				
`Ln(x)`	`Real`	`Real`	Натуральный логарифм	`System`
`LnXP1(x)`	`Real`	`Real`	Натуральный логарифм от (`X+1`). Используется, когда значение `X` близко к 0	`Math`

Log10(x)	Real	Real	Логарифм числа по основанию 1.	Math
Log2(x)	Real	Real	Логарифм числа по основанию 2	Math
LogN(x)	Real	Real	Логарифм числа по указанному основанию.	Math
Экспоненциальные				
Exp(x)	Real	Real	Экспонента числа	System
IntPower(y,x)	y-Real, x-Integer	Real	Возведение y в целочисленную степень x	Math
IntPower(y,x)	y-Real, x- Real	Real	Возведение y в вещественную степень x	Math
Разного назначения				
Abs(x)	Real|Integer	Тип параметра	Абсолютное значение (модуль) числа	System
Ceil(x)	Real	Integer	Ближайшее меньшее целое	Math
Floor(x)	Real	Integer	Ближайшее большее целое	Math
Frac(x)	Real|Integer	Тип параметра	Дробная часть числа	System
Int(x)	Real|Integer	Тип параметра	Целая часть числа	System
Pi	-	-	π = 3.141592653…	System
Random	-	-	Псевдослучайное число, равномерно распределенное в диапазоне 0…[1]	System
Random(x)	Integer	Integer	Псевдослучайное целое число, равномерно распределенное в диапазоне 0…[x-1]	System
Randomize	-	-	Инициализация генератора псевдослучайных чисел	System

Round(x)	Real	Integer	Округляет число до ближайшего целого	System
Sign(x)		Integer	Определяет знак числа	Math
Sqr(x)	Real	Real	Число, возведенное в квадрат	System
Sqrt(x)	Real	Real	Квадратный корень числа	System
Trunc(x)	Real	Integer	Ближайшее целое число, меньшее или равное x, если x≥0, и большее или равное x, если x<0	System

Функции преобразования типов данных

Обращение	Тип параметра	Тип результата	Описание	Библиотечный модуль
Преобразование строк в другие типы				
StrToInt(S)	String	Integer	Преобразование символов строки S в целое число	Sys Utils
StrToFloat(S)	String	Real	Преобразование символов строки S в вещественное число	Sys Utils
Обратное преобразование				
FloatToStr(x)	Real	String	Преобразование вещественного числа x в строку	Sys Utils
IntToStr(x)	Integer	String	Преобразование целого числа x в строку	Sys Utils
FloatToStrF(x,Format, precision, digits)	x-Real, Format-спецификация формата, precision- Integer, digits-Integer	String	Преобразование вещественного числа в форматированную строку	Sys Utils

FormatFloat (Format, x)	Format-String, x- Real	String	Преобразование вещественного числа в строку символов в соответствии с указанным форматом	Sys Utils

Процедуры общего назначения

Нижеперечисленные процедуры не имеют параметров.

Обращение	Описание
Beep	Выдает звуковой сигнал, вызывая функцию Windows Api MessageBeep
Exit	Выполняет немедленный выход из текущей процедуры
Break	Выполняет немедленный выход из текущего оператора цикла for, while или repeat. Управление передается на следующий после цикла оператор
Continue	Прерывает текущую итерацию оператора цикла for, while или repeat и переходит на начало следующей итерации

Стандартные функции и процедуры для работы со строками

Функции для работы с данными строкового типа

Обращение к функции	Действие	Пример
Copy (S, Poz, N)	Выделяет из строки S подстроку длиной N символов, начиная с позиции Poz. Здесь N и Poz - целочисленные выражения	S:='IBM-PC'; S1:=Copy(S,5,2); Результат: S1='PC'
Concat(S1,…,Sn)	Выполняет сцепление (конкатенацию) строк S1,S2,..,Sn в одну строку	S1:='Test'; S2:='-'; S3:='5'; S:=Concat(S1,S2,S3); Результат: S:='Test-5'

`Length(S)`	Определяет текущую длину строки `S`. Результат – значение целого типа	`S:='Test-5';`
		`n:=Length(S);`
		
		Результат: `n=5`

`LowerCase(S)`	Возвращает копию строки `S`, в которой все символы преобразованы в символы нижнего регистра (в строчные буквы). Работает только с буквами латинского алфавита	`S:= 'STUDENT';`
		`S:=LowerCase(S);`
		
		Результат:`S='student'`

`UpperCase(S)`	Возвращает копию строки `S`, в которой все символы преобразованы в символы верхнего регистра (в прописные буквы). Работает только с буквами латинского алфавита	`S:= 'student';`
		`S:=UpperCase(S);`
		
		Результат:`S='STUDENT'`

`Trim(S)`	Удаляет из строки `S` лидирующие и завершающие пробелы, а также управляющие символы	`S:='  Студент  ';`
		`S:=Trim(S);`
		
		Результат: `S='Студент'`

`TrimLeft(S)`	Возвращает копию строки `S` с удаленными лидирующими пробелами и управляющими символами	`S:='  Студент  ';`
		`S:=TrimLeft(S);`
		
		Результат:
		`S='Студент  '`

`TrimRight(S)`	Возвращает копию строки `S` с удаленными завершающими пробелами и управляющими символами	`S:='  Студент  ';`
		`S:=TrimRight(S);`
		
		Результат:
		`S='  Студент'`

`Chr(X)`	Возвращает символ, указанный его ASCII-кодом. Здесь `X` – значение целого типа. Результат – значение типа `Char`.	`for i:=65 to 71 do S:=S+Chr(i);` Результат: `S='ABCDEFG '`

`Ord(S)`	Возвращает ASCII-код указанного символа. Здесь `S` – переменная типа `Char`. Результат – значение целого типа.	`S:='F';`
		`Cod:=Ord(S);`
		
		Результат: `Cod=70`

`Pos(S1, S2)`	Обнаруживает первое появление в строке `S2` подстроки `S1`. Результат – целое число, равное номеру позиции, где находится первый символ подстроки `S1`. Если такое появление не обнаружено, то результат равен 0.	`S:='abcdef';`
		`n:=Pos('cd',S);`
		
		Результат: `n=3`

Процедуры для работы с данными строкового типа

Обращение к процедуре	**Действие**	**Пример**

`Delete(S, Poz, N);`	Удаление `N` символов из строки `S`, начиная с позиции `Poz`.	`S:='abcdef';`
		`Delete(S,3,2);`
		
		Результат: `S='abef'`

`Insert(S1,S2,Poz);`	Вставка строки `S1` в строку `S2`, начиная с позиции `Poz`.	`S:='ЭВМ PC';`
		`Insert('IBM-',S,5);`
		
		Результат:
		`S='ЭВМ IBM-PC'`

`Str(X[:N[:M]],S);`	Преобразовывает числовое значение `X` в строку `S`. Параметр `X` является выражением целочисленного или действительного типа, а параметры `N` и `M` – это целочисленные выражения, определяющие форматирование строки `S`.	`x:= -23.60;`
		`Str(x:6:1,S);`
		
		Результат: `S=' –23.6'`
		`a:=-23.60;`
		`b:=20.5;`
		`Str(a+b:6:1,S);`
		
		Результат: `S='  –3.1'`

`Val(S,x,Cod);`	Преобразовывает строку `S` в числовое значение. Параметр `S` должен содержать последовательность символов, которая может быть воспринята как действительное число со знаком. Параметр `x` может быть как действительным, так и целочисленным. В параметр `Cod` заносится код выполнения преобразования: 0, если преобразование выполнено успешно, или номер позиции, в которой произошла ошибка	`S:='–23.60';`
		`val(S,x,Cod);`
		
		Результат: `x=–23,6`

Библиографический список

1. *Пашеку, Хавьер.* Программирование в Borland Delphi 2006 для профессионалов = Delphi for .NET Developer's Guide / Хавьер Пашеку. – М. : Вильямс, 2006. – 944 с. – ISBN 0-672-32443-1.

2. *Рубенкинг, Нил Дж.* Язык программирования Delphi для «чайников». Введение в Borland Delphi 2006 = Delphi for Dummies / Нил Дж. Рубенкинг. – М. : Диалектика, 2007. – 336 с. – ISBN 0-7645-0179.

Оглавление

Учебное издание

ОЗЕРОВА Марина Игоревна

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ
Delphi

Практикум

Подписано в печать 11.11.11.
Формат 60x84/16. Усл. печ. л. 6,75. Тираж 100 экз.
Заказ
Издательство
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых
600000, Владимир, ул. Горького, 87.