А.С. Кудрявцев

ПРОГРАММИРОВАНИЕ В DELPHI

Учебное пособие

Санкт-Петербург
2011

Министерство образования и науки Российской Федерации

**Государственное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный
технологический университет растительных
полимеров»**

А.С. Кудрявцев

ПРОГРАММИРОВАНИЕ В DELPHI

УЧЕБНОЕ ПОСОБИЕ

**Санкт-Петербург
2011**

ББК 31.38я7
К 889
УДК 681.3.6(075)

А.С. Кудрявцев
Программирование в Delphi: учебное пособие/ ГОУВПО СПбГТУРП. СПб.,2011. - 102 с.: ил. 47.

В пособии рассматриваются общие характеристики и структура программной системы Delphi, графический интерфейс ее среды программирования, основные конструкции используемого языка программирования и технология программирования.

Пособие предназначено для бакалавров, обучающихся по направлению подготовки 220700 «Автоматизация технологических процессов и производств».

Рецензенты: кафедра информационных технологий СПб института управления и права (заведующий кафедрой, профессор, кандидат технических наук О.А. Кононов);
профессор кафедры автоматики и автоматизации производственных процессов СПбГУНиПТ, доктор технических наук, профессор В.А. Балюбаш.

Рекомендовано к изданию Редакционно-издательским советом университета в качестве учебного пособия.

Редактор и корректор Т.А. Смирнова
Технический редактор Л.Я. Титова Темплан 2011,поз.49

Подп. к печати 03.02.2011. Формат 60×84/16. Бумага тип.№1. Печать офсетная.
Печ.л. 6,5. Уч.-изд.л. 6,5. Тираж 100 экз. Изд.№ 49 Цена «С». Заказ .

Ризограф ГОУВПО Санкт-Петербургского государственного технологического университета растительных полимеров, 1980095, СПб., ул. Ивана Черных, 4.

Предисловие

Программная система Delphi, описание которой приводится в предлагаемом пособии, определяется специалистами по программированию как одна из наиболее качественных систем разработки прикладного программного обеспечения.

Основой для создания Delphi стал язык программирования Pascal, разработанный в 60-х годах 20-го столетия профессором цюрихского университета Николсом Виртом в качестве учебного языка для студентов. Позднее, в 1983 году, один из учеников Вирта создал компанию Borland, в которой Pascal, пройдя несколько этапов в своем развитии, превратился в мощное средство разработки программ. Возникло целое семейство языков Pascal, ведущее место среди которых занял язык Turbo Pascal. Этот язык, разработанный применительно к операционной системе MS DOS, в последующем нашел применение и при работе на компьютерах, оснащенных операционной системой Windows.

В начале 1990-х годов фирма Borland приступила к разработке нового программного продукта, ориентированного на использование возможностей, представленных операционной системой Windows, а также применение новых к тому времени технологий объектно-ориентированного программирования. Цель разработки состояла в обеспечении существенного сокращения затрат времени на создание прикладных программ пользователями.

В литературе по Delphi рассматриваются два варианта ее предназначений и возможностей применения. В соответствии с первым Delphi представляет собой простую для освоения систему, которую могут применять начинающие пользователи. Исходя из второго определения, Delphi предназначается для программистов – профессионалов. Приведенные определения, несмотря на свою кажущуюся несовместимость, не противоречат одно другому. Освоение основных, используемых чаще всего, средств Delphi позволяет начинающему пользователю разрабатывать программы, имеющие практическую значимость. Стремление углубить свои познания в использовании заложенных разработчиками в систему многочисленных средств и возможностей потребует от пользователя приложения изрядных усилий, что в итоге повысит уровень его профессионализма. Следует заметить, что начальное освоение Delphi не представит больших сложностей для студентов, так как при изучении ранее в школе информатики они осваивали программирование на одном из алгоритмических языков – чаще всего Pascal. В системе Delphi используется язык программирования Object Pascal (переименованный в последних версиях системы в язык Delphi), представляющий модифицированную версию Pascal.

Разработка пользователями программ в Delphi начинается с использования средств ее графического интерфейса для создания формы выходного документа решаемой задачи. Уместно вспомнить, насколько трудоемкой и кропотливой была эта работа при написании программ, например, на языке Pascal. Сокращение объема работы программистов и, соответственно, затрат

времени на разработку программ в Delphi достигается также за счет обращения к программным кодам стандартных конструкций, включенных разработчиками в состав системы, а также разрабатываемых и включаемых в нее самими пользователями.

Оценивая должным образом существенно повышающие эффективность труда программистов достоинства системы Delphi, следует иметь в виду, что включенные в нее стандартные средства не могут охватить разнообразные конструкции алгоритмов, подлежащих решению задач. Значительная часть работы по написанию программных кодов применительно к конкретным алгоритмам остается прерогативой программиста. Поэтому до начала разработки прикладных программ в Delphi, точно также как и при их написании на любом из алгоритмических языков, целесообразно, а порой и просто необходимо, описание алгоритма в удобном для его последующего программирования виде, в частности, представление в виде блок - схемы. Осваивающим программирование будущим инженерам нужно хорошо представлять, что умения программировать еще недостаточно для того, чтобы писать программы сложных задач. Для этого нужно научиться понимать и описывать логику алгоритмов. Именно с этой целью во все главы пособия включено большое количество примеров, в которых приводятся не только программные коды, но и блок – схемы алгоритмов, относительно которых эти коды написаны.

Предлагаемое учебное пособие написано применительно к преподаваемой на кафедре АТПП дисциплине “Программирование и основы алгоритмизации”. Содержание включенного в пособие учебного материала и его объем определились исходя из целей и тематики дисциплины, сформулированных в программе ее изучения.

При написании раздела 1.2 использованы материалы, предоставленные Т. С. Смирновой, которую автор благодарит также за многие полезные советы, полученные в процессе работы над пособием.

Автор благодарен студенткам факультета АСУТП Ярославе Котовой, выполнившей подготовку программных кодов примеров и получение решений по ним для включения в пособие, Ксении Колокольцевой и Татьяне Селивановой, подготовивших текст рукописи пособия к изданию.

Глава 1. Общая характеристика Delphi

1.1.Понятие Delphi

Система Delphi включает язык программирования и программный пакет, определенный разработчиками как среда программирования, в которой этот язык реализуется.

В рассматриваемой системе используется язык программирования Object Pascal, являющийся расширенной модификацией языка Pascal.

Среда программирования представляет собой комплекс программ, создающих удобное окружение для быстрой разработки пользователями прикладных программ, именуемых для краткости приложениями, их отладке и запуску. Среда программирования включает: графический интерфейс, библиотеки созданных разработчиками программ с возможным включением в них программ, разрабатываемых пользователями, редактор текстов программ, встроенный высокоскоростной компилятор.

При создании системы термин Delphi был использован в качестве наименования среды программирования. Позднее, начиная с седьмой версии, это имя было распространено разработчиками системы как на среду, так и на язык программирования.

Разработка приложений в Delphi отличается от их разработки на алгоритмических языках, например на Pascal, где всю работу по написанию текста программы выполняет программист. В Delphi программист получил возможность в процессе разработки приложения наглядно (визуально) показывать среде программирования,какие включенные в нее объекты он хотел бы использовать в своей работе и дописывать только те фрагменты программы, создание которых не может быть обеспечено за счет использования средств и возможностей среды.

Приложения, находящиеся в стадии разработки, называются в Delphi, также как и в других аналогичных программных системах, проектами.

Проект собирается из многих элементов среды программирования: форм, пиктограмм, картинок, программных модулей и других. Каждый элемент размещается в определенном файле. Проект любого разрабатываемого приложения представляет собой совокупность файлов. Основными файлами проекта являются:

- файлы описания форм, в которых описывается вид формируемых пользователем документов;
- файлы программных модулей, содержащие программные коды на языке Object Pascal,сгенерированные средой программирования и дополненные программистом;
- главный файл проекта, который представляет главный программный блок, подключающий все файлы модулей, входящих в проект.

Компилятор последовательно обрабатывает файлы проекта и строит из них выполняемый файл, т. е. создает приложение.

1.2. Графический интерфейс Delphi

В графическом интерфейсе Delphi программные функции, обеспечивающие общение пользователя со средой программирования в процессе разработки и выполнения приложений, представлены графическими элементами, отображаемыми на экране монитора компьютера, а также средствами управления ими.

Использование средств и возможностей графического интерфейса Delphi осуществляется обращением к его окнам и меню, обеспечивающим разработку, редактирование, компиляцию и выполнение программ. Графический интерфейс содержит большое количество окон различного назначения, к которым пользователь может обращаться в процессе работы. Окна по мере надобности могут вызываться на экран и убираться с него.

Знакомство с графическим интерфейсом и работа с ним начинается с его *основного окна*, которое открывается на экране монитора компьютера после запуска Delphi. Запуск Delphi производится также как и запуск любой программной системы, работающей под управлением операционной системы Windows, или через программное меню или обращением к ярлыку вызываемого продукта. Следует заметить, что для разных версий Delphi вид основного окна может быть несколько различен, но в основных чертах окна одинаковы. На рис. 1.1 приведен вид основного окна наиболее широко используемой в настоящее время версии Delphi 7.

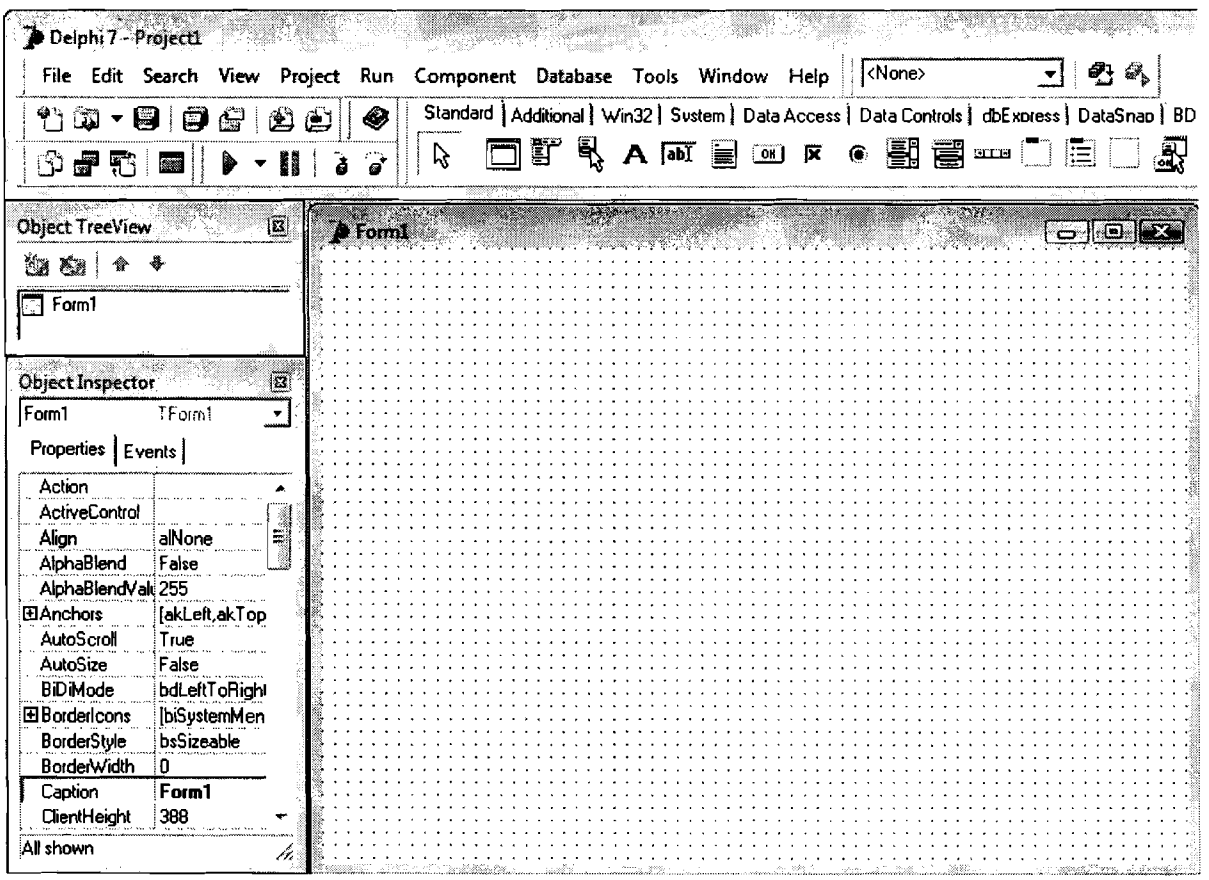

Рис. 1.1. Основное окно графического интерфейса Delphi

После запуска Delphi в его основном окне представлено пять окон:

- главное меню - **Delphi 7**;
- окно формы – **Form 1**;
- окно редактора свойств объектов – **Object Inspector**;
- окно просмотра списка объектов – **Object Tree View**;
- окно редактора кода – **Unit1.pas** (размещается за окном формы).

В первую очередь обратимся к расположенному в центре основного окна окну, в заголовке которого после запуска Delphi записано его наименование - Form 1. Это наименование автоматически присваивается системой в качестве имени, разрабатываемому в окне приложению, но может быть заменено программистом на другое. Рассматриваемое окно называется окном формы или просто *Формой*. Форма содержит все общие элементы,присущие окнам Windows: значок, заголовок, кнопки «Свернуть», «Развернуть», «Закрыть». В поле Формы отображаются исходные характеристики, результаты решения задач, а также вспомогательная информация, повышающая восприятие получаемого в результате решения задачи документа.

После запуска системы Форма представляет собой чистое поле. В нем программист размещает нужные для создаваемого приложения элементы, выбирая их из набора входящих в графический интерфейс типовых объектов, позиционирует их, устанавливает нужные размеры, меняет свойства.

Под Формой «спрятано» окно *Редактора Кода* (рис. 1.2). Активизация Редактора Кода осуществляется щелчком мыши по той части его окна, которое видно из-под Формы, либо нажатием на клавишу F12. При повторном нажатии этой клавиши вновь активизируется Форма.

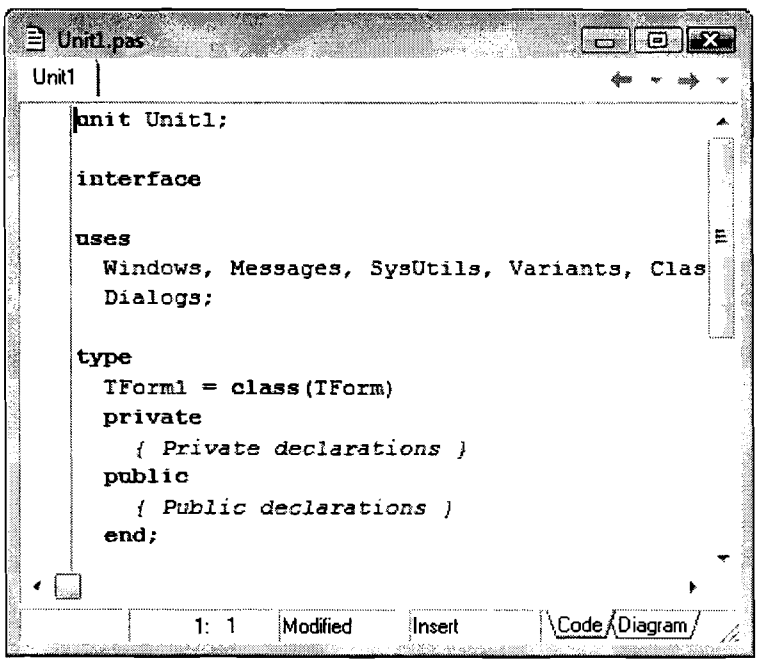

Рис 1.2. Окно Редактора Кода

В окне Редактора Кода размещается программный код на языке Delphi, соответствующий созданному программистом в Форме документу. Здесь же программист описывает алгоритм решения задачи.

В верхней части основного окна расположено *Главное Меню Системы*, включающее меню команд, панели инструментов и панели компонентов.

Меню команд размещено в верхней строке Главного Меню Системы. Меню команд содержит набор команд, необходимых для работы в среде программирования. Команды объединены в отдельные меню (группы) по назначению. Набор меню и состав входящих в них команд несколько различен в разных версиях Delphi. В общем же меню Delphi стандартны и понятны каждому, кто работал в операционной системе Windows.

Ниже дается описание назначения каждого из них:

File – работа с файлами.

Edit – работа с областью обмена, размещение компонентов на форме.

Search – поиск, замена заданного символа или строки в тексте.

View – отображение различной информации.

Project – управление объектом: добавление и удаление файлов, сборка проекта, установка параметров проекта.

Run – запуск и отладка программы.

Component – разработка новых компонентов, установка готовых компонентов.

Database – запуск программ, облегчающих построение приложений баз данных.

Tools – настройка параметров интегрированной среды разработки, запуск вспомогательных программ.

Window – активизация нужного окна интегрированной среды разработки.

Help – получение справочной информации.

Ниже меню команд в левой части главного окна находится *панель кнопок быстрого набора*, дублирующих наиболее часто используемые команды. На ней размещено шестнадцать кнопок - аналогов основных команд, обеспечивающих более быстрый запуск нужной команды сравнительно с обращением к меню.

Содержание меню будет изучаться по мере надобности в процессе освоения системы. Здесь приводится лишь несколько команд, чаще всего используемых при разработке приложений в среде Delphi. Создание нового проекта приложения начинается с команды **File/New Application**. По этой команде открывается новый проект приложения с пустой Формой, как это видно на рис. 1.1. Сохранить на диске готовый проект или его заготовку можно командой **File/ Save Project As** или **File/Save All**. Удобно также для сохранения использовать быстрые кнопки – третью или четвертую слева в верхнем ряду на рис. 1.1. Открыть ранее сохраненный проект можно командой **File/Open** или **File/Open Project** (вторая слева быстрая кнопка в верхнем ряду на рис. 1.1). Если же пользователь недавно работал с этим проектом, то удобнее воспользоваться командой **File/Reopen** или кнопкой справа от быст-

рой кнопки **Open** (см. рис. 1.1). Эта команда дает возможность быстро выбрать проект из числа тех, с которыми пользователь работал последнее время. Для компиляции и запуска на выполнение приложения надо выполнить команду **Run/Run** (быстрая кнопка с зеленой стрелкой, пятая в нижнем ряду на рис. 1.1).

Правее панели кнопок быстрого набора размещена панель палитры компонентов библиотеки визуальных компонентов, именуемая для краткости *палитрой компонентов*. Компонент предварительно можно определить как объект, который является отдельным строительным блоком программы при ее разработке. Более строгое определение компонента будет приведено в разделе 1.4.

Разработчики среды Delphi поместили в палитру компонентов то, что посчитали достаточным для создания любых приложений. Среди компонентов находятся кнопки, надписи, стандартные диалоговые окна, меню и др. Все множество компонентов разделено на группы. Каждая группа размещена на своей странице, вкладки которых приведены в верхней части палитры компонентов. Например, на странице с вкладкой **Standard** размещены стандартные компоненты пользовательского интерфейса, которыми чаще всего ограничиваются разработчики несложных приложений, на странице с вкладкой **Additional** – дополнительные компоненты пользовательского интерфейса.

Для размещения нужного компонента в Форме следует:

- перейти к нужной вкладке в палитре компонентов и активизировать ее, установив на нее курсор и щелкнув левой клавишей мыши;
- установить курсор на пиктограмму выбранного компонента и активизировать его;
- установить курсор в том месте Формы, где компонент должен быть размещен и щелкнуть левой клавишей мыши – компонент окажется в Форме.

Пользуясь курсором и мышью, можно изменять размеры компонентов и их расположение в Форме.

Например, для размещения в Форме текущей даты следует активизировать вкладку **Win32**, на странице которой обратиться к компоненту **Date Time Picker**. Если в эту же Форму нужно, кроме того, добавить компонент **Button**, используемый обычно в качестве управляющей кнопки, следует перейти на страницу с вкладкой **Standard** и щелкнуть мышью по пиктограмме **OK**, соответствующей указанному компоненту. Затем щелкнуть мышью в выбранном месте Формы, где появится окно с надписью `Button1` (рис. 1.3).

В левом нижнем углу основного окна расположено окно *Инспектора Объектов* - **`Object Inspector`** (рис 1.4).

Окно Инспектора Объектов – это окно свойств. Оно состоит из двух вкладок: *вкладки свойств* – **Properties** и *вкладки событий* – **Events**. На пер-

вой вкладке устанавливаются свойства компонентов, на второй – им присваиваются нужные значения. В заголовке окна указывается имя компонента, для которого в данный момент может вестись корректировка значений его свойств. Как только очередной компонент окажется в Форме, в заголовке окна Инспектора Объектов появляется его наименование, а в таблице окна – присущие ему свойства и соответствующие им значения, которые программист может изменять.

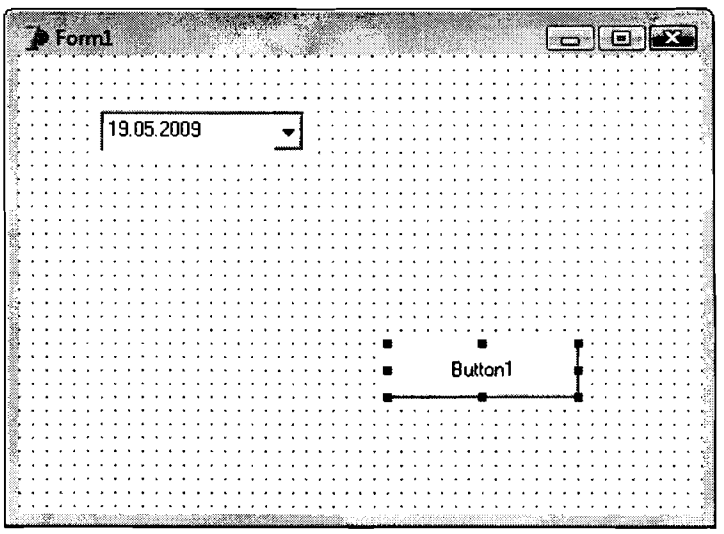

Рис. 1.3. Компонент `Button` на Форме

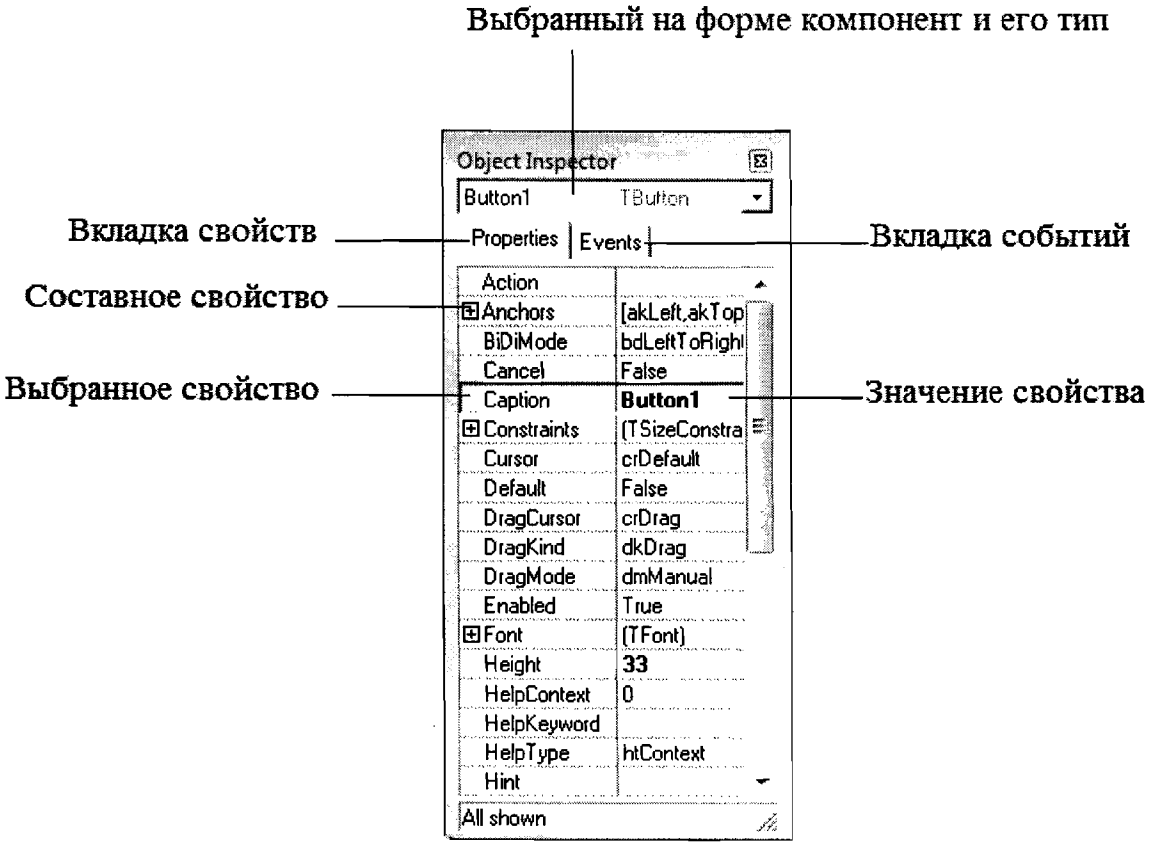

Рис. 1.4. Окно Инспектора Объекта

Например, для того чтобы изменить на кнопке `Button` её имя с `Button 1` на `Compute`,нужно обратиться к окну вкладки свойств Caption и изменить текст в соответствующем ему окне вкладки событий путем набора с клавиатуры. По мере набора строки каждая буква будет автоматически появляться на кнопке, т.е. свойство этого компонента будет изменяться.

Если в процессе разработки приложения возникает необходимость изменить значения свойств ранее включенных в Форму компонентов, то компонент, с которым будет проводиться эта работа, следует предварительно активизировать.

Над окном Инспектора Объектов находится окно **`Object Tree View`** – *окно Компонентов Формы* (рис 1.5).

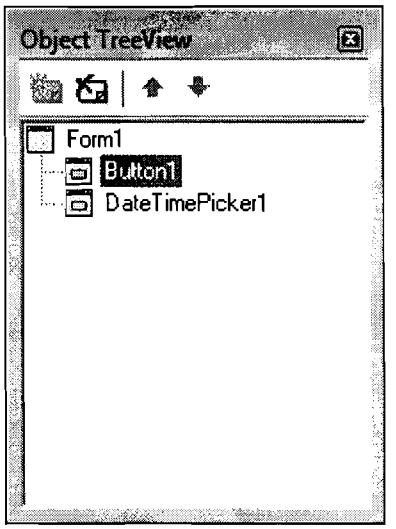

Рис 1.5. Окно компонентов Формы

В этом окне отображаются все компоненты, размещенные в форме в виде дерева, отражающего их вложенность. Из рис. 1.5 следует, что изображенное здесь дерево компонентов имеет один корневой элемент `Form1` и два вложенных компонента `Button1 и Date.`

1.3. Технология визуального программирования в Delphi. Пример разработки программы

Технология разработки приложений, применяемая в Delphi, определяется как технология визуального программирования.

Визуальное программирование – программирование, предусматривающее создание приложений с помощью наглядных (визуальных) средств вместо написания их текста.

В Delphi в качестве таких визуальных средств используются компоненты, размещаемые программистом в поле Формы. Каждому компоненту соответствует программный код, включенный в среду разработки приложений. При этом программист разрабатывает только форму получаемого в ре-

зультате решения задачи выходного документа, а соответствующий используемым компонентам текст программы генерируется автоматически и отображается в окне Редактора Кода. Работающие в Delphi программисты обычно не ограничиваются только использованием стандартного набора компонентов, а дополняют их самостоятельно разработанными.

С помощью средств визуальной разработки программисты получили возможность создавать приложения из готовых элементов, дописывая при этом только определенную часть текста программы. В некоторых случаях удается вообще не писать вручную ни одной строки программного кода. Однако в большинстве разработанных приложений,в частности, обеспечивающих получение решений различных инженерных задач, описание алгоритма на языке программирования остается прерогативой программиста.

Для разработки приложений в Delphi можно и не использовать визуальное программирование, выполняя их написание только средствами языков Object Pascal или Delphi.

При использовании технологии визуального программирования процесс разработки приложений разделяется на два этапа.

На первом этапе программист размещает в Форме необходимые компоненты, позиционирует их, устанавливает нужные размеры, меняет свойства, используя для этого средства среды разработки. При этом решаются две задачи. Первая из них, возлагаемая на программиста, заключается в создании им формы выходного документа. Знакомым с программированием на какомлибо алгоритмическом языке, например Pascal, известно, насколько трудоемкой и кропотливой является там работа по описанию в программе позиций и параметров каждой представленной в выходном документе характеристики. Ниже, при рассмотрении примера разработки приложения, можно будет увидеть и оценить, насколько проста эта работа в Delphi. Вторая задача решается системой и заключается в формировании программного кода применительно к размещенным программистом в Форме компонентам при проектировании здесь выходного документа.

На втором этапе программистом выполняется написание программного кода, определяемого содержанием алгоритма, относительно которого разрабатывается приложение. Из этого следует, что для программиста и при работе в Delphi остается необходимым знание языка программирования и умение писать на нем программы. При программировании сложных алгоритмов полезно, а порой и просто необходимо, их представление в виде,удобном для написания программ, например, в виде блок-схем.

Обратимся к примеру, в котором вначале рассмотрим предварительные этапы работы программиста по подготовке к разработке приложения в Delphi, а затем и сам процесс его разработки.

В качестве такого примера возьмем задачу понятную, а возможно и интересную специалистам любого профиля деятельности, которая формулируется в следующем виде: требуется по заданному росту человека определить его идеальный вес.

Алгоритм решения данной задачи выражается следующей зависимостью:

```
идеальный вес = рост - 105.
```

Первым и единственным для этой задачи подготовительным шагом к разработке приложения в Delphi будет формирование получаемого в результате решения задачи выходного документа. Для этого в вычерченной на листе бумаги прямоугольной форме следует разместить всё то, что хотелось бы увидеть в Форме в результате решения задачи (рис. 1.6).

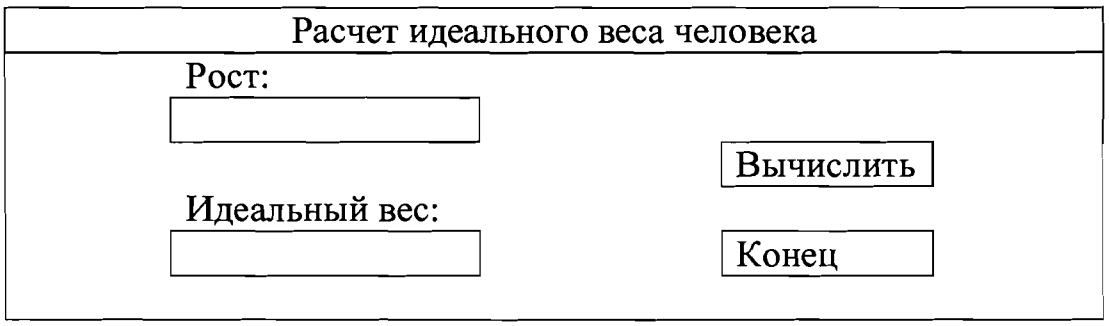

Рис. 1.6. Вид выходного документа

На рисунке выходного документа слева изображены два окна, в которых будут размещаться числовые значения исходной и искомой характеристик. Над каждым окном расположена надпись, определяющая значение размещаемой в нем величины. Справа размещены две управляющие кнопки. Щелчок мышью по верхней из них задает режим работы программы, щелчок по нижней - приводит к завершению работы.

Алгоритм решения рассматриваемой задачи прост и поэтому нет необходимости в его дополнительном описании.

Начало разработки приложения состоит в переносе программистом содержания сформированного на листе бумаги выходного документа в Форму.

Эту работу можно начать с замены имени Формы – `Form1` на заголовок выходного документа (рис. 1.6), хотя этим можно заняться и в любой последующий момент. Для указанной замены в окне Инспектора Объектов следует найти свойство Caption, так как именно оно определяет содержание надписи, а затем убрать из окна его значений надпись `Form1` и вместо неё набрать наименование формируемого документа. В рассматриваемом примере таким наименованием будет: “Расчет идеального веса человека”. По мере набора новое наименование будет появляться в заголовке Формы.

Далее следует перейти к переносу в поле Формы изображенных на рисунке элементов выходного документа.

Процедуру выполнения этой работы рассмотрим на примере размещения в Форме одной из управляющих кнопок. С этой целью следует щелкнуть мышью по пиктограмме **OK** с подсказкой `Button` (кнопка). Затем курсор устанавливается в нужном месте Формы и после очередного щелчка мы-

шью, здесь будет размещена заготовка кнопки с надписью `Button1` (рис. 1.7).

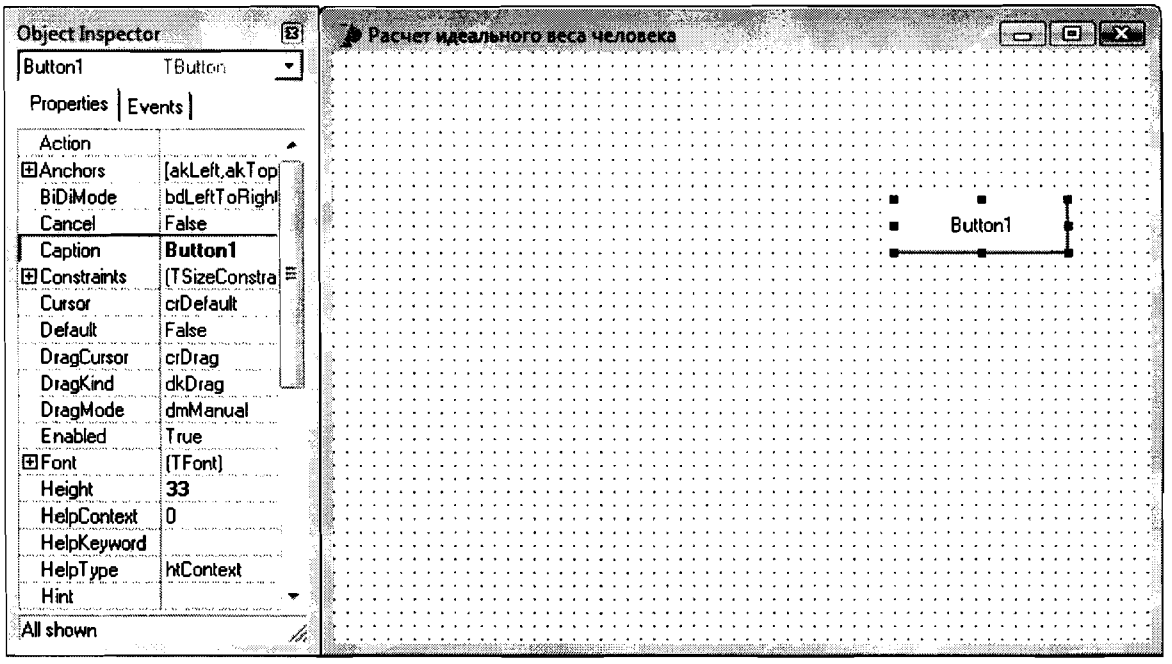

Рис. 1.7. Заготовка кнопки на Форме

Следующим шагом будет изменение наименование кнопки. Для этого следует обратиться к окну Инспектора Объектов, в котором представлен список свойств компонента `Button`. В нем выбирается свойство Caption, так как оно определяет содержание надписи. Заменим в свойстве Caption стандартное значение `Button1` на **Вычислить** (рис. 1.8).

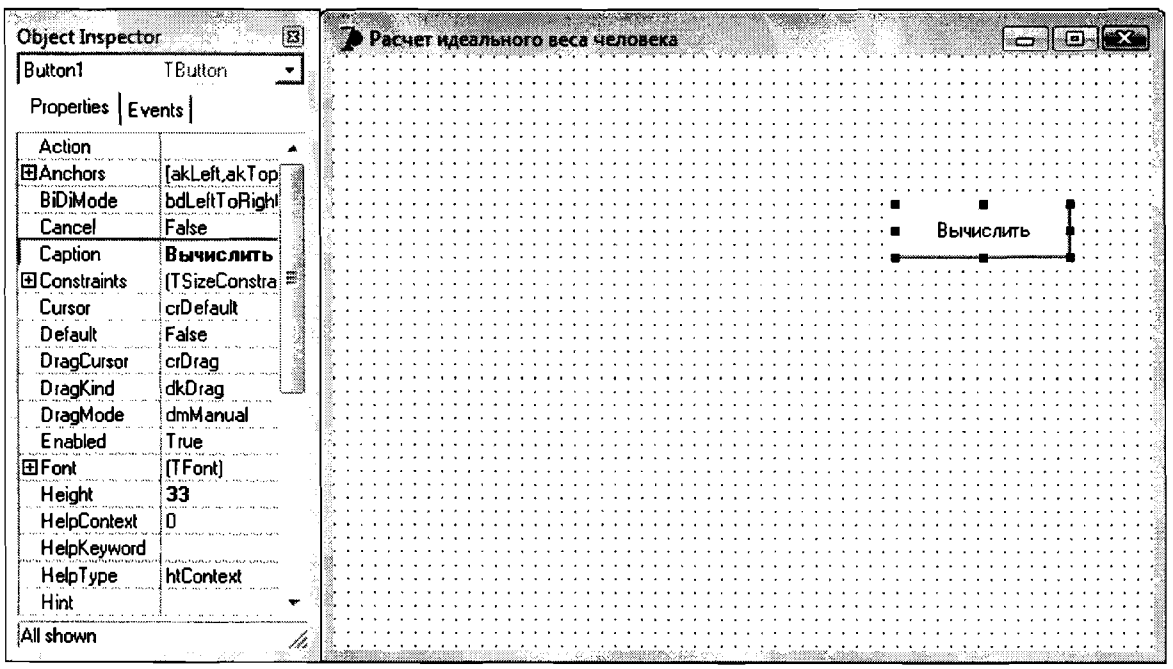

Рис. 1.8. Переименованная кнопка

Точно так же в Форму помещается вторая кнопка с надписью **Конец**.

Для представления значений исходной и искомой характеристик могут быть использованы поля ввода и вывода. Их создание и последующая работа с ними обеспечивается компонентом Edit, пиктограмма которого на панели компонентов имеет вид **ab**. После щелчка по этой пиктограмме в Форме появится поле ввода, содержащее текст `Edit1`. Для удаления этого стандартного наименования в очередной раз следует обратиться к окну Инспектора Объектов и удалить для свойства Text имя `Edit1`. В результате содержимое поля ввода в Форме очистится. Аналогично формируется и поле вывода.

Последние два элемента, которые следует разместить в Форме – это надписи над полями ввода и вывода. Для выполнения этой работы обратимся к компоненту Label, которому соответствует пиктограмма **A**. Разместим вначале надпись над полем ввода. Средой программирования ей будет присвоено имя `Label1`. Соответственно, надпись над полем вывода будет именована как `Label2`. Последовательно дважды обратившись к свойству Caption в окне Инспектора Объектов, заменим стандартные наименования надписей на их обозначения, определенные в выходном документе.

В результате описанной работы в Форме будет сформирован интерфейс будущей программы (рис. 1.9).

Рис. 1.9. Форма для задачи расчета идеального веса человека

Однако интерфейс - это еще не работающая программа. Для проверки нажмем на панели кнопок кнопку с подсказкой Run (Запуск) или выберем в меню команду Run/Run. Этим действием выполняется компиляция и запуск приложения на выполнение. Понажимайте кнопки **Вычислить** и **Конец**. Видно, что они работают. Активизируйте поле ввода и затем с цифровой клавиатуры введите в него какое - либо число. Однако вычисления не производятся, т.е. нет реакции на нажатие кнопок. В Редакторе Кода можно увидеть часть текста программы, сформированную средой программирования. Программисту следует дополнить его, дописав необходимый программный код, обеспечивающий производство вычислений. Для этого предварительно сле-

дует сделать двойной щелчок мышью по кнопке **Вычислить**, вызвав тем на экран окно редактора кода с фрагментом программы работы данной кнопки. Программист должен дописать сюда необходимый для запуска вычислений программный код между служебными словами begin и end:

```
procedure TForm1.Button1Click(Sender:TObject);
begin
   Edit2.Text := IntToStr(StrToInt(Edit1.Text)-105);
end;
```

При внимательном рассмотрении видно, что на языке программирования описан алгоритм вычисления веса. Знакомым с Pascal понятно, что эта запись представляет собой оператор присваивания.

Такую же процедуру следует проделать и с кнопкой **Конец**. Здесь программист должен дописать одно слово Close (окончание, закрытие). Соответствующий фрагмент программы будет иметь вид:

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

Процедура Close, вызываемая при нажатии кнопки **Конец**, завершает работу программы.

После выполнения всех описанных выше работ программа может быть запущена для выполнения. С этой целью следует вновь обратиться к команде Run/Run и запустить программу. Затем активизировать поле ввода и ввести туда величину роста человека (например, 170 см). Для производства вычислений следует щелкнуть по кнопке **Вычислить**. В поле вывода появится результат работы программы (рис. 1.10).

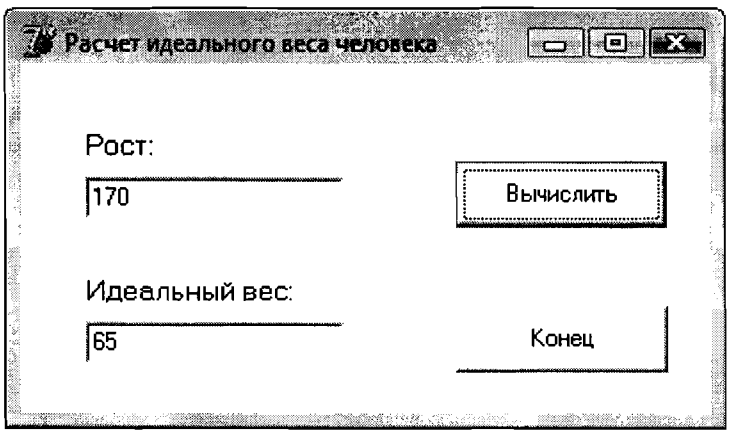

Рис. 1.10. Результат работы программы вычисления веса

При необходимости повторных вычислений следует вновь активизировать окно ввода, задать новое значение роста и получить соответствующий результат. Таким образом, можно последовательно выполнять вычисления для любых наборов значений исходных данных.

Для того чтобы завершить работу программы следует щелкнуть по кнопке **Конец**.

В третьей главе, используя полученные к тому времени знания по Delphi, мы вновь обратимся к рассмотренному выше примеру, сформулируем его в усложненной постановке, представляющей больший практический интерес, рассмотрим соответствующий программный код и получим решение.

1.4. Технологии программирования. Сущность объектно-ориентированного подхода к программированию

Технология программирования - это совокупность методов и средств разработки программ и порядок применения этих методов и средств.

На ранних этапах развития программирования, когда программы писались в виде последовательностей машинных команд, какая-либо технология программирования отсутствовала. С увеличением размеров и усложнением программ в них стали выделять обособленные части, которые оформлялись как подпрограммы. Часть таких подпрограмм объединялась в библиотеки, нужные из которых затем можно было включать в разрабатываемые программы. Это положило начало *процедурному программированию*, в котором создаваемая программа представляется совокупностью процедур-подпрограмм. Одна из подпрограмм является главной, и с нее начинается выполнение программы. Процедурный подход потребовал структурирования будущей программы, разделения ее на отдельные процедуры. При разработке отдельной процедуры о других процедурах требовалось знать только их назначение и способ вызова. Появилась возможность корректировать отдельные процедуры, не затрагивая остальной части программы, сокращая за счет этого затраты труда и машинного времени на разработку и модернизацию программ.

Следующим шагом в углублении структурирования программ стало *структурное программирование*, при котором программа в целом и отдельные процедуры рассматривались как последовательности канонических структур: линейных участков, циклов, разветвлений. Появилась возможность читать и проверять программу как последовательный текст, что повысило производительность труда программистов при разработке и отладке программ. С целью повышения структурности программы были выдвинуты требования к большей независимости подпрограмм. Подпрограммы должны связываться с вызывающими их программами только путем передачи им аргументов, использование в них переменных, принадлежащих другим подпрограммам или главной программе, стало считаться нежелательным.

Процедурное и структурное программирование затронули, прежде всего, процесс описания алгоритма как последовательности шагов, ведущих от варьируемых исходных данных к искомому результату. Для решения специальных задач стали разрабатываться языки программирования, ориентированные на конкретные классы задач: вычислительные задачи, системы управления базами данных, имитационное моделирование и т.д.

Постоянно расширяющееся применение программ в самых разных областях человеческой деятельности привело к необходимости повышения

оперативности написания программ, а также надежности программного обеспечения. Одним из направлений совершенствования программирования стало повышения уровня типизации данных. Теория типов данных исходит из того, что каждое используемое в программе данное принадлежит одному и только одному типу данных. Тип данного определяет множество возможных его значений и набор операций, допустимых над этим данным. Данное конкретного типа в ряде случаев может быть преобразовано в данное другого типа, но такое преобразование должно быть явно представлено в программе. Стремление повысить уровень типизации языка программирования привело к появлению языка Pascal, который считается строго типизированным языком, хотя в нем разрешены некоторые неявные преобразования типов, например, целого в вещественное. Применение строго типизированного языка позволяет еще при трансляции исходного текста написанной программы выявить возможные ошибки использования данных, повышая этим быстроту ее разработки и надежность готового продукта. Все универсальные языки программирования, несмотря на различия в синтаксисе и используемых ключевых словах, реализуют одни и те же канонические структуры: операторы присваивания, циклы и разветвления. Во всех современных языках присутствуют предопределенные (базовые) типы данных (целые и вещественные арифметические типы, символьный и строковый тип), имеется возможность использования агрегатов данных, в том числе массивов и структур (записей). Вместе с тем при разработке программ для решения конкретных прикладных задач, желательна возможно большая концептуальная близость текста программы к описанию задачи. Достижение этого обеспечивается объектно-ориентированным подходом к программированию.

Основополагающей идеей *объектно-ориентированного программирования* является объединение данных и обрабатывающих их процедур в единое целое - *объекты*.

Основные идеи объектно-ориентированного подхода опираются на следующие положения:

- программа представляет собой модель некоторого реального процесса, части реального мира;
- модель процесса может быть описана в программе как совокупность взаимодействующих между собой объектов;
- объект описывается набором параметров, значения которых определяют его состояние, и набором действий, которые могут над ним выполняться или которые может выполнять он;
- взаимодействие между объектами осуществляется посылкой специальных сообщений от одного объекта к другому; сообщение, полученное объектом, может потребовать от него выполнения определенных действий, например, изменения состояния.

Объекты, описанные одним и тем же набором параметров, и способные выполнять один и тот же набор действий представляют собой класс однотипных объектов. Соответственно любой объект является экземпляром некоторого класса. С позиций объектно-ориентированного подхода к програм-

мированию класс объектов можно рассматривать как тип данного, а отдельный объект - как данное этого типа.

Класс представляет собой структуру, в которой объединены данные, определяемые как поля, и методы – процедуры и функции, с помощью которых реализуется работа класса. Можно сказать, что поля определяют состояние объекта, а методы характеризуют функциональные возможности объекта, создаваемого на основе соответствующего класса. Основная идея такой структуры заключается в том, чтобы избежать повторений написания программного кода, а использовать созданные ранее фрагменты кода для выполнения необходимых функций. Поэтому класс можно рассматривать как шаблон или прототип, на основе которого создаются объекты. Применительно к такому пониманию, объект – это место в памяти, где хранятся данные, с ним связанные, а весь код для их обработки берется из соответствующего класса. То есть код, который может иметь различные размеры, существует в единственном экземпляре.

Возможности использования классов не определяются только изложенным выше. Очень часто при изменении постановок задач возникает необходимость внесения корректировок в их алгоритмы, а затем, соответственно, и в разрабатываемые программы. Объектно-ориентированный подход предусматривает для подобных ситуаций возможность создания на основе одного класса, определяемого как базовый или исходный, других классов, внося в них только необходимые изменения. Программный код исходного класса остается при этом неизменным, а код класса, созданного на его основе, будет содержать только новые свойства, не дублируя содержания исходного. При этом вновь созданный класс будет обладать всеми возможностями исходного класса, или, говоря более строго, наследует его возможности.

Классы объектов часто строятся так, чтобы они образовывали иерархическую структуру. Например, класс “Студент”, описывающий абстрактного студента, может служить основой для построения классов “Студент 1 курса”, “Студент 2 курса” и т.д., которые обладают всеми свойствами студента вообще и некоторыми дополнительными свойствами, характеризующими студента конкретного курса. Аналогично, при разработке пользовательского интерфейса могут использоваться как программный код базового класса “Окно”, так и коды классов специальных окон, например, окна информационных сообщений, окна ввода данных и т.п.

Объект производного класса обладает всеми свойствами базового класса и некоторыми собственными свойствами. Он может реагировать на те же типы сообщений от других объектов, что и объект базового класса и на сообщения, имеющие смысл только для того производного класса, к которому он непосредственно принадлежит. Применительно к этому механизм работы объектно-ориентированного программирования описывается изложенной ниже последовательностью действий. При вызове какого-то метода для работы с объектом этот метод сначала ищется в классе, к которому объект отнесен. Если метод найден, то он запускается на выполнение. В противном случае производится обращение к родительскому классу, где вновь произво-

дится поиск необходимого метода. При отрицательном результате дальнейший поиск продолжается вверх по иерархическому дереву, вплоть до исходного класса. Именно в этом заключается идеология объектно-ориентированного программирования.

При разработке приложений (прикладных программ) объект можно рассматривать как отдельный элемент программы, который может выполнять определенные задачи и свойства которого можно изменять. В частности, объектами являются и окно формы (Form) и размещаемые в нем компоненты: Button, Edit, Label и другие.

Применительно к изложенному, каждый компонент может рассматриваться как функциональный элемент графического интерфейса, обладающий определенными свойствами и являющийся отдельным строительным блоком при разработке приложений. С позиций языка объектно-ориентированного программирования компоненты представляют собой классы, порожденные прямо или косвенно от класса TComponent.

Объединение этих двух определений дает цельное представление, что такое компонент. При работе с компонентами из среды визуальной обработки видна их лицевая сторона. Однако, как только пользователь начинает управлять компонентами программно, он сталкивается с их программной стороной. Таким образом, среда Delphi обеспечивает симбиоз визуального и объектно-ориентированного программирования.

Для того чтобы продуктивно применять объектно-ориентированный подход для разработки программ, потребовались языки программирования, поддерживающие этот подход, т.е. позволяющие строить описание классов объектов, образовывать данные объектных типов, выполнять операции над объектами. Одним из первых языков такого типа стал Object Pascal, развившийся в язык Delphi. В этих языках все данные являются объектами некоторых классов, а общая система классов строится как иерархическая структура на основе предопределенных базовых классов.

Ниже рассматриваются базовые классы, на которых строятся все конструкции среды программирования Delphi.

Класс TObject

Все объекты языка Object Pascal являются производными от базового (родительского класса) `TObject`, а, следовательно, автоматически наследуют все его методы. В результате любой объект способен, например, сообщить свое имя, тип и указать, является ли он производным от того или иного класса. Очень удачным во всем этом механизме является то, что программист избавлен от необходимости беспокоиться о реализации стандартных функций – все их можно непосредственно использовать по своему прямому назначению. *Данный класс лежит в основе всей иерархии классов Delphi.* Он обладает самыми общими методами, присущими любому объекту языка Object Pascal. Класс TObject описывает основные принципы поведения объектов

во время работы программы (создание, уничтожение, обработка событий и т.д.).

Класс TPersistent

Класс `TPersistent` является прямым наследником класса `TObject`. На уровне этого класса реализованы основные методы копирования содержимого объектов.

Класс TComponent

Класс `TComponent` является прямым наследником класса `TPersistent` и наследником класса `TObject`. Класс `TComponent` является основным родительским классом для всех классов, описывающих компоненты Delphi. В этот класс входит набор самых общих свойств, имеющихся у каждого компонента Delphi .

Класс TControl

Класс `TControl`, наследник `TComponent`, служит основным классом для всех визуальных элементов управления. Если такой элемент управления является стандартным элементом Windows, то он базируется на еще одном промежуточном классе `TWinControl`(наследнике класса `TControl`).

Класс TForm

Форма – это важнейший компонент Delphi, на котором основана вся работа этой системы по проектированию и разработке приложений. Форма (Класс *TForm*) содержит обширный набор свойств, методов и событий, позволяющих легко настраивать и организовывать самые сложные алгоритмы её функционирования.

Поскольку пользователю больше всего приходится работать именно с Формой, то знание ее основных свойств, представленных в приведенной ниже таблице, существенно повысит эффективность его работы.

Таблица 1.1. Свойства класса TForm и их назначение

Свойство	Назначение
`ActiveControl`	Устанавливает фокус для элемента на форме
`Align`	Изменяет размеры и положение формы
`AlphaBlend`, `AlphaBlendValue`	Определяют прозрачность формы
`Anchors`	Определяют угол формы для привязки к координатам
`AutoShroll`	Если `True`, то полосы прокрутки только при необходимости

Продолжение табл.1.1

Свойство	Назначение
AutoSize	Если True, то границы могут изменяться автоматически при изменении содержимого
BiDiMode	Двунаправленный режим для порядка считывания
Active	Содержит значение True, если форма имеет фокус ввода
BorderIcons	Определяет пиктограмму в заголовке окна
BorderStyle	Вид границ формы
BorderWidth	Ширина рамки
Caption	Название формы, помещаемое в заголовке
Canvas	Область рисования формы
ClientHeight, ClientWidth	Размеры клиентской части формы (без заголовка)
Color	Цвет формы
Constraints	Ограничители, устанавливающие пределы автоматического изменения размеров формы
Ctl3D	Вид формы – объемный (3D) или нет
Cursor	Определяет вид курсора при наведении указателя мышки на форму
DefaultMonitor	Определяет монитор, в котором отображается форма
DockSite	Содержит значение True, если к форме разрешено "пристыковываться" другим окнам
DropKind, DragMode	Определяют возможности формы при операциях перетаскивания элементов
Enabled	Определяют реакцию формы на события мыши, клавиатуры и таймеров
Font	Установка шрифтов для формы
FormStyle	Стиль формы
Heigth	Высота формы с заголовком и рамкой
HelpContex	Используется для организации справочника
HelpFile	Название файла справки для формы
HelpKeyword	Ключевое слово для справочника
HelpType	Используется для организации справочника
Hint	Содержит текст этикетки, появляющейся при наведении на форму указателя мыши
HorzShrollBar	Свойства горизонтальной полосы прокрутки
Icon	Пиктограмма, обозначающая форму, когда она свернута
KeyPreview	Содержит значение True, если форма должна получать информацию о нажатых клавишах раньше, чем расположенные на ней объекты
Left	Координаты угла привязки
Menu	Ссылка на главное меню формы (TMenu)
ModalResult	Значение, возвращаемое формой, если она работает как модальное диалоговое окно
Name	Идентификатор (имя) формы для обращения к ней в программе
OldCreateOrder	Определяет момент выполнения событий **OldCreate** и OnDestroy относительно конструктора и деструктора
ParentBiDiMode	Использование режима, установленного в базовом классе

Окончание табл.1.1

Свойство	Назначение
ParentFont	Использование режима, установленного в базовом классе
PixelsPerInch	Число пикселей на дюйм. Применяется для настройки размера формы в зависимости от экранного разрешения
Position	Положение формы на экране в момент ее открытия в программе
PrintScale	Масштабирование формы при выводе на печать
Scaled	Содержит значение `True`, если размер формы будет подгоняться в соответствии со значением свойства `PixelsPerInch`
ScreenSnap	Разрешение на стыковку с границей экрана
ShowHints	Разрешение отображения этикетки
SnapBuffer	Установка зоны в пикселях для стыковки с границей экрана
Tag	Связывает числовое значение с формой
Top	Координаты угла привязки
TransparentColor	Разрешает подсветку при установленном режиме прозрачности
TransparentColorValue	Определяет цвет подсветки
UseDockManager	Разрешение режима стыковки при перетаскивании
Visible	Свойства вертикальной полосы прокрутки
Wigth	Содержит значение `True`, если форма будет видима во время работы программы
WindowState	Ширина формы с рамкой
	Состояние формы (свернута, развернута, нормальный размер)

Знание приведенных в таблице свойств облегчит пользователю работу со многими другими объектами, так как и для них также используется большинство рассмотренных применительно к Форме свойств.

Форма может отображаться в двух видах: как графическое отображение и как текстовый файл. Для переключения между режимами необходимо щелкнуть по Форме правой кнопкой мыши, а затем в появившемся контекстном меню выбрать пункт `View as Text` (Отображать как текст) или `View as Form` (Отображать как форма). В текстовом режиме будут показаны все свойства Формы с установленными значениями, а также свойства всех входящих в нее элементов управления, т.е. тех элементов, для которых Форма является хозяином.

Следует заметить, что практически невозможно кратко, но понятно описать все свойства Формы. Необходимо самому попробовать все доступные ее значения и понять их действие. Следует самому общаться с компьютером, так как, только изучая справочную информацию невозможно изучить графический интерфейс Delphi.

В заключение рассмотрим наиболее часто используемые компоненты, предварительно заметив, что с их помощью можно создавать графические интерфейсы многих приложений, почти не прибегая к ручному кодированию.

Для этого нужно знать не только их свойства и методы, но и хорошо понимать организацию использования для создания необходимого графического интерфейса.

Компонент TLabel

Компонент `TLabel` (Надпись) служит для отображения коротких надписей в форме. Обычно надписи используют как заголовки к другим компонентам, но часто их используют и как табло для отображения информации. Для размещения в форме компонента следует выполнить следующие действия:

1. Выбрать вкладку Standart и щелкнуть по пиктограмме компонента `TLabel` (буква **A**).
2. Щелкнуть на Форме в том месте, где должна размещаться надпись. Особенно точным быть не обязательно, компонент всегда можно перемещать по Форме.

После размещения компонента в Форме в Инспекторе Объектов будут представлены его свойства. Большинство из них те же, что и для Формы. Список свойств начинается со свойства `Align`, для которого существует семь значений. Обратившись последовательно к каждому из них можно увидеть, как изменяется отображение надписи на Форме. Таким же образом можно рассмотреть и содержание остальных свойств.

Компонент TButton

Компонент `TButton` (Кнопка) также расположен на вкладке Standard. Проделайте самостоятельно те же операции, что и для надписи, и поместите кнопку в форму. Посмотрите на свойства кнопки в Инспекторе Объектов, которые аналогичны свойствам надписи.

Компонент TEdit

У компонента `TEdit` (Текстовое поле) свойства `Caption` (Заголовок) нет. Вместо него используется свойство `Text` (Текст). Это свойство содержит введенные пользователем данные в текстовом виде (тип `string`). Первоначально это свойство содержит строку, совпадающую с именем элемента управления (`Edit1`, `Edit2` и т.д.). Текущее содержимое свойства Text (Текст) каждого текстового поля формы лучше удалить и ввести вместо этого пустую строку. Изменять содержимое поля можно с помощью свойства Text как из Инспектора Объектов, так и из программы.

Следует еще раз обратить внимание на то, что и свойство Text и свойство Caption имеют тип string. В соответствии с этим и числовые исходные данные и полученные в результате работы по программе искомые характеристики задаются и выводятся как переменные строкового типа. Поэтому в

программе после ввода для обработки должно быть предусмотрено преобразование переменных строкового типа в числовые данные, целые или вещественные, а затем для отображения в Форме результатов работы программы требуется их обратное преобразование.

Компонент TMemo

Обойтись надписями для вывода информации можно только в простейших случаях. Если же необходимо вывести большой объем информации (например, полный почтовый адрес или произвольный комментарий), то понадобится несколько строк текста. В таком случае следует использовать компонент `TMemo` (Текстовая область), расположенный на вкладке Standard.

У него кроме уже знакомых свойств есть и специфические свойства, такие как, например, WantReturns, которое необходимо для установки режима использования клавиши <Enter>. При вводе текста для перехода на новую строку (к новому абзацу) обычно используется эта клавиша. Однако в диалоговых окнах Windows указанная клавиша часто применяется для завершения ввода. Способ использования клавиши <Enter> и определяется значением свойства WantReturns. Если оно имеет значение True, то клавиша <Enter> позволяет переходить к новой строке внутри текстовой области, в противном случае она служит для завершения ввода и перехода к следующему элементу управления.

Рассмотренные выше компоненты имеют первоочередное значение для пользователей. Общее же число компонент в Delphi достаточно велико. Часть из них будет рассматриваться в процессе занятий по изучению системы, остальные, по мере надобности, придется осваивать в будущем самостоятельно.

Глава 2. Основные понятия языка Delphi

2.1. Состав языка

Delphi как и любой другой формализованный язык включает алфавит и создаваемые из него по определенным правилам типовые конструкции.

Алфавит языка Delphi содержит следующие символы:

•латинские буквы: `A,B,C,…,x,y,z;`

•цифры:`0,1,2,…,9;`

• специальные символы: `+ , - , / , = , <, > , [ ,] , . , ( , ),; , : , { , } , $ , # , _ , @ , ' , ^ .`

Из символов алфавита формируют ключевые слова и идентификаторы. *Ключевые слова* – это зарезервированные слова, которые имеют специальное значение для компилятора и используются только в том смысле, в котором они определены (операторы языка, типы данных и т.п.). *Идентификатор* – это имя программного объекта. К программным объектам относятся:

константы, переменные, метки, типы данных, объекты, классы, свойства, процедуры, функции, модули и программы. Идентификатор представляет собой совокупность букв, цифр и символа подчеркивания. Первый символ идентификатора – буква или знак подчеркивания, но не цифра. Идентификатор не может содержать пробел. Прописные и строчные буквы в именах не различаются, например `ABC`, `abc` , `Abc` – одно и то же имя. Каждое имя (идентификатор) должно быть уникальным и не совпадать с ключевыми словами. Идентификатор может иметь любую длину, однако в Delphi только первые его 255 символов являются значимыми.

В тексте программы можно использовать комментарии. Если текст начинается с двух символов "косая черта" `//` и заканчивается символом перехода на новую строку, заключен в фигурные скобки `{}` или располагается между парами символов `(* и *)`, то компилятор его игнорирует. Например:

```
{Комментарий может
 выглядеть так.}
(* или так *)
// А если вы используете такой способ,
// то каждая строка должна начинаться с двух символов
"косая черта".
```

Комментарии удобно использовать как для пояснений к программе, так и для временного исключения фрагментов программы при отладке.

2.2. Ключевые слова

Ниже в таблице 2.1 приведен перечень наиболее часто используемых ключевых слов (далеко не всех) с кратким комментарием об их назначении.

Таблица 2.1.Ключевые слова

Ключевое слово	Комментарий
and	Булев оператор И
array	Массив
begin	Начало блока
case	Оператор выбора. Используется при выборе из многих вариантов
class	Определяет тип *класс*
const	Определяет константы, т.е. изменяемые переменные, но в Delphi есть режим, допускающий изменение констант в теле программы
div	Целочисленное деление
do	Определяет начало исполнимой части в операторах цикла
downto	Определяет направление итерации в операторе for
else	Используется в операторах выбора case и условном операторе if
end	Обычно используется совместно с ключевым словом begin и отмечает конец блока
file	Устанавливает тип переменной как файл. Используется при работе с файлами
for	Используется в операторах цикла for…to и for…downto

Окончание табл.2.1

Ключевое слово	Комментарий
function	Используется при объявлении функции
goto	Переход на метку
if	Используется в операторах выбора if…then и if…then…else
interface	Определяет тип интерфейса. Используется при опережающем объявлении интерфейса
label	Метка. Используется совместно с ключевым словом goto. Может быть выражена любым идентификатором или числом от 0 до 9999
mod	Остаток от деления целых чисел
not	Булев оператор отрицания
of	Используется во многих операторах как связующее ключевое слово
or	Булев оператор ИЛИ
out	Используется при объявлении процедуры, функции или метода. Предупреждает о том, что данный параметр используется только для выдачи значений
procedure	Используется при объявлении процедур
program	Определяет имя программы, которое должно быть выражено идентификатором
record	Определяет тип *запись*
repeat	Используется в операторе цикла repeat…until
set	Ключевое слово для объявления множества
shl	Логический оператор сдвига влево
shr	Логический оператор сдвига вправо
string	Используется при объявлении строковых типов
then	Используется в операторах if…then и if…then…else
to	Используется в операторе for…to
type	Определяет раздел объявления типов
unit	Модуль. Обычно это функционально законченный фрагмент программы, сохраняемый в файле с таким же именем
until	Используется в операторе repeat…until
uses	Определяет раздел подключаемых модулей
var	Определяет раздел переменных
while	Используется в операторе while…do
xor	Булев оператор Исключающее ИЛИ

2.3.Понятие данных. Типы данных в Delphi

Для решения задачи в любой программе выполняется обработка каких-либо данных. *Данные* хранятся в памяти компьютера и могут быть самых различных типов: целыми и вещественными числами, символами, строками, массивами. *Типы данных* определяют способ хранения чисел или символов в памяти компьютера. Они задают размер ячейки, в которую будет записано то или иное значение, определяя тем самым его максимальную величину или точность задания. Участок памяти (ячейка), в котором хранится значение определенного типа, называется *переменной*. У переменной есть *имя* (идентификатор) и *значение*. Имя служит для обращения к области памяти, в которой

хранится значение. Во время выполнения программы значение переменной можно изменить. Перед использованием любая переменная должна быть описана. Описание выполняется в определенном разделе программы сразу же после ее заголовка. *Описание переменной* в Delphi осуществляется с помощью служебного слова `var`:

```
var   имя переменной: тип переменной;
```

Если объявляется несколько переменных одного типа, то описание выглядит следующим образом:

```
var
переменная1, переменная2, … , переменная N: тип переменных;
```

Например:

```
var a: integer;        //объявлена целочисленная переменная.
b, c: real;          //объявлены две вещественные переменные.
```

Константа - это величина, которая не изменяет своего значения в процессе выполнения программы. *Описание константы* в Delphi имеет вид:

```
Const   имя константы = значение;
```

Например:

```
Const
      h=3;            // целочисленная константа.
      b=-7.5;         // вещественная константа.
      c='abcde';     // символьная константа.
```

Символьный тип данных

Данные символьного типа в памяти компьютера всегда занимают один байт. Это связанно с тем, что обычно под величину символьного типа отводят столько памяти, сколько необходимо для хранения любого из 256 символов клавиатуры.

Описывают символьный тип с помощью служебного слова `char`.

Например:

```
var   c: char;
```

В тексте программы значение переменных и константы символьного типа должны быть заключены в апострофы: `'a', 'b', '+'`.

Целочисленный тип данных

Данные целочисленного типа хранятся в памяти компьютера как целые числа. Диапазон значений данных этого типа, занимающих в памяти компьютера от одного до восьми байтов, представлен в таблице 2.2.

Таблица 2.2. Целочисленные типы данных

Тип	Диапазон	Размер (байты)
Byte	0...255	1
Word	0...65535	2
LongWord	0...4294967295	4
ShortInt	-128...127	1

Окончание табл.2.2

Тип	Диапазон	Размер (байты)
Integer	-2147483648...2147483647	4
LongInt	-2147483648...2147483647	4
Int64	$-2^{63}...2^{63}$	8
Cardinal	0...4294967295	4

Описание целочисленных переменных в программе может быть таким:

```
var
 i, j: integer;
 W: cardinal;
 L1, L2: longint;
```

При программировании целесообразно использовать данные типа Integer и Cardinal, поскольку они позволяют достичь максимальной производительности программы при переходе на новые модели компьютеров, например, компьютеров, построенных на основе новых 64 – разрядных процессоров.

Вещественный тип данных

Данные вещественного типа хранятся в памяти компьютера в форме чисел с плавающей точкой. При этом число представлено в экспоненциальной форме записи вида $mE\pm p$,где m – мантисса (целое или дробное число с десятичной точкой), p – порядок (целое число). Операции над числами с плавающей точкой отнимают у процессора больше времени и требуют больше памяти, чем целые числа. Поэтому переменные вещественного типа рекомендуется использовать только тогда, когда нельзя ограничиться использованием целочисленных переменных.

Вещественное число в Delphi может занимать от четырех до десяти байтов. Диапазоны значений вещественного типа представлены в табл. 2.3.

Таблица 2.3. Вещественные типы данных

Тип	Диапазон	Размер (байты)
Single	1,5E – 45...3,4E + 38	4
Real	2,9E – 39...1,7E + 38	8
Double	5,0E – 324...1,7E +308	8
Extended	3,4E – 4932...3,4E + 4932	10

Примеры описания вещественных переменных:

```
var
r1, r2: real; D: double;
```

Применительно к типу Real можно сказать то же, что было сказано выше относительно Integer и Cardinal.

Тип дата – время

Тип данных дата – время `TDateTime` предназначен для одновременного хранения даты и времени. В памяти компьютера он занимает восемь байтов и фактически представляет собой вещественное число, где в целой части хранится дата, а в дробной – время.

Логический тип данных

Данные *логического типа* могут принимать только два значения: истина `(true)` или ложь `(false)`. В стандартном языке Паскаль был определен лишь один логический тип данных – `boolean`.Логические типы данных, определенные в Delphi, представлены в табл. 2.4

Таблица 2.4. Логические типы данных

Тип	Размер (байты)
Boolean	1
ByteBool	1
Bool	2
WordBool	2
LongBool	4

Пример объявления логической переменной:

```
var   FL: Boolean;
```

Создание новых типов данных

Несмотря на достаточно мощную структуру встроенных типов данных, в Delphi предусмотрен механизм создания новых типов. Для этого используют служебное слово `type`:

```
type   новый тип данных = определение типа;
```

Когда новый тип данных создан, можно объявлять соответствующие ему переменные:

```
var   переменных : новый тип данных;
```

Применение этого механизма мы рассмотрим в следующих разделах.

Перечислимый тип данных

Перечислимый тип задается непосредственным перечислением значений, которые он может принимать:

```
var имя переменной:(значение 1, значение 2, … , значение N
);
```

Такой тип может быть полезен, если необходимо описать данное, которое принимает ограниченное число значений. Например:

```
var
   animal:   (cat , dog) ;
```

```
        color:   (red , black , white) ;
```

Применение перечислимых типов делает программу нагляднее:

```
type          // Создание нового типа данных - времена года.
    year times = (winter , spring , summer , autumn );
    var           // Описание соответствующей переменной.
       yt: year times;
```

Интервальный тип

Интервальный тип задается границами своих значений внутри базового типа:

```
var   имя переменной: минимальное значение типа ..
максимальное значение типа;
```

Обратите внимание, что в данной записи два символа точки рассматриваются как один, поэтому пробела между ними не допускается. Кроме того, левая граница диапазона не должна превышать правую. Например:

```
var
   date: 1.. 31;
   symb: 'a' .. 'h';
```

Применим механизм создания нового типа данных:

```
type
    // Создание перечислимого типа данных - дни недели.
    Week days = ( Mo , Tu , We , Th , Sa , Su);
    // Создание интервального типа данных - рабочие дни.
    Working days = Mo .. Fr;
var
   days: Working days;
```

Структурированные типы

Структурированный тип данных характеризуется множественностью образующих его элементов. В Delphi это *массивы, строки, записи, множества* и *файлы.*

Массив – совокупность данных одного и того же типа. Число элементов массива фиксируется при описании типа и в процессе выполнения программы не изменяется.

Для *описания массива* используют ключевые слова `array … of:`

```
имя массива: array  [список индексов]  of тип данных;
```

где

- имя массива – любой допустимый в Delphi идентификатор;
- тип данных – любой тип, используемый в Delphi.
- список индексов – перечисления диапазонов измерения номеров элементов массива; количество диапазонов совпадает с количеством измерений массива; диапазоны отделяются друг от друга запятой, а границы диапазона – двумя символами точки:

```
[индекс1 нач .. индекс1 кон, индекс2 нач .. индекс2 кон, … , ]
```

Например:

```
var
   // Одномерный массив из 10 целых чисел.
   a: array [1..10]  of  byte;
   // Двухмерный массив вещественных чисел (3 строки, 3
столбца).
   b: array [1..3, 1..3]   of   real;
```

Еще один способ описать массив – создавать новый тип данных. Например, так:

```
type
    //объявляется новый тип данных - трехмерный массив це-
лых
    чисел.
    massiv = array  [0..4, 1..2,  3..5]  of word;
 var   // Описывается переменная соответствующего типа.
    M: massiv;
```

Для доступа к элементу массива достаточно указать его порядковый номер, а если массив многомерный (например, таблица), то несколько номеров:

```
имя массива [номер элемента]
```

Например: `a [5], b [2, 1], M[3, 2, 4].`

Строка – последовательность символов. В Delphi строка трактуется как массив символов, то есть каждый символ строки пронумерован, начиная с единицы.

При использовании в выражениях строка заключается в апострофы. Описывают переменные строкового типа так:

```
имя переменной:  string;
```

или

```
имя переменной:  string (длина строки);
```

Например:

```
const   S='СТРОКА';
  var
      Str1: string;
      Str2: string[ 255 ];
      Stroke: string [100];
```

Если при описании строковой переменной длина строки не указывается, это означает, что она составляет 255 символов. В приведенном примере строки `Str1` и `Str2` одинаковой длины.

Запись – это структура данных, состоящая из фиксированного количества компонентов, называемых полями записи. В отличие от массива поля записи могут быть различного типа. При объявлении типа записи используют ключевые слова `record … end`:

```
имя записи = record  список полей end;
```

Здесь имя записи – любой допустимый в Delphi идентификатор, список полей – описания полей записи. Например:

```
// Описана структура записи.
// Каждая запись содержит информацию о студенте и
// состоит из двух полей - имя и возраст.
type
    student = record
```

```
            name: string;
             age: byte;
        end;
    var  // Объявлены соответствующие переменные - три объекта
типа запись.
         a, b, c: student;
```

Доступ к полям записи осуществляется с помощью составного имени:

```
имя записи. имя поля
```

Например:

```
a. name: = 'Ivanov Ivan';
a. age: = 18;
b. name: = a.name;
```

Файл – это именованная область внешней памяти компьютера. Файл содержит компоненты одного типа (любого типа Delphi, кроме файлов). Длина созданного файла не оговаривается при его объявлении и ограничивается только емкостью диска, на котором он хранится. В Delphi можно объявить *типизированный файл*:

```
имя файловой переменной = file  of  тип данных;
```

нетипизированный файл:

```
имя файловой переменной = file;
```

и текстовый файл:

```
имя файловой переменной = TextFile;
```

Например:

```
var
f1: file of byte;
f2: file;
f3: TextFile;
```

2.4. Выражения и операции

Обработка данных в программе выполняется в выражениях. *Выражение* – это фрагмент программы, задающий порядок действий над данными и обеспечивающий вычисление некоторого значения. Оно может быть как числом, целочисленным или вещественным, так и характеристикой логического типа.

Например:

$(t + \sin(x))/2 \times t$ – результат вещественного типа;

a < = b + 2 – результат логического типа.

Выражение состоит из операндов (констант, переменных, обращений к функциям), круглых скобок и знаков операций. В таблице 2.5 представлены основные операции Delphi.

Таблица 2.5. Основные операции Delphi

Знак операции	Действие	Тип операндов	Тип результата
+	сложение	целый/ вещественный	целый/ вещественный
+	сцепление строк	строковый	строковый
-	вычитание	целый/ вещественный	целый/ вещественный
*	умножение	целый/ вещественный	целый/ вещественный

Окончание табл.2.5

Знак операции	Действие	Тип операндов	Тип результата
/	деление	целый/ вещественный	вещественный
div	целочисленное деление	целый	целый
mod	остаток от деления	целый	целый
not	отрицание	целый/ логический	целый/ логический
and	логическое И	целый/ логический	целый/ логический
or	логическое ИЛИ	целый/логический	целый/логический
xor	исключающее ИЛИ	целый/логический	целый/логический
shl	сдвиг влево	целый	целый
shr	сдвиг вправо	целый	целый
in	вхождение в множество	множества	логический
<	меньше	любой	логический
>	больше	любой	логический
< =	меньше или равно	любой	логический
> =	больше или равно	любой	логический
=	равно	любой	логический
< >	не равно	любой	логический

В сложных выражениях порядок, в котором выполняются операции, соответствует приоритету операций. В Delphi приняты следующие *приоритеты*:

1. not
2. *, /, div, mod, and, shl, shr.
3. +, - , or, xor .
4. =, < >, > ,< , > = , < = .

Использование скобок в выражениях позволяет менять порядок вычислений. Перейдем к рассмотрению основных операций языка.

Арифметические операции

Операции + , - , * , / относят к *арифметическим операциям*. Их назначение понятно и не требует дополнительных пояснений.

Операции целочисленной арифметики (применяется только к целочисленным операндам):

- div – целочисленное деление (возвращает целую часть частного, дробная часть отбрасывается): 11div4 = 2;
- mod – остаток от деления: 11mod4 = 3.

Операции отношения

Операции отношения применяются к двум операндам и возвращают в качестве результата логическое значение. Таких операций семь: >, > =, <, < =, =, < >, in. Результат операции отношения – логическое значение true (истина) или false (ложь).

Назначение операций >, > =, <, < =, =, < > понятно (см. табл. 2.5). Поясним, как работает операция in. Первым операндом этой операции должно быть любое выражение, вторым – множество, состоящее из элементов того же типа. Результат операции true (истина), если левый операнд принадлежит множеству указанному справа.

Логические операции

В Delphi определены следующие *логические операции* or, and, xor, not. Логические операции выполняются над логическими значениями true (истина) и false (ложь). В таблице 2.6 приведены результаты логических операций.

Таблица 2.6. Логические операции

A	B	Not A	A and B	A or B	A xor B
t	t	f	t	t	f
t	f	f	f	t	t
f	t	t	f	t	t
f	f	t	f	f	f

В логических выражениях могут использоваться операции отношения, логические и арифметические.

2.5.Стандартные функции

В язык Delphi включено большое количество разнообразных стандартных функций. В частности, в таблице 2.7 приведен ряд часто используемых арифметических стандартных функций.

Таблица 2.7. Арифметические функции

Обозначение	Тип аргументов	Тип результата	Действие
	Стандартные арифметические функции		
abs (x)	целый/ вещественный	целый/ вещественный	модуль числа
sin (x)	вещественный	вещественный	синус
cos (x)	вещественный	вещественный	косинус
arctan (x)	без аргумента	вещественный	арктангенс
pi	вещественный	вещественный	число π
exp (x)	вещественный	вещественный	экспонента e^x
ln (x)	вещественный	вещественный	натуральный логарифм
sqr (x)	вещественный	вещественный	квадрат числа
sqrt (x)	вещественный	вещественный	корень квадратный
int	вещественный	вещественный	целая часть числа
frac (x)	вещественный	вещественный	дробная часть числа
round (x)	вещественный	целый	округление числа

Окончание табл.2.7

Обозначение	Тип аргументов	Тип результата	Действие
`trunc (x)`	вещественный	целый	отсекание дробной части числа
`random (x)`	целый	целый	случайное число от 0 до n
Функции, определенные в программном модуле **`Math`**			
`acros (x)`	вещественный	вещественный	арккосинус
`arcsin (x)`	вещественный	вещественный	арксинус
`arccot (x)`	вещественный	вещественный	арккотангенс
`arctan2(y,x)`	вещественный	вещественный	арктангенс y/x
`cosecant (x)`	вещественный	вещественный	косеканс
`sec (x)`	вещественный	вещественный	секанс
Функции, определенные в программном модуле **`Math`**			
`cot (x)`	вещественный	вещественный	котангенс
`tan (x)`	вещественный	вещественный	тангенс
`lnXP1 (x)`	вещественный	вещественный	логарифм натуральный от (x + 1)
`log10 (x)`	вещественный	вещественный	десятичный логарифм
`log2 (x)`	вещественный	вещественный	логарифм по основанию два
`logN (n, x)`	вещественный	вещественный	логарифм от x по основанию n

Функции из нижней части таблицы будут работать только в том случае, если в тексте основной программы после ключевого слова `Unit` указать имя `Math`.

Определенную проблему представляет возведение X в степень n. Если значение степени n – целое положительное число, то можно n раз перемножить X (что дает более точный результат и при целом n предпочтительней) или воспользоваться формулой:

$$\begin{cases} X^n = e^{n\ln(x)}, x > 0 \\ X^n = -e^{n\ln|x|}, x < 0 \end{cases},$$

которая программируется с помощью стандартных функций языка:

`exp (n*ln(x))` – для положительного x;

`exp (n*ln(abc(x))` – для отрицательного x.

Данную же формулу можно использовать для возведения `x` в дробную степень n, где `n` – обыкновенная правильная дробь вида `k/l`, а знаменатель `l` нечетный. Если знаменатель `l` четный, это означает извлечение корня четной степени, следовательно, есть ограничения на выполнение операции.

При возведении числа X в отрицательную степень, следует помнить, что

$$x^{-n} = \frac{1}{x^n}.$$

Таким образом, для программирования выражения, содержащего возведение в степень, надо внимательно проанализировать значения, которые

могут принимать X и n, так как в некоторых случаях возведение X в степень n невыполнимо.

Ряд стандартных функций, предназначенных для *работы со строками*, представлен в таблице 2.8.

Таблица 2.8. Функции обработки строк

Обозначение	Тип аргументов	Тип результата	Действие
		Работа со строками	
length (S)	строка	целое	текущая длина строки S
concat (S1, S2, ...)	строки	строка	объединение строк S1, S2
copi (S, n, m)	строка, целое, целое	строка	копирование n символов строки S, начиная с m–й позиции
delete (S, n, m)	строка, целое, целое	строка	удаление n символов из строки S, начиная с m–й позиции
insert (S, n, m)	строка, целое, целое	строка	вставка n символов в строку S, начиная с m–й позиции
pos (S1, S2)	строки	целое	номер позиции, с которой начинается вхождение S2 в S1
		Преобразование строк в другие типы	
StrToDateTame (S)	строка	дата и время	преобразует символы из строки S в дату и время
StrToFloat (S)	строка	вещественное	преобразует символы из строки S в вещественное число
StrToInt (S)	строка	целое	преобразует символы из строки S в целочисленное число
		Обратное преобразование	
DateTameToStr (V)	дата и время	строка	преобразует дату и время в строку
FloatToStr (V)	вещественное	строка	преобразует вещественное число в строку
IntToSrt (V)	целое	строка	преобразует целочисленное число в строку
FloatToStrF(V,F,P,D)	вещественное	строка	преобразует вещественное число V в строку символов с учетом формата F и параметров P, D

Рассмотрим подробнее структуру функции `FloatToStrF (V, F, P, D)`. Обычно ее используют для форматированного вывода вещественного числа. В таблице 2.9 содержатся значения параметров этой функции.

Таблица 2.9. Параметры функции `FloatToStrF`

Формат	Назначение
`ffExponent`	Экспоненциальная форма представления числа, P - мантисса, D- порядок: 1,2345E + 10

Окончание табл.2.9

Формат	Назначение
ffFixed	Число в формате с фиксированной точкой, P - общее количество цифр в представлении числа, D – количество цифр в дробной части: 12,345
ffGtneral	Универсальный формат, использует наиболее удобную форму представления числа
ffNumber	Число в формате с фиксированной точкой, использует символ разделителя тысяч при выводе больших чисел

Примером работы функции FloatToStrF служит фрагмент программы:

```
var   n: integer;   m: real;   St: string;
   begin
      n:=5; m:=4.8; St:= 'Иванов А.';
      // Выражение chr (13) - символ окончания строки.
      // Для вывода вещественного числа m отводятся четыре
позиции
      //вместе с точкой, две позиции после точки.
      Label1.Caption:='Студент' +St+'сдал' + IntToSrt (n)+
'экзаменов.'
      +chr(13)+'Средний бал составил' + FloatToStrF ( m ,
ffFixed ,4,2);
      End.
```

Его результатом будет фраза:

```
Студент Иванов А. сдал 5 экзаменов.
Средний бал составил 4.80
```

С подробным описанием приведенных выше и множеством других функций можно познакомиться в справочной системе Delphi.

2.6. Понятие оператора. Оператор присваивания

Операторы в языке программирования представляют конструкции, определяющие выполнение некоторого законченного действия. Оператор заканчивается точкой с запятой.

Следует заметить, что иногда в литературе оператор определяет действие, которое должно быть выполнено одним или несколькими операндами.

В этом случае в качестве оператора может рассматриваться, например, знак + (плюс).

В предлагаемом пособии рассматриваемые ниже операторы будут определяться исходя из первой формулировки.

Наиболее широко используемым оператором является оператор присваивания. Присваивание – это занесение в память.

В Delphi знак присваивания состоит из двух символов: двоеточия и знака равенства. Символы «:=» всегда пишут слитно. Пробелы допускаются перед символом двоеточия и после символа равенства.

В общем случае оператор присваивания имеет вид:

```
имя переменной: = значение; ,
```

где значение – это выражение, переменная, константа или функция. Выполняется оператор так: сначала вычисляется значение выражения, указанного в правой части оператора, а затем его результат записывается в область памяти, имя которой указанно слева. Например, запись `a:= b` означает, что переменной `a` присваивается значение `b`. Типы переменных `a` и `b` должны совпадать или быть совместимыми для присваивания, то есть тип, к которому принадлежит переменная `b`, должен находиться в границах типа переменной `a`.

Обратимся еще к одному примеру:

```
i:= i+1;
```

Смысл его заключается в том, что к значению переменной `i` прибавляется единица, и полученное значение вновь присваивается переменной `i`.

Оператор присваивания, как и любой другой оператор Delphi заканчивается точкой с запятой.

Рассмотрим работу с оператором присваивания на примерах.

Пример 2.1.Разобрать программу перевода температуры, заданной в градусах по шкале Цельсия, в градусы по шкалам Фаренгейта и Кельвина. Для перевода используются следующие зависимости:

$$F = 1{,}8C + 32;$$

$$K = C + 273;$$

Примем, что исходные данные и результаты решения будут представлены на форме в виде, приведенном на рис 2.1 .

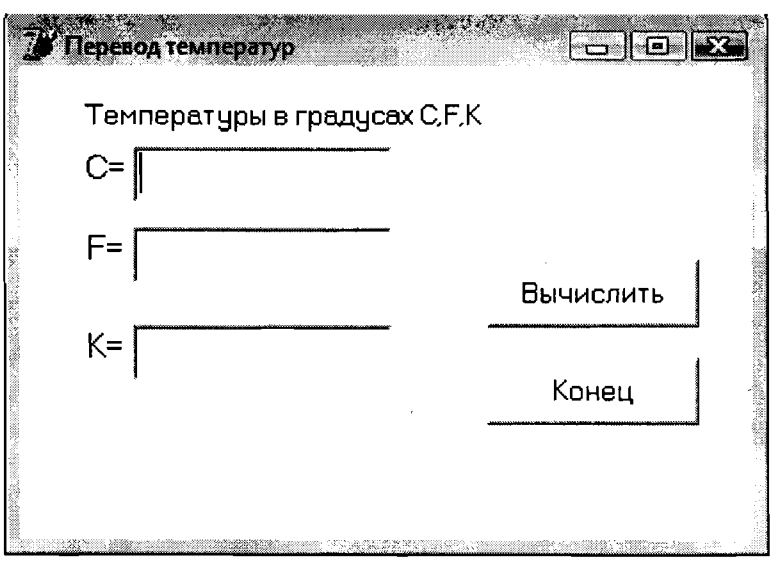

Рис 2.1. Вид Формы к примеру 2.1

Текст написанного программистом программного кода, который следует включить в разрабатываемую программу, имеет следующий вид:

```
var c,f,k:real;
begin
c:=StrtoFloat(Edit1.Text);
f:=1.8*c+32;
```

```
k:=c+273;
Edit2.Text:=FloattoStr(f);
Edit3.Text:=FloattoStr(k);
end;
```

Полезно обратить внимание на то, что при наборе программного кода последний оператор можно сформировать путем копирования предшествующего ему с последующим внесением необходимых изменений.

После запуска программы на выполнение для расчетов по каждому последующему варианту, начиная со второго, можно установить курсор в поле ввода ранее набранного значения исходной характеристики, удалить это число, набрать нужную величину и щелкнуть по кнопке. В ***полях вывода*** результаты вычислений автоматически появятся вместо предшествующих.

В следующем примере рассмотрим построение программы с использованием стандартных функций, а также последовательность разработки всей программы, по которой в компьютере будет выполняться решение задачи.

Пример 2.2. По заданным длинам катетов *a* и *b* вычислить длину гипотенузы *c*, а также величины углов α и β прямоугольного треугольника (рис 2.2).

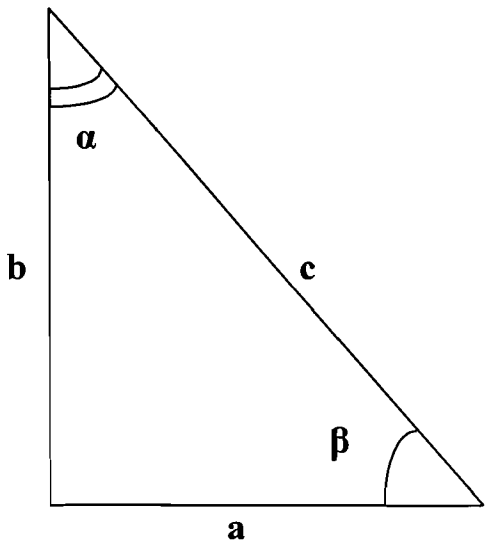

Рис.2.2. Прямоугольный треугольник

Прежде чем приступить к написанию программы, вспомним математические формулы, необходимые для решения задачи. Гипотенуза с вычисляется по формуле:

$$c = \sqrt{a^2 + b^2} \ .$$

Углы треугольника α и β рассчитываются следующим образом:

$$\alpha = arctg\left(\frac{a}{b}\right)$$

$$\beta = \pi/2 - \alpha.$$

Решение задачи можно разбить на следующие этапы:

1.Задание значений a и b (ввод величин a и b в память компьютера).

2.Расчет значений c, α и β по приведенным выше формулам.

3.Вывод значений c, α и β.

Попытайтесь самостоятельно разобрать вид окна данной программы. Разместите компоненты на Форме, как показано на рис 2.3, и измените их заголовки в соответствии с таблицей 2.10.

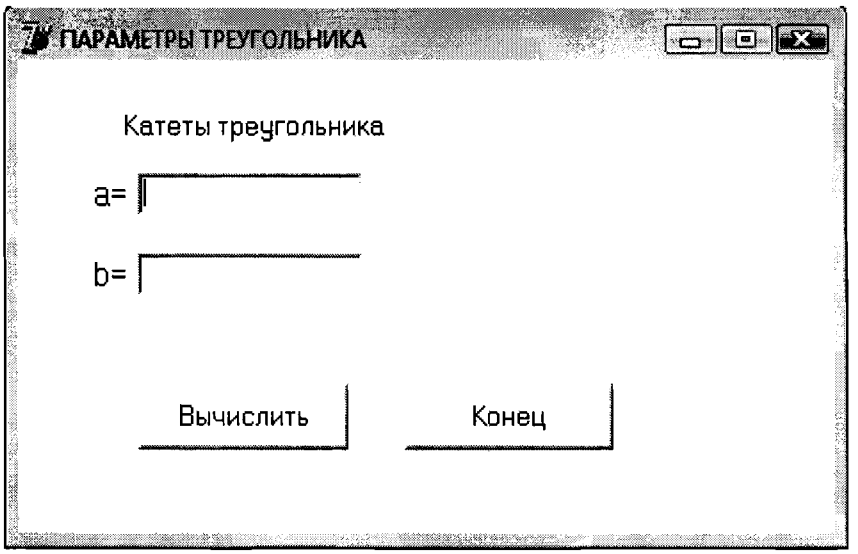

Рис.2.3. Вид Формы к примеру 2.2

Таблица 2.10.Заголовки компонентов Формы, представленной на рис.2.3

Компонент	**Свойство Caption**
form1	параметры треугольника
label1	катеты треугольника
label2	a =
label3	b =
label4	гипотенуза c =
label5	угол alfa =
label6	угол betta =
button1	вычислить
button2	конец

Итак, проект Формы готов. На вкладке Unit1 в окне программного кода Unit1.pas Delphi автоматически сформировал структуру модуля, перечислив названия основных разделов. Двойной щелчок по кнопке **Вычислить** приведет к созданию процедуры TForm1.Button1Click в разделе implementation:

```
Procedure TForm1.Button1Click(Sender: TObject);
Begin

end;
```

и ее описанию в разделе `interface`. Понятно, что создание процедуры не содержит ни одной команды. Задача программиста заполнить шаблон описаниями и операторами. Все команды, указанные в процедуре между словами `begin` и `end`, будут выполнены при щелчке по кнопке **Выполнить**. В нашем случае процедура `TForm1.Button1Click (Sender: TObject)` будет иметь вид:

```
Procedure TForm1.Button1Click(Sender: TObject);
 Var a,b,c,alfa,betta:real;
 begin
    a:=StrToFloat(Edit1.Text);
    b:=StrToFloat(Edit2.Text);
    c:=sqrt(sqr(a)+sqr(b));
    alfa:=arctan(a/b);
    betta:=pi/2-alfa;
    alfa:=alfa*180/pi;
    betta:=betta*180/pi;
    Label4.Caption:='Гипотенуза c ='+FloatToStr(c);
    Label5.Caption:='Угол alfa='+FloatToStr(alfa);
    Label6.Caption:='Угол betta='+FloatToStr(betta);
 end;
```

Как видно, программистом было написано всего 11 команд, предназначенных для решения поставленной задачи. Остальной текст в окне редактора создается автоматически.

После завершения работы следует вернуться к Форме, щелкнув по клавише **F13**, а затем дважды щелкнуть кнопкой мыши по кнопке **Конец**. В появившуюся в окне ***редактор кодов*** процедуру следует вставить оператор **Close**. Полученную в результате программу можно запускать на ***Трансляцию*** и ***Выполнение.***

Пример 2.3. Длительность эксперимента была получена в секундах. Требуется представить это время в часах, минутах и секундах.

Исходная характеристика: S- время эксперимента в секундах.

Искомые характеристики:Z- количество часов,M- кол-во минут, S – количество секунд протекания эксперимента.

Алгоритм решения данной задачи можно описать в следующем виде:

1. Для нахождения количества часов разделить S на 3600(секунд в часе) и взять целую часть результата деления (вспомним деление уголком).
2. Остаток деления разделить на 60 и вновь взять целую часть, что составит количество минут. Полученный при этом остаток составит число секунд. В том случае,если остаток не будет соответственно выводить 0.

Для математического описания алгоритма полезно использовать принятые в Delphi знаки операций целочисленного деления `div` и `mod`. В этом случае алгоритм можно записать, введя промежуточную переменную `ost`(остаток деления):

1. `z= s div 3600;`
2. `ost= s mod 3600;`
3. `M= ost div 60;`
4. `ost= ost mod 60;`
5. `s= ost.`

Для нахождения исходной и искомой характеристик предусмотрим окно,имеющее вид (рис 2.4) :

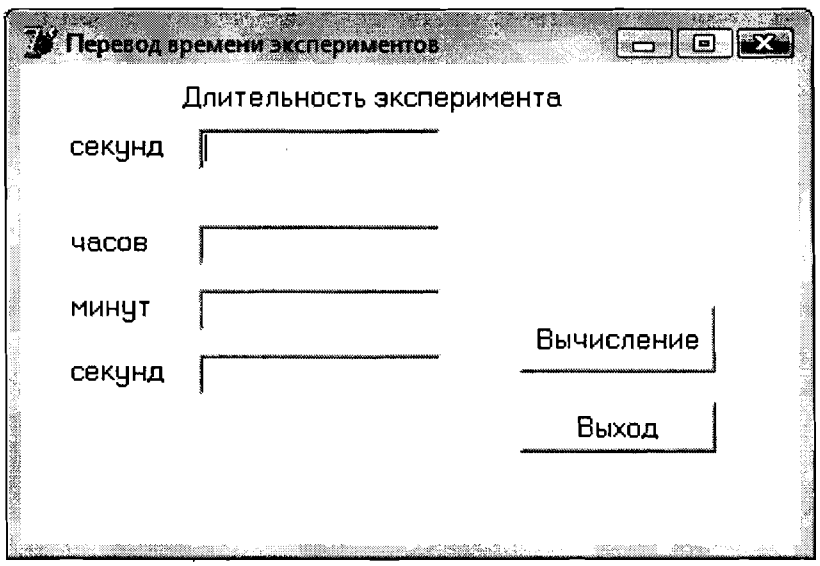

Рис 2.4. Вид Формы к примеру 2.3

Исходя из алгоритма и окна формы,программный код, разработанный программистом, может иметь вид:

```
var s,m,z,ost:integer;
begin
s:=StrToInt(Edit1.Text);
z:=s div 3600;
ost:=s mod 3600;
m:=ost div 60;
ost:=ost mod 60;
s:=ost;
Edit2.Text:=IntToStr(z);
Edit3.Text:=IntToStr(m);
Edit4.Text:=IntToStr(s);
end;
```

Глава 3. Программирование разветвляющихся алгоритмов в Delphi

3.1.Основные конструкции разветвляющихся алгоритмов, средства их описания в Delphi

Решение большинства задач редко сводится к простому последовательному вычислению по формулам. Чаще порядок расчетов зависит от определенных условий, например, от значений исходных данных или промежуточных результатов, полученных на предыдущих шагах вычисления алгоритма. В подобных случаях процесс дальнейших вычислений может разветвляться. Разветвляющийся процесс – одна из трех базовых типовых конструкций алгоритмов. Графическое отображение возможных структур алгоритмов, описывающих разветвляющиеся процессы в виде блок-схем, приведено на рис.3.1.

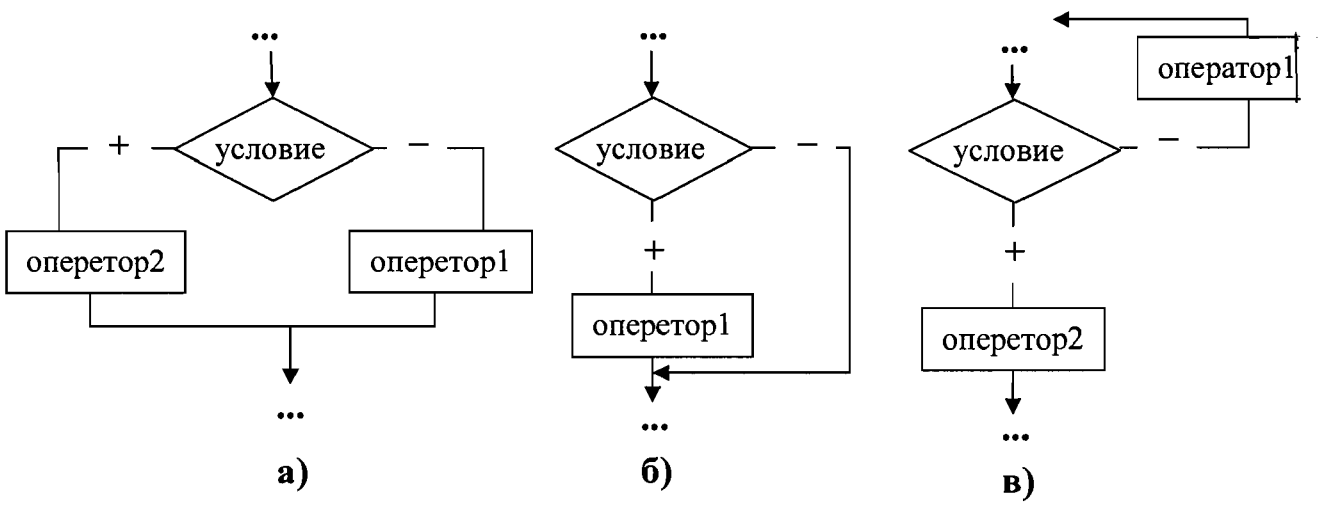

Рис 3.1. Возможные структуры разветвляющихся алгоритмов

Ветвление задает выполнение одного (оператор 1) либо другого (оператор 2) оператора в зависимости от выполнения или невыполнения некоторого заданного условия (рис 3.1а). В зависимости от содержания алгоритма оператор на одной из ветвей может отсутствовать(рис3.1б). Задание на выполнение операторов может быть изменено разработчиком алгоритма путем изменения содержания задаваемого для проверки условия. Блок-схема алгоритма, представленная на рис.3.1в, определяет, что по одной из ветвей осуществляется возврат в верхнюю часть алгоритма для повторного выполнения каких-то определенных действий. Например, при проверке в условии правильности введенного значения исходной характеристики и обнаружении ошибки, может возникнуть необходимость повтора операции ввода.

На ветвях одного ветвления могут содержаться другие ветвления. Такие разветвляющиеся алгоритмы определяются, как алгоритмы, имеющие структуру вложенных ветвлений.

Любой из представленных на рис.3.1 операторов может в конкретном случае быть простым, т.е. представленным одним оператором или состав-

ным, включающим несколько простых операторов. Эта особенность записи операторов учитывается при написании программ. В Delphi при описании составных операторов используются операторные скобки begin и end. В общем виде составной оператор в программе представляется записью:

begin
оператор 1;
.....
оператор n;
end.

При такой форме записи компилятор воспринимает группу операторов как единый составной.

Для описания разветвляющихся процессов в Delphi используются два условных оператора if и case.

3.2. Условный оператор if

Для организации вычислений в зависимости от какого-либо задаваемого условия с разветвлением на два направления используется условный оператор if, который в общем виде записывается:

```
if условие then оператор 1 else оператор 2; ,
```

где `if...then...else` – зарезервированные слова Delphi, `условие` – выражение логического типа, способное принимать одно из двух значений: истина или ложь, `оператор 1` и `оператор 2` – любой оператор языка Delphi.

Работает условный оператор следующим образом. Сначала вычисляется выражение, записанное в `условии`. Если оно имеет значение истина (`True`), выполняется `оператор 1`. В противном случае, когда выражение имеет значение ложь (`Else`), `оператор 1` игнорируется, и управление передается на `оператор 2`.

Например, чтобы сравнить значения a и b, можно создать следующий программный код:

```
a:= StrToInt (Edit1.Text) ;
b:= StrToInt (Edit2.Text) ;
if a=b then
label1.Caption:= 'значение a равно значению b'
else
label1.Caption:= 'значение a не равно значению b';
```

Если в задаче требуется, чтобы в зависимости от условия выполнялся не один оператор, а несколько, их необходимо описать как *составной оператор.*

Альтернативная ветвь `else` в *условном операторе* может отсутствовать, если в ней нет необходимости:

```
If условие then оператор; .
```

В таком «усеченном» виде условный оператор работает так: `оператор` (группа операторов) либо выполняется, либо пропускается, в зависимости от значения выражения, представляющего `условие`.

Пример применения условного оператора без альтернативной ветви else может быть таким:

```
a:=StrToInt(Edit1.Text);
b:=StrToInt(Edit2.Text);
c:=0;
//Значение переменной c изменяется только при условии, что
a   не равно b.
If (a<>b) then c:=a+b;
//Вывод на экран значения переменой c выполняется в любом
случае.
Label3.Caption:='Значение переменой c='+IntToStr(c);
```

Условные операторы могут быть вложены друг в друга. При вложениях условных операторов всегда действует правило: ветвь `else` считается принадлежащей ближайшему оператору `if`. Например, в записи:

```
If условие 1 then
If условие 2 then
оператор A
else оператор Б;
```

`оператор Б` относится к `условию 2`. В конструкции:

```
if условие 1 then
begin
if условие 2 then
оператор A;
end
else оператор Б;
```

он принадлежит оператору if с условием 1.

Для сравнения переменных в условных выражениях применяют операции отношения: `=, < >, <, >, <=, >=`. Условные выражения составляют с использованием логических операций `and`, `or` и `not`. В Delphi приоритет операций отношения меньше, чем логических операций, поэтому составные части сложного логического выражения заключают в скобки. Допустим, нужно проверить,принадлежит ли переменная x интервалу `[a,b]`. Условный оператор будет иметь вид: `if (x>=a) and (x<=b) then...` .Запись `if x>=a and x<=b then...` неверна, так как фактически будет вычисляться значение выражения `x>= (a and x) <=b`.

После завершения действий по выбранной, исходя из выполнения или невыполнения заданного условия, ветви оператора if автоматически происходит переход к следующему по порядку записи оператору программы. Графически такой переход показан на рис.3.1 в позициях **а** и **б**. Однако в определенных случаях возникает необходимость изменения такого порядка. Подобная ситуация представлена, на рис.3.1**в**. Не исключено, что и для двух предыдущих вариантов для каждой ветви может быть задан выход в различные последующие точки программы. Выполнение этого обеспечивается за счет включения в оператор if оператора безусловного перехода, общая запись которого имеет вид:

goto метка; .

Результатом такой записи будет переход к оператору, помеченному соответствующей меткой. Метка записывается перед оператором, которому передается управление продолжением программы, и отделяется от него двоеточием:

метка: оператор; .

Для использования метка должна быть предварительно описана в разделе описаний программы с использованием ключевого слова label, за которым записывается имя метки. Если меток несколько,они перечисляются через запятую. Метки обозначаются либо как целые положительные числа либо как переменные. Например:

Label 1, 2, m, n1, n2, error.

Следует заметить, что без особой необходимости использовать оператор goto не следует, так как это затрудняет работу компилятора.

Рассмотрим использование оператора `if` на примерах.

Пример 3.1. Написать программу вычисления параметра w по формулам:

$$m=f\cdot p;$$
$$w=k\cdot m.$$

Значение коэффициента k выбирается по результатам вычисления параметра m, исходя из условия: при $m<5$ $k=0,9$; при $m\geq 5$ $k=0,7$. По результатам вычислений выдать значения k и w.

Исходные данные: f и p – переменные вещественного типа.

Искомые характеристики: k и w –переменные вещественного типа.

Промежуточная характеристика: m – переменная вещественного типа.

Исходные данные и результаты работы программы представить в форме по виду рис. 3.2.

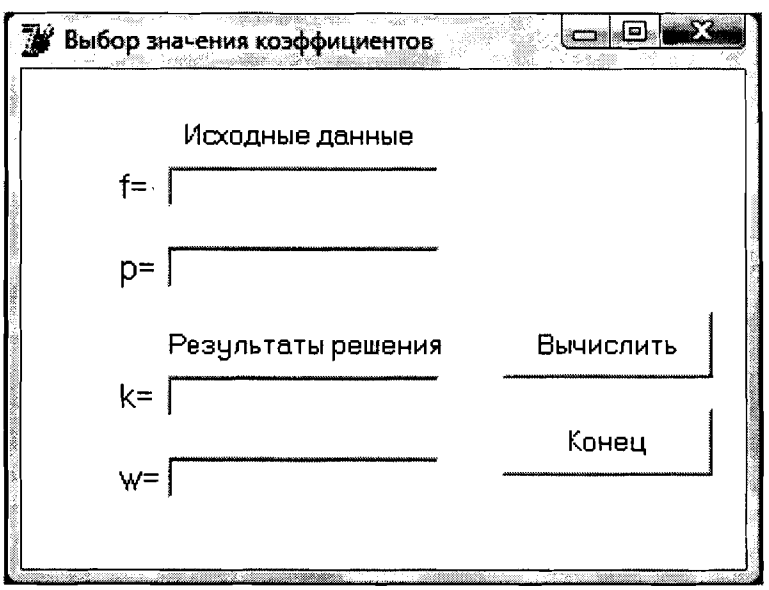

Рис 3.2. Вид Формы к примеру 3.1

Программный код, написанный программистом и вводимый в окно редактора, имеет вид:

```
//Описание переменных.
var m,f,p,k,w:real;
begin
//Из полей ввода Edit1 и Edit2 считываются введенные стро-
ки,
//с помощью функции StrToFloat(x) преобразовываются в
//вещественные числа, которые присваиваются переменным
//f и p.
f:=StrToFloat(Edit1.Text);
p:=StrToFloat(Edit2.Text);
//Вычисление значения переменной m, от которой
//зависят значения коэффициента k.
m:=f*p;
//Если m<5, то значение коэффициента k=0,9,
if m<5 then k:=0.9
//иначе при m>=5 k=0,7.
else k:=0.7;
//Вычисление значения переменной w.
w:=k*m;
//Для вывода результатов вычислений используется функция
//FloatToStr(x), которая преобразовывает вещественную
//переменную x в строку. Значение коэффициентов k и w
//выводится на Форму в поля вывода Edit.
Edit3.Text:=FloatToStr(k);
Edit4.Text:=floatToStr(w);
end;
```

Для наглядности в код включены операторные скобки, устанавливаемые системой, а для лучшего понимания содержания программы – комментарий.

Пример 3.2. Для условий предыдущего примера значения коэффициента k определяются следующими условиями: при $m<5$ k=0,9; при $m\geq5$ и $m<10$ k=0,8; при $m\geq10$ k=0,7.

В этом случае программный код, написанный программистом, примет вид:

```
var m,f,p,k,w:real;
begin
f:=StrToFloat(Edit1.Text);
p:=StrToFloat(Edit2.Text);
m:=f*p;
if m<5 then k:=0.9
else if m<10 then k:=0.8
else k:=0.7;
w:=k*m;
Edit3.Text:=FloatToStr(k);
Edit4.Text:=floatToStr(w);
end;
```

Пример 3.3. Написать программу для вычислений по выражению z=2/x, учитывая невозможность деления на 0.

Исходная характеристика: x – вещественное число.

Искомая характеристика: z – вещественное число.

Исходная и искомая характеристики должны быть представлены в форме по виду рис.3.3.

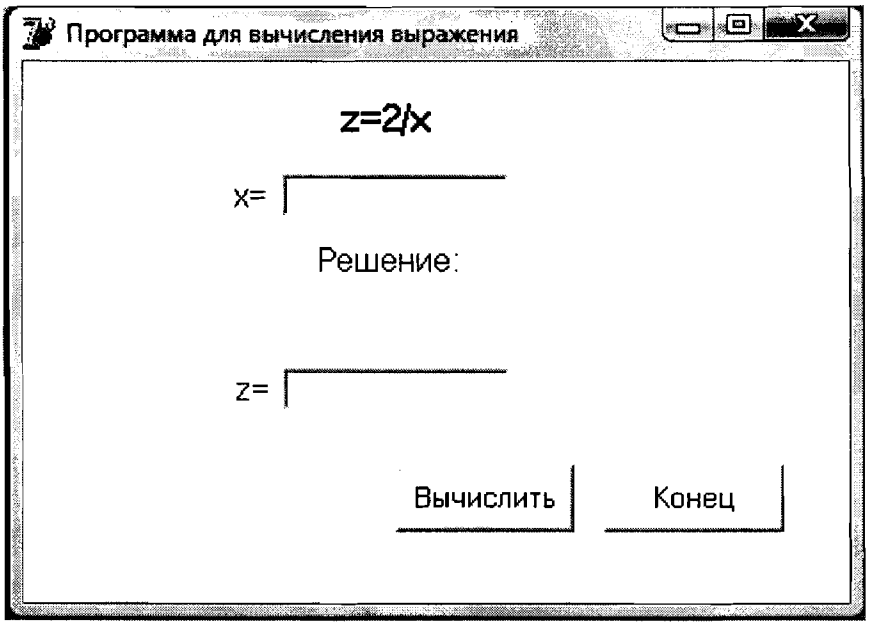

Рис 3.3. Вид Формы к примеру 3.3

В том случае, когда будет введено значение x, равное 0, в форме должно быть выведено сообщение: «X не может быть равным 0. Введите другое значение X.». Поле вывода z остается пустым. При других значениях x в поле вывода z должен быть представлен результат вычислений.

Написанный программистом программный код, в котором использована структура вложенных ветвлений:

```
var x,z:real;
begin
x:=StrToFloat(Edit1.Text);
//Если x=0, то в объект Label4 выводится текст "X не
//может быть равным 0.Введите другое значение X".
if x=0 then Label4.Caption:='X не может быть равным 0.
Введите другое значение X.'
else
begin z:=2/x;Edit2.Text:=FloatToStr(z);
end;
end;
```

Пример 3.4. Написать программу вычислений периметра и площади треугольника. Исходные данные должны удовлетворять следующему условию: длина каждой стороны должна быть меньше суммы длин двух других

сторон. Предусмотреть также, что длина каждой стороны должна быть задана положительным числом.

Исходные характеристики: a, b, c – стороны треугольника, их длины задаются вещественными числами.

Искомые характеристики: P – периметр треугольника, S – площадь треугольника. P и S – переменные вещественного типа.

Промежуточная характеристика: q – полупериметр треугольника, переменная вещественного типа.

Алгоритм вычислений:

$$P=a+b+c;$$

$$q=P/2;$$

$$S=\sqrt{q\cdot(q-a)\cdot(q-b)\cdot(q-c)}.$$

Исходные и искомые характеристики следует представить в Форме по виду рис.3.4.

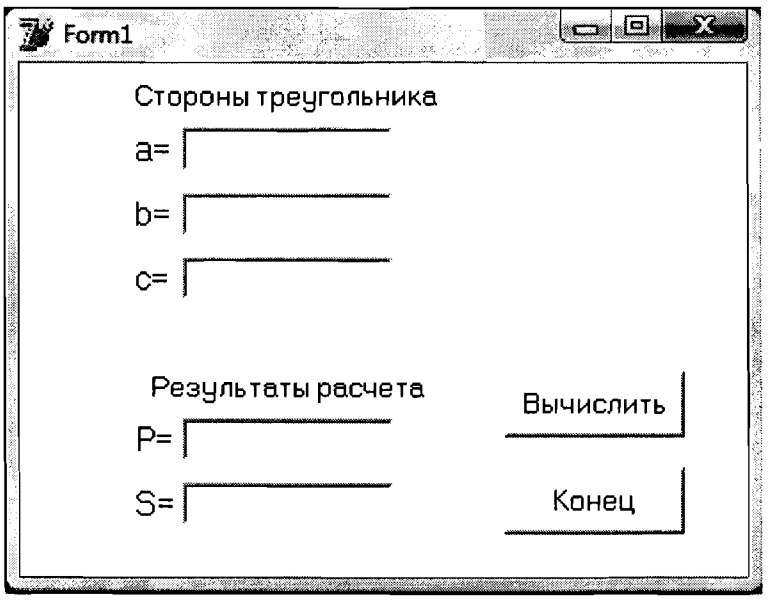

Рис.3.4.Вид Формы к примеру 3.4

В том случае, когда хотя бы одно из приведенных в постановке задачи условий не выполняется, в Форме должно быть выведено сообщение: «Длины сторон заданы ошибочно. Повторить ввод». Поля вывода P и S остаются пустыми. При выполнении условий сюда выводятся результаты вычислений.

Программа для рассматриваемой задачи может быть написана с использованием структур вложенных ветвлений. Ниже приводится иной вариант программы, в которую введено сложное логическое выражение. Такая конструкция позволяет в одном операторе if сразу «отфильтровать» все неверные варианты исходных данных.

В этом случае программный код, написанный программистом, примет вид:

```
var a,b,c,q,p,s:real;
```

```
begin
a:=StrToFloat(Edit1.Text);
b:=StrToFloat(Edit2.Text);
c:=StrToFloat(Edit3.Text);
if (a>0) and (b>0) and (c>0) and (a<b+c) and
(b<a+c) and (c<a+b)
then
  begin
  Label1.Caption:='';
  p:=a+b+c;
  q:=p/2;
  s:=sqrt(q*(q-a)*(q-b)*(q-c));
  Edit4.Text:=FloatToStrF(p,ffFixed,4,1);
  Edit5.Text:=FloatToStrF(s,ffFixed,6,1);
  end
else
  begin
  Edit4.Text:='';
  Edit5.Text:='';
  Label1.Caption:='Длины сторон заданы ошибочно.
  Повторить ввод.';
  end;
```

Пример 3.5. Вновь обратимся к рассмотренному в главе 1 примеру соответствия роста и веса человека и «модернизируем» его.

Пусть нужно не только определить идеальный вес, но и сравнив его с фактическим весом человека, сделать вывод и представить его в виде соответствующего сообщения, исходя из следующих условий:

– Если фактический вес отличается от идеального не более чем ±5 кг, то выдается сообщение «У вас нормальный вес».

– Если фактический вес меньше идеального больше чем на 5 кг, выдается сообщение «Вам надо поправиться на ...кг» (с указанием количества килограммов).

– Если фактический вес превышает идеальный больше чем на 5 кг, выдается сообщение «Вам нужно похудеть на ... кг».

Исходные характеристики: r – рост человека, vf – фактический вес человека.

Обе исходные переменные целочисленного типа.

Искомые характеристики: vi – идеальный вес человека, O – отклонение фактического веса человека от его идеального веса.

Обе искомые переменные вещественного типа.

Блок схема алгоритма решения поставленной задачи приведена на рис3.6.

Рис 3.5. Вид Формы к примеру 3.5

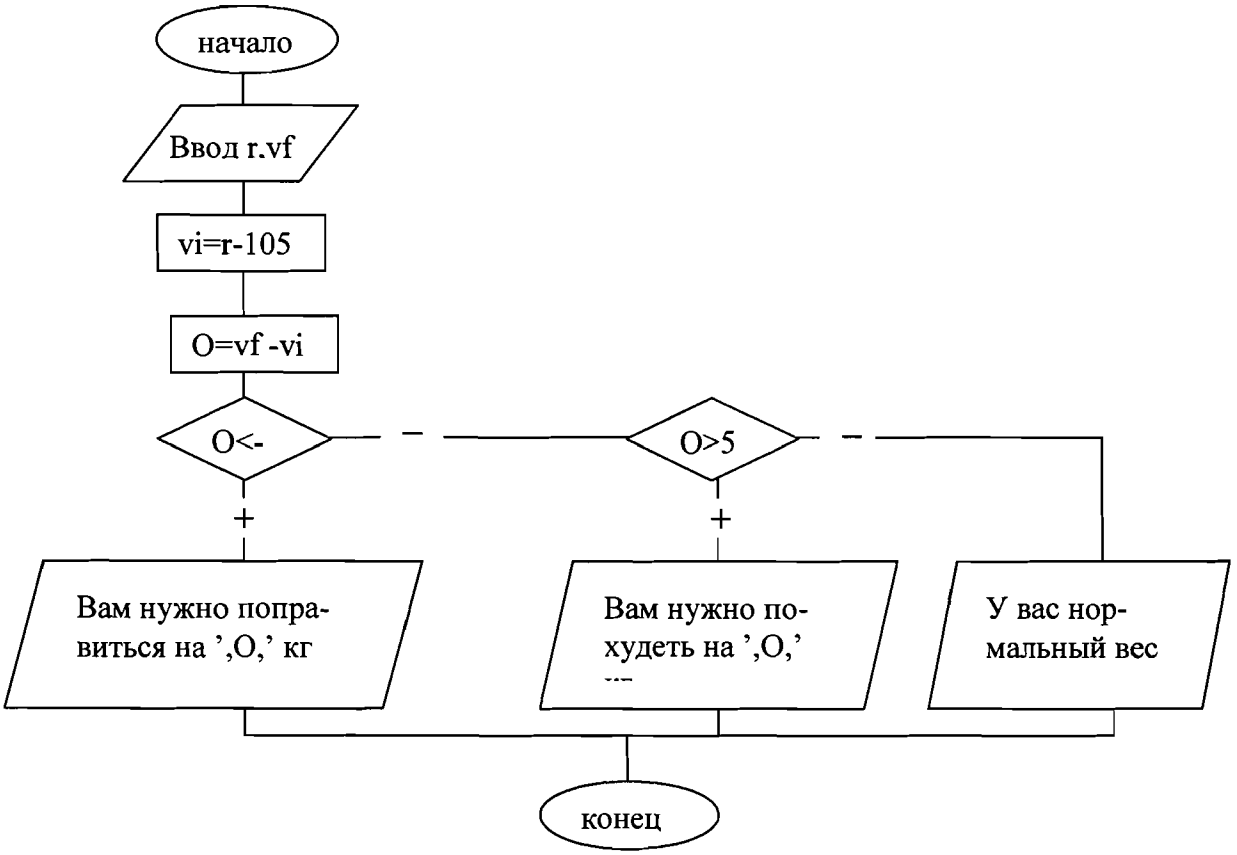

Рис 3.6. Блок-схема алгоритма определения веса человека

Программный код, написанный применительно к приведенному алгоритму и виду Формы, имеет вид:

```
var r,vf,vi,o:real;
begin
 r:=StrToFloat(Edit1.Text);
```

```
 vf:=StrToInt(Edit2.Text);
 vi:=r-105;
 Edit3.Text:=FloatToStr(vi);
 o:=vf-vi;
 if o<-5 then Label1.Caption:='Вам нужно поправиться на'
                              +FloatToStr(abs(o))+' кг'
 else if o>5 then Label1.Caption:='Вам нужно похудеть
                              на'+FloatToStr(o)+' кг'
 else  Label1.Caption:='У вас нормальный вес;
end;
```

3.3.Оператор case

Оператор `case` необходим в тех случаях, когда в зависимости от значений какой–либо переменной надо выбрать более двух возможных направлений реализации алгоритма. Оператор имеет вид:

```
case выражение of
значение 1: оператор 1;
значение 2: оператор 2;
...
значение N: оператор N;
else
альтернативный оператор
end;
```

Здесь `выражение`- переменная перечислимого типа(включая `char` и `Boolean`), `значение 1`, `значение 2`, ...,`значение N`- это конкретное значение управляемой переменной или выражение, при котором необходимо выполнить соответствующий `оператор`, игнорируя остальные варианты. `Значения` в каждом наборе должны быть уникальны, то есть они могут появляться только в одном варианте. Пересечение значений наборов для различных вариантов является ошибкой.

Оператор работает следующим образом. Вычисляется значение `выражения`. Затем выполняется `оператор`, помеченный `значением`, совпадающим со значением выражения. То есть, если `выражение` принимает `значение` равное 1,выполняется `оператор` 1. Если `выражение` принимает `значение` равное 2, -оператор 2 т.д. Если `выражение` не принимает ни одного из `значений`, выполняется `альтернативный оператор`, расположенный после ключевого слова `else`.

Альтернативная ветвь `else` может отсутствовать, тогда оператор имеет вид:

```
case выражение of
значение 1: оператор 1;
значение 2: оператор 2;
...
значение N: оператор N;
end;
```

Кроме того, в операторах case допустимо использование составного оператора. Например:

```
case выражение of
значение 1: begin оператор A; оператор B; end;
значение 2: begin оператор C; оператор D; оператор E; end;
…
значение N: оператор N;
end;
```

Рассмотрим применение оператора case на следующем примером.

Пример 3.6. По заданному номеру месяца m вывести на печать название время года.

Для решения данной задачи необходимо проверить выполнение четырех условий. Если заданное число m равно 12, 1 или 2,то это зима, если m попадает диапазон от 3 до 5 – весна, лето определяется принадлежностью числа m диапазону от 6 до 8 и, соответственно, при равенстве переменной m 9,10 или 11 – это осень. Понятно, что область возможных значений переменной m находится в диапазоне от 1 до 12, и если пользователь введет число, не входящее в этот интервал, появится сообщение об ошибке. Написанный программистом программный код может имеет вид:

```
var m: byte;
Begin
m:=StrToInt(Edit1.Text);
case m of
//В зависимости от значения m на печать
//выводится название времени года.
12,1,2:     label1.Caption:=' зима ';
3..5:       label1.Caption:=' весна ';
6..8:       label1.Caption:=' лето ';
9..11:      label1.Caption:='осень ';
else
//сообщение о недопустимом значении переменной m.
label1.Caption:=' ошибка ввода!!! ';
end;
end;
```

Для приведенных ниже примеров, исходя из заданных постановок, дать описание алгоритмов, а затем, используя операторы if и case, разработать программные коды обработки информации.

Пример 3.7. Для вычисления площади круга исходные данные могут задаваться как радиус, диаметр или длина окружности. В зависимости от варианта задания исходной характеристики написать программу, которая запрашивала бы значение заданной характеристика по ее заданному виду, а затем производила вычисление площади круга.

Пример 3.8. Имеется пронумерованный перечень деталей: 1 – шуруп, 2 – винт, 3 – болт, 4 – гвоздь, 5 –гайка.

Составить программу, которая по номеру детали будет выводить на экран ее наименование.

Пример 3.9.Написать программу, которая по задаваемому номеру единицы измерения (1 – сантиметр, 2 – дециметр, 3 – метр, 4 – километр) и длине отрезка L выдавала бы соответствующее значение его длины в метрах.

Глава 4. Операторы цикла

4.1. Общая характеристика операторов цикла

Во многих задачах для получения решения приходится многократно, более одного раза, проводить одни и те же действия над различными значениями исходных и промежуточными данных. Такой процесс получения решения называется циклическим процессом. Один проход цикла определяется как шаг или итерация.

В Delphi для написания циклических программ предлагается три вида операторов:

- While...do(пока...выполнять) – оператор цикла с предусловием;
- Repeat…until(повторять...до) – оператор цикла с постусловием;
- For…do(на протяжении...выполнять) – оператор цикла с заданным количеством повторений.

На рис 4.1 изображены блок-схемы алгоритмов, соответствующие названным выше операторам.

Тело цикла – оператор или совокупность операторов, ради повторения которых организуется цикл. Если тело цикла состоит более чем одного оператора, то в программе они должны быть представлены как составной оператор.

Начальные установки служат для того, чтобы да входа в цикл задать значения переменных, которые в нем используются.

Одна из таких переменных определяется как параметр цикла. *Параметром цикла* называется переменная, которая используется в выражении при проверке условия проведения цикла и принудительно изменяется на каждой итерации, причем, как правило, на одну и ту же величину. Эта величина может задаваться в начальных условиях в виде переменной или записываться как числовая константа в операторе модификации параметра цикла. Таким же образом в начальных условиях или непосредственно в выражении задается предельная величина изменения параметра цикла, что определяет количество итераций.

Выражение задается как выражение логического типа. В нем на каждом шаге цикла текущее значение параметра цикла сравнивается с заданной граничной величиной его значения. Истинностью или ложностью результата определяется продолжение или прерывание цикла. Учет полученного резуль-

тата различен для операторов while и repeat, что будет рассмотрено в последующих программах.

Для оператора for предусматривать изменение значений параметра цикла нет необходимости. Оно производиться автоматически в заданном диапазоне значений этого параметра.

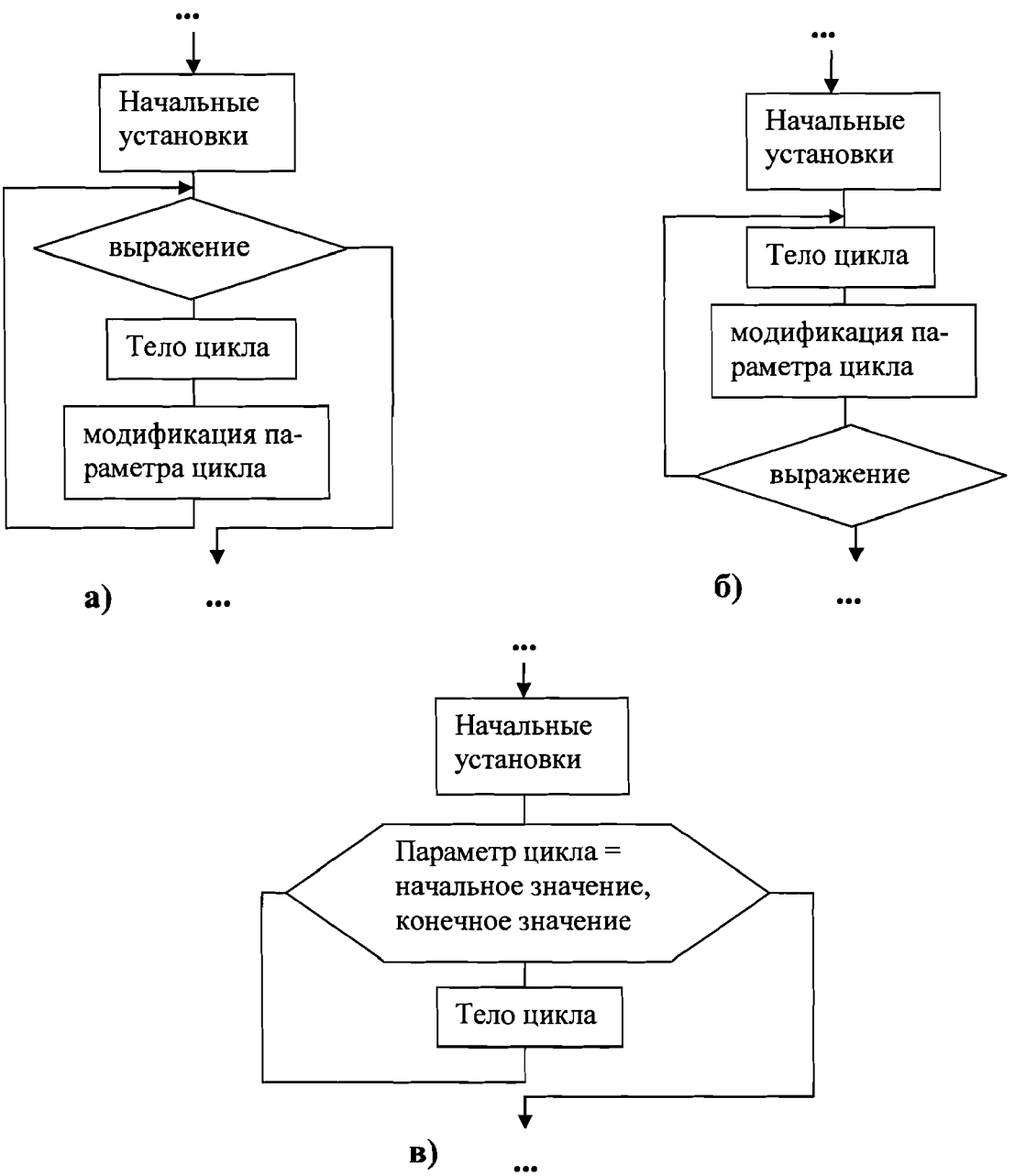

Рис 4.1. Блок-схема алгоритмов цикла:
а - с предусловием; б – с постусловием; в – с заданным количеством повторений

4.2. Оператор цикла с предусловием

Оператор реализующий цикл с предусловием имеет вид:

`While выражение do тело цикла; оператор модификации цикла.`

Работает данный оператор следующим образом. Вычисляется значение выражения. Если оно истинное (`True`) выполняется оператор, входящий в тело цикла и оператор модификации параметра цикла, В противном случае цикл заканчивается и управление передается оператору, записанному в программе непосредственно за оператором `while`.

Рассмотрим работу цикла с предусловием на примерах.

Пример 4.1. Вычислить $P=a^n$, где n – целое положительное число.

Исходные данные: a – основание степени, переменная вещественного типа; n – показатель степени, целочисленная переменная.

Искомая характеристика: p – степень, переменная вещественного типа.

Промежуточная характеристика: i – параметр цикла, целочисленная переменная, принимающая значения от 1 до n с шагом, равным 1.

Значения исходных данных в форме размещаются: a – в поле ввода `Edit1`, n – в поле ввода `Edit2`. Результат вычислений должен быть отображен в поле надписи `Label1`.

Блок – схема алгоритма приведена на рис. 4.2.

Из рассмотрения блок-схемы видно, что программа относительно нее могла быть написана с использованием операторов `if` и `goto`. Изучающим программирование рекомендуется самостоятельно разработать такую программу.

Ниже приводится фрагмент программного кода, написанного программистом с использованием оператора while:

```
var i,n:integer;
    a,p:real;
begin
 a:=StrToFloat(Edit1.Text);
 n:=StrToInt(Edit2.Text);
 p:=1;
 i:=1;
   while i<=n do
   begin
   p:=p*a;
   i:=i+1;
   end;
 Label1.Caption:='P='+FloatToStr(p);
end;
```

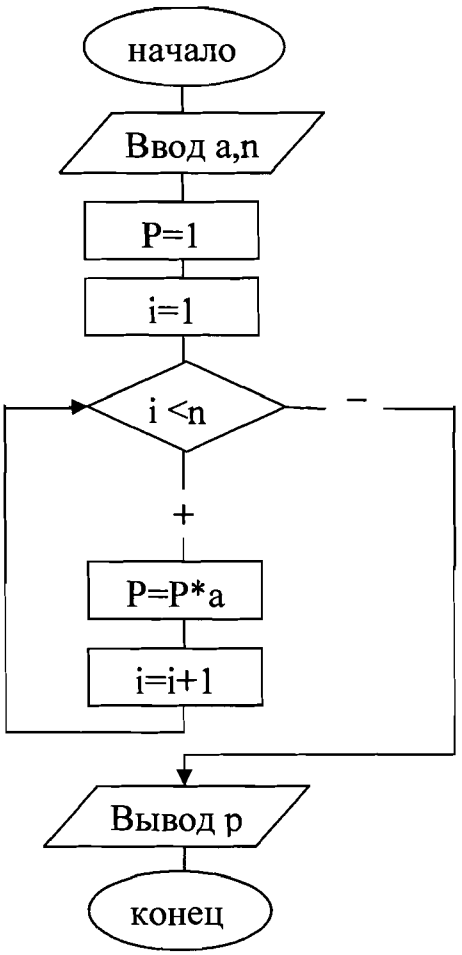

Рис 4.2. Блок-схема алгоритма $P=a^n$

Пример 4.2. Вычислить факториал числа n:

$$F = n! = \prod_{i=1}^{n} i = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n.$$

Исходная характеристика: n – целочисленная положительная переменная, значение которой нужно вычислить.

Искомая характеристика: f – результат вычислений, целочисленная переменная.

Промежуточная переменная: i – параметр цикла, значение которого используется при вычислении значения f (i - целочисленная переменная, изменяющаяся с шагом равным 1 от 1 до n).

Отличие рассматриваемого примера от предыдущего заключается в том, что здесь параметр цикла i «участвует» в производстве вычислений.

Программный код, обеспечивающий получение решения, может иметь вид:

```
var i,f,n:integer;
begin
 n:=StrToInt(Edit1.Text);
```

```
 f:=1;
 i:=1;
 while i<=n do
  begin
   f:=f*i;
   i:=i+1;
  end;
 Label1.Caption:='f='+IntToStr(f);
end;
```

Пример 4.3. Найти наибольший общий делитель (НОД) двух натуральных чисел A и B.

Исходные данные: A и B, целочисленные переменные.

Искомая характеристика: A – НОД, целочисленная переменная.

Для решения поставленной задачи воспользуемся алгоритмом Евклида: каждый раз будем уменьшать большее из чисел на величину меньшего до тех пор, пока оба значения не станут равными, как показано в таблице 4.1.

Таблица 4.1. Поиск НОД для чисел A=25 и B =15

исходные данные	первый шаг	второй шаг	третий шаг
A = 25	A = 10	A = 10	A = 5
B = 15	B = 15	B = 5	B = 5

A (НОД) = 5

В блок - схеме алгоритма, представленной на рис 4.3, для решения используется цикл с предусловием, то есть тело цикла повторяется до тех пор, пока A не равно B.

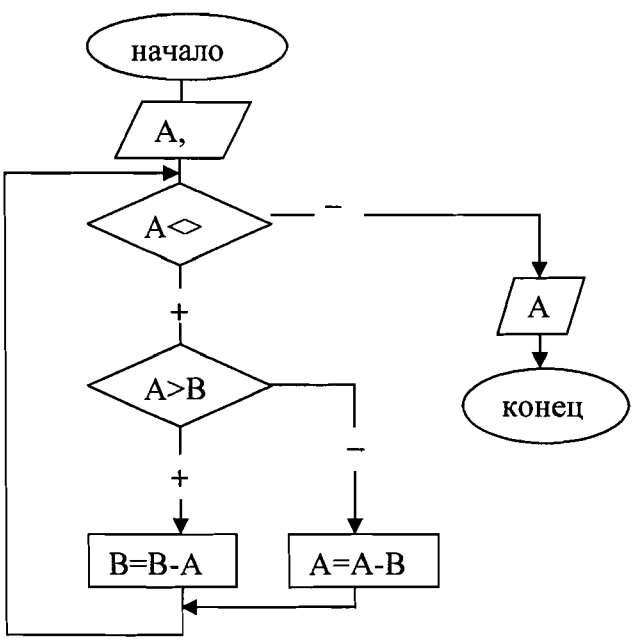

Рис.4.3. Блок - схема алгоритма поиска наибольшего общего делителя

Для организации ввода чисел A и B разместим на Форме два объекта типа надпись (Label1, Label2) и два поля ввода (Edit1, Edit2). Результаты работы программы – значение НОД – выведем в Label3. Решение задачи обеспечивается следующим программным кодом:

```
var a,b:integer;
begin
 a:=StrToInt(Edit1.Text);
 b:=StrToInt(Edit2.Text);
 //Если числа a и b не равны, выполнять тело цикла.
 while a<>b do
 //Если число a больше, чем b, уменьшить его значение на b,
 if a>b then a:=a-b
 //иначе уменьшить значение числа b на a.
 else b:=b-a;
 Label3.Caption:='НОД='+IntToStr(a);
end;
```

4.3. Оператор цикла с постусловием

В Delphi цикл с постусловием реализован конструкцией:

```
repeat
тело цикла;
модификация параметра цикла;
until
выражение;
```

Работает оператор следующим образом. Вначале выполняются операторы, составляющие тело цикла, и оператор модификации параметра цикла. Затем вычисляется значение выражения. Цикл выполняется до тех пор, пока значение выражения ложно. Как только результат станет истинным, произойдет выход из цикла.

Нетрудно заметить, что цикл с постусловием всегда будет выполнен хотя бы один раз, в отличие от цикла с предусловием, который может не выполниться ни разу.

Рассмотрим применение оператора repeat на примере.

Пример 4.4. Для условий примера 4.3 написать программный код с использованием оператора repeat.

Результат работы программы не изменится при замене программного кода с оператором цикла while на приведенный ниже фрагмент программы:

```
var a,b:integer;
begin
 a:=StrToInt(Edit1.Text);
 b:=StrToInt(Edit2.Text);
 //Повторять тело цикла
 repeat
 if a>b then a:=a-b
 else b:=b-a;
 Label3.Caption:='НОД='+IntToStr(a);
 //до тех пор, пока не выполнится условие.
```

```
 until a=b;
end;
```

Изучающим программирование рекомендуется самостоятельно написать программные коды с использованием оператора `repeat` для условий примеров 4.1 и 4.2, проверив затем работу программ на компьютере.

4.4.Оператор с заданным количеством повторений

Операторы цикла с условием универсальны. С их помощью может быть описан любой циклический процесс. Однако, если количество повторений цикла однозначно определено до начала его выполнения, проще использовать оператор цикла `for`. Соответствующая ему блок – схема алгоритма представлена на рис. 4.2. Оператор `for` имеет двоякое написание в зависимости от того увеличивается или уменьшается значение параметра цикла после очередной итерации:

```
For параметр цикла:= начальное значение to конечное значение
                                                    do тело цикла


For параметр цикла:= начальное значение downto конечное
                                          значение do тело цикла
```

Начальное и конечное значения параметра цикла могут быть заданы: переменными, выражениями, числовыми константами. В первых двух случаях его значения должны быть определены до входа в цикл. В последнем случае числовые величины записываются прямо в операторе `for`.

В процессе выполнения оператора параметр цикла изменится с шагом равным единице. Для многих задач, где количество повторений цикла не задано явно, его можно привести к вычислению с шагом равным единице. Предположим, что параметр цикла x принимает значения в диапазоне от X_n до X_k, изменяясь с каким – то шагом dx. Тогда количество повторений тела цикла (n) можно определить по формуле:

$$n = \frac{X_n - X_k}{dx} + 1,$$

округлив результат деления до целого числа.

Покажем целесообразность использования оператора for, вновь обратившись к рассмотренным выше примерам.

Пример 4.5. Написать программный код с использованием оператора for для условий примера 4.1.

Блок – схема алгоритма для рассмотренного подхода приведена на рис. 4.4.

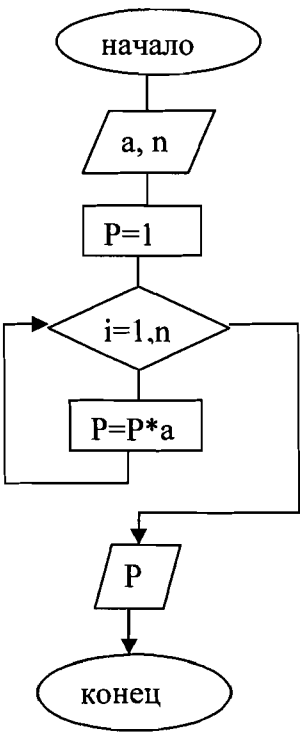

Рис.4.4. Блок – схема с блоком модификация для $P=a^n$

Далее приведен фрагмент программы, составленной для решения поставленной задачи:

```
var i,n:byte;
    a,p:real;
begin
 a:=StrToFloat(Edit1.Text);
 n:=StrToInt(Edit2.Text);
 p:=1;
 for i:=1 to n do
 p:=p*a;
 Label3.Caption:='a в степени n равно'+FloatToStr(p);
end;
```

Пример 4.6. Написать программный код с использованием оператора for для условий примера 4.2.

Изучающим программирование рекомендуется самостоятельно составить блок – схему алгоритма для условий рассмотренного примера с использованием блока «модификация».

Правильность блок – схемы можно проверить на приведенном ниже фрагменте программы.

```
var i,n,factorial:integer;
begin
 n:=StrToInt(Edit1.Text);
 factorial:=1;
 for i:=2 to n do
 factorial:=factorial*i;
```

```
 Label2.Caption:='N!='+IntToStr(factorial);
End;
```

4.5. Окно Information. Последовательный вывод результатов решения

Особенность всех приведенных выше примеров состоит в том, что результат выполнения вычислений выражен одним числом, которое затем отображалось в Форме. Однако, для многих задач результаты решения, получаются и должны быть показаны на экране компьютера после прохождения каждого шага цикла. Для такого последовательного отображения данных предназначено окно Information, вызов которого обеспечивается обращением к формату MessageDlg, имеющему структуру:

```
MessageDlg(описание выходных данных,
           MtInformation,[mbok],0);
```

Рассмотрим применение описанного выше оператора на примере, закрепив кроме того полученный навык в использовании всех операторов цикла.

Пример 4.7. Вычислить и последовательно вывести на экран значения функции

$$y = e^{\sin x} \cdot \cos x,$$

на отрезке $[0;\pi]$ с шагом 0,1.

а) Программный код для цикла while:

```
var x,y:real;
begin
 //Присваивание параметру цикла начального значения.
 x:=0;
 while x<=pi do
  begin
  y:=exp(sin(x))*cos(x);
  //Вывод на экран пары x и y.
  MessageDlg('x='+FloatToStr(x)+
             ';y='+FloatToStr(y),MtInformation,[mbok],0);
  x:=x+0.1;
  end;
end;
```

б) Программный код для цикла repeat:

```
var x,y:real;
begin
 x:=0;
 repeat
  y:=exp(sin(x))*cos(x);
  MessageDlg('x='+FloatToStr(x)+
             ';y='+FloatToStr(y),MtInformation,[mbok],0);
  x:=x+0.1;
```

```
 until x>pi;
end;
```

в)Программный код для цикла for:

```
var x,y:real;
    i,n:integer;
begin
 n:=round((pi-0)/0.1)+1; //Количество повторений цикла.
 x:=0;
 //i-параметр цикла, изменяется от 1 до n с шагом 1.
 for i:=1 to n do
  begin
  y:=exp(sin(x))*cos(x);
  MessageDlg('x='+FloatToStr(x)+
             ';y='+FloatToStr(y),MtInformation,[mbok],0);
  x:=x+0.1;
  end;
end;
```

4.6. Задание значения данных с использованием диалогового окна ввода

Для многих задач характерна необходимость ввода большого количества значений исходных данных. Это можно делать, включая в программу массивы, что будет рассмотрено позднее, или используя диалоговое окно ввода.

Диалоговое окно ввода - это стандартное окно, которое появляется на экране в результате вызова функции `InputBox`. В общем виде оператор ввода данных с использованием этой функции записывают так:

```
имя переменной:=InputBox(заголовок окна, подсказка, значение
переменной);
```

где

- `заголовок окна` – строка, определяющая название окна;
- `подсказка` – текст поясняющего сообщения;
- `значение переменной` – строка, которая будет находиться в поле ввода при появлении окна на экране;
- `имя переменной` – переменная строкового типа, которой будет присвоено `значение переменной` из поля ввода.

После выполнения фрагмента программы:

```
var S:string;
begin
S:=InputBox('Заголовок окна',
'Подсказка:введите исходные данные','Данное значение')
 end;
```

появится окно, представленное на рис.4.5. У пользователя есть возможность изменять текст в поле ввода. Щелчок по кнопке **ОК** приведет к тому, что в переменную, указанную слева от оператора присваивания, будет

занесена строка, находящаяся в поле ввода. В данном случае в переменную S будет записана строка "Данное значение". Щелчок по кнопке Cancel закроет окно ввода.

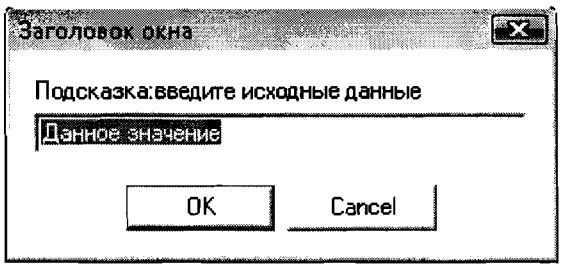

Рис.4.5. Окно ввода

Учитывая, что функция InputBox возвращает строковое значение при вводе числовых данных, применяют функции преобразования типов:

```
var S:string; gradus, radian: real;
begin
S:= InputBox('Ввод данных', 'введите величину угла в радианах', '0,000');
gradus:= StrToFloat(S);
radian := gradus*pi/180;
MessageDlg('Величина угла в градусах'+FloatToStr(radian),
MtInformation,[mbOk],0);
end;
```

Диалоговое окно можно применять при решении задач, обрабатывающих числовые последовательности. Рассмотрим примеры таких задач.

Пример 4.8. Задается последовательность из N вещественных чисел. Определить наибольший элемент последовательности.

Исходные характеристики: N - целое число; X – вещественное число, определяет значение текущего элемента последовательности.

Искомая характеристика: Max – вещественное число, элемент последовательности с наибольшим значением.

Промежуточная переменная: i – параметр цикла, номер вводимого элемента последовательности.

В памяти компьютера отводится ячейка, например с именем Max – *максимум*. Далее предполагают, что первый элемент последовательности - наибольший и записывают его в Max. Затем вводится второй элемент последовательности и сравнивается с предлагаемым максимумом. Если окажется, что второй элемент больше, его записывают в ячейку Max. В противном случае никаких действий не предпринимают. Потом переходят к вводу следующего элемента последовательности, и алгоритм повторяется сначала. В результате в ячейке Max сохранится элемент последовательности с наибольшим значением.

Фрагмент программы на Delphi:

```
var i,N:integer;
```

```
    max,x:real;
    s:string;
begin
 //Ввод количества элементов последовательности.
 N:=StrToInt(Edit1.Text);
 //Ввод первого элемента последовательности.
 s:=InputBox('Ввод элементов последовательности',
'Введите число.','0');
 x:=StrToInt(s);
 //Предположим, что первый элемент максимальный - Max=x.
 max:=x;
 //Параметру цикла присваивается значение i=2.
for i:=2 to N do
 begin
   //Ввод следующих элементов последовательности.
   s:=InputBox('Ввод элементов последовательности',
  'Введите число.','0');
   x:=StrToInt(s);
   //Если найдется элемент, превышающий максимум,
   //записать его в ячейку Max - теперь это предполагаемый
   //максимум.
   if x>max then max:=x;
 end;
 //Вывод наибольшего элемента последовательности.
 MessageDlg('Знач-ие наибольшего элемента'+FloatToStr(max),
 MtInformation,[mbok],0);
end;
```

Алгоритм поиска наибольшего элемента в последовательности представлен на рис. 4.6.

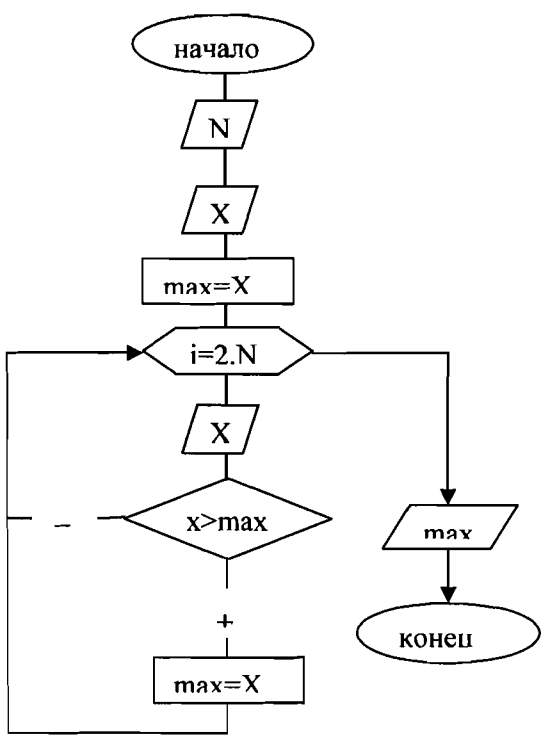

Рис. 4.6. Поиск наибольшего числа в последовательности

Пример 4.9. Для задаваемой последовательности из n чисел, значения которых пошагово в процессе ввода присваиваются переменной x, вычислить их сумму. Алгоритм решения этой задачи приведен на рис.4.7.

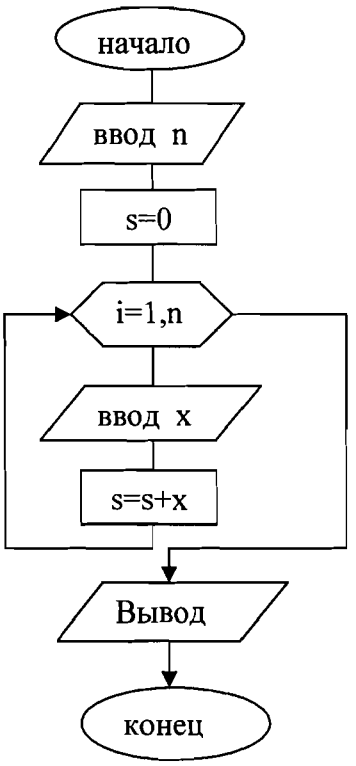

Рис.4.7. Блок – схема суммирования значений переменной x

Ниже приводится фрагмент программного кода, написанного программистом:

```
var i,N:integer;
    S:string;
    x,sum:real;
begin
 S:=InputBox('Ввод числа элементов
 последовательности','Введите число.','0');
 N:=StrToInt(S);
 i:=0;
 sum:=0;
 for i:=1 to N do
   begin
   S:=InputBox('Ввод элемента последовательности',
   'Введите число.','0');
   x:=StrToFloat(S);
   sum:=sum+x;
   end;
 Edit1.Text:=IntToStr(N);
 Edit2.Text:=FloatToStr(sum);
end;
```

Пример 4.10. Для условий примера 4.9 найти произведение вводимых чисел. Алгоритм данной задачи отличается от предыдущего (рис.4.8) только

тем, что при вычислении произведения его начальное значение должно приниматься равным единице (P=1).

Решение задачи обеспечится следующим программным кодом:

```
var i,N:integer;
    S:string;
    x,P:real;
begin
 S:=InputBox('Ввод числа элементов
 последовательности','Введите число.','0');
 N:=StrToInt(S);
 i:=0;
 P:=1;
 for i:=1 to N do
   begin
   S:=InputBox('Ввод элемента последовательности',
   'Введите число.','0');
   x:=StrToFloat(S);
   P:=P*x;
   end;
 Edit1.Text:=IntToStr(N);
 Edit2.Text:=FloatToStr(P);
end;
```

Пример 4.11. Во вводимой с клавиатуры последовательности чисел требуется умножить каждое число на заданное значение коэффициента k.

Результат каждого вычисления должен быть последовательно отображен в окне Information.

В указанном окне выводу очередного результата вычислений должно предшествовать сообщение: “Значение X с учетом коэффициента k =”.

Блок-схема алгоритма корректировки задаваемых значений переменной X за счет значения коэффициента k приведена на рис. 4.8.

Решение задачи обеспечится следующим программным кодом:

```
var i,N:integer;
    S:string;
    x,k,kx:real;
begin
 S:=InputBox('Ввод числа элементов
 последовательности','Введите число.','0');
 N:=StrToInt(S);
 S:=InputBox('Ввод коэффициента k','Введите число.','0');
 k:=StrToFloat(S);
 i:=0;
 for i:=1 to N do
   begin
   S:=InputBox('Ввод элемента последовательности',
   'Введите число.','0');
   x:=StrToFloat(S);
   kx:=k*x;
   MessageDlg('Значение X с учетом коэффициента k =
```

```
    '+FloatToStr(kx),
    MtInformation,[mbok],0);
    end;
end;
```

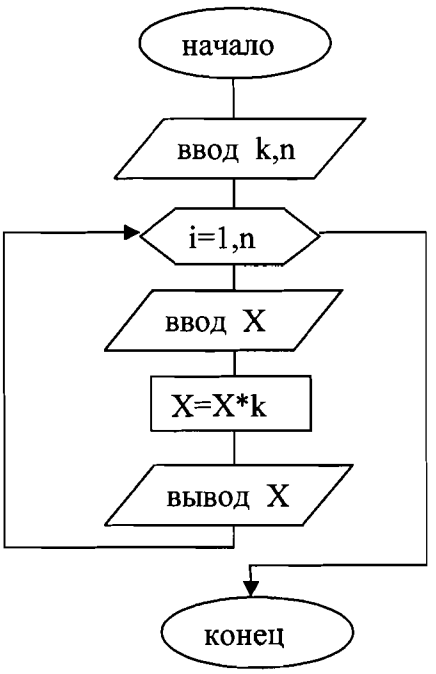

Рис.4.8. Блок – схема корректировки значений переменной x

4.7. Программирование вычислений рекуррентных последовательностей

Из курса математики известно, что рекуррентная последовательность – это бесконечный ряд чисел, каждое из которых, за исключением начального, выражается через предыдущее.

Для ряда переменных a_1 ... a_k , значения которых представляют рекуррентную последовательность, взаимосвязь между соседними элементами, дающая возможность вычислить значение каждого последующего элемента относительно предыдущего ему, выражается в общем виде формулой:

$$a_i = F(a_{i-1}).$$

Эта формула называется *рекуррентной формулой.*

Алгоритм вычисления значений рекуррентной последовательности описывается совокупностью заданного начального значения ряда и рекуррентной формулой. Применительно к рассматриваемому примеру это можно записать как:

$$a_i = \begin{cases} a_1, \text{если } i = 1, \\ F(a_{(i-1)}), \text{если } i > 1. \end{cases}$$

Решению задач относительно рекуррентных последовательностей всегда предшествует установление вида их рекуррентных формул.

Например,
для арифметической прогрессии рекуррентная формула имеет вид:

$$a_i = a_{i-1} + s$$

где s- шаг последовательности;

для факториала: $a_i = a_{i-1} \times i$.

С рекуррентными последовательностями можно связать следующие часто встречающиеся задачи следующего рода:

- вычислить заданный (n - й) элемент последовательности;
- математически обработать определенную часть последовательности (например, вычислить сумму или произведение n членов);
- подсчитать количество элементов в заданном диапазоне последовательности, удовлетворяющих определенному условию;
- вычислить и сохранить в памяти для дальнейшей обработки значения заданного количества членов последовательности.

Рассмотрим на примерах разработку алгоритмов и написание программ для решения задач применительно к рекуррентным последовательностям.

Пример 4.12. Просуммировать первые n элементов геометрической прогрессии, в которой первый член равен 1, а шаг прогрессии – 2.

Для поставленной задачи алгоритм можно записать:

$$a_i = \begin{cases} 1, \text{если } i = 1, \\ 2 \cdot a_{i-1}, \text{если } i > 1. \end{cases}$$

$$S = \sum_{i=1}^{i=n} a_i.$$

Соответствующий программный код будет иметь вид:

```
var a,n,i,S:integer;
begin
n:=StrToInt(Edit1.Text);
a:=1;
S:=a;
for i:=2 to n do
  begin
  a:=2*a;
  S:=S+a;
  end;
Edit2.Text:=IntToStr(S);
end;
```

Пример 4.13. Вычислить значение n – го члена ряда Фибоначчи.

Ряд Фибоначчи представляет собой ряд чисел, в котором первый элемент равен 0, второй – 1, а каждый последующий представляет собой сумму двух предыдущих (0,1,1,2,3,5,8,13,21…).

Рекуррентная формула вычисления значений ряда имеет вид:

$a_i = a_{i-1} + a_{i-2}$.

Применительно к ней написан следующий программный код:

```
var n,i,f,f1,f2:integer;
begin
f1:=0; f2:=1;
n:=StrToInt(Edit1.Text);
case n of
 1: Edit2.Text:=IntToStr(f1);
 2:
  begin
  MessageDlg('1 число ряда Фибоначчи равно '+ IntToStr(f1),
  MtInformation,[mbok],0);
  MessageDlg('2 число ряда Фибоначчи равно '+ IntToStr(f2),
  MtInformation,[mbok],0);
  Edit2.Text:=IntToStr(f2);
  end;
else
 begin
  MessageDlg('1 число ряда Фибоначчи равно ' + In-
tToStr(f1),
  MtInformation,[mbok],0);
  MessageDlg('2 число ряда Фибоначчи равно '+ IntToStr(f2),
  MtInformation,[mbok],0);
  for i:=3 to n do
  begin
  f:=f1+f2;
  MessageDlg(IntToStr(i)+' число ряда Фибоначчи равно '+
  IntToStr(f),
  MtInformation,[mbok],0);
  f1:=f2;
  f2:=f;
  end;
  Edit2.Text:=IntToStr(f);
end;
end;
end;
```

Здесь понадобились три переменные (f1,f2,f) для вычисления значения очередного члена последовательности, поскольку для этого нужно запомнить значения двух предыдущих.

Пример 4.14. Для заданного значения вещественной переменной x вычислить сумму ряда

$$S = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots.$$

Вычисления прекращаются тогда, когда значение очередного члена станет меньше заданного значения переменной Ɛ, т. е.

$$\frac{x^i}{i!} < \varepsilon.$$

Нетрудно увидеть, что между элементами последовательности, образующими сумму, есть рекуррентная зависимость. Если слагаемые в рассмотренном выражении записать в виде

$$a_0 = x^0 = 1;\ a_1 = x^1;\ a_2 = \frac{x^2}{2!};\ a_3 = \frac{x^3}{3!};\ ...,$$

то алгоритм данной рекуррентной последовательности может быть описан следующим образом:

$$a_i = \begin{cases} 1, \text{если } i = 0, \\ a_{i-1} \times \frac{x}{i}, \text{если } i > 0. \end{cases}$$

Программный код, обеспечивающий решение данной задачи, имеет вид:

```
var a,x,s,eps:real;
    i:integer;
begin
x:=StrToFloat(Edit1.Text);
eps:=StrToFloat(Edit2.Text);
a:=1;
s:=0;
i:=0;
while abs(a)>eps do
  begin
  s:=s+a;
  i:=i+1;
  a:=a*x/i;
  end;
Edit3.Text:=FloatToStr(s);
end;
```

Глава 5. Работа с массивами в Delphi

5.1. Общая характеристика массивов

Для упорядочения хранения данных с целью удобства обращения к ним и их обработки создаются массивы данных.

Массив данных (или информационный массив) – упорядоченный набор однородных данных, характеризующих какой-либо объект, процесс и т.д., рассматриваемых как единое целое и упорядоченных таким образом, что их описание однозначно определяет положение каждого элемента и путь доступа к нему.

Массив в Delphi определяется как индексный массив, представляющий собой именованный набор однотипных переменных, расположенных в памяти компьютера непосредственно друг за другом, доступ к которым осуществляется по индексу или индексам.

Индекс массива – целое число, либо значение типа, приводимого к целочисленному, указывающее на конкретный элемент массива.

Количество используемых индексов массива может быть различным. Массивы с одним индексом называются одномерными, с двумя – двумерными и т.д. Одномерному массиву в математике соответствует понятие векто-

ра, двумерному – матрицы. Чаще всего применяются массивы с одним или двумя индексами, реже - с тремя. Еще большее количество индексов встречаются крайне редко.

В простейшем случае массивы имеют постоянную размерность. Такие массивы называются статическими.

Массивы, размеры которых могут меняться в процессе выполнения программы, называются открытыми (динамическими). Эти массивы дают возможность более гибкой работы с данными, так как позволяют не прогнозировать заранее их хранимые объемы, а регулировать размер массива в соответствии с возникающей потребностью.

5.2.Описание массивов

Имена массивов формируются по тем же правилам, что и имена переменных. Массивы в Delphi определяются как данные структурированного типа. Их описание рассматривалось в разделе 2.3, в котором была дана характеристика используемых типов данных. Дополнительно к изложенному там, полезно иметь в виду, что значение конечного индекса описываемого массива более рационально задавать не при описании массива, а раньше, присваивая его константе. Этим достигается большая простота корректировки программы при возникновении необходимости изменить размер массива.

Например:

```
const
m=20;
n=10;
var
A:array [1..m] of Integer;
mass: array[1..n] of real;
```

Второй способ – создание нового типа данных и описание переменной этого типа. Создают новый тип в блоке описания, после ключевого слова type:

```
type
имя типа = array [список идексов] of  тип данных;
```

Здесь имя типа – любой допустимый в Delphi идентификатор, имя нового типа данных, тип данных – любой тип языка или созданный тип; список индексов – диапазон изменения номеров элементов создаваемой структуры.

Например:

```
Type
massiv=array[0..12] of real;
abc=array[-3..6] of integer;
matr=array[1..3] of massiv;
new_mass=array[1..4,0..6,-7..8,3..11] of real;
var
x,y:massiv;
z:abc;
```

```
M,A:matr;
Q:new_mass;
```

При описании открытого массива указывается тип входящих в него элементов, но не определяются границы изменения индексов:

Имя открытого массива: array of array of…тип.

Например:

Var mas 1: array of real;

mas 2: array of array of integer.

Нижняя граница открытого массива всегда равна нулю. Верхняя граница определяется стандартной функцией:

high(имя массива).

Например:

For i:=0 to high(X) do.

Здесь значение X должно быть определенно в программе ранее.

5.3. Ввод и вывод элементов массива

Одномерные массивы

Ввод и вывод значений элементов одномерных массивов осуществляется поэлементно. Блок-схемы алгоритмов ввода и вывода элементов для этих массивов изображены на рис 5.1. и 5.2.

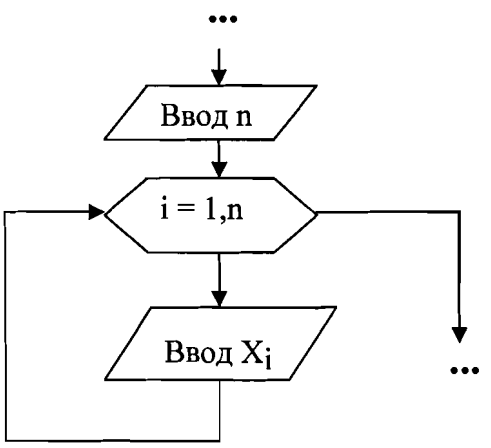

Рис 5.1. Алгоритм ввода одномерного массива

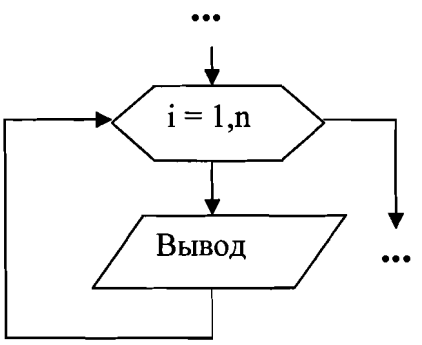

Рис 5.2. Алгоритм вывода одномерного массива

Одним из способов ввода значений одномерного массива в Delphi является использование функции InputBox:

```
Procedure TForm1.Button1Click(Sender: Tobject);
Var i,n:byte; X:array [1..20] of real;
begin
n:=StrToInt(Edit1.Text);//Количество элементов массива.
For i:=1 to n do        //Поэлементный ввод.
X[i]:=StrToFloat(InputBox("Ввод элементов массива"
,"Введите "+IntTOStr(i)+ "элемент", '0,00'));
```

Результатом работы такой подпрограммы будет многократное появление на экране окна ввода, куда следует поочередно вводить значения элементов массива. Для аналогичного вывода можно применять функцию MessageDlg:

```
for i:=1 to n do
MessageDl("X["+IntTOStr(i)+"]='FloatToStr(X[i]),
MtInformation,[mbOk],0),
```

которая будет открывать отдельное окно для каждого элемента.

Чтобы у пользователя была возможность просмотреть элементы массива одновременно, можно сформировать из них строку, а затем вывести ее в Форму или в окно информации (рис 5.3), написав для этого программу:

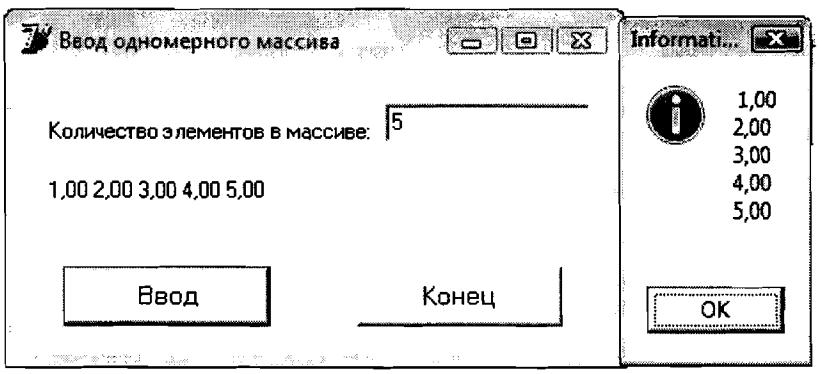

Рис.5.3. Вывод массива

```
var i,n:byte;
    X:array[1..20] of real;
    S:string;
begin
 n:=StrToInt(Edit1.Text);
 for i:=1 to n do
 X[i]:=StrToFloat(InputBox('Ввод элементов массива',
 'Введите '+IntToStr(i)+ 'элемент','0,00'));
 S:='';
 //В переменную строкового типа записывается пустая строка.
 for i:=1 to n do
 //Выполняется слияние элементов массива, преобразованных в
 //строки, и символов пробела - результат строка, в которой
 //элементы массива перечислены через пробел.
 S:=S+FloatToStrF(X[i],ffFixed,5,2)+' ';
 Label2.Caption:=S; //Вывод строки в Форму.
```

```
 S:=' ';
 //В переменную строкового типа записывается пустая строка.
 for i:=1 to n do
 //Элементы массива выводятся в столбец, т.к. к каждому
 //преобразованному элементу добавляется символ окончания
 //строки.
 S:=S+FloatToStrF(X[i],ffFixed,5,2)+chr(13);
 MessageDlg(S,MtInformation,[mbok],0);
 //Вывод в виде сообщения.
end;
```

Ввод и вывод значений одномерных массивов может выполняться по излагаемым ниже правилам ввода-вывода двумерных, в этих случаях одномерный массив рассматривается как вырожденная матрица.

Двумерные массивы (матрицы)

Ввод и вывод значений матрицы осуществляются построчно, как показано в блок-схеме на рис 5.4.

Средства Delphi обеспечивают возможность отображения вводимых и выводимых двумерных массивов в Форме в виде соответствующих им по размерности матриц.

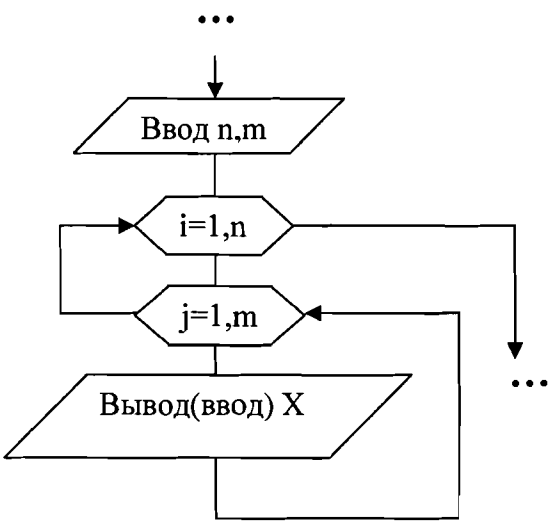

Рис 5.4. Ввод и вывод элементов матрицы

Для этого используется компонент, который называется `StringGrrid`, он представляет собой таблицу, ячейки которой представлены для чтения или редактирования (рис 5.5).На панели **Tool Palette** этот компонент находится в группе **Additional.** Управлять видом таблицы можно при помощи свойств Инспектора Объектов.

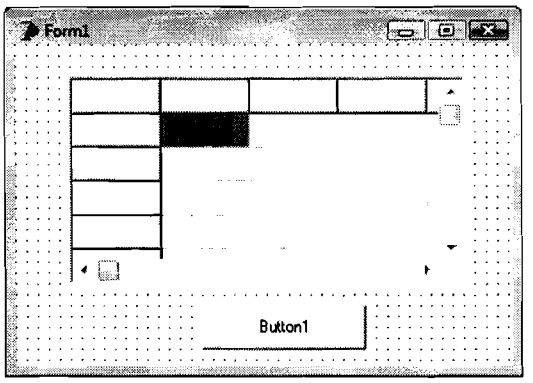

Рис.5.5.Размещение компонента «Таблица» на Форме

Число строк и столбцов таблицы можно задавать, изменяя свойства ColCont и RowCont соответственно. Свойства Col и Row –номер столбца и номер строки (индексы) выделенной ячейки. Обратиться к ячейке таблицы можно при помощи свойства Cells[Col, Row]. Многие параметры таблицы определены в свойстве Options. Наиболее важным из них является возможность редактирования таблицы goEditting.

Создадим небольшую программу, с помощью которой можно осуществить ввод массива целых чисел, а затем вывести его в обратном порядке. Для простоты освоения излагаемого материала ограничимся работой с одномерным массивом, напомнив, что такой массив можно рассматривать как вырожденную матрицу.

Разместим на форме надпись, поле ввода, две таблицы и одну кнопку. Зададим свойства компонентам StringGrid1 и StringGrid2, как показано в таблице 5.1.

Таблица 5.1.Свойства компонентов StringGrid1 и StringGrid2

Свойство	StringGrid1	StringGrid2	Описание свойства
Height	60	60	высота
Width	386	385	ширина
ColCount	10	10	количество столбцов
RowCount	1	1	количество строк
Options.goEditing	true	false	возможность редактирования

Поле ввода понадобится для определения элементов в будущем массиве. В первую таблицу будем вводить элементы исходного массива, а во вторую - выводить преобразованный массив (рис5.6). Выполнение этого будет обеспечено следующей программой:

```
procedure TForm1.Button1Click(Sender: TObject);
var n,i:integer;
    X:array[1..10] of integer;
begin
 n:=StrToInt(Edit1.Text); //Количество элементов в массиве.
 for i:=1 to n do         //Ввод массива.
```

```
 //Из поля таблицы считывается элемент, преобразовывается
 //в число и присваивается элементу массива.
 X[i]:=StrToInt(StringGrid1.Cells[i,0]);
 for i:=1 to n do        //Вывод массива.
 //Элемент массива преобразовывается в строку и
 //помещается в поле таблицы.
 StringGrid2.Cells[i,0]:=IntToStr(X[n-i+1]);
end;
```

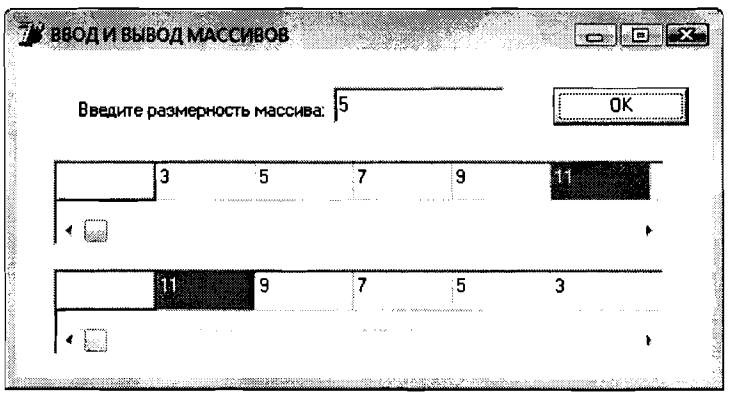

Рис.5.6.Форма для ввода и вывода массивов

5.4. Операции над массивами

Для работы над массивами, как единым целым, в Delphi определена только операция присваивания, применяемая для массивов одинаковой структуры.

Например:

```
Var c, d: array [1..20] of real;
Begin
.....
c:=d;
.....
end.
```

В этом случае в элементы массива **c** будут записаны значения соответствующих элементов массива **d**.

Все остальные возможные действия с массивами выполняются путем обращения к их элементам. Обращение к элементам массива осуществляется с помощью индекса для одномерного или индексов для двумерного массива. В качестве индексов могут использоваться положительные целые числа, переменные или выражения. Например: a[1], a[10], a[i], a[i+3].

С элементами массива можно выполнять все действия, которые доступны для переменных того же типа.

Алгоритмы, с помощью которых ведется обработка одномерных массивов, аналогичны алгоритмам обработки последовательности простых переменных (вычисление суммы, произведения, поиск элементов по определенному признаку и т.д.). Как правило, это циклические алгоритмы и соот-

ветствующие им программы. Отличие состоит в том, что вместо многоразового обращения в цикле к одной переменной осуществляется последовательное обращение к элементам массива. Структура цикла обработки элементов подобна структуре циклов ввода и вывода их значений.

При работе с массивами целесообразно предусмотреть в начале программы ввод значений их элементов посредством создания отдельного цикла, а затем с помощью другого цикла организовать процесс обработки введенных данных. Это позволит пользователю одноразово задать все исходные данные, а затем не вмешиваться в работу программы, что выгодно при решении объемных задач. Значения элементов массива могут не вводиться, а быть присвоены в разделе описаний как константы. Например:

Const s:mas=(0,5,7,9.5,15e1); .

Рассмотрим работу с массивами на примерах, предусмотрев при этом возможность самостоятельной работы изучающих программирование на основе знаний и практических навыков, полученных ими при изучении материала предыдущей и начала данной глав.

Для приведенных ниже трех примеров рекомендуется самостоятельно разработать виды форм выходных документов, блок-схемы алгоритмов и написать программы, сравнив затем разработанные варианты циклов обработки данных с приведенными в примерах.

Пример 5.1. Вычислить: $S = \sum_{i=1}^{n} x_i$.

Соответствующий алгоритму фрагмент программы суммирования будет иметь вид:

```
S:=0;
for i:=1 to n do
S:=S+X[i];
```

Пример 5.2. Вычислить: $P = \prod_{i=1}^{n} x_i$.

Соответствующий фрагмент программы будет иметь вид:

```
P:=1;
for i:=1 to n do
P:=P*X[i];
```

Пример 5.3. Вычислить среднее арифметическое значений n элементов одномерного массива X.

Фрагмент программы, обеспечивающий заданное вычисление, имеет вид:

```
S:=0;
for i:=1 to n do
 S:=S+X[i];
ar:=S/n;
```

Пример 5.4. Найти максимальный элемент одномерного массива X, состоящего из n элементов, а также номер этого элемента.

Исходные данные: X[i] – элементы массива (i=1,2,…n).

Искомые характеристики: Max – максимальный элемент массива, Nmax – номер максимального элемента массива.

Соответствующий фрагмент программы имеет вид:

```
Max:=X[1];
Nmax:=1;
for i:=2 to n do
if X[i]>Max then
 begin
 Max:=X[i];
 Nmax:=i;
 end;
MessageDlg('Max='+FloatToStr(Max)+chr(13)+'Nmax='+
IntToStr(Nmax),MtInformation,[mbok],0);
```

Алгоритм решения задачи следующий. Пусть в переменной с именем Max хранится значение максимального элемента массива, а в переменной с именем Nmax – его номер. Предположим, что первый элемент массива является максимальным и запишем его в переменную Max, а в Nmax – его номер (то есть 1). Затем все элементы, начиная со второго, сравним в цикле с максимальным. Если текущий элемент массива оказывается больше максимального, записываем его в переменную Max, а текущее значение индекса i- в переменную Nmax. Процесс определения максимального элемента в массиве изображен при помощи блок-схемы на рис 5.7.

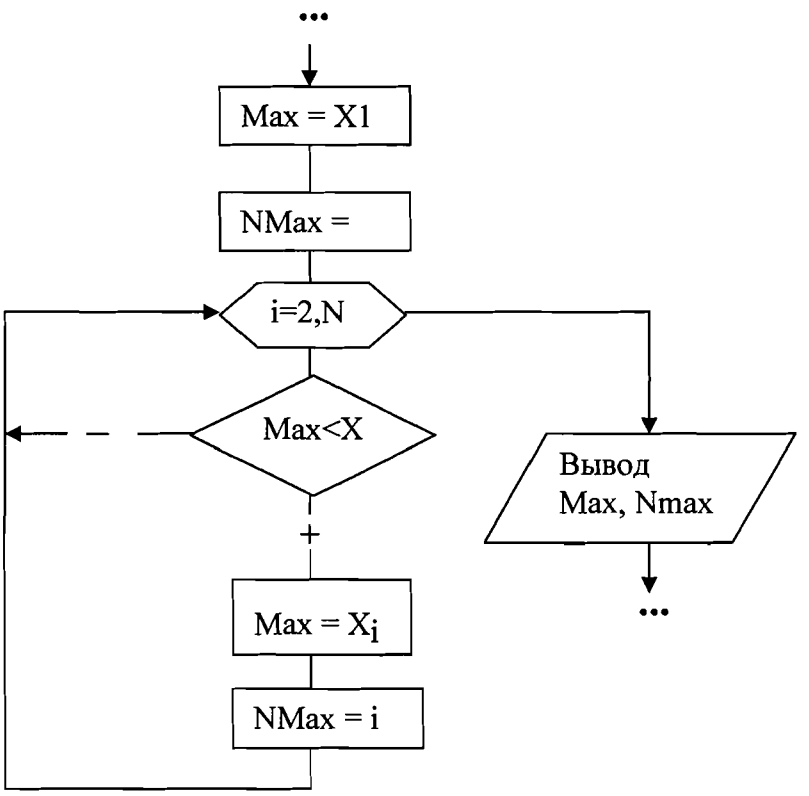

Рис 5.7. Поиск максимального элемента и его номера в массиве

Пример 5.5. Произвести сортировку элементов в одномерном массиве X из n элементов так, чтобы они расположились по возрастанию, т.е. $X_1 \leq X_2 \leq \ldots \leq X_n$.

Существует несколько алгоритмов сортировки. Рассмотрим один из них, который называется «метод пузырька».

Сортировка пузырьковым методом популярна из-за простоты алгоритма схожего с процессом движения пузырьков в резервуаре с водой, когда каждый пузырек находит собственный уровень. Алгоритм использует метод обменной сортировки, основанный на выполнении в цикле операций сравнения и, при необходимости, обмена соседних элементов.

Рассмотрим алгоритм пузырьковой сортировки более подробно. Сравним нулевой элемент массива с первым. Если нулевой элемент окажется больше первого, поменяем их местами. Те же действия выполняем для первого и второго, второго и третьего, i-го и (i+1)-го, предпоследнего и последнего элементов. В результате этих действий самый большой элемент станет на последнее место (n). Теперь повторим данный алгоритм сначала, но последний элемент, рассматривать не будем, так как он уже занял свое место. После проведения данной операции самый большой элемент оставшегося массива станет на место (n-1). Будем повторять эти действия до тех пор, пока не упорядочим весь массив.

В таблице 5.2 представлен процесс упорядочивания элементов в массиве.

Таблица 5.2. Процесс упорядочивания элементов в массиве по возрастанию

Номер элемента	**1**	**2**	**3**	**4**	**5**
Исходный массив	7	3	5	4	2
Первый просмотр	3	5	4	2	7
Второй просмотр	3	4	2	5	7
Третий просмотр	3	2	4	5	7
Четвертый просмотр	2	3	4	5	7

Нетрудно заметить, что для преобразования массива, состоящего из n элементов, необходимо просмотреть его n-1 раз, каждый раз уменьшая диапазон просмотра на один элемент. Блок-схема описанного алгоритма приведена на рис 5.8.

Для перестановки элементов используется буферная переменная b, в которой временно хранится значение элемента, подлежащего замене.

Текст программы, сортирующий элементы в массиве по возрастанию методом «пузырька»:

```
var i,n,j:byte;
    b:real;
    X:array[1..100] of real;
    S:string;
begin
 n:=StrToInt(Edit1.Text); //Количество элементов массива.
```

```
for i:=1 to n do          //Поэлементный ввод.
X[i]:=StrToFloat(InputBox('Ввод элементов массива',
                'Введите'+IntToStr(i)+ ' элемент','0,00'));
for j:=1 to n-1 do
for i:=1 to n-j do
if X[i]>X[i+1] then
//Если текущий элемент больше следующего, то
begin
 b:=X[i];
 //сохранить значение текущего элемента.
 X[i]:=X[i+1];
 //Заменить текущий элемент следующим.
 X[i+1]:=b;
 //Заменить следующий элемент текущим.
end;
//Вывод упорядоченного массива.
S:=' ';
for i:=1 to n do
S:=S+FloatToStrF(X[i],ffFixed,5,2)+' ';
Label2.Caption:=S;
end;
```

Заметим, что для перестановки элементов в массиве по убыванию их значений необходимо при сравнении элементов массива изменить знак > на <.

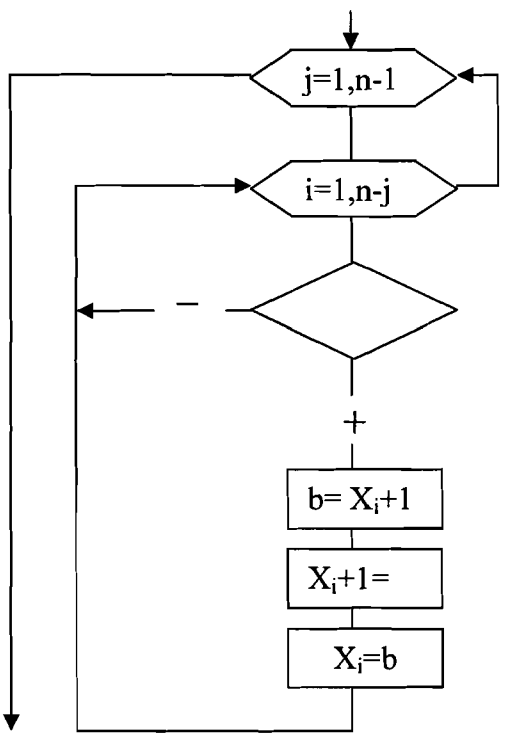

Рис 5.8.Сортировка массива пузырьковым методом

Пример 5.6. Произвести выбор всех четных элементов массива A, состоящего из к целых положительных чисел, с записью их в массив B.

Дан массив A состоящий из k целых положительных чисел. Записать все четные по значению элементы массива A в массиве B.

Алгоритм решения задачи заключается в следующем. Нужно последовательно перебирать элементы массива A. Если среди них есть четные числа, то они записываются в массив B. Допустим, что первый четный элемент хранится в массиве A под номером 2, второй и третий – под номерами 4 и 5 соответственно, а четвертый – под номером 7. В массиве B этим элементам будут присвоены номера 1, 2, 3 и 4. Значит, для их формирования необходимо определить дополнительную переменную. Рассмотрим алгоритм, который изображен в виде блок-схемы на рис 5.9. Здесь индексы массива A хранятся в переменной i, а для номеров массива B зарезервирована переменная m. Операция, выполняемая в блоке 2, означает, что в массиве может не быть искомых элементов. Если же условие в блоке 5 выполняется, переменная m увеличивается на единицу, а значение соответствующего элемента массива A записывается в массив B под номером m (блок 7). Условный блок 8 необходим, чтобы проверить,выполнилось ли хотя бы раз условие поиска (блок 5).

Приведенный ниже фрагмент программы реализует описанный алгоритм:

```
procedure TForm1.Button1Click(Sender: TObject);
var k,i,m:integer;
    A,B:array [1..10] of integer;
begin
 k:=StrToInt(Edit1.Text);
 for i:=1 to k do
 A[i]:=StrToInt(StringGrid1.Cells[i,0]);
 m:=0;
 for i:=1 to k do
  begin
  if A[i] mod 2=0 then
   begin
   m:=m+1;
   B[m]:=A[i];
   end;
  end;
 if m<>0 then
 for i:=1 to m do
 StringGrid2.Cells[i,0]:=IntToStr(B[i])
 else
 MessageDlg('B массиве нет четных
             элементов',MtInformation,[mbok],0);
end;
```

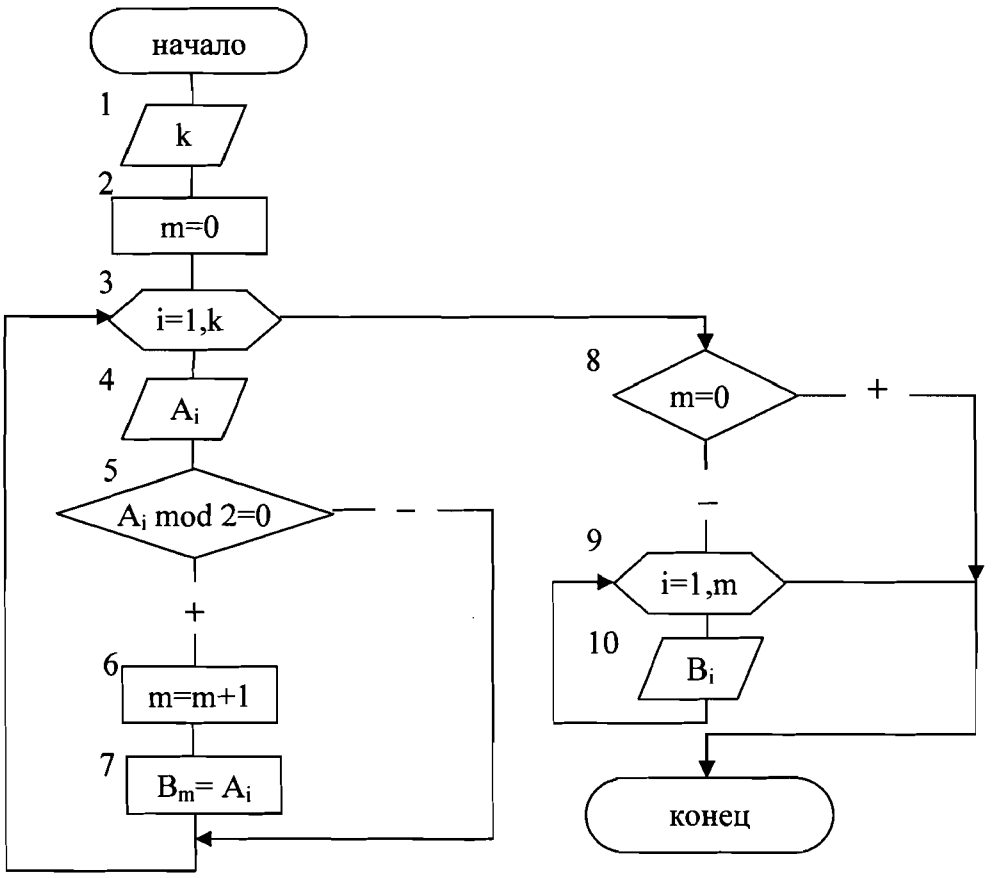

Рис 5.9. Алгоритм формирования массива B из соответствующих элементов массива A

Пример 5.7. Разместить в Форме матрицу с исходными данными размером 5x5 и матрицу,транспонированную относительно исходной.

Разместим в Форме две надписи Label1 и Label2 со свойствами Caption «Заданная матрица» и «Транспонированная матрица», два компонента StringGrid, изменив их свойства, как показано в табл. 5.3, и кнопку BitBnt1 со свойствами King=bkOk.

Таблица 5.3. Свойства компонентов StringGrid

Свойства	StringGrid1	StringGrid2	Описание свойства
Top	30	30	Расстояние от верхней границы таблицы до верхнего края формы
Left	15	240	Расстояние от левой границы таблицы до левого края формы
Height	130	130	высота таблицы
Width	200	200	ширина таблицы
ColCount	5	5	количество столбцов
RowCount	5	5	количество строк
DefaultColWidth	30	30	ширина столбца
DefaultRowHeight	20	20	высота строки
Options.goEditing	true	false	возможность редактирования таблицы

Далее приведен текст программы, которая будет выполняться, если пользователь щелкнет по кнопке **Ok**. Результат работы программы представлен на рис 5.10:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
const n=4;m=4;      //Размерность матрицы.
var
i,j:byte;           //Индексы матрицы:
//i - строки, j - столбцы.
A:array [1..n,1..m] of integer; //Целочисленная матрица.
begin
//Исходные данные считываются из ячеек таблицы на форме
//и их значения записываются в двумерный массив A.
for i:=1 to n do    //Цикл по номерам строк.
for j:=1 to m do    //Цикл по номерам столбцов.
//Обращение к элементам матрицы происходит по строкам.
A[i,j]:=StrToInt(StringGrid1.Cells[i,j]);
//Элементы матрицы A выводятся в ячейки таблицы на форме.
for i:=1 to n do    //Цикл по номерам строк.
for j:=1 to m do    //Цикл по номерам столбцов.
//Обращение к элементам матрицы происходит по столбцам.
StringGrid2.Cells[i,j]:=IntToStr(A[j,i]);
end;
```

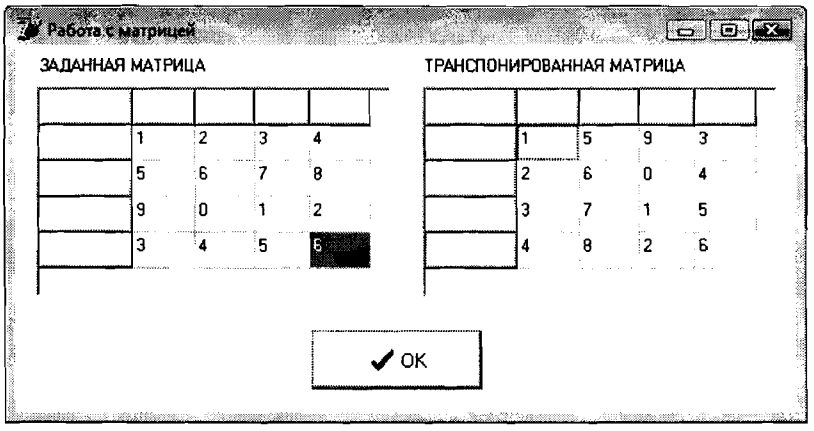

Рис.5.10. Ввод и вывод элементов матрицы

Глава 6. Подпрограммы в Delphi

6.1. Понятие подпрограммы

Из теории алгоритмов известно понятие *вспомогательного алгоритма.* Его определяют как алгоритм решения некоторой подзадачи из основной решаемой задачи. В таком случае алгоритм решения исходной задачи определяется как *основной алгоритм.* В языках программирования вспомогательные алгоритмы называются *подпрограммами.*

Подпрограмма – это фрагмент программного кода, представляющий собой именованную логически законченную группу операторов языка, кото-

рую можно вызывать для выполнения в различных местах программы любое количество раз.

В Delphi используется два вида подпрограмм: *процедуры* и *функции.*

Для того чтобы подпрограмма выполнялась, её нужно вызвать. Вызов записывается в том месте программы, где требуется получить результаты работы подпрограммы. Подпрограмма вызывается по имени. Процедура вызывается с помощью отдельного оператора, а функция – из правой части оператора присваивания. Например:

```
inc(i)              //вызов процедуры;
y := sin(x) + 1     //вызов функции.
```

В примерах и в качестве процедуры и в качестве функции использованы стандартные модули Delphi. Первый из них - `inc(i)` увеличивает значение переменной `i` на единицу. Его действие аналогично действию оператора присваивания `i:=i+1`. Второй – `sin(x)`, возвращает синус аргумента `x`, который задает угол в радианах. В обоих случаях по заданному имени выполняется обращение к соответствующему программному коду, включенному в программное обеспечение системы Delphi, с использованием которого затем производится вычисление относительного заданного значения аргумента.

В процессе написания программ пользователям приходится самим разрабатывать программы в виде процедур и функций, задавая их имена и соответствующие программные коды.

Информация, передаваемая в подпрограмму для обработки, называется *параметрами*, а результат вычислений – *значениями*. Обращение к подпрограмме называют *вызовом*. Перед вызовом подпрограмма должна быть обязательно *описана* в разделе описаний. Описание подпрограммы состоит из *описания* и *тела*. В заголовке объявляется имя подпрограммы и в круглых скобках ее параметры, если они есть (для функции необходимо сообщить тип возвращаемого ею результата). Тело подпрограммы следует за заголовком и состоит из описаний и исполняемых операторов.

Любая подпрограмма может содержать описание других подпрограмм. Константы, переменные и типы данных могут быть объявлены как в основной программе, так и в подпрограммах различной степени вложенности. Переменные, константы и типы, объявленные в основной программе до определения подпрограмм, называются *глобальными*. Они доступны всем функциям и процедурам. Переменные, константы и типы, описанные в какой- либо подпрограмме, доступны только в ней и называются *локальными.*

Чтобы правильно определить область действия идентификаторов (переменных) в подпрограммах, необходимо придерживаться следующих правил:

• каждая переменная, константа или тип перед использованием должны быть описаны;

• областью действия переменной, константы или типа является подпрограмма, в которой они описаны;

• все имена в пределах подпрограммы, где они объявлены, должны быть уникальными и не совпадать с именем самой подпрограммы;

• одновременные локальные и глобальные переменные – это разные переменные, обращение к таким переменным в подпрограмме трактуется как обращение к локальным переменным (глобальные переменные не доступны);

•при обращении к подпрограмме доступны объекты, которые объявлены в ней до ее описания.

Обмен информацией между вызываемой и вызывающей функциями осуществляется с помощью механизма *передачи параметров.* Список переменных, указанный в заголовке функции, называется *формальными параметрами*, или просто *параметрами* подпрограммы. Все переменные из этого списка могут использоваться внутри подпрограммы. Список переменных в операторе вызова подпрограммы – это *фактические параметры*, или *аргументы*.

Механизм передачи параметров обеспечивает замену формальных параметров фактическими и позволяет выполнять подпрограмму с различными данными. Между фактическими параметрами в операторе вызова и формальными параметрами в заголовке подпрограммы устанавливается взаимно однозначное соответствие.

Количество, типы и порядок следования формальных и фактических параметров должны совпадать.

Передача параметров выполняется следующим образом. Вычисляются выражения, стоящие на месте фактических параметров. В памяти выделяется место под формальные параметры в соответствии с их типами. Затем формальным параметрам присваиваются значения фактических. Выполняется проверка типов, и при их несоответствии выдается диагностическое сообщение.

Формальные параметры процедуры можно разделить на два класса: параметры – значения и параметры – переменные. При передаче данных через *параметры – значения* подпрограмма работает с копиями фактических параметров и доступа к исходным значениям аргументов у нее нет. При передаче данных через *параметры – переменные* подпрограмма обращается к адресам фактических параметров, и, соответственно, имеет доступ к ячейкам памяти, в которых хранятся значения аргументов. Таким образом, *параметры – значения* подпрограмма изменить не может, в отличие от *параметров – переменных.* Следовательно, *параметры – значения* могут использоваться только в качестве входных данных подпрограммы, а *параметры - переменные* – в качестве входных и выходных данных.

6.2. Процедуры в Delphi

Описание процедуры имеет вид:

```
procedure имя процедуры(список формальных параметров);
label
     список меток;
```

```
const
     список констант;
type
     список типов;
var
    список переменных;
begin
     //Тело процедуры.
end;
```

Начинается описание с *заголовка процедуры*, где `procedure` – ключевое слово языка, имя процедуры - любой допустимый в Delphi идентификатор, список формальных параметров - имена формальных параметров и их типы, разделенные точкой с запятой:

```
procedure name1 (r:real; i:integer; c:char);
```

Однотипные параметры могут быть перечислены через запятую:

```
procedure name2 (a,b:real; i,j,k:integer;);
```

Список формальных параметров необязателен и может отсутствовать:

```
procedure name3;
```

Если в заголовке процедуры будут применяться параметры – переменные, то перед ними необходимо указать служебное слово `var`:

```
procedure name4 (x,y:real; var z:real;);
                                        //x,y    -    параметры-
значения,
                                      //z - параметр-переменная.
```

После заголовка следует *тело процедуры*, которое состоит из раздела описаний (констант, типов, переменных, процедур и функций, используемых в процедуре) и операторов языка, реализующих алгоритм процедуры. Раздел описаний в процедуре может отсутствовать, если в нем нет необходимости.

Для *обращения к процедуре* необходимо использовать оператор вызова:

```
имя процедуры(список фактических параметров);
```

Фактические параметры в списке оператора вызова отделяются друг от друга запятой:

```
a:=5.3; k:=2; s:='a';
name1(a, k, s);
```

Если при описании процедуры формальные параметры отсутствовали, то и при описании их быть не должно:

```
name3;
```

Рассмотрим построение и использование процедур на примере.

Пример 6.1. Найти разность средних арифметических значений двух вещественных массивов из 10 элементов.

Разработанный программистом для решения этой задачи программный код будет иметь вид:

```
const n=10;
type mas=array[1..n] of real;
var a,b:mas;
    i:integer;
```

```
        dif,av a,av b:real;
      procedure average (x:mas; var av:real);                {1}
      var i:integer;
      begin
       av:=0;
       for i:=1 to n do av:=av+x[i];
       av:=av/n;
      end;
      begin
{2}
       for i:=1 to n do
         a[i]:=StrToFloat(InputBox('Ввод элементов
         массива','Введите '+IntToStr(i)+' элемент','0,00'));
       for i:=1 to n do
         b[i]:=StrToFloat(InputBox('Ввод элементов
         массива','Введите '+IntToStr(i)+' элемент','0,00'));
       average(a,av a);                                       {3}
       average(b,av b);                                        {4}
       dif:=av_a-av_b;
       Edit1.Text:=FloatToStr(dif);
      end;
```

Описание процедуры `average` расположено в строках с {1} по {2}. В строках {3} и {4} эта процедура вызывается для обработки сначала массива a, затем – массива b. Массивы передаются в качестве аргументов. Результат вычисления среднего арифметического возвращается в главную программу через второй параметр процедуры. Список аргументов при вызове как бы накладывается на список параметров, поэтому они должны попарно соответствовать друг другу.

6.3. Функции в Delphi

Описание функции имеет вид:

```
function имя функции(список формальных параметров):
         тип результата;
label
     список меток;
const
     список констант;
type
     список типов;
var
    список переменных;
begin
     //Тело функции.
```

Заголовок функции содержит: служебное слово-`function`; любой допустимый в Delphi идентификатор - `имя функции`; имена формальных параметров и их типы, разделенные точкой с запятой - `список формальных параметров`; тип возвращаемого функцией значения - `тип резуль-`

тата (функции могут возвращать скалярные значения целого, вещественного, логического, символьного, или ссылочного типа).

Примеры описания функций:

```
function fun1(x:real):real;
function fun2(a,b:integer):real;
```

Тело функции состоит из раздела описаний (констант, типов, переменных, процедур и функций, используемых в процедуре) и операторов языка, реализующих ее алгоритм. Раздел описаний в функции может отсутствовать, если в нем нет необходимости.

Функция вычисляет одно значение, которое передается через ее имя. Для этого в теле функции всегда должен быть один оператор, присваивающий значение имени функции.

Например:

```
function fun2(a,b:integer):real;
begin
fun2:=(a+b)/2;
end;
```

Обращение к функции осуществляется по ее имени с указанием списка формальных параметров:

```
имя функции (список формальных параметров);
```

Например:

```
y:=fun1(1.28)
z:= fun1(1.28)/2+fun2(3.8);
```

Рассмотрим построение и использование функций на примере.

Пример 6.2. Разработать подпрограмму в виде функции для условий примера 6.1.

```
const n=10;
type mas=array[1..n] of real;
var a,b:mas;
    i:integer;
    dif:real;
function average(x:mas):real;                                    {1}
  var i:integer;
      av:real;
  begin
   av:=0;
   for i:=1 to n do av:=av+x[i];
   average:=av/n;                                                {2}
  end;
begin
 for i:=1 to n do
   a[i]:=StrToFloat(InputBox('Ввод элементов
   массива','Введите '+IntToStr(i)+' элемент','0,00'));
 for i:=1 to n do
   b[i]:=StrToFloat(InputBox('Ввод элементов
   массива','Введите '+IntToStr(i)+' элемент','0,00'));
 dif:=average(a)-average(b);                                     {3}
 Edit1.Text:=FloatToStr(dif);
end;
```

Оператор {1} – заголовок функции. Оператор {2} присваивает вычисленное значение имени функции. В {3} функция вызывается дважды: сначала для одного массива, затем для другого.

Относительно приведенной ниже постановки задачи будут рассмотрены примеры с программными кодами возможных видов процедур, а также - соответствующей подпрограммы – функции.

Требуется написать программу определения наибольшего общего делителя для вычисленных значений выражений:a+b, |a-b|, a×b. Формально это можно записать как НОД(a+b, |a-b|, a×b).

В основу решения данной задачи может быть положен следующий алгоритм: если значения x, y, z – натуральные числа, то НОД(x, y, z)=НОД(НОД (x, y), z). Иначе говоря, нужно найти НОД двух величин, а затем НОД полученного значения и третьего числа.

Очевидно, что вспомогательным алгоритмом для решения рассматриваемой задачи будет алгоритм получения наибольшего общего делителя двух чисел. Эта задача решается с помощью алгоритма Евклида, содержание которого было приведено в примере 4.3 главы 4. Рассмотрим возможные подходы к получению решения поставленной задачи.

Пример 6.3. В приведенном ниже программном коде обмен аргументами и результатами между основной программой и процедурой производится через параметры (формальные и фактические).

Ввод значений исходных данных и вывод результата производится через поля ввода и вывода.

```
var A,B,C:integer;
Procedure Evklid(M,N:integer; var K:integer);
  begin
   while M<>N do
   if M>N then M:=M-N
   else N:=N-M;
   K:=M;
  end;
begin
 A:=StrToInt(Edit1.Text);
 B:=StrToInt(Edit2.Text);
 Evklid(A+B,abs(A-B),C);
 Evklid(C,A*B,C);
 Edit3.Text:=IntToStr(C);
end;
```

В рассмотренном примере формальные параметры M и N являются параметрами – значениями. Это аргументы процедуры. При обращении к ней в первый раз им соответствуют значения выражений a+b и abs(a-b), во второй – c и a×b. Параметр k является параметром – переменной. В ней помещается результат работы процедуры. В обоих обращениях к процедуре соответствующим фактическим параметром является переменная c. Через эту переменную основная программа получает результат.

Пример 6.4. Использование процедуры без параметров применительно к рассматриваемой задаче может быть описано:

```
var A,B,K,M,N:integer;
Procedure Evklid;
begin
 while M<>N do
 if M>N
 then M:=M-N
 else N:=N-M;
 K:=M;
end;
begin
 A:=StrToInt(Edit1.Text);
 B:=StrToInt(Edit2.Text);
 M:=A+B;
 N:=abs(A-B);
 Evklid;
 M:=K;
 N:=A*B;
 Evklid;
 Edit3.Text:=IntToStr(K);
end;
```

В программе NOD1 переменные M,N,k – локальные, используемые внутри процедуры; переменные A,B,C – глобальные. Соответственно, внутри процедуры переменные A,B,C не используются. Связь между внешним блоком и процедурой осуществляется через её параметры.

В программе NOD2 все переменные являются глобальными. В процедуре Evklid нет ни одной локальной переменной и, соответственно, нет параметров. Поэтому переменные M и N, используемые в процедуре, получают свои значения через оператор присваивания в основном блоке программы. Результат вычислений присваивается глобальной переменной k, значение которой затем выводится на экран.

Использование механизма передачи через параметры делает процедуру более универсальной, независимой от основной программы. Однако в некоторых случаях оказывается удобней использовать передачу через глобальные переменные. Обычно это применяется при создании процедур, работающих с большими объемами данных. В таком случае глобальное взаимодействие экономит объем задействованной памяти компьютера.

Пример 6.5. Программа решения рассматриваемой задачи с использованием функции будет иметь вид:

```
var A,B,Rez:integer;
Function Evklid(M,N:integer):integer;
begin
 while M<>N do
 if M>N
 then M:=M-N
 else N:=N-M;
 Evklid:=M;
```

```
end;
begin
 A:=StrToInt(Edit1.Text);
 B:=StrToInt(Edit2.Text);
 Rez:=Evklid(Evklid(A+B,abs(A-B)),A*B);
 Edit3.Text:=IntToStr(Rez);
end;
```

Из программы видно, что тело функции отличается от тела процедуры только тем, что в функции результат присваивается переменной с тем же именем, что и имя функции. Правила соответствия между формальными и фактическими параметрами остаются теми же.

Сравнивая приведенные выше программы, можно сделать вывод, что программа NOD3 имеет определенные преимущества перед предшествующими ей. Функция позволяет получить результат путем выполнения одного оператора присваивания.

6.4. Рекурсивные подпрограммы

В математике рекурсивным называют определение любого понятия через себя самого. Классическим примером рекурсивного алгоритма является определение факториала любого числа большего или равного нулю:

$$n! = \begin{cases} 1, \text{если } n = 0; \\ n(n-1), \text{если } n > 0. \end{cases}$$

Здесь функция факториала определена через факториал.

Вариант 0!=1 является заданным по определению. Это значение является опорным, от которого начинается вычисление всех последующих значений факториала, используя их предыдущие вычисления.

Вычисление факториала с использованием циклической программы рассматривалось в примере 4.6 главы 4. Возможным для этой цели может быть и использование рекурсивной подпрограммы.

Под рекурсией в программировании понимают программу, которая вызывает сама себя. Рекурсивные подпрограммы (функции и процедуры) используются для компактного описания рекурсивных алгоритмов. Так, например, функция для вычисления факториала может иметь вид:

```
function factor(N:integer):integer;
begin
 if n=0
 then factor:=1
 else factor:=n*factor(n-1);
end;
```

Рассмотрим организацию работы приведенной функции. При вычислении функции, например, с аргументом,равным,3 произойдет повторное обращение к функции factor(2). Это потребует обращение к функции factor(1). И наконец, при обращении к factor(0) будет получено исходное числовое значение.

Последовательность рекурсивных обращений к функции должна обязательно выходить на определенное значение. Весь маршрут последователь-

ных обращений компьютер запоминает в специальной области памяти, называемой *стеком.*

Таким образом, выполнение рекурсивной функции происходит в два этапа: прямой ход – заполнение стека; обратный ход – цепочка вычислений по обратному маршруту, сохраненному в стеке.

Достоинством рекурсивных функций является обеспечение компактной записи программы, а недостатком-большой расход памяти на повторные вызовы функций и затраты времени на выполнение программы. В частности, рассмотренная ранее программа вычисления факториала будет выполняться более оперативно.

В заключение рассмотрим еще один пример рекурсивного алгоритма и описание его с использованием программы функции.

Пример 6.6. В главе 4 был рассмотрен пример 4.13 вычисления значения n-го члена ряда Фибоначчи. Написать программу, используя для получения решения подпрограмму – функцию.

Алгоритм рекурсивной функции fibonachi изображен на блок – схеме рис.6.1.

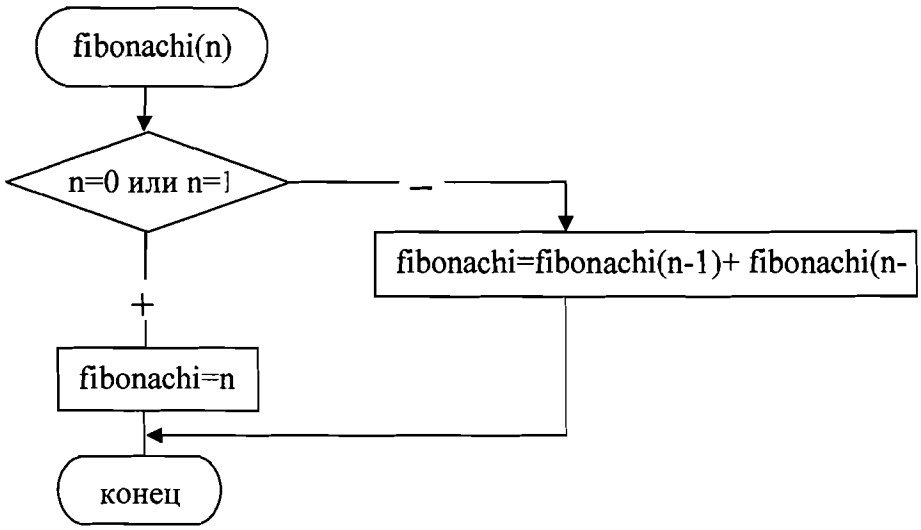

Рис.6.1. Рекурсивный алгоритм вычисления числа Фибоначчи

Текст подпрограммы:

```
function fibonachi(n:word):word;
begin
 if (n=0) or (n=1) then
 fibonachi:=n
 else
 fibonachi:=fibonachi(n-1)+fibonachi(n-2);
 end;
var x:word;
begin
 x:=StrToInt(Edit1.Text);
 Label2.Caption:=IntToStr(x)+' число Фибоначчи '+
                 IntToStr(fibonachi(x));
 end;
```

Глава 7. Графика в Delphi. Построение графиков и диаграмм

Delphi предоставляет пользователю достаточно широкие возможности по созданию разнообразных графических объектов. В частности, это возможность построения графиков и диаграмм различных видов, что удобно для оценки и анализа результатов решения многих инженерных задач. Построение графиков и диаграмм в Delphi может быть осуществлено с использованием различных программных средств. Наиболее просто и оперативно эта работа выполняется при обращении к компоненту `TChart`.

Компонент `TChart` предназначен для работы с диаграммами различных типов, в том числе и объемными. Он расположен последним на странице Additional. У этого компонента очень много свойств, которые подробно описаны в справке по Delphi.

Работа по созданию приложения, предназначенного для создания графиков и диаграмм, начинается с помещения компонента `TChart` в Форму. Затем необходимо установить свойства компонента с помощью Инспектора Объектов и собственного редактора компонента `TChart`. Вызов редактора компонента осуществляется двойным щелчком мыши по компоненту, либо щелчком правой кнопкой и выбора из контекстного меню команды Edit Chart. Окно редактора компонент, изображенное на рис. 7.1 и 7.2, состоит из вкладок Chart и Series.

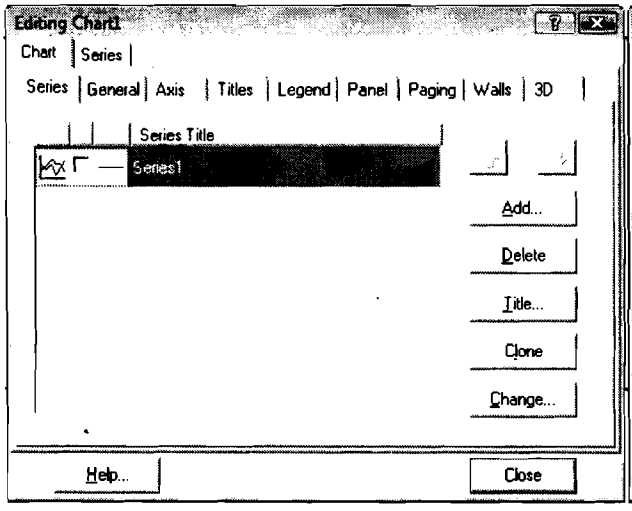

Рис.7.1. Свойства компонента `TChart`, вкладка Chart

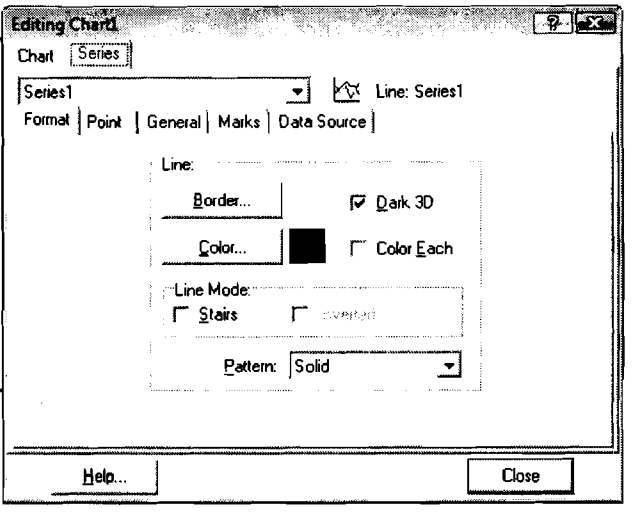

Рис.7.2. Свойства компонента `TChart`, вкладка Series

Вкладки используются для установки общих свойств графика:

- Chart – установки общих свойств графика;
- Series – форматирование отдельных линий данных, отображаемых на графике.

Компонент `TChart` обеспечивает выполнение многих действий. Ниже рассматриваются только основные из них. С остальными можно ознако-

миться во встроенной справке Delphi или просто опробовать их, экспериментируя с графиками и диаграммами.

Для того чтобы подключить компонент TChart к работающему приложению Delphi, недостаточно только изменения свойств компонента. Для ввода отображаемых значений непосредственно из программы надо использовать методы класса Series. Для управления значениями, по которым строится диаграмма (график функции), используются методы Add, AddXY, Delete, Clear. Рассмотрим их более подробно.

```
function Add(const AValue: Double; const ALabel:string;
AColor: TColor): Longint;
```

К диаграмме добавляется значение, указанное параметром AValue, параметр ALabel определяет подпись под значением, AColor определяет цвет столбца; результатом функции является номер значения в массиве значений диаграмм.

```
function AddXY(const AXValue, AYValue: Double; const
ALabel: string; AColor: TColor): Longint;
```

Заданием (AXValue, AYValue) добавляется новая точка в график функции. Параметры AXValue, AYValue определяют координаты добавляемой точки. Необязательный параметр ALabel задает текст метки, соответствующей добавляемой точке. Этот текст может выводиться около соответствующих делений координатной оси или как текст значения Y. Параметр AColor тот же, что и в методе Add.

Метод Clear очищает серию от записанных ранее данных.

```
Procedure Delete (Index: LongInt);
```

Обеспечивается удаление из диаграммы точки с индексом Index.

Обращение к методам имеет вид Chart1.Series[i].Метод (...). Здесь i - номер графика, с которым осуществляется действие.

Рассмотрим использование компонента TChart на примерах.

Пример 7.1. С помощью компонента класса TChart построить график функции f(x) = sin(x) на интервале [-3;3].

Для решения задачи разместим в Форме компонент Chart1, кнопки График, Очистить и Закрыть (рис.7.3). Дважды щелкнем по компоненту Chart1. Появится окно, аналогичное, изображенному на рис.7.1.

В этом проекте мы будем рисовать один график, поэтому на вкладке Chart\Series щелкнем один раз по кнопке Add для добавления одной серии в компонент Chart1. В появившемся окне (см. рис.7.4) выключим флажок 3D и выберем тип добавляемого графика (тип серии данных) Line.

Перейдем с вкладки Series на вкладку Titles и введем название графика (рис.7.5).

Теперь установим некоторые свойства компонента в Форме. Кнопку Очистить и компонент Chart1 сделаем невидимыми, для этого их свойство Visible установим в False.

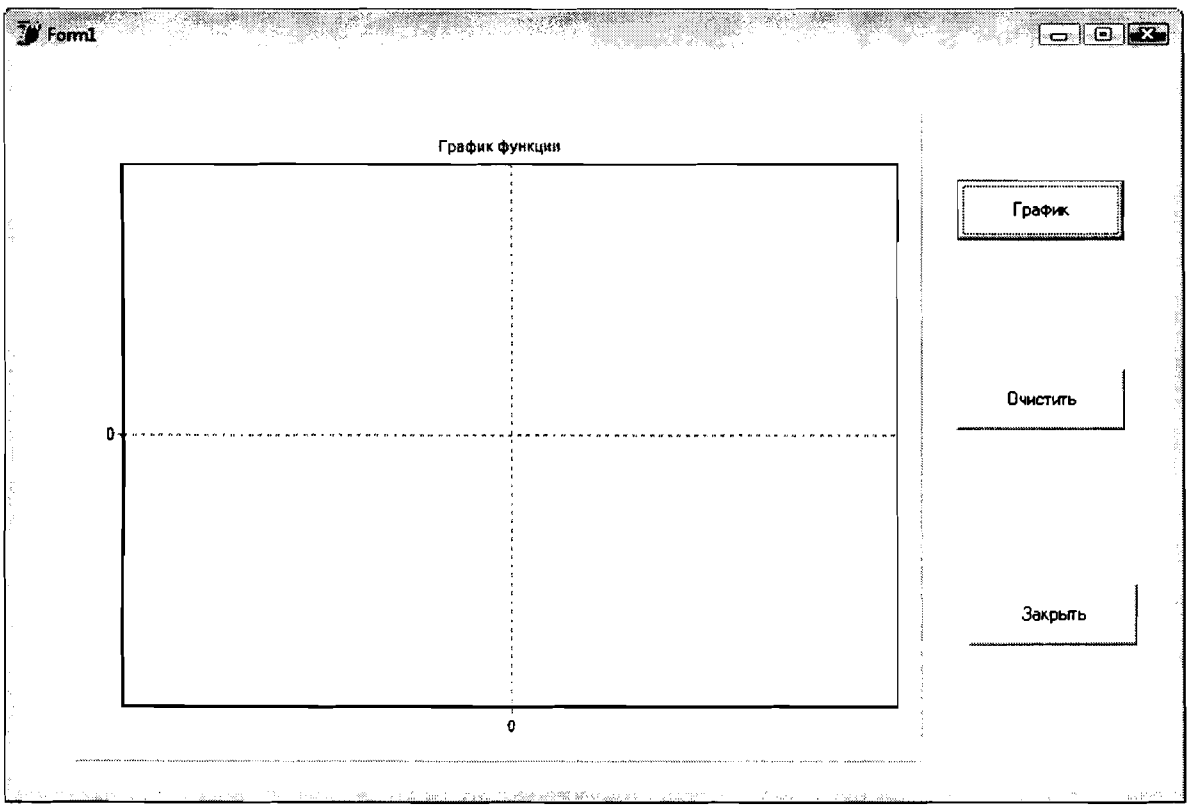

Рис.7.3. Форма для решения примера 7.1

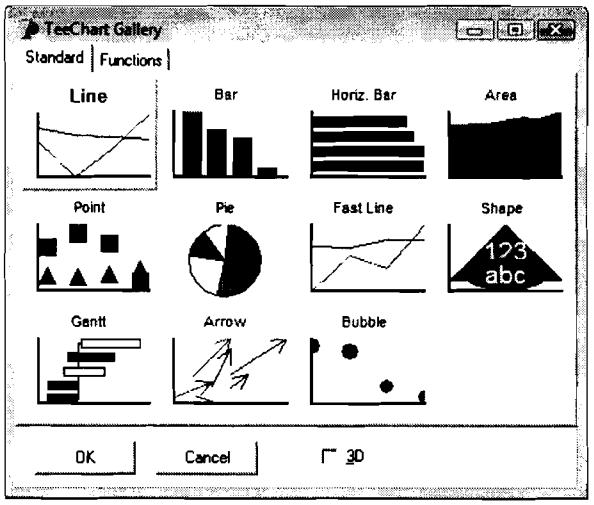

Рис.7.4. Выбор типа добавляемого графика

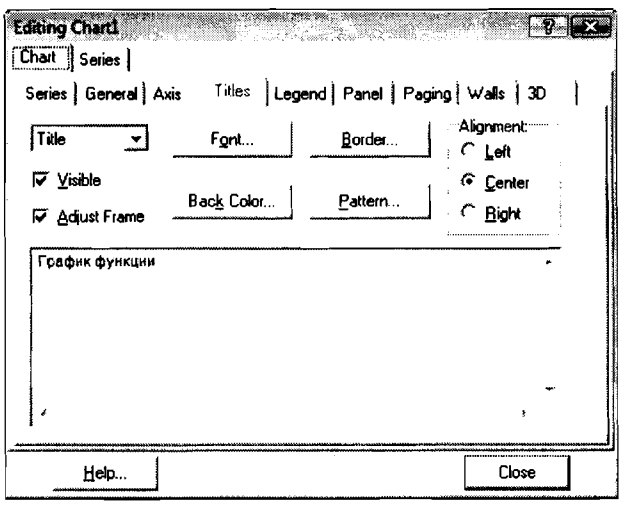

Рис.7.5. Ввод названия графика

В свойствах `Chart1` перейдем на вкладку Chart\Legend и выключим свойство `Visible`. Затем осталось написать тексты обработчиков событий трех кнопок. Обработчик кнопки **Закрыть** будет, как всегда, состоять из одного оператора:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
Close;
end;
```

Текст обработчика кнопки **График** может быть следующим:

```
procedure TForm1.Button1Click(Sender: TObject);
var x,xkon:real;
    s:string;
begin
//Кнопка График становится невидимой, кнопка
//Очистить видимой.
Button2.Visible:=True;
Button1.Visible:=False;
Chart1.Visible:=True;
//Определяем левую границу графика.
s:=InputBox('Ввод левой границы графика',
'Введите число.','0,00');
x:=StrToInt(s);
//Определяем правую границу графика.
s:=InputBox('Ввод правой границы графика',
'Введите число.','0,00');
xkon:=StrToInt(s);
repeat
//Изображаем очередную точку.
Chart1.Series[0].AddXY(x,sin(x));
x:=x+0.1;
until x>xkon;
end;
```

Текст обработчика кнопки **Очистить** может быть следующим:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
//Кнопка График становится видимой,
//кнопка Очистить невидимой.
Button2.Visible:=False;
Button1.Visible:=True;
//Очистка серии данных.
Chart1.Series[0].Clear;
end;
```

При запуске программы Форма имеет вид, представленный на рис. 7.6. При щелчке по кнопке **График** на экране появится график функции (рис.7.7), при щелчке по кнопке **Очистить** Форма примет вид, показанный на рис. 7.8. Как видно из примера, можно изменить любой параметр компонента `Chart1` из текста программы.

Пример 7.2. Построить графики переходной характеристики, описанной зависимостью вида:

$$h(t) = k\left(1 - e^{-t/T}\right).$$

Время протекания процесса задается от 0 с шагом равным 1. Коэффициент T принимается равным 2,2. Расчет и построение графика выполнить относительно значений коэффициента k, равных 1,1 и 2,4.

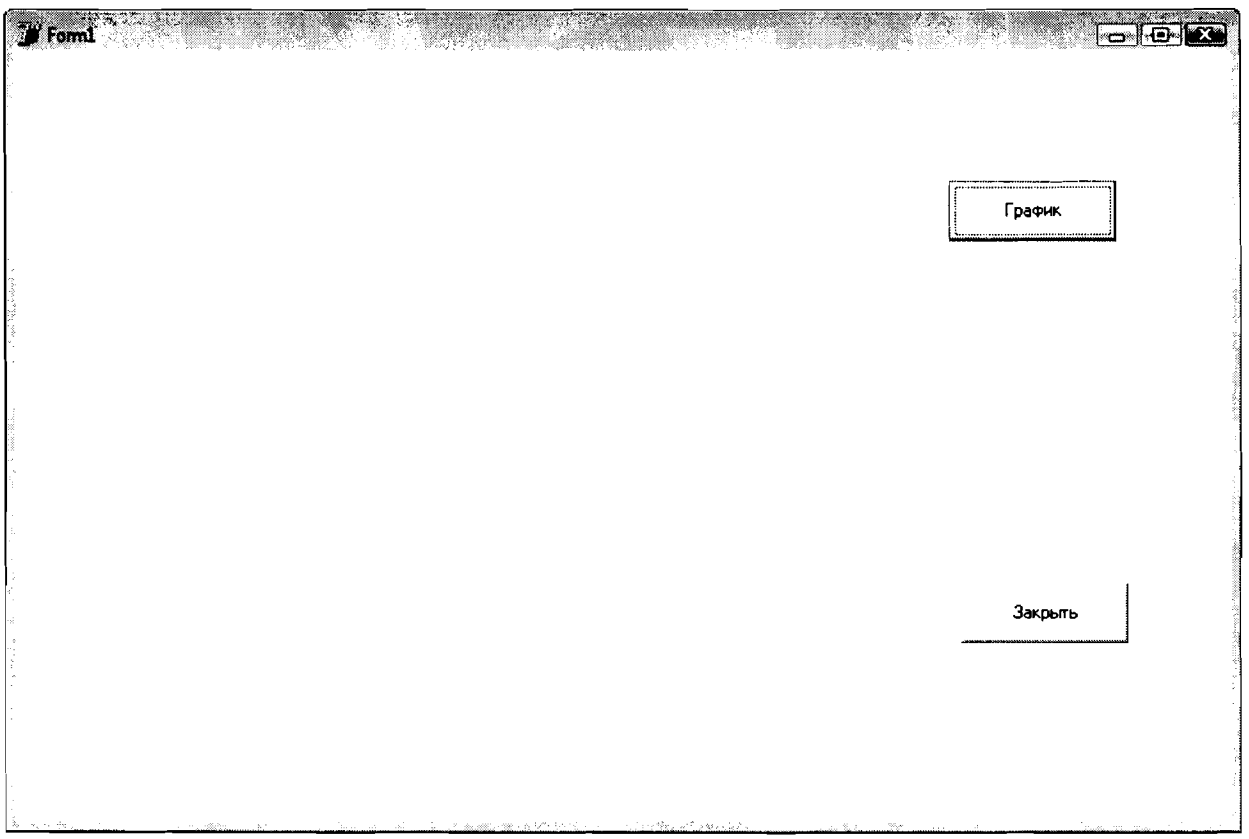

Рис.7.6. Стартовое окно программы из примера 7.1

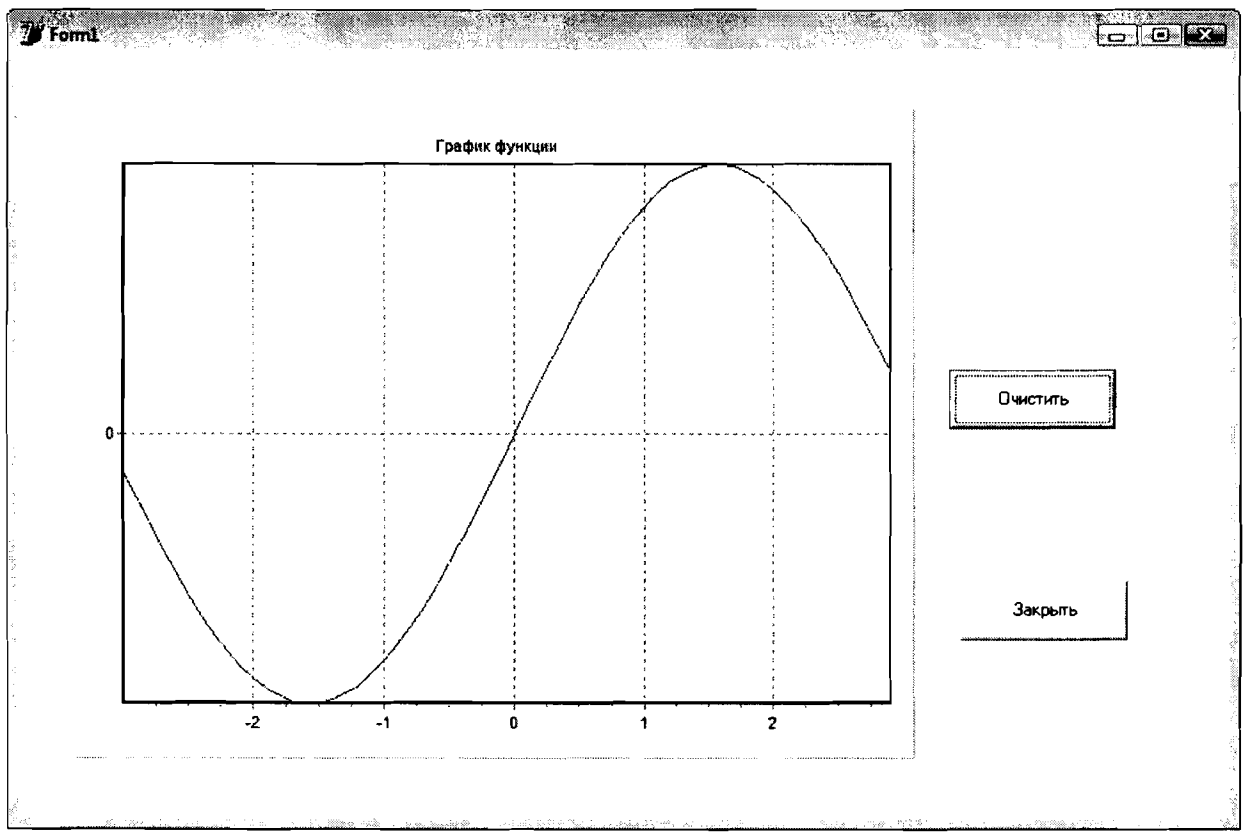

Рис.7.7. Окно программы с графиком функций y=sin(x)

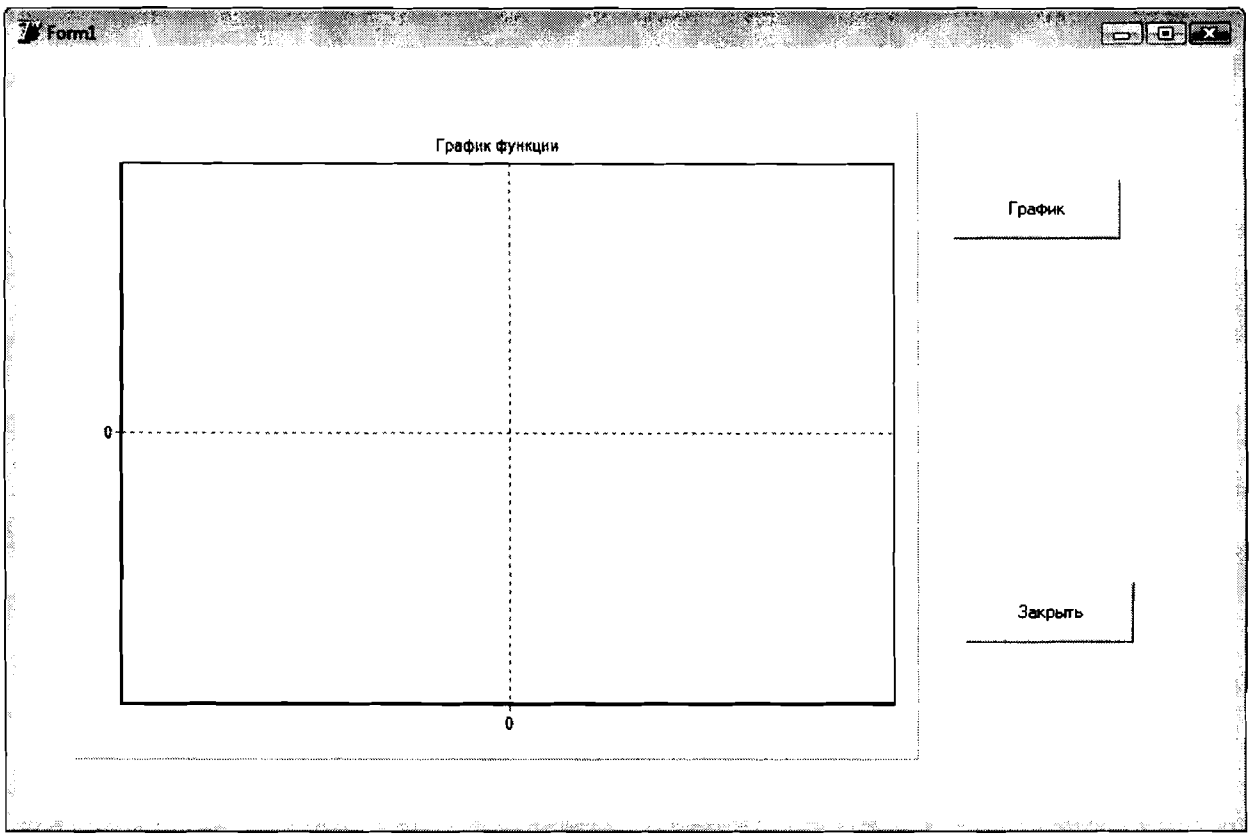

Рис.7.8. Окно программы после щелчка по кнопке Очистить

Отличие этого примера от предыдущего состоит в том, что на одном компоненте TChart будет размещено два различных графика. Осуществляется эта операция с помощью ввода двух серий значений Series1 и Series2, каждая из которых будет отвечать за один из графиков функций. Все остальные операции производятся аналогично примеру 7.1. В результате выполнения программы появиться окно, представленное на рис.7.9.

Программный код, обеспечивающий получение решения, будет иметь вид:

```
procedure TForm1.Button1Click(Sender: TObject);
const k1=1.1;k2=2.4; Tc=2.2;
var x,h1,h2,t,tkon:real;
    s:string;
begin
Button2.Visible:=True;
Button1.Visible:=False;
Chart1.Visible:=True;
t:=0;
s:=InputBox('Введите время протекания процесса',
'Введите число.','0,00');
tkon:=StrToInt(s);
repeat
x:=-t/Tc;
h1:=k1*(1-exp(x));
Chart1.Series[0].AddXY(t,h1);
```

```
h2:=k2*(1-exp(x));
Chart1.Series[1].AddXY(t,h2);
t:=t+1;
until t>tkon;
end;
```

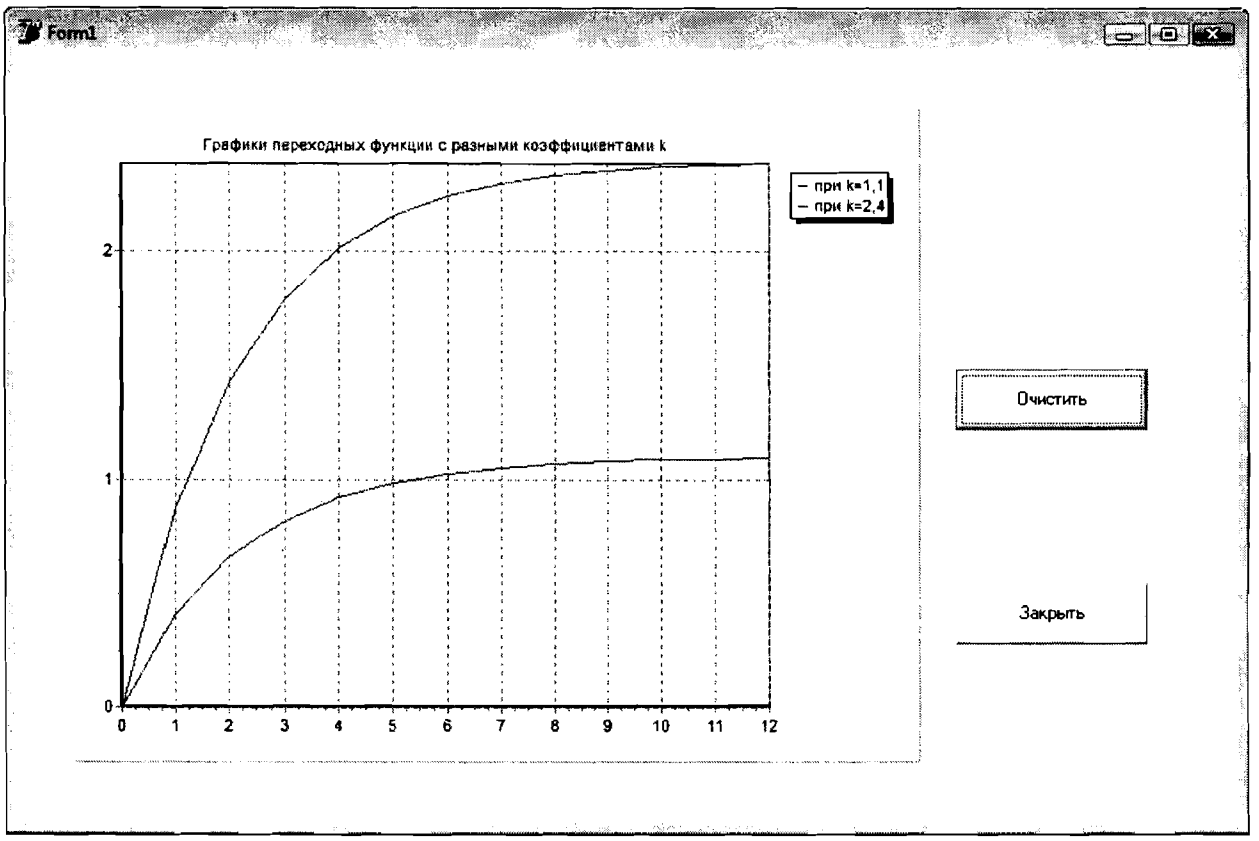

Рис.7.9. Окно программы в результате выполнения примера 7.2

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

Суриков В.Н., Кудрявцев А.С., Петров Г.А., Хардиков Е.В. Основы алгоритмизации инженерных задач: учебное пособие/ СПБГТУРП. – СПб., 2008.

Кудрявцев А.С. Программирование и основы алгоритмизации: учебно-методическое пособие/ СПБГТУРП. – СПб., 2007.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

Архангельский А.Я. Программирование в Delphi. Учебник по классическим версиям Delphi. – М.: Бином, 2008.

Галисеев Г.В. Программирование в среде Delphi 7. – М.: Издательский дом «Вильямс», 2003.

Пестриков В.М., Маслобоев А.И., Федоров О.К. Основы программирования в системе Borland Delphi: учебное пособие/ СПБГТУРП. – СПб., 2004.

Чеснокова О.В. Учимся программировать на Delphi 2007. – М.: NT Press, 2008.

Фаронов В.В. Программирование на языке высокого уровня: учебник для вузов. – СПб.: Питер, 2005.

ОГЛАВЛЕНИЕ