МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНЕГО СПЕЦИАЛЬНОГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ УЗБЕКИСТАН

Н.А.Каримова

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ТЕХНИЧЕСКИХ СИСТЕМАХ.ч.2

учебник

Рекомендуется Министерством высшего среднего специального образования Республики Узбекистан в качестве учебника

ТАШКЕНТ- 2020

УДК: 004,43 (075)

КВС 32.973.26–018.1

К 15

К 15 Кармова Н.А. Информационные технологии в технических системах. (Учебник). Часть 2–Т.

ISBN 978–9943–11–840–13

В этом учебнике описываются методы программирования, которые являются одним из основных разделов «Информационные технологии в технических системах». В учебнике представлен обзор современных языков и технологий программирования, интегрированного поля Borland C ++ Builder 6, его возможностей и использования его основных структур. Процессы программирования в технических системах иллюстрируются на примерах. Учебник предназначен для студентов бакалавриата по всем дисциплинам, изучающим информационные технологии в технических системах. Учебник также рекомендуется для студентов, преподавателей и независимых студентов, изучающих естественные науки.

Mazkur darslikda “Texnik tizimlarda axborot texnologiya-lari” fanining asosiy bo‘limlaridan bo‘lgan dasturlash texno-logiyalari haqida bayon etilgan. Darslikda zamonaviy dasturlash tillari va texnologiyalari, Borland C++ Builder 6 integrallashgan sohasi, uning tashkil etuvchilari va asosiy konstruktsiyalaridan foydalanish hususiyatlari haqida to‘liq ma’lumotlar berilgan. Texnik tizimlardagi jarayonlarni dasturlash misollar yordamida ko‘rsatilgan.

Darslik “Texnik tizimlarda axborot texnologiyalari” fanini o‘rganayotgan barcha ta’lim yo‘nalishidagi bakalavr talabalariga mo‘ljalangan. Shuningdek,

darslik magistrantlar, professor-o‘qituvchilar, hamda fanni mustaqil o‘rganuvchilariga tavsiya etiladi.

This tutorial describes programming methods, which are one of the main sections of “Information Technology in Technical Systems”. The textbook provides an overview of modern programming languages and technologies, the integrated field of Borland C ++ Builder 6, its capabilities and the use of its basic structures. The programming processes in technical systems are illustrated by examples. The textbook is intended for undergraduate students in all disciplines studying information technology in technical systems. The textbook is also recommended for students, teachers and independent students studying natural sciences.

Рецензенты:

Ш.М.Гулямов - Ташкентский государственный технический университет, профессор кафедры "АПП", д.т.н., профессор;

Х.Н.Зайниддинов - ТУИТ, профессор кафедры «АТ», т.ф.н., профессор.

Введение

Использование информационных систем в экономике, управлении, связи, научных исследованиях, образовании, сфере услуг, торговле, финансах и других областях человеческой деятельности является важной областью информационного и социального развития. Преимущества, которые могут быть достигнуты за счет использования компьютерных технологий, будут увеличиваться с увеличением объема обработки информации. Масштабные реформы, проводимые сегодня в нашей стране, требуют формирования непрерывной системы образования. Подготовка недавно продуманных, хорошо обученных специалистов, способных успешно управлять рыночной экономикой, является требованием времени, особенно тех, кто имеет широкий доступ к информационным технологиям. Бурное развитие общества тесно связано с внедрением информационных технологий во всех областях. Это требует от молодых специалистов знания современных технологий программирования. Национальная учебная программа реализуется в соответствии с положениями Закона Республики Узбекистан «Об образовании» и основана на анализе национального опыта и последних достижений современного программирования.

Первая глава учебника содержит всесторонний обзор современных языков программирования и технологий, их использования и классификации.

Вторая глава знакомит с интегрированным доменом Borland C ++ Builder 6, его составляющими, процессом разработки приложений и палитрой компонентов.

Третья глава посвящена базовому дизайну и возможностям языка программирования C ++, категориям данных, стандартным функциям и разработке программ в консольных и визуальных средах.

В четвертой главе подробно описывается программирование линейных процессов и классификация операторов в системе программирования Borland C ++ Builder 6.

Пятая глава учебника подробно описывает примеры процессов программирования на C ++ Builder 6, оператор безусловного перехода, оператор условного перехода и операторы выбора.

Шестая глава фокусируется на C ++ Builder 6 Программирование Повторяющееся программирование, Повторение предварительных условий, Последующие повторяющиеся процессы и Репликация параметров.

Седьмая глава посвящена работе с массивами в системе программирования C ++ Builder 6. Понятие массива и его типов, описание массивов в C ++ и память элементов массива описаны подробно.

Восьмая глава описывает структуру категории C ++ и ее описание, структурный элемент и его действия, а также использование смешанной категории C ++ в языке программирования.

Девятая глава посвящена функциям и процедурам в C ++, реализации структурного программирования и реализации инженерных проблем в объектно-ориентированном программировании.

В десятой главе учебника подробно описывается, как система программирования C ++ обрабатывает файловые и справочные категории, справочную категорию и ее роль в программе.

Одиннадцатая глава учебника описывает классы и объекты на алгоритмическом языке C ++. Определения абстракции, преемственности и полиморфизма.

В двенадцатой главе учебника подробно описываются графические возможности Borland C ++ Builder 6, методы визуализации технических проблем, возможности графических модулей и способы их использования.

В тринадцатой главе учебника подробно описаны интегрированная база данных Borland C ++ Builder 6, база данных импорта и экспорта с программным обеспечением и методы обработки базы данных с использованием компонентов.

Глава 1. СОВРЕМЕННЫЕ ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ В ТЕХНИЧЕСКИХ СИСТЕМАХ

Ключевые слова: программирование, языки программирования, объектно-ориентированные технологии программирования, классификация языков программирования, языки программирования с открытым исходным кодом.

1.1. Языки программирования и их возникновение

С момента создания первых программируемых вычислительных машин человечество разработало более восьми тысяч языков программирования. Их количество растет с каждым годом. Некоторые языки могут использоваться небольшим числом пользователей их разработчиками, в то время как другие языки программирования используются миллионами пользователей для создания приложений.

Языки программирования предназначены для написания компьютерных программ, которые включают в себя набор правил, которые позволяют компьютеру выполнять определенные вычислительные процессы и управлять различными объектами. Большинство языков программирования используют специальные структуры для управления процессом идентификации, управления и вычисления структур данных.

В зависимости от уровня развития языки программирования можно разделить на несколько периодов. Языки первого периода восходили к 50-м годам, когда появились первые компьютеры. Примеры включают язык ассемблера.

Языки второго периода варьируются от 50 до 60 лет. За это время был создан новый облик Ассамблер с меняющейся концепцией. Использование этого языка повысило скорость и надежность программ.

70-е годы считаются четвертым поколением языков. Эти языки предназначены для крупномасштабных проектов, которые часто используются для решения конкретных задач. Программы выполняют задачи, используя проблемно-ориентированные языковые инструменты, то есть действия, которые выражают единственную функцию.

В 1949 году Джон Мучли разработал систему Short Code, которая была первым языком языков программирования высокого уровня. В 1951 году Грейс Хоппер была первой, кто создал компилятор A-O.

В 1954 году был создан ФОРТРАН (FORmula TRANslation). Эта программа была создана Джоном Бекусой в IBM. Фортран широко используется для научных и инженерных расчетов.

В 1958 году LISP (язык обработки LISt) перечисляет программы с линейными списками. Это язык программирования высокого уровня, созданный Джоном Маккарти.

В 1959 году был разработан COBOL (Общий бизнес-ориентированный язык). Это бизнес-ориентированный язык программирования высокого уровня. Этот язык программирования был создан Грейс Хоппер.

В третьем периоде - в 1960-х годах - стали появляться универсальные языки высокого уровня, которые позволили бы им решать проблемы в самых разных областях. Отличный дизайн, простота и автоматизированное проектирование этих языков помогли программистам повысить их производительность. Многие из этих языков все еще используются сегодня. ALGOL был создан в 1960 году, а ALGOritmic Langauge - это алгоритмический язык, используемый при расчете научных и технических вопросов.

В 1964 году Дартмутский университет под руководством Джона Кемени и Томаса Курца разработал язык программирования BASIC. Этот язык предназначен для выполнения простых, понятных и сложных вычислений.

PASCAL был опубликован в 1971 году и назван в честь французского ученого Блеза Паскаля. Он используется для создания индивидуальных программ для решения различных типов задач. Большинство языков программирования являются основой. Этот язык программирования широко используется в средних школах и колледжах.

В 1960-х и 1970-х годах были разработаны основные идеи и концепции языков программирования. На этой основе создаются и улучшаются современные языки программирования.

1.2. Современные языки программирования

В настоящее время существует множество современных языков программирования, предназначенных для решения многих проблем народного хозяйства с использованием компьютеров. Вот список самых популярных языков программирования: Опрос GitHub среди ведущих мировых ИТ-разработчиков. Давайте рассмотрим некоторые из самых современных языков программирования:

1. JavaScript - это многопартийный язык программирования, который поддерживает объектно-ориентированные, функциональные стили. JavaScript - это язык программирования высокого уровня, который обрабатывает команды в браузере клиента, например, на компьютере конечного пользователя, что снижает нагрузку на сервер и увеличивает скорость программы. JavaScript был разработан Netscape. Этот язык программирования широко используется в интернет-технологиях.

2. Java - это объектно-ориентированный язык программирования, основанный на классах, который легко переносить и запускать на как можно большем количестве платформ. Язык программирования Java был разработан Sun Microsystems. Позднее компания была приобретена Oracle. В 1995 году состоялось первое официальное появление. Поскольку приложения Java обычно реализуются в пользовательских байтовых кодах, они могут работать

на любой компьютерной архитектуре, использующей виртуальную машину Java. Язык программирования Java был создан Джеймсом Гослингом и изначально предназначался для программирования электронных устройств. Позже он использовался для написания клиентского и серверного программного обеспечения. Язык программирования Java является одним из самых продолжительных, стабильных и мощных языков в сети. Javascript широко используется на всех платформах, операционных системах и устройствах. В настоящее время он широко используется в большинстве языков программирования.

3. Python - это язык программирования высокого уровня, который легче всего читать благодаря своей простоте, удобочитаемости и синтаксису. Разработано Гвидо ван Россумом в 80-х годах. Python обычно используется как функция языка сценариев, которая позволяет разработчикам писать большое количество легко читаемых и функциональных кодов за короткие промежутки времени, но также обеспечивает динамическое, объектно-ориентированное, процедурное и функциональное программирование. Благодаря своей гибкости Python является одним из наиболее широко используемых языков программирования сегодня. Широко используется в веб-приложениях.

4. Ruby - это динамический объектно-ориентированный язык программирования с открытым исходным кодом, который является одним из самых молодых языков, разработанных в 1990-х годах ученым-программистом Юкихиро Мацумото. Не нужно изучать много команд, которые используют простой синтаксис для чтения и записи на этом языке. Поэтому изучать этот язык относительно легко. Хотя сам язык является объектно-ориентированным, обеспечение процедурного, функционального и императивного программирования делает этот язык гораздо более гибким. Широко используется в веб-приложениях.

5. PHP (Hypertext Preprocessor - «PHP: Hypertext Processor») является одним из наиболее распространенных языков для разработки и разработки динамических веб-сайтов. PHP был разработан в 1995 году. PHP обрабатывает данные на сервере, и пользователь получает необходимую информацию в HTML. PHP - это язык программирования с открытым исходным кодом, поэтому тысячи модулей были написаны и готовы к настройке. Он обеспечивает взаимодействие со многими существующими системами управления базами данных (MySQL, MySQLi, SQLite, PostgreSQL, Oracle (OCI8), Oracle, Microsoft SQL Server, Sybase, ODBC, mSQL, IBM DB2, Cloudscape и Apache Derby, Informix, Ovrimos SQL, Lotus Notes, DB ++, DBM, dBase, DBX, FrontBase, FilePro, Ingres II, SESAM, Firebird / InterBase, доступ к файлам Paradox, MaxDB).

6. C ++ - это язык программирования общего назначения в статическом стиле, основанный на компиляции. Он был разработан Биярном Страуструпом в Bell Labs в 1979 году, чтобы расширить возможности языка программирования C и представить ООП (объектно-ориентированное программирование). Первоначально называвшийся «C with Classes», он был переименован в C ++ в 1983 году с его нынешним названием. Язык C ++ используется для производства компонентов, связанных с операционными системами, клиентскими программами, компьютерными играми, приложениями для повседневного использования и многими другими приложениями. C ++ может быть отличным инструментом для многих бесплатных и коммерческих платформ. Например, на платформе x86 есть GCC, Visual C ++, компилятор Intel C ++, Embarcadero (Borland) C ++ Builder и многие другие.

7. CSS (каскадные таблицы стилей - каскадные таблицы стилей) - это язык программирования, используемый для описания макетов документов с использованием языка разметки. Предназначен для разделения веб-страниц на разделы и дизайна веб-сайтов. Разделение расширяет возможности

использования веб-сайтов и расширяет их использование современных технологий. Это простое в использовании веб-приложение, предназначенное для чтения, отображения определенного формата для печати и последовательного отображения мобильного контента в различных вариантах.

8. C # (th sharp) - объектно-ориентированный язык программирования. Разработано Microsoft в 1998-2001 годах командой инженеров во главе с Андерсом Хайлсбергом. Он используется в качестве языка разработки приложений в Microsoft .NET Framework. Включено в состав компилятора csc.exe .NET Framework для реализации на C #. C # - это язык программирования с открытым исходным кодом.

9. C - это язык программирования общего назначения, скомпилированный в статических методах. Язык C был разработан Деннисом Ритчи в лаборатории Bell Labs в 1972 году. Первоначально основанный на платформе операционной системы UNIX. Более поздние версии были разработаны для работы на многих платформах. Это основа для разработки языков программирования, таких как C ++, Java, C #, JavaScript и Perl. Поэтому изучение этого языка может привести к пониманию других языков. C используется для разработки приложений низкого уровня, потому что это удобно для разработки программных приложений.

10. Go - это всеобъемлющий язык программирования Google. Первая разработка языка программирования Go началась в сентябре 2007 года, и его прямым проектом занимались Роберт Грисмер, Роб Пайк и Кен Томпсон. Официально язык программирования Go был выпущен для публики в ноябре 2009 года.

Go разработан как язык системного программирования для создания высокопроизводительных приложений для современных распределенных систем и многоядерных процессоров. Это можно рассматривать как попытку заменить язык Си.

При разработке особое внимание уделялось обеспечению высокого уровня компиляции. Программы, написанные на Go, написаны для объектного кода и не требуют для запуска виртуальной машины.

11. Objective-C - это объектно-ориентированный язык программирования компиляции. Этот язык программирования основан на парадигмах языка программирования C и Smalltalk. Используется Apple. В частности, объектная модель построена на языке программирования Smalltalk, то есть сообщения отправляются объектам. Компилятор Objective-C встроен в систему GCC и доступен на большинстве основных платформ.

Язык в основном используется для Mac OS X (какао) и систем GNUstep. Интерфейс OpenStep используется для создания объектно-ориентированных приложений.

12. Scala - это всеобъемлющий язык программирования, который включает в себя как функциональное, так и объектно-ориентированное программирование. Простой и компактный язык программирования для приложений. Разработано в 2003 году Мартином Одерским на основе платформ Java и .Net. Язык программирования Scala используется многими компаниями-разработчиками программного обеспечения (Coursera, LinkedIn, Twitter, UBS).

Развитие информационных технологий требует широкого использования современных языков программирования. На основе опросов, проведенных за рубежом, мы можем посмотреть на следующую статистику, основанную на степени, в которой современные языки программирования могут использоваться для создания программных приложений (рис. 1.1).

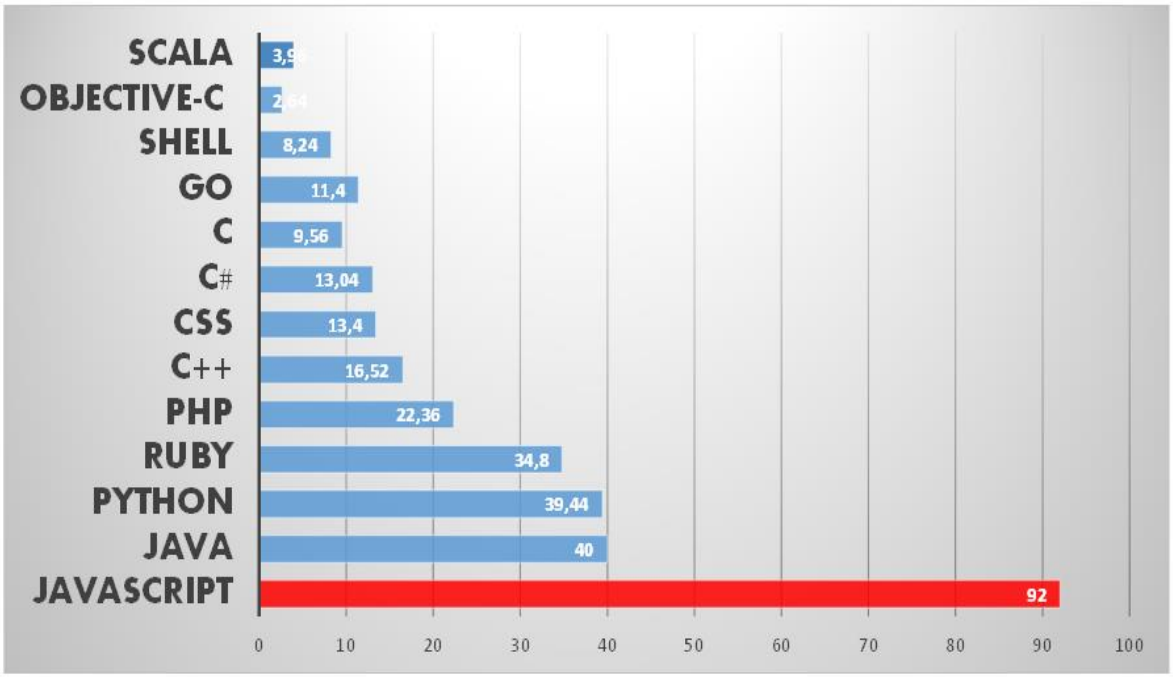


Рис. 1.1. Современные языки.

В настоящее время языки программирования находятся на подъеме, и языки программирования широко используются для решения проблем в определенных областях.

1.3. Классификация языков программирования

Исходя из языков программирования, рассмотренных выше, не все языки программирования являются однородными. Хотя некоторые языки программирования являются объектно-ориентированными, другие основаны на устройствах. Имея это в виду, мы можем классифицировать языки программирования следующим образом:

1) низкие и продвинутые языки программирования. Низкоуровневые языки программирования являются машинными языками. Машинно-зависимые языки, в свою очередь, подразделяются на машинные языки и машинные языки. Они являются частью первого поколения языка программирования. Второе поколение языков программирования низкого уровня является примером языков ассемблера. Язык программирования

Ассемблер позволяет вводить код с помощью макрокоманд. Например, программирование на языке ассемблера выглядит следующим образом:

```
.386
.model flat, stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
.data
    msg db "Hello, world", 13, 10
    len equ $-msg
 .data?
    written dd ?
 .code
start:
    push    -11
    call    GetStdHandle
    push    0
    push    OFFSET written
    push    len
    push    OFFSET msg
    push    eax
    call    WriteFile
```

```
        push    0
        call    ExitProcess
end start
```

С помощью приведенного выше кода вы можете поместить слово Hello, world (Hello world) на экран.

Языки программирования первого и второго поколения, учитывая особенности устройства, позволяют нам повысить производительность функций, требуемых в этом процессоре.

Быстрое развитие информационных технологий к 1970-м годам потребовало от программистов перехода от программирования машинного кода к языкам программирования высокого уровня. Языки высокого программирования включают семантическую модель языка, то есть мышление человека. Это облегчает смену программ и перемещение их с одного устройства на другое. Это было третье поколение языков программирования.

Языки программирования четвертого и пятого поколений дополняются тем, что должна делать программа. На большинстве этих языков это было сделано путем перепрограммирования программного обеспечения для конкретной компьютерной архитектуры и операционной системы. Языки программирования высокого уровня обеспечивают преимущество в эффективности;

2) безопасные и небезопасные языки программирования. Язык программирования считается безопасным, когда скомпилированная программа принята компилятором и не выходит за пределы возможных действий. Это может вызвать логические ошибки в программе, даже если она принята компилятором. Хотя существуют логические ошибки, сгенерированные данные, являются безопасным языком программирования, если целостность данных не нарушена. Часто используются дополнительные

библейские тройки, которые позволяют им исправлять ошибки в приеме и эффективно использовать категории. Это обеспечит прямую ссылку на память компьютера.

Категории, включенные в небезопасные языки, будут приниматься по мере их появления. На небезопасных языках это может привести к сбою приложений и невозможности доступа к памяти;

3) компиляция и интерпретация языков программирования. Компиляция означает, что исходный код этой программы преобразуется в целевой машинный код с помощью специальной программы, называемой компьютер - атор. Результат сохраняется в виде отдельного файла и может быть запущен как дополнение к программе.

Интерпретируемые языки программирования не преобразуют исходный код в машинный код. Он реализует исходный код с помощью специальной программы-интерпретатора, которая выполняется непосредственно центральным процессором.

Некоторые языки программирования используют два разных способа выполнения программы. Языки компиляции широко используются в быстрой и мелкомасштабной разработке программного обеспечения и повышают эффективность. Интерпретативные языки программирования широко используются для создания приложений, которые позволяют работать с некоторыми аппаратными средствами операционной системы. Недостатком является то, что это требует больше ресурсов и низкой скорости реализации. В зависимости от области применения, языки компиляции и интерпретации имеют свои особенности;

4) парадигма программирования. Парадигма программирования - это набор идей и концепций, которые определяют, как писать компьютерные программы (программный подход). Основные модели программирования могут включать в себя:

- императивное программирование;

- декларативное программирование;

- системное программирование;

- функциональное программирование;

- логическое программирование;

- объектно-ориентированное программирование

Некоторые языки программирования могут сочетать в себе многие особенности рассматриваемых классификаций.

1.4. Современные технологии программирования

Изначально программирование было удивительным изобретением, которое позволяло программистам напрямую интегрировать программы непосредственно в память главного компьютера через блок переключения. Программы были написаны в машинном коде в двоичной системе подсчета. Часто ошибки были допущены при написании программ на машинном языке. Кроме того, программа в машинных кодах была чрезвычайно сложна для понимания.

Со временем компьютеры становились все более и более популярными, и появился более высокий уровень процедурных языков. Первым из них был Фортран. Однако основное влияние на развитие объектно-ориентированного подхода позже выявили такие процедурные языки, как ALGOL. Процедурные языки позволяют программисту разделить программное обеспечение для обработки информации на несколько процедур более низкого порядка. На таком низком уровне такие процедуры диктуют общую структуру программы. Последовательные ссылки на эти процедуры определяют выполнение программ, состоящих из процедур.

Эта новая парадигма программирования была гораздо более продвинутой, чем парадигма программирования на машинном языке, а процедуры являлись основным инструментом ее структурирования. Поскольку каждая процедура должна была программировать свои методы доступа к данным, изменения в представлении данных привели бы к изменениям во всех аспектах программы, в которой этот доступ был выполнен. Таким образом, даже малейшая коррекция приведет к ряду изменений во всей программе.

Были предприняты попытки устранить некоторые недостатки модульного программирования, такие как процедурное программирование на таких языках, как Modula2. Модульное программирование разбивает программу на несколько составных частей или, другими словами, модулей. Если процедурное программирование разделяет данные и процедуры, модульное программирование, напротив, объединяет их. Модуль состоит как из самих данных, так и из процедур обработки данных. Если другие части программы должны использовать модуль, они не будут затронуты интерфейсом модуля. Модули скрывают всю внутреннюю информацию в других частях программы.

Тем не менее, модульное программирование не лишено недостатков. Модули не являются расширяемыми, что означает, что модуль может быть изменен шаг за шагом без прямого доступа к коду и без прямой модификации. Кроме того, один модуль не может использоваться в другом при разработке одного модуля без передачи его функций другому, хотя модуль может указывать тип, но не один модуль.

Модульное программирование представляет собой гибридную схему для процедуры, и программа может быть разбита на несколько процедур. Однако теперь процедура не выполняется на необработанных данных, а контролирует модули.

Объектно-ориентированные технологии программирования - это технологии программирования, появившиеся в ответ на кризис программного обеспечения. Причиной этого кризиса является то, что методы структурированного программирования не смогли предоставить программное обеспечение для растущего числа проблем. В результате планы по различным проектам были нарушены, затраты превысили бюджет, функциональность программного обеспечения была скомпрометирована, а ошибки увеличились.

Одним из наиболее важных аспектов программного обеспечения является его сложность. Ни один программист не может полностью учитывать все функции системы. Вот почему большая команда программистов и других специалистов вовлечена в его разработку. Следовательно, цели, которые имеют непосредственное отношение к проблеме, включают целенаправленное управление этой командой. Традиционные языки программирования использовали принцип «разделяй и управляй» для решения таких сложностей. Таким образом, проблема делится на более мелкие проблемы, а затем для каждого выпуска разрабатывается отдельная программа. Технологии объектно-ориентированного программирования используют другой подход. Основное внимание уделяется элементам, необходимым для решения проблем, связанных с различными абстракциями проблемной области. Эти абстракции были разработаны программистами. Разработчики исследовали конкретную область и выбрали отдельные объекты. Определенные особенности, которые могут использоваться, чтобы решить проблемы для этих объектов, были идентифицированы. В зависимости от потребностей, действия для каждого свойства определены. Впоследствии для каждого реального объекта в изучаемой области был разработан программный объект.

Как вы знаете, для каждой компьютерной проблемы написание собственной программы требует написания. Расширение класса таких вопросов, безусловно, приведет к созданию новых программ. Когда «старые» языки программирования недоступны для нового программирования, или

необходимы новые языки программирования для улучшения процесса программирования.

Объектно-ориентированное программирование - это новая область программирования, в которой программная система считается набором взаимосвязанных объектов, и каждый объект относится к определенному классу, а каждый класс создает различную генеалогию. Отдельный класс рассматривается как совокупность данных и набор действий над ними. Доступ к элементам этого класса возможен только через действия, определенные в этом классе. Взаимосвязь между данными программы и действиями обеспечивает надежность программных систем над традиционными языками программирования. Основное понятие объектно-ориентированного программирования - это объект и класс.

Для объектно-ориентированного программирования требуются годы, оставленные в стороне от традиционного или традиционного программирования. В результате объектно-ориентированное программирование становится намного проще, более наглядным и становится отличным инструментом для решения многих проблем при разработке программного обеспечения.

Учитывая вышеизложенные идеи, технологии объектно-ориентированного программирования могут решить следующие проблемы:

- устраняет недостатки в традиционных языках программирования;
- решение проблем, которые не могут быть решены с помощью традиционных языков программирования или которые могут быть решены с большим трудом;
- диапазон данных, которые могут быть обработаны, и их типы намного шире, чем у традиционных языков программирования;
- создает удобный интерфейс;
- контроль различных типов ввода и вывода;

- вы легко организуете новые данные, классы, модули и управляющие данные;
- генерировать и обрабатывать различные эффекты звука и движения с помощью мультимедийных и анимационных инструментов;
- легко обрабатывать такие вопросы, как база данных и обработка данных, запросы SQL;
- работа с OLE-контейнерами в приложениях Windows;
- создает удобную систему удобного программного обеспечения;
- создание установочных дисков для переноса программного обеспечения на другие компьютеры;
- устраняет возможные ошибки в редактировании текста программы и т.д.

Ясно, что существует большой пробел, который можно решить, решая традиционные проблемы с использованием традиционных языков программирования, и он отвечает многим требованиям современного программирования.

Вопросы по главе 1

1. Языки программирования и их происхождение.

2. Как классифицируются языки программирования?

3. На чем основаны технологии объектно-ориентированного программирования?

4. Какие языки программирования включены в современные языки программирования?

5. Зачем использовать языки программирования с открытым исходным кодом?

Тестовые вопросы:

1. Выберите правильную строку для объектно-ориентированных языков программирования:

а) C ++, Java;

б) Фортран, Java;

в) Accembler, C #;

г) Кобол, HTML;

2. Какая компания является языком программирования Go?

а) Microsoft

б) Google

в) C ++

г) Delphi

3. В какой строке правильно отображается язык программирования на основе машинного кода?

а) Паскаль

б) Accembler

в) Java

г) C #

4. Выберите соответствующее поле для языка программирования с открытым исходным кодом.

а) Accembler

б) C ++

в) Java

г) Паскаль

5. Какой язык программирования используется для разработки веб-страниц?

а) C ++

б) Delphi

в) Fortran

г) PHP

Глава 2. СИСТЕМА ПРОГРАММИРОВАНИЯ BORLAND C++ BUILDER 6 И ЕЁ СОСТАВЛЯЮЩИЕ

Ключевые слова: приложение, система программирования, библиотека компонентов, приложение, палитра компонентов, интегрированная область Borland C ++ Builder 6.

2.1. Система программирования Borland C++ Builder 6 и её особенности

Компьютерное программирование быстро развивается в последние годы, и все больше и больше людей интересуются программированием. Современные инструменты программирования предназначены для обеспечения высокого класса приложений.

Создание инструмента программирования на C ++ проложило путь не только профессиональным программистам, но и обычным программистам. За очень короткое время Borland разработал ряд диалектов C ++. Новейшие платы C ++ включают в себя основные функции, такие как создание и обработка базы данных, совместное использование данных через Интернет, использование объектно-ориентированного программирования и создание новой библиотеки компонентов (VCL) для визуального программирования. ,

Давайте подробнее рассмотрим Borland C ++ Builder 6, простую в использовании платформу программирования. Borland C ++ Builder 6 - это объектно-ориентированная среда программирования для приложений Windows. Среда программирования Borland C ++ Builder 6 основана на современной технологии визуального проектирования с объектно-ориентированной идеей программирования.

Borland C ++ Builder 6 - это сочетание нескольких важных технологий:

- Компилятор скомпилирован в машинный код высокого уровня;

- объектно-ориентированные компонентные модели;

- Визуальное рисование программных приложений;

- Высокий масштабирующий инструмент для создания базы данных.

Borland C ++ Builder 6 - это простой в использовании инструмент для запуска приложений Windows, который автоматизирует компьютерное программирование, уменьшает количество ошибок и упрощает работу программиста. В Borland C ++ Builder 6 программа построена на основе современных технологий визуального проектирования с учетом теории объектно-ориентированного программирования. Как вы знаете, программирование - это сложный процесс, но система Borland C ++ Builder 6 значительно упрощает эту задачу и, в зависимости от проблемы, программист генерирует 50-80% своей работы.

Система Borland C ++ Builder 6 сокращает время, необходимое для проектирования и создания приложений, и упрощает создание приложений, работающих в среде Windows.

Borland C ++ Builder 6 использует ряд современных систем управления базами данных и технологий программирования.

Объектно-ориентированная модель компонентов позволяет создавать новые приложения с использованием готовых объектов, а также создавать пользовательские объекты пользователя.

Объекты Borland C ++ Builder 6 по умолчанию объединяют более 270 базовых классов. Классы Borland C ++ Builder 6 образуют библиотеку визуальных компонентов - VCL со сложной иерархической структурой. Существуют сотни классов, которые являются частью VCL.

В технологии визуального программирования объект представляет собой диалоговое окно и элементы управления (область ввода и вывода, кнопки управления, переключатели и т. Д.).

Программирование в Borland C ++ Builder 6 основано на двух взаимосвязанных процессах:

- Визуальный дизайн программы;

- Процесс ввода (записи) программных кодов.

В процессе визуального проектирования дизайн приложения формируется. Когда разработчик завершает процесс визуального проектирования, Borland C ++ Builder 6 автоматически начинает генерировать программный код. На протяжении всего проекта разработчика программа будет заполнять код приложения специальными операторами языка C ++.

2.2. Установка системы Borland C++ Builder 6

Borland C ++ Builder 6 предназначен для операционных систем Windows. Есть несколько шагов, которые вы можете предпринять, чтобы загрузить программное обеспечение на ваш компьютер.

Шаг 1 Откройте диск с помощью Borland C ++ Builder 6. Найдите файл bcb6.exe в открытом каталоге. Дважды щелкните файл bcb6.exe. Размер файла составляет 186 МБ.

Шаг 2 Архив необходим для загрузки файла bcb6.exe в память компьютера (рисунок 2.1). Для загрузки файлов во временный каталог (C: \ Temp \ BCB6) требуется 731 МБ свободного места. Если объем памяти компьютера меньше 731 МБ, программа не может быть установлена.

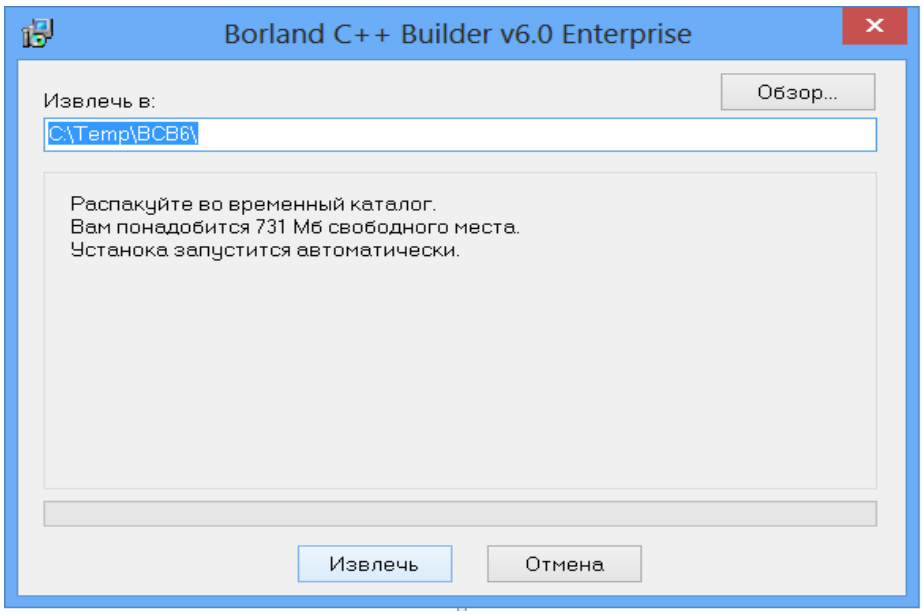


Рис. 2.1. Загрузить файлы во временный каталог.

Если в памяти компьютера есть свободное место, нажмите кнопку «Поиск», и процесс начнется. Процесс извлечения файлов выполняется следующим образом (рис. 2.2).

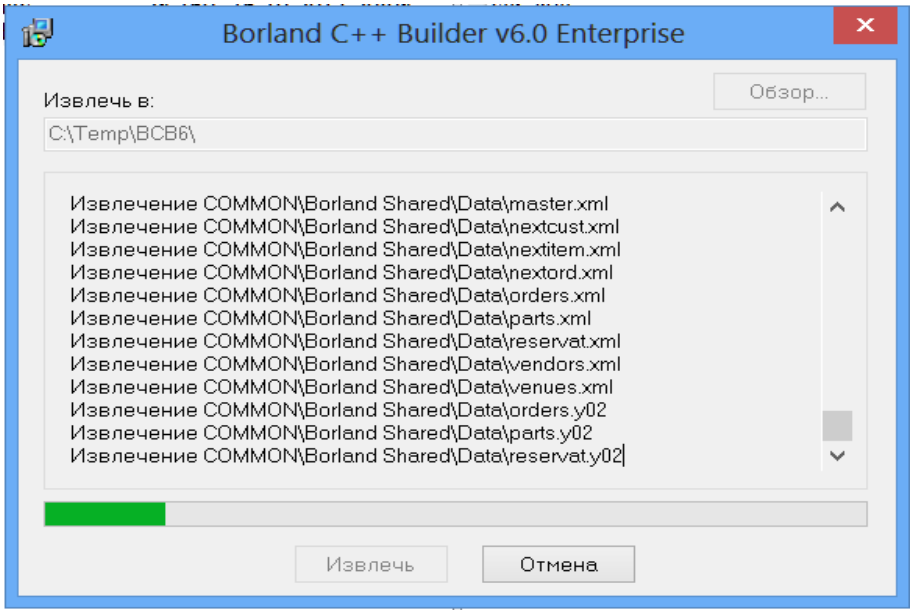


Рис. 2.2. Процесс извлечения файлов.

Шаг 3. После загрузки файлов во временный каталог на экране появятся следующие вкладки:

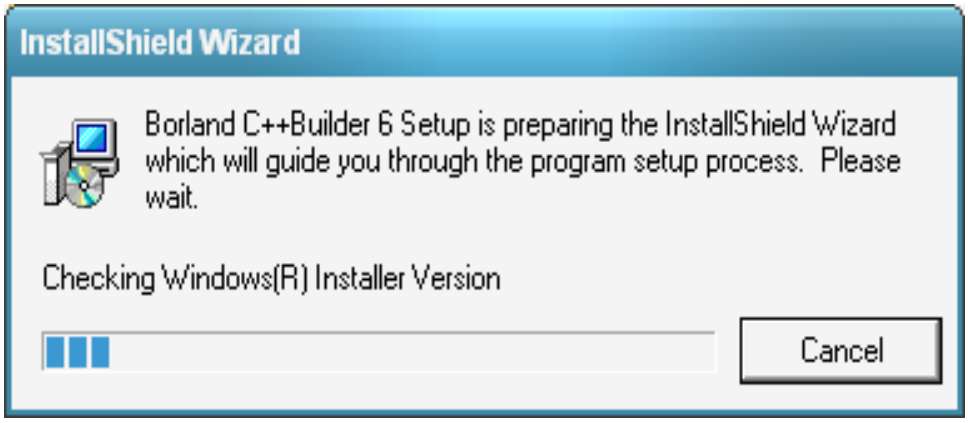


Рис. 2.3. Мастер InstallShield.

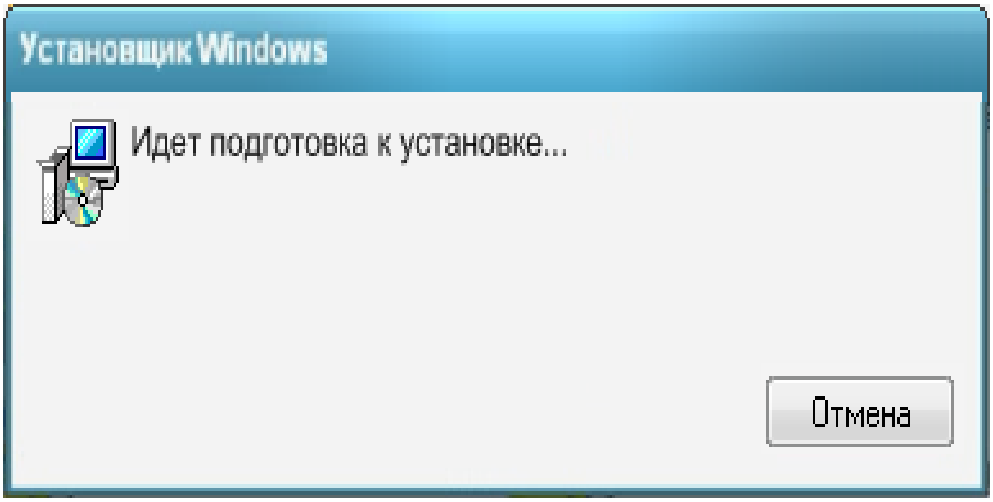


Рис. 2.4. Окно установщика Windows.

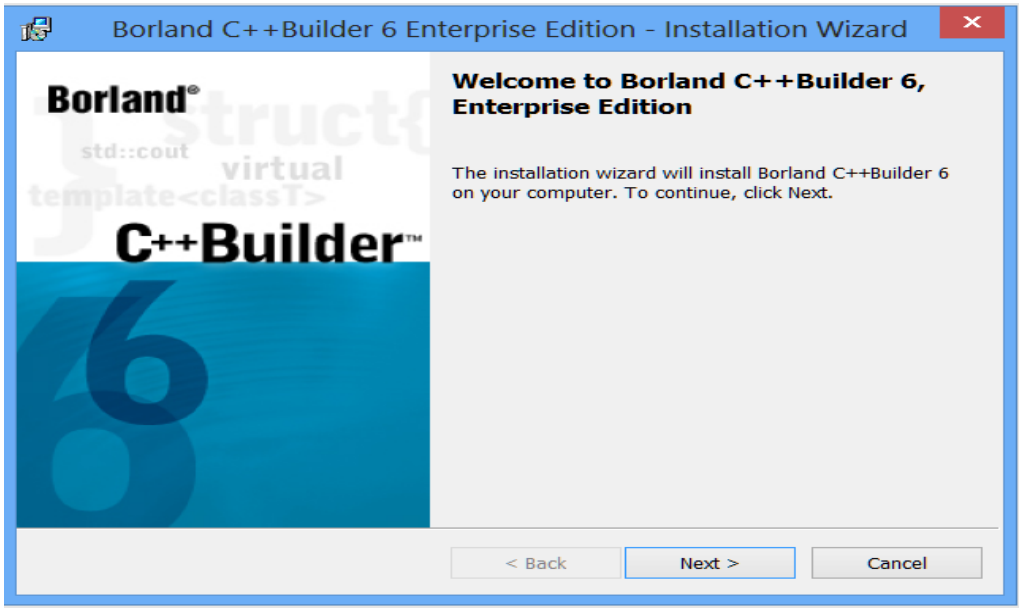


Рис. 2.5. Окно мастера установки программного обеспечения.

Нажмите кнопку «Далее» во всплывающем окне. Процесс установки начнется.

Шаг 4. После нажатия Next на экране появится окно на рисунке 2.6. Сгенерированное окно потребует от вас ввести серийный номер и код активации программы.

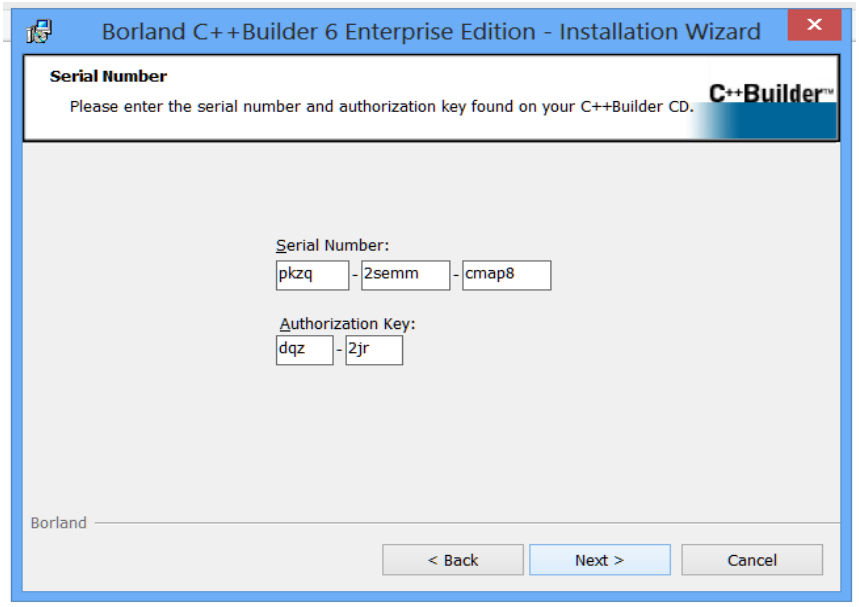


Рис. 2.6. Вкладка для серийного номера и кода активации.

После ввода серийного номера и кода активации программы нажмите кнопку «Далее».

Шаг 5. Появится окно для принятия условий лицензии Borland (рис. 2.7). В окне появляются кнопки, которые позволяют вам выбирать. То есть «Я принимаю условия лицензионного соглашения» и «Я не принимаю условия лицензионного соглашения». Если вы выберете «Я принимаю условия лицензионного соглашения», кнопка «Далее» будет активирована.

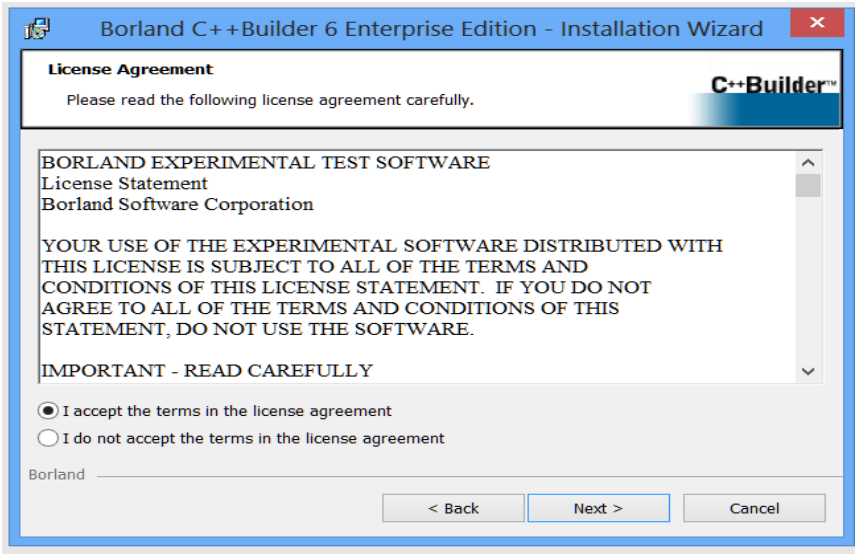


Рис. 2.7. Окно для принятия условий лицензии.

Шаг 6. После нажатия Next на экране появится окно на рисунке 2.8. Сгенерированное окно попросит вас выбрать тип операционной среды программы.

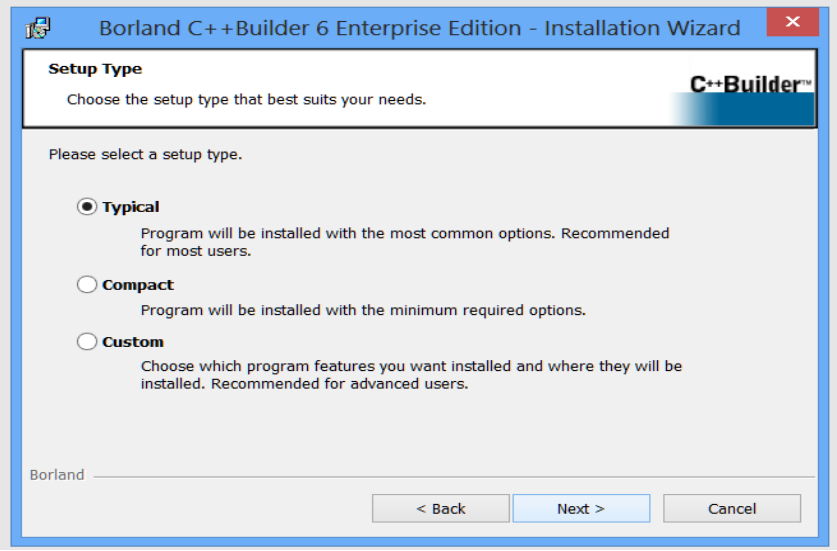


Рис. 2.8. Вкладка для выбора типа среды приложения.

Существует три типа операционной среды:

- типичный;

- компактный;

- Пользовательский.

Существуют разделы, в которых можно указать типичную типичную программу, компактный вид программы и выбор программных модулей. Соответствующий раздел будет отмечен специальным символом, после чего нажмите «Далее».

Шаг 7. Появится окно с путем к каталогу, в котором будет установлена программа (рис. 2.9). Вы должны указать отдельный путь к каталогу для каждого из четырех программных модулей в окне. Вы можете изменить расположение каталога приложения, нажав кнопку «Изменить». Если папка, в которой должна быть установлена программа, будет нажата, а затем будет выполнен следующий шаг.

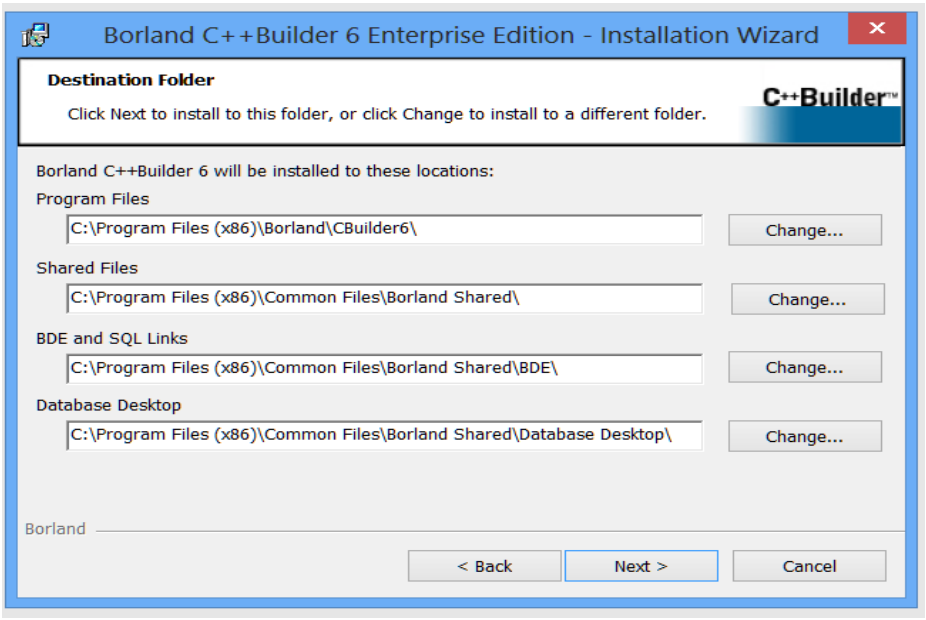


Рис. 2.9. Список каталогов, в которые будет установлена программа.

Шаг 8. Если в памяти компьютера достаточно свободного места, появится окно для успешной установки программы (рис. 2.10).

Когда вы закончите все процессы, нажмите Далее. Рис. 2.11 появляется на экране. Рекомендуется перезагрузить компьютер для полной функциональности Borland C ++ Builder 6, установленного в окне.

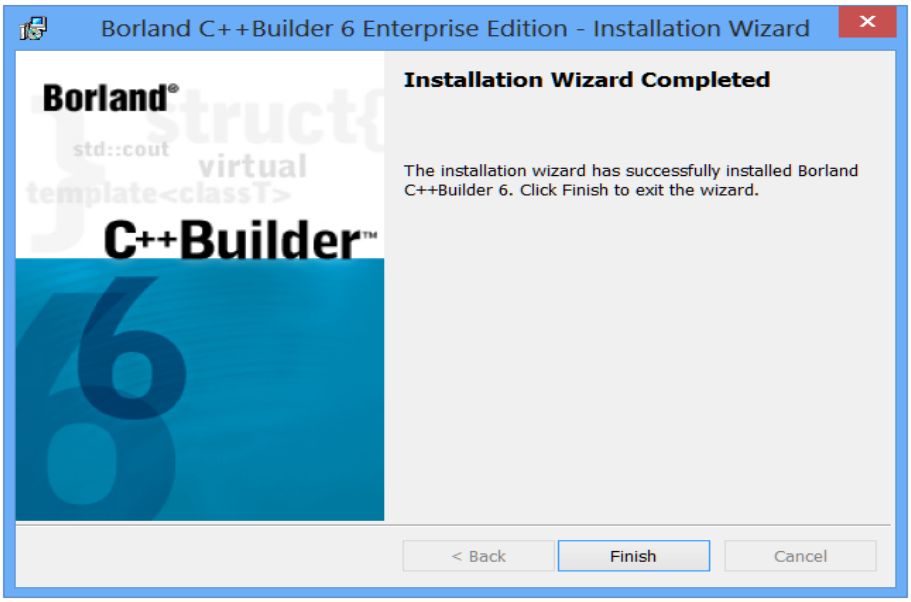


Рис. 2.10. Учебник об успешной установке программы.

Рис. 2.11. Borland C ++ Builder 6 Информация об установщике.

Нажмите «Да» во всплывающем окне. После этого процесс загрузки будет завершен.

После загрузки программы на стартовом экране откроется новое окно (рис. 2.12).

Несколько программных модулей появятся в сгенерированном программном меню. Из модулей выбран C ++ Builder 6, который предлагает возможность запуска программы.

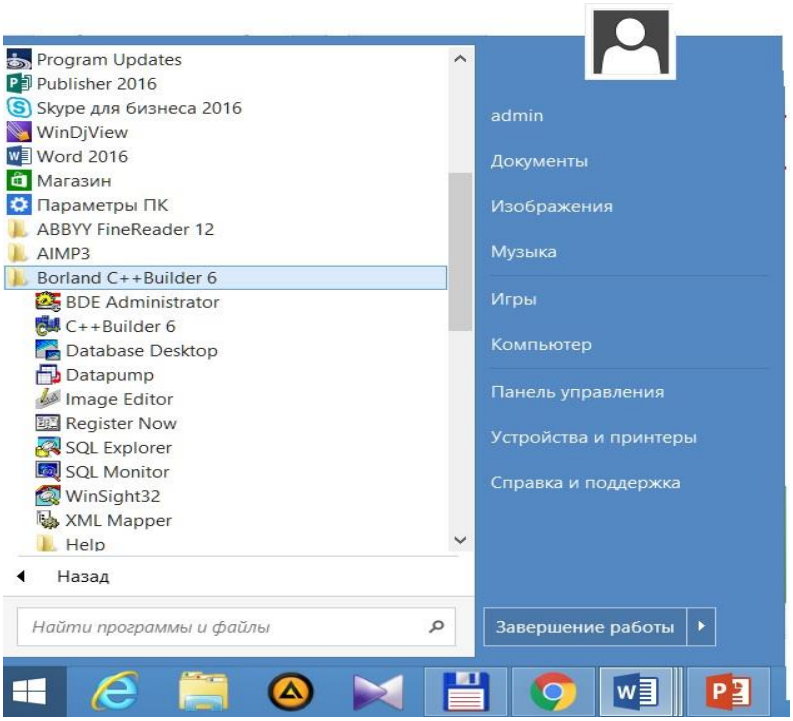


Рис. 2.12. Просмотрите программу в меню «Пуск».

2.3. Интегрированная среда Borland C++ Builder 6

Программирование в Borland C ++ Builder 6 организовано посредством двух взаимосвязанных процессов:

- Визуальный дизайн программы;

- Процесс ввода программных кодов в консольной среде.

В процессе визуального проектирования формируется дизайн приложения программы. Когда разработчик завершает процесс визуального проектирования, Borland C ++ Builder 6 автоматически начинает генерировать программный код. На протяжении всего проекта разработчика программа будет заполнять код приложения специальными операторами языка C ++.

Borland C ++ Builder 6 также работает как другие операционные системы Windows:

Автозагрузка => Все программы => Borland C ++ Builder 6 => C ++ Builder 6.

После загрузки программы на экране появится окно (рисунок 2.13). Borland C ++ Builder 6 включает в себя следующие пять основных окон:

1) окно - C ++ Builder 6 (Project1);

2) окно формы (Form1);

3) окно редактирования свойств объекта (инспектор объектов);

4) окно просмотра списка объектов (просмотр дерева объектов);

5) окно ввода кода программы (Unit.cpp).

Главное окно (Project1) находится в верхней части экрана с заголовком в первой строке, а именно с именем проекта (C ++ Builder 6 - Project1) (рис.2.14).

Второй ряд содержит основные меню по горизонтали. Главное меню позволит вам получить доступ ко всем командам и функциям, необходимым для создания программы.

В следующей строке слева находятся клавиши быстрого доступа. Объединяются последовательно в зависимости от задачи, которую они выполняют. Они позволяют быстрый доступ к часто используемым командам. Справа находится палитра визуальных компонентов VCL (Библиотека визуальных компонентов, Библиотека визуальных компонентов). Операционная система Windows включает в себя визуальные компоненты для создания приложений. Палитра визуального компонента разделена на группы из нескольких частей. С помощью этой визуальной палитры компонентов вы можете быстро и легко создавать приложения.

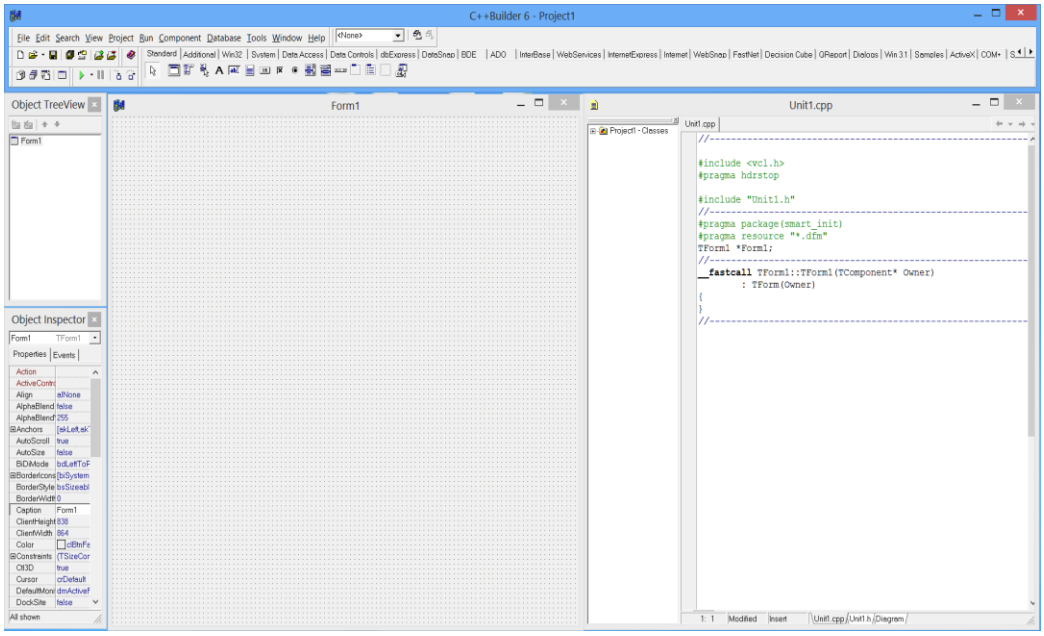


Рис. 2.13. Borland C ++ Builder 6 интегрированная область.

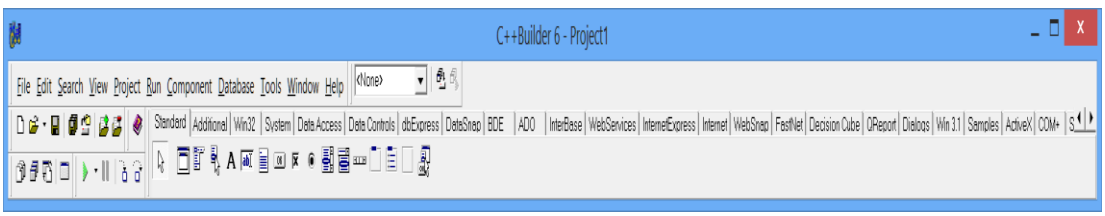


Рис. 2.14. Главное окно.

Окно формы (Form1) - это предварительный просмотр создаваемой программы (рисунок 2.15). Окно форм является основой для приложений C ++ Builder 6, где вы можете размещать компоненты в создаваемой программе. Начинается с названия программы.

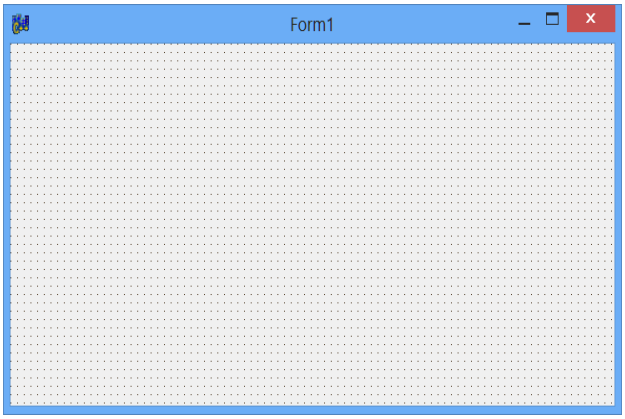
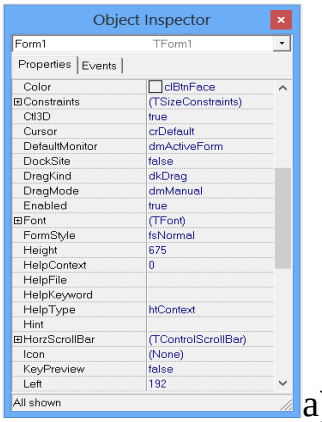


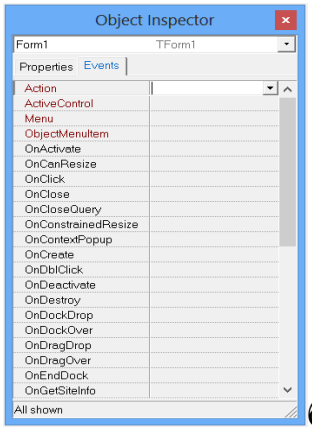
Рис. 2.15. Окно формы.

Окно «Инспектор объектов» (Object Inspector) используется для редактирования свойств и событий объекта. В объектно-ориентированном программировании программа представляет собой систему объектов, и каждый объект может иметь ряд свойств. Свойство состоит из данных и как им управлять. Свойства объекта - это свойства, данные объекту, его внешний вид, местоположение и состояние. Объект также может устанавливать различные события. Действие называется управлением, таким как щелчки мыши, перемещение курсора и т. д.

Окно «Инспектор объектов» предназначено для установки свойств и параметров событий двух страниц: свойств (свойства: рис. 2.16a) и событий (события: рис. 2.16b).



а)



б)

Рис. 2.16. Окно инспектора объектов.

а) окно свойств; б) вкладка «События».

На странице «Свойства» задаются свойства выбранного объекта или компонента. Например, вы можете установить запись объекта, используя Caption, а свойство Color устанавливает цвет объекта.

Страница «События» определяет событие, т. Е. Состояние его запуска, для выбранного объекта в приложении «Форма». Каждый случай имеет имя по умолчанию. Например, OnClick - это одно нажатие левой кнопки мыши, OnDblClick - одно нажатие левой кнопки мыши.

Окно просмотра списка объектов (Object Tree View) описывает компоненты, используемые в программе в виде дерева. Показывает расположение компонентов и их положение (рис. 2.17).

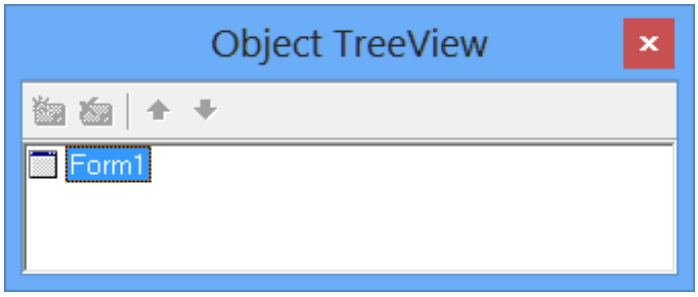


Рис. 2.17. Просмотр списка объектов

Окно ввода кода программы (Unit.cpp) используется для ввода и редактирования кода (текста) создаваемой новой программы (рис. 2.18). Программа будет включать в себя последовательность операторов, которые понадобятся для выполнения определенного процесса.

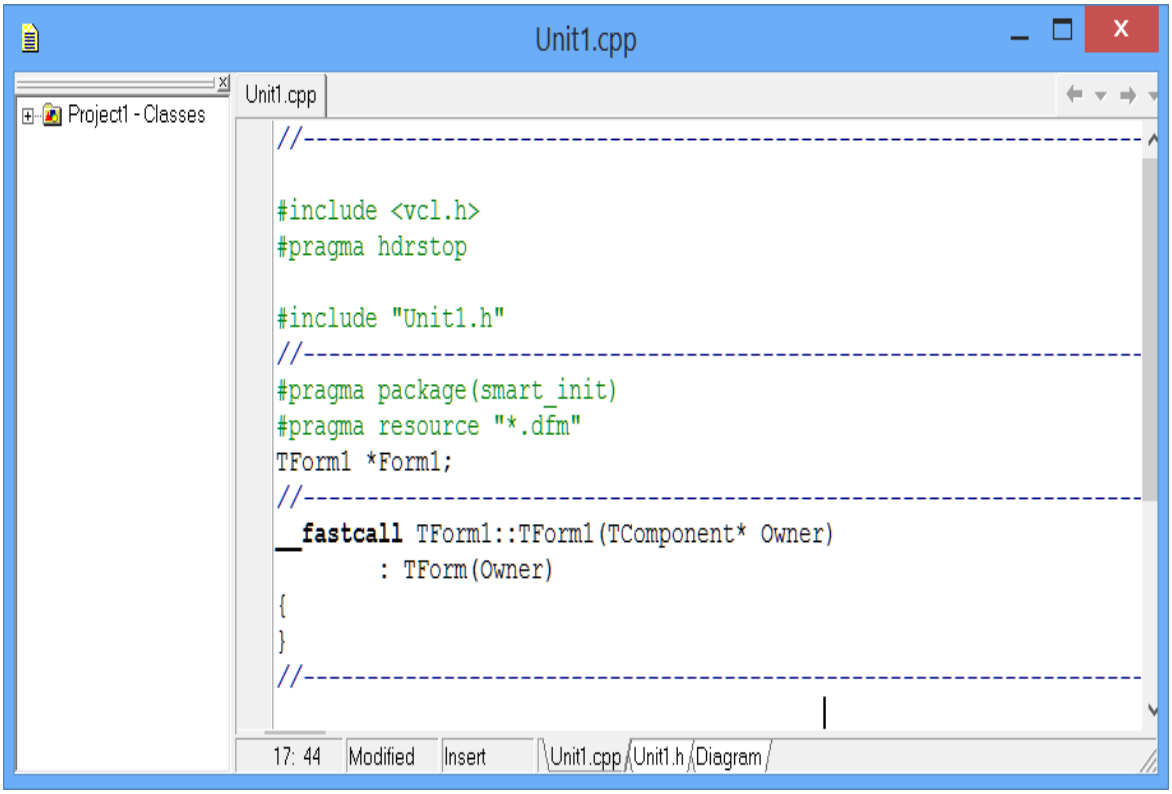


Рис. 2.18. Окно ввода кода программы.

Основные меню включают в себя:

-File - содержит необходимые команды для работы с файлами;

-Edit - содержит необходимые команды для редактирования содержимого файла;

-Поиск - раздел файлов, набор команд, позволяющий искать нужные части содержимого модуля;

-Просмотр - установка необходимой палитры инструментов в окне программы, кода проекта, а также набора команд для открытия и просмотра менеджера проектов;

-Компиляция - это компонент запуска проекта и программы, команды компиляции;

-Run - это набор команд, необходимых для запуска и остановки программы;

-Options - набор команд для настройки параметров конфигурации окна среды;

-Инструменты - позволяет использовать сервис и дополнительные утилиты;

-Помощь - это набор команд для вызова помощи.

Главное меню включает в себя множество разделов.

2.4. Палитра компонентов Borland C++ Builder 6

Стандартная палитра компонентов. Первая страница палитры компонентов содержит 16 объектов, и большинство программ используют эти объекты для создания.

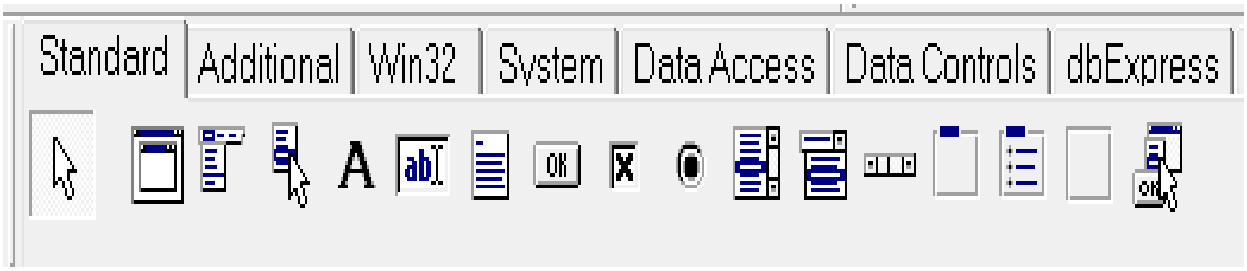


Рис. 2.19. Стандартная палитра компонентов.

Палитра компонентов и макет каждой страницы могут быть изменены. Это означает, что вы можете заменить существующие компоненты и добавить новые.

Ниже приводится краткое описание стандартных компонентов Borland C ++ Builder 6 и их использования:

Таб. 2.1. Палитра стандартных компонентов

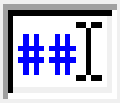
Вид	Выполнимая функция
	Курсор - это не компонент, а инструмент для отказа от объекта.
	Frame - служит окном для размещения других компонентов. Этот компонент очень похож на дизайнера форм.
	MainMenu - позволяет добавить главное приложение в приложение. Новый раздел меню добавляется через функцию «Элементы».
	PopupMenu - Создать подменю. Это меню отображается при щелчке правой кнопкой мыши.
	Label используется для отображения текста на экране. Тексты вводятся через свойство Caption.
	Edit - отображать короткие тексты и разрешать пользователю вводить собственный текст в конце программы.
	Memo означает работу с многорядными накидками. Введите данные через свойство Lines.
	Button - действие или процесс, когда кнопка нажата во время выполнения программы.
	CheckBox - В левой части экрана отображается текстовая строка, позволяющая выбрать несколько параметров.
	RadioButton - это круглая кнопка, которая позволяет выбирать из нескольких ситуаций.
	ListBox - это компонент для отображения информации о списке.
	ComboBox - Выбирает комбинацию области редактирования и списка параметров текста.
	Scrollbar - это полоса прокрутки. Часто редактируемые или видимые данные используются при выходе из экрана.
	GroupBox - используется для информирования операционной системы Windows о том, как расположены объекты.
	RadioGroup - используется для создания списков и выполнения программ, выбрав один из списков.
	Панель - Создает пустую панель, которая может содержать другие компоненты.
	Action List - Управляет взаимодействием между элементами интерфейса и приложением.

Палитра дополнительных компонентов. Палитра дополнительных компонентов позволяет нам еще больше улучшить пользовательский интерфейс приложения.

Рис. 2.20. Палитра дополнительных компонентов.

Компоненты, перечисленные в палитре «Дополнительные компоненты», показаны в следующей таблице:

Таб. 2.2. Палитра дополнительных компонентов

Вид	Выполнимая функция
	BitBtn используется для выполнения определенного процесса. Кнопка как кнопка, но изображения могут быть установлены на нее.
	SpeedButton - это кнопка для создания быстрой панели инструментов для команд. Обычно, только изображение (глиф) помещается на эту кнопку.
	MaskEdit - аналогично Edit, но для ввода можно щелкнуть. Формат определяется в свойстве EditMask.
	StringGrid - используется для визуализации текстовых данных в табличном представлении.
	DrawGrid - отображает пользовательские данные в виде таблицы.
	Image - отображает графические изображения в форме. Он устанавливает нужный файл изображения с помощью функции изображения.
	Shape - форма использует простые графические объекты (круги, квадраты).
	ScrollBox - создает пути, которые позволяют отображать неэкранные объекты в форме.

Палитра компонентов системы. Используется для создания мультимедийных приложений в операционной системе Windows.

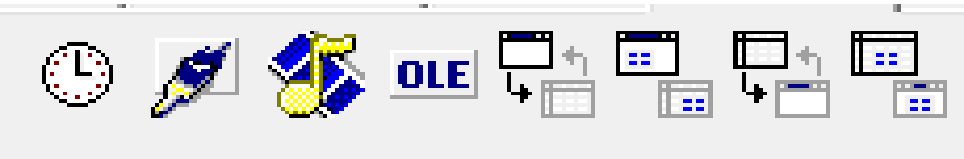
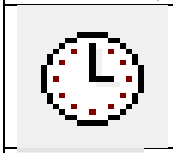


Рис. 2.21. Палитра компонентов системы.

Компоненты в палитре компонентов системы перечислены в следующей таблице:

Таб. 2.3. Палитра компонентов системы

Вид	Выполнимая функция
	Таймер - используется для установки и контроля временных интервалов в программе.
	PaintBox это место для рисования. Обработчик состояния вернет соответствующие щелчки мыши на Paintbox.
	MediaPlayer - создает панель форм для использования различных форматов файлов и управления мультимедийными инструментами.
	OleContainer - вводит механизм интеграции и подключения объектов OLE, который позволяет приложению передавать данные между различными приложениями в среде Windows.

Диалоги палитры компонентов. Палитра компонентов Dialogs содержит компоненты, которые вызывают стандартное диалоговое окно Windows. Внешний вид диалоговых окон зависит от версии среды Windows. Объекты на этой странице не отображаются во время выполнения приложения, и вам необходимо программно вызывать соответствующие диалоги.

Рис. 2.22. Диалоги палитры компонентов.

Компоненты, перечисленные в палитре компонентов Dialogs, перечислены в следующей таблице:

Таб. 2.4. Палитра компонентов диалогов

Вид	Выполнимая функция
	OpenDialog - создает диалоговое окно для открытия файлов. Вы можете установить фильтр для имени файла и расширения.
	SaveDialog - создает диалог для сохранения файлов.
	OpenPictureDialog - создает диалоговое окно для открытия графических файлов изображений. Вы сможете увидеть изображение в диалоговом окне.
	SavePictureDialog - создает диалоговое окно для хранения графических файлов.
	FontDialog - создает диалоговое окно, которое позволяет вам выбрать размер шрифта и внешний вид приложения.
	ColorDialog - Windows создает диалоговое окно для выбора цвета из цветовой палитры.
	PrintDialog - создает диалоговое окно для выбора и управления печатными носителями, установленными на вашем компьютере.
	PrinterSetupDialog - создает диалоговое окно для настройки принтера.
	FindDialog - открывает диалог правописания, который позволяет настроить параметры поиска в приложении.
	ReplaceDialog - открывает диалог текстового поиска, который позволяет заменить фрагменты слова, найденные в приложении.

Палитра компонентов ADO. Позволяет вам общаться с вашей базой данных с активными объектами данных (ADO). База данных связана с

компонентами Microsoft OLE. Эта палитра компонентов предназначена для работы с базой данных.

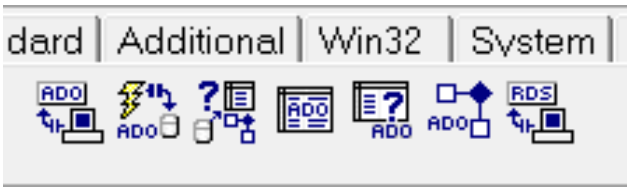
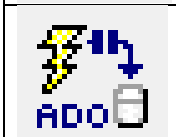
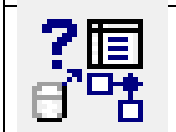
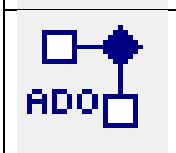


Рис. 2.23. Палитра компонентов ADO.

Компоненты в палитре компонентов ADO перечислены в следующей таблице:

Таб. 2.5. Палитра компонентов ADO

Вид	Выполнимая функция
	ADOConnection - это компонент, обеспечивающий доступ к базе данных.
	ADOCommand - это компонент, который позволяет вам выполнять команды SQL на основе вашей существующей базы данных.
	ADODataSet - обеспечивает доступ к одной или нескольким базам данных ADO.
	ADOTable - для связи с существующей базой данных в объектной базе данных.
	ADOQuery - запросы ADO, позволяющие выполнять команды SQL для извлечения данных из базы данных ADO.
	ADOStoredProc - ADO - это хранимая процедура, которая позволяет приложениям получать доступ к хранимым данным через интерфейс ADO.
	RDSConnection - это соединение RDS, используемое для создания серверных приложений из этой команды.

Палитра компонентов доступа к данным. Палитра компонента Доступ к данным позволяет приложению создавать элементы управления базой данных.

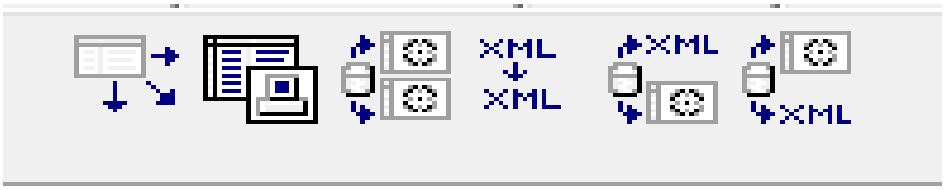
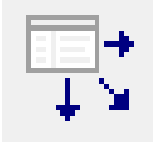
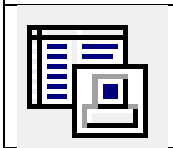
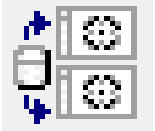
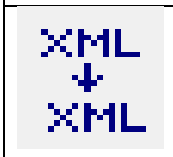


Рис. 2.24. Палитра компонентов доступа к данным.

Компоненты в палитре компонентов доступа к данным перечислены в следующей таблице:

Таб. 2.6. Палитра компонентов доступа к данным

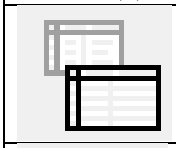
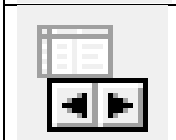
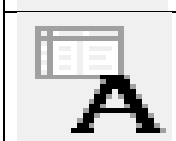
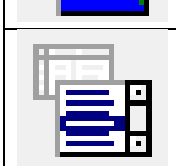
Вид	Выполнимая функция
	DataSource - позволяет редактировать и получать доступ к записям базы данных напрямую, связываясь с таблицей TTable или TAdo.
	ClientDataSet - это база данных, основанная на клиентских технологиях. Используется для создания клиентской базы данных.
	DataSetProvider является поставщиком сбора данных. Он используется в базе данных, которую необходимо кэшировать.
	XML Transform - используется для преобразования и возврата документов XML в пакет данных.

Палитра компонентов управления данными. Палитра компонента «Элементы управления данными» позволяет приложению создавать собственные инструменты для управления базами данных пользователей.

Рис. 2.25. Палитра компонентов управления данными.

Компоненты в палитре компонентов Data Controls перечислены в следующей таблице:

Таб. 2.7. Палитра компонентов доступа к данным

Вид	Выполнимая функция
	DBGrid - объект используется для отображения отчетов базы данных, таблиц и запросов в табличном представлении.
	DBNavigator - это набор ключей, который позволяет вам управлять записями базы данных.
	DBText - позволяет вставлять текст в столбец базы данных.
	DBMemo - добавляет несколько строк текста в столбец базы данных.
	DBImage - отображает доступные изображения в базе данных.
	DBListBox - используется для генерации данных компиляции в базе данных.
	TDBChart - используется для генерации различных типов графики.

Вопросы по главе 2

1. Какие технологии программирования Borland сочетал с C++ Builder 6?

2. Каковы основные компоненты основного рабочего окна Borland C++ Builder 6?

3. Каковы основные группы инструментов окна Borland C++ Builder 6?

4. Какие палитры компонентов доступны в системе Borland C++ Builder 6?

5. Какие компоненты являются палитрой компонентов по умолчанию?

Тестовые вопросы:

1. Объект окна Object Inspector ?

а) Приносит компоненты, установленные в форме.

б) Устанавливает различные свойства и события для выбранного компонента;

в) Описывает список функций для компонента;

г) Контролирует компонентные свойства формы ;

2. Почему используется компонент «Редактировать»?

а) Извлечь данные, использованные в анкете;

б) Для ввода и извлечения данных, используемых в форме;

в) составить описательный текст в форме;

г) чтобы начать форму;

3. Какой компонент используется для вставки и печати многострочных текстовых форм?

а) Memo;

б) Edit;

в) Label;

г) RadioGroup;

4. Как называются основные окна Borland C ++ Builder 6?

а) Project, Form, Object Inspector, Object tree View, Component;

б) Project, Form, Object Inspector, Object tree View, Standart;

в) Project, Form, Object Inspector, Object tree View, Unit;

г) Project, Form, Object Inspector, Object tree View, Additional;

5. Какая страница окна Инспектора объектов содержит списки событий?

а) Properties;

б) Standard;

в) Events;

г) Additional;

Глава 3. ОСНОВНАЯ КОНСТРУКЦИЯ ЯЗЫКА С++ И ОСОБЕННОСТИ ЕЁ ИСПОЛЬЗОВАНИЯ

Ключевые слова: алфавит С ++, переменные, инварианты, идентификаторы, операторы, производные, объекты, категории данных, компиляторы, трансляторы, стандартные функции, файлы, арифметические и логические переменные, локальные и глобальные переменные.

3.1. Основные составляющие языка С++

Алфавит С ++. Алфавит С ++ включает в себя следующие символы:

- заглавные и строчные буквы латинского алфавита (A, B, C, ... a, b, c).

- арабские цифры от 0 до 9 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9).

- Специальные символы (",; * / | \:;% ?! = <> {} [] () # & ^ ~).

Алфавитные символы образуют лексему языка, в том числе: слова, комментарии, идентификаторы, значки действий, неизменяемые и переменные, символы разделения.

Комментарии могут быть сделаны в любом месте программы. Они обычно пишутся после "//".

Служебные слова. Идентификаторы, которые не используются в языке, то есть программисты в качестве имен переменных, называются служебными словами.

Следующие слова доступны в С ++:

int	extern	else	char	register	for
float	typedef	do	double	static	while
struct	goto	switch	union	return	case
long	sizeof	default	short	break	entry
unsigned	continue	auto	if		

Идентификаторы. Идентификаторы состоят из латинских букв, подчеркиваний и цифр. Идентификатор должен начинаться с латинского алфавита или подчеркивания. Следует помнить, что есть разница между прописными и строчными буквами, например, MAX, CUM и суммой.

Переменные. Переменные, которые могут изменять значения во время выполнения программы, называются переменными. Имена переменных могут состоять из букв и цифр. Существует разница между прописными и строчными буквами при установке переменных. Каждая из переменных (буквы «A» и «a» обозначают 2 переменные) должна иметь свое имя, категорию, местоположение и значение. Применение к переменной делается от ее имени. Расположение переменной памяти для переменной является ее адресом. Переменная должна быть описана перед использованием. Переменные могут быть объявлены локально и глобально. Локально объявленные переменные применяются только к определенному процессу. Глобальные переменные позволяют применять в любом месте программы. Локальное и глобальное объявление переменных зависит от характера программы.

Константы. Переменные, такие как переменные, представляют ячейки памяти, предназначенные для хранения данных. В отличие от переменных, они не меняют значение программы. Когда оно объявляется неизменным, оно должно быть установлено на значение, которое не может быть изменено позже.

C ++ определяет два типа литеральных и литеральных переменных.

Литературные переменные непосредственно включены в программу. Например:

int myAge=39;

Unchecked - это переменная с именем.

Например:

const int St=16;

Определенные переменные легче использовать, чем буквальные. Потому что, если вы хотите изменить значение литеральной переменной с тем же именем, вы должны изменить значение всей программы, и единственная переменная - это изменить значение единицы.

Действия. Вы можете использовать следующее действия в C ++:

1) арифметические операции: +, -, /, *,%. Все действия обычно выполняются. При выполнении операции деления категория результатов определяется категорией задействованных значений. Если операция разбиения выполняется для всех чисел, результатом является целое число, то есть десятичная точка опускается (10/4 = 2). Если результат должен быть получен в виде истинного (десятичного) числа, число (.) Ставится после числа на рисунке или на полях, например, 9./5=1.8. Знак% (оператор модуля) представляет остаток, созданный делением целого числа на целое число. Например: 7% 3 = 1;

2) сравнения: = = (равно?); ! = (не равно); <; >; > =; <=;

3) логические действия: && (и) логическое умножение; || (или) логическое дополнение; ! (не) логическое отрицание.

Логические операции могут выполняться над произвольными числами. Если результат равен true, результат равен 1, если результат равен false, результат равен 0. Как правило, разница в 0 (ноль) считается истинной. Например: (i> 50) && (j == 24) или (s1 <s2) && (s3> 50 || s4 <= 20);

или $6 \leq x \leq 10$ записывается как (x> = 6) && (x <= 10).

4) процедуры оценки:

а) Символ действия значения - «=», который обычно использует значение для определенной переменной, например a = 5; б = 2 * c; x = y = z = 1;

б) Действие приращения (++) используется в двух значениях: значение переменной будет увеличено на 1 (a ++) после обращения к переменной, а значение переменной увеличится до 1 (++ a);

в) Акт декремента (-) выполняется так же, как операция впрыска, но только для уменьшения.

Например: s = a + b ++ (добавление a к b и затем увеличение значения b до 1);

s = a + (- b) (уменьшите значение b до 1, а затем добавьте его к a).

г) Сокращения также используются в C ++ (Таблица 3.1):

Таб. 3.1. Сокращенные действия

Аббревиатура	Полная заметка
x += a;	x = x + a;
x -= a;	x = x - a;
x *= a;	x = x * a;
x /= a;	x = x / a;
x %= a;	x = x % a;

Директива. Директивы предназначены для обработки исходного текста перед компиляцией. Любая директива начинается с "#". В каждой строке может быть написана только одна директива. Например, текст приложения #include <myfile> содержит содержимое файла объявления с именем myfile. Файл декларации содержит различную информацию, необходимую для успешного выполнения программы.

В таблице 3.2 перечислены действия, используемые в языке C ++.

Таб. 3.2. Действия, используемые в C ++

Арифметические операции	Случайные действия	Относительные действия	Логические операции
+ плюс	& и	= = равно	&& и
-минус	|или	!= не равно	|| или
*умножения	<< двинуть влево	> больше	! инверсия
/деление	>> двинуть вправо	>= больше или равно	
% получить модуль	~ инверсия	< меньше	
-унарный минус		<=меньше или больше	
+ унарный плюс			
Увеличение ++			
-- сокращение			

3.2. Типы данных

В программе категория данных определяет набор значений для данных, а также действия, которые необходимо предпринять. Для компиляции команд компилятор должен точно знать, сколько памяти хранится и какие действия предпринять. Все это продиктовано описанием категории данных. Результаты переменных, переменных и действий, используемых в программе, должны быть конкретными. Языковые категории.

C ++ подразделяются на базовые (basic-base) и комбинированные (структурные) типы. Обычная категория включает в себя все категории: истинные, логические и символические:

- bool (логическое);

- char (символ);

- int (целое);

- float (настоящее);

- double (это настоящий тип двойного определения).

На их основе формируются подкатегории. Составные категории включают в себя массивы, структурированные, индексы и классы.

Все категории. Все категории используются для описания целых чисел. Данные класса C ++ предоставляются в следующих типах:

- short (короткие);

- long (длиное);

- signed (знак);

- unsigned (без знака).

Целые числа со знаком: int со знаком, short int, int, long int и все не указывающие числа: unsigned int, unsigned short int, unsigned long int.

Без знака отрицательный представляет целое число. Программа может заменять короткие, длинные, подписанные и беззнаковые имена, такие как короткие int, длинные int, подписанные int и unsigned int. Кроме того, всем номерам автоматически присваивается подписанная категория int.

Истинный Тип. S ++ определяет типы float, double и long double. Они все работоспособны. Фактическое число состоит из мантиссы и порядка, а мантисса определяет точное число и порядок его значений. Вещественным числам с точкой пометки автоматически присваивается двойная категория. Вы можете указать категорию чисел, указав суффиксы F, f (float) и L, l (long). Например, 3.14F, 2E + 6L (длинная двойная категория).

Логическая категория. Значения логической категории принимают значения true и false. Они могут участвовать в арифметических операциях. При преобразовании этих значений в целую категорию, false-0 будет установлен в true-1, соответственно. Широко используется в процессах сетевого программирования.

Тип символа (символа). C ++ имеет 3 категории символов: char, подписанный char и unsigned char. Каждому символу назначается только 1 байт в памяти:

sizeof (char) = sizeof (char со знаком) = sizeof (char без знака) = 1

В следующей таблице перечислены категории, используемые в C ++, и диапазоны их значений (таблица 3.3):

Таб. 3.3. Список категорий, используемых в C ++, и их диапазоны значений

Категория	Диапазон значений	Размер (в байтах)
bool	true va false	1
signed char	-128 127	1
unsigned char	0 255	1
signed short int	-32768 32767	2
unsigned short int	0 .... 65 535	2
signed int	-2147483648 2147483647	4
unsigned int	0 4294967295	4
signed long int	-2147483648....2147483647	4
unsigned long int	0 4294967295	4
float	3.4e-38 .... 3.4e+38	4
double	1.7e-308 .... 1.7e+308	8
long double	3.4e-4932 .... 3.4e+4932	10

3.3. Стандартные функции

Программы могут использовать командные функции по умолчанию в структуре команд. Математические функции используются в арифметических операциях. В следующей таблице перечислены стандартные математические и другие функции (таблица 3.4).

Таб. 3.4. Стандартные математические и другие функции

Выражение	функции	Выражение	функции
sin x	sin(x)	$\sqrt{x}$	sqrt(x); pow(x,1/2.)
cos x	cos(x)	$\|x\|$	abs(x) yoki fabs(x)
tg x	tan(x)	arctan x	atan(x)
ex	exp(x)	arcsin x	asin(x)
ln x	log(x)	arccos x	acos(x)
lg x	log10(x)	$\sqrt[3]{x^2}$	pow(x,2/3.)
xa	pow(x,a)	log2x	log(x)/log(2)
ceil	Округление вещественного числа до взрослого		
floor	Округлите дробь вещественного числа (уберите дроби)		
fmod (a/b)	Рассчитать остаток a / b		

Следующие функции выполняют функцию изменения категории данных (Таблица 3.5):

Таб. 3.5.Функции изменения категории данных

Выражение функции в C ++	Функция функции
IntToStr(k) StrToInt (s) FloatToStr (n) StrToFloat (s)	Вернуть целое число k в строку Переместить с на целое число Преобразовать вещественное число в строку Переместить строку s в реальные числа

Давайте возьмем пример вышеуказанных стандартных математических операций. Например, $\frac{-b+\sqrt{b^2-4ac}}{2a}$ → (-b + sqrt(b*b-4*a*c))/ (2*a); yoki (-b + pow(b*b-4*a*c,1/2))/(2*a);

$\frac{\sin x^2+\cos y^2}{\sqrt{x+y}}\rightarrow$ (sin(pow(x,2)))/(pow(x+y,1/2));

e sin x + tg2(x+3) → exp(sin(x)) + pow(tan(x+3),2);

3.4. Структура программы в Borland C++ Builder 6

В C ++ программа состоит из ряда директив препроцессора, описаний и описаний глобальных объектов и функций. Директивы направляют программу до ее компиляции. Определения включают в себя функции и объекты. Активы необходимы для описания обработки данных в приложении. Функции определяют действия в программе.

Описания предоставляют компилятору имя и свойства объектов и функций, определенных в разных частях программы или других файлах.

Программа представлена одним или несколькими текстовыми файлами. Текстовый файл будет отсортирован по строкам. Каждая строка заканчивается символом стрелки.

Программа проходит три этапа обработки:

- преобразование текста с использованием препроцессора;

- компилировать;

- удалить (редактировать или удалить все ссылки).

Следует отметить, что использование производной #include <ad filename> не добавляет необходимые стандартные библиотеки в программу, но добавление ссылок выполняется на этапе компиляции.

После успешного завершения вышеупомянутых шагов будет сгенерирован код машины.

Функция препроцессора заключается в преобразовании текста программы для компиляции. Определяет порядок обработки препроцессора с использованием директив программы. Каждая директива начинается с "#", например #include или #define. #define указывает, как изменить текст в тексте, а «#» включает текстовые файлы для включения в программу. На рисунке 3.1 показаны процессы обработки заявки.

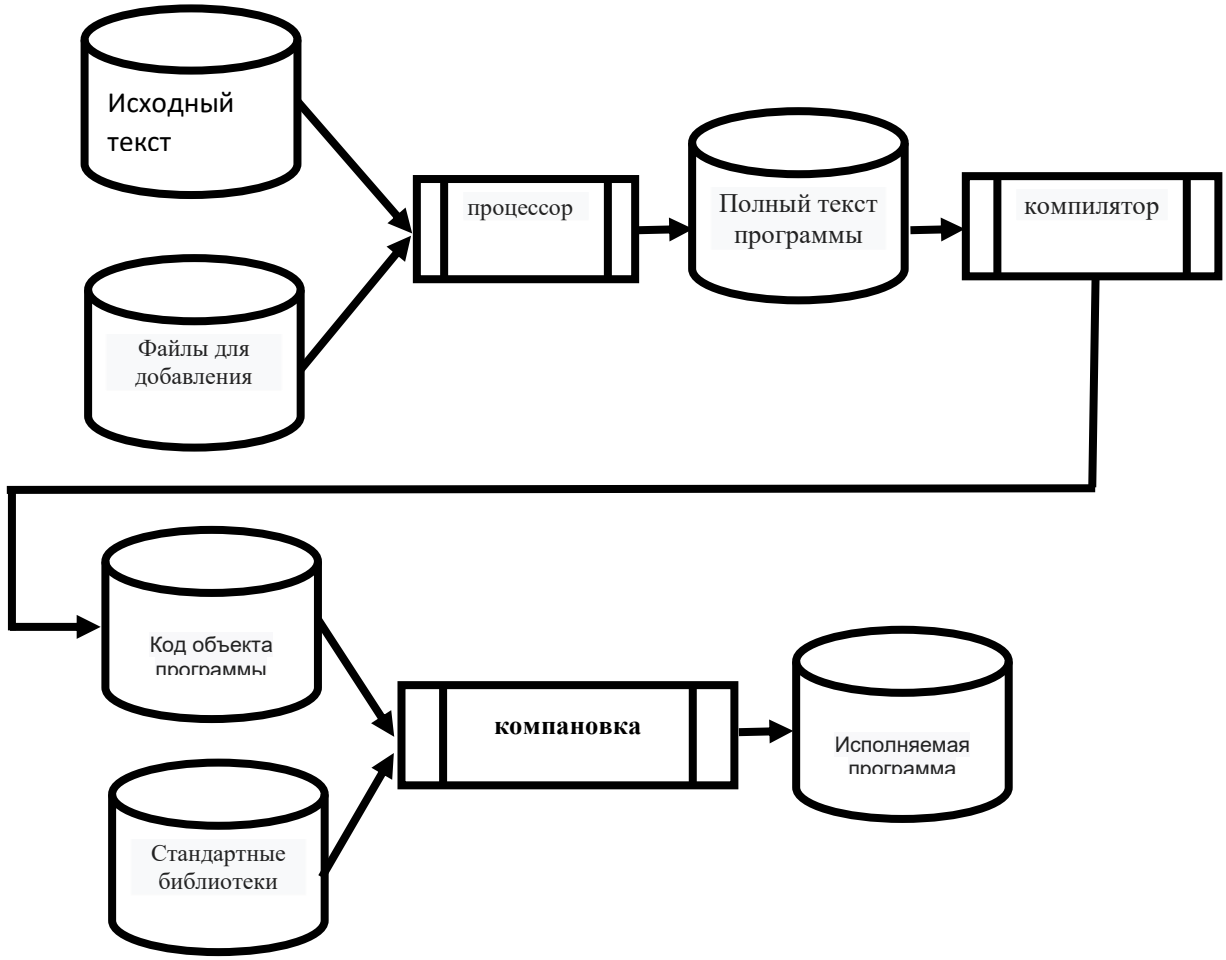


Рис. 3.1. Процессы обработки программного обеспечения.

Директива #include добавляет необходимый текст файла в текст программы из каталога файлов объявлений, который используется совместно

со стандартными библиотеками компилятора. Список заголовочных файлов для стандартных библиотек устанавливается в стандарте языка C ++.

Определения и определения глобальных объектов и функций должны предоставляться через основную функцию main. Поскольку это основная функция программы, без нее программа не может быть выполнена. В результате разработки и выполнения программ на C ++ создаются различные приложения. Вы можете создавать и выполнять программы в двух средах: в консоли и в среде приложений в визуальной форме.

Таким образом, общий вид программы выглядит следующим образом:

Директивы препроцессора

Void main( )

{

Описание активов;

Операторы выполняются;

}

Давайте посмотрим на каждую часть программы C ++.

1. Директивы - директива # include <file.h> означает инструкцию. C ++ использует необходимые директивы, в зависимости от структуры программы, то есть от необходимости. Они обозначены символом <>. Вот несколько основных рекомендаций:

- # include <stdio.h> - Для простой программы ввода / вывода на C.

Где std означает стандарт, i - вход, o - выход;

- # include <iostream.h> - использовать в обычном C ++ для ввода / вывода;

- # include <math.h> - использовать стандартные математические функции;

- # include <conio.h> - используется для формирования текстового интерфейса;

- # include <string.h> - используется для выполнения действий над строковой переменной;

- # include <stdlib.h> - для вызова файлов стандартной библиотеки;

- # include <time.h> - использовать часы на компьютере;

- # include <graphics.h> - Использовать графические возможности C ++.

Эти файлы представляют собой специальные рекламные файлы библиотеки и хранятся в отдельной папке с именем include. Эти файлы содержат прототипы функций, категории, переменные и определения переменных.

Директивы будут сканировать программу до ее компиляции.

2. Макросы - # определяют значение макроса. Например:

#define y sin (x + 25) - y = sin (x + 25) (данное значение);

#define pi 3.1415 - pi = 3.1415

#define s (x) x * x - s (x) = x * x (; не проверено)

3. Объявление глобальных переменных. Переменные, объявленные в основной функции, являются локальными, а переменные, объявленные вне функции, называются глобальными переменными. Глобальные переменные используются для всех программных объектов. Вы можете объявить переменную непосредственно перед использованием, тогда она будет локальной. Имя глобальной переменной может совпадать с локальной переменной. В этом случае значение локальной переменной изменит только ее значение в текущей функции, при этом глобальная переменная будет запущена, когда она выйдет из функции.

4. Основной функцией является main (). Эта функция должна быть в программе. Вообще говоря, программа на C ++ считается функцией. Функция main () {запускается и закрывается в конце программы}. основной - базовый. Категория отображается перед этой функцией. Если функция main () возвращает простое слово или фразу и не возвращает результата, она возвращает слово void. Функция main () вызывается операционной системой, а не программой. Операционная система не должна возвращать значение, потому что она не использует его. Поэтому мы должны указать тип функции main () как void. Каждая функция имеет свой аргумент, поэтому скобки родительской функции main () передаются ее параметру Иногда это может быть головная боль. Чтобы вернуть эту функцию, обычно используется оператор return. Возвращение 0 (ноль) означает, что операционная система закончила нормально работать. Категория return должна совпадать с категорией возврата в операторе функции. Например, int main () и 0 (ноль) - все категории. После этой функции объявляются локальные переменные, программы обработки деталей и их фактические параметры. Затем пишутся основные операторы программы (ввод / вывод, освоить и т. Д.). Если эти операторы ограничены, они заключаются в отдельные скобки {}. На C ++ программа написана строчными буквами. Некоторые операторы могут вводиться заглавными буквами, в этом случае они упоминаются отдельно. Операторы отмечены знаком «;». Операторы могут быть написаны в ряд. Программа также может делать комментарии, которые заключены в / * .... * /. Если комментарий заканчивается строкой, он пишется после //. Например:

```
main () // Основные функции C ++
cout << "Send function 1 =" << f1; // Введите функцию f1.
cin << "Функция 1 -" << A1; // A1 может функционировать.
```

3.5. Создание консольного приложения в Borland C++ Builder 6

Консольные приложения могут быть созданы в Borland C ++ Builder 6 различными способами. Самый простой способ сделать это:

1. Запустите среду Borland C ++ Builder 6:

Пуск=> Все программы=> Borland C++ Builder 6 => C++ Builder 6

2. В главном меню «Файл» выберите «Новый», а затем «Другое»:

File=> New=> Other

3. Выберите “Console Wizard” в специальном окне, чтобы сохранить формы и проекты, и нажмите кнопку OK (рисунок 3.2).

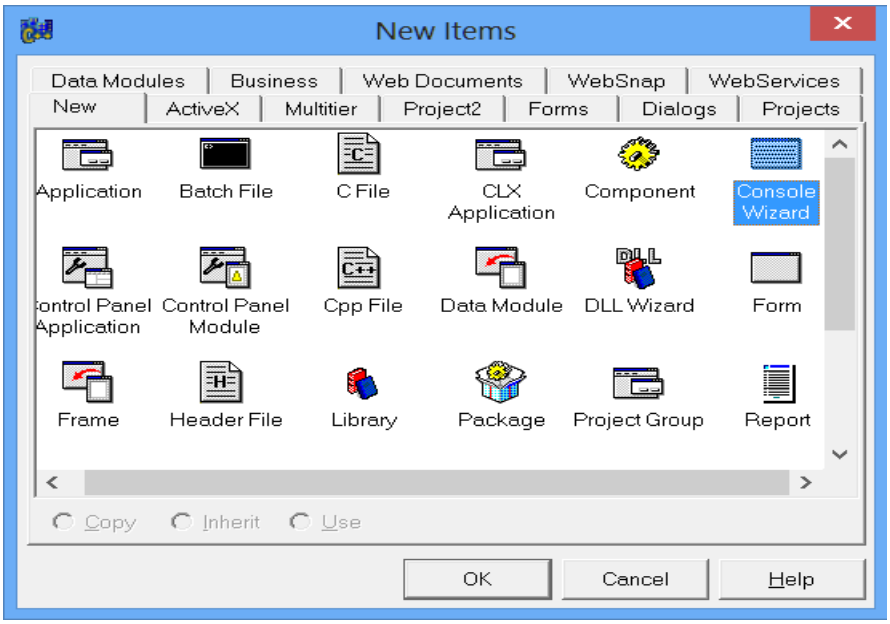


Рис. 3.2. Выберите консольную среду.

В результате на экране откроется окно проекта (с именем расширения unit.cpp).

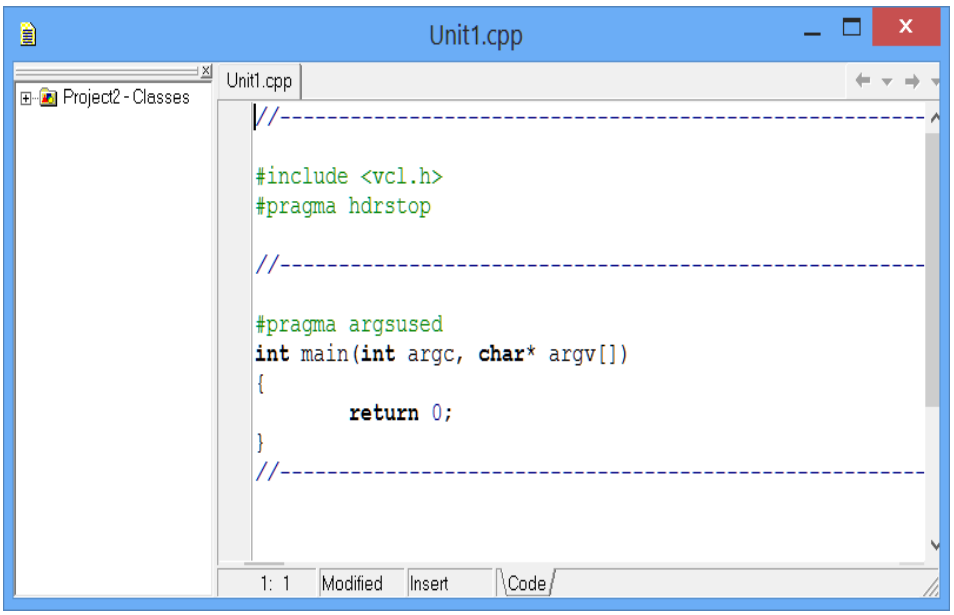


Рис. 3.3. Изображение консольной среды.

Скобки {} включают текст программы файла проекта (после int main (int argc, char * argv []).

```
//---------------------------------------------------------

#include <vcl.h>

#pragma hdrstop


//---------------------------------------------------------

#pragma argsused

int main(int argc, char* argv[])

{

// вводится текст программы

return 0;

}

//---------------------------------------------------------
```

Вы должны сохранить скомпилированную программу перед ее запуском. Чтобы сохранить ее, вам нужно выбрать командную строку «Файл => Сохранить все» в главном меню. Рекомендуется сохранять каждый проект в отдельной новой папке. Проект автоматически называется «Project1.bpr», который может быть изменен программистом. Здесь «1» - это номер каждого последовательного названия проекта (номера 1, 2, 3, ...). После сохранения проекта мы передадим проект. Для этого вам необходимо ввести следующую последовательность команд из главного меню: «Run => Run» или нажмите функциональную клавишу [F9]. После этого проект будет скомпилирован, на экране появится следующее окно (рисунок 3.4).

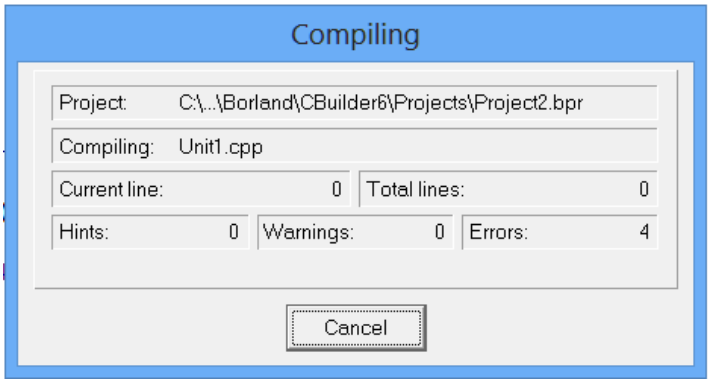


Рис. 3.4. Окно компиляции проекта.

После запуска программы на экране появится стандартное окно программы, в которое вводятся значения, и будут отображаться результирующие значения.

Как только файл проекта создан, следующие файлы будут автоматически созданы и сохранены в одной папке:

- Project1.bpr - главный файл проекта, в котором хранятся и запускаются ключевые компоненты;

- Unit1.cpp - текстовый файл программы;

- Project1.exe - это файл приложения или исполняемый файл. Этот файл компилируется с использованием компилятора или компилятора. При нажатии [F9] исполняемый файл генерируется автоматически. Исполняемый файл является автономным файлом и не требует каких-либо других файлов или программ. Вы можете запустить его как другие программы;

- Project1.res - это файл ресурсов проекта.

В консольной среде давайте посмотрим на слово hello world. Для этого мы открываем новую консольную среду. Следующий текст программы будет вставлен в открытой среде:

```
#include <vcl.h>
#include <iostream.h>
#include <conio.h>
#pragma hdrstop

//---------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
  cout << "Hello, World!" << endl;
  getch();
  return 0;
}
//---------------------------------------------------------
```

В результате появится следующая версия программы:

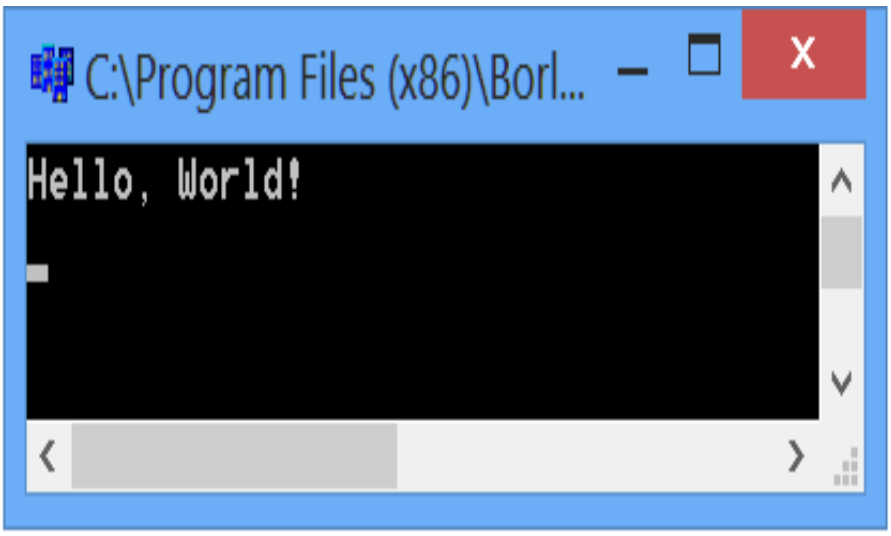


Рис. 3.5. Окно результатов.

3.6. Создание проектов в приложении «Форма»

Borland C ++ Builder 6 использует интерфейс RAD (Rapid Application Development). Этот интерфейс делает программу легкой для разработчика, то есть компьютер выполняет много сложных задач, подготавливает основную программу, а разработчик завершает проект с помощью операторов языка C ++ и использует готовые инструменты. Форма приложения в окне Форма основана на технологии визуального программирования, а форма (Форма) принимается как сложный структурированный объект. Форма также может вместить несколько других объектов. Объект представляет собой диалоговое окно и элементы управления (поля ввода и вывода, клавиши управления и т. Д.). В Borland C ++ Builder 6 приложение формы содержит различные компоненты (объекты), принадлежащие библиотеке визуальных компонентов (Visual Component Library - VCL).

В визуальном программировании создание и выполнение программы в основном основаны на двух взаимосвязанных процессах:

- Визуальное оформление процесса, то есть оформление формы;

- написать (написать) код приложения, соответствующий приложению.

Существует специальное окно кода для написания кода программы, которое предназначено для ввода и редактирования текста программы. Эти коды написаны на C ++.

После загрузки Borland C ++ Builder 6 появится окно формы. В окне «Коды» исходный код программы, представляющей эту форму, распечатывается системой, и щелкните Unit1.cpp - окно, чтобы перейти к окну кода (рис. 3.6).

Программный код генерируется в виде модуля. Как упоминалось ранее, модуль представляет собой программу, которая описывает категории компонентов в приложении формы, категории данных, используемые в программе, и способы управления ими. Окно «Unit1.cpp» пустого модуля формы будет выглядеть так:

```
//---------------------------------------------------------

#include <vcl.h>

#pragma hdrstop

#include "Unit1.h"


//---------------------------------------------------------

#pragma package(smart_init)

#pragma resource "*.dfm"

TForm1 *Form1;


//---------------------------------------------------------

__fastcall TForm1::TForm1(TComponent* Owner)

  : TForm(Owner)
```

```
{

}

//---------------------------------------------------------
```

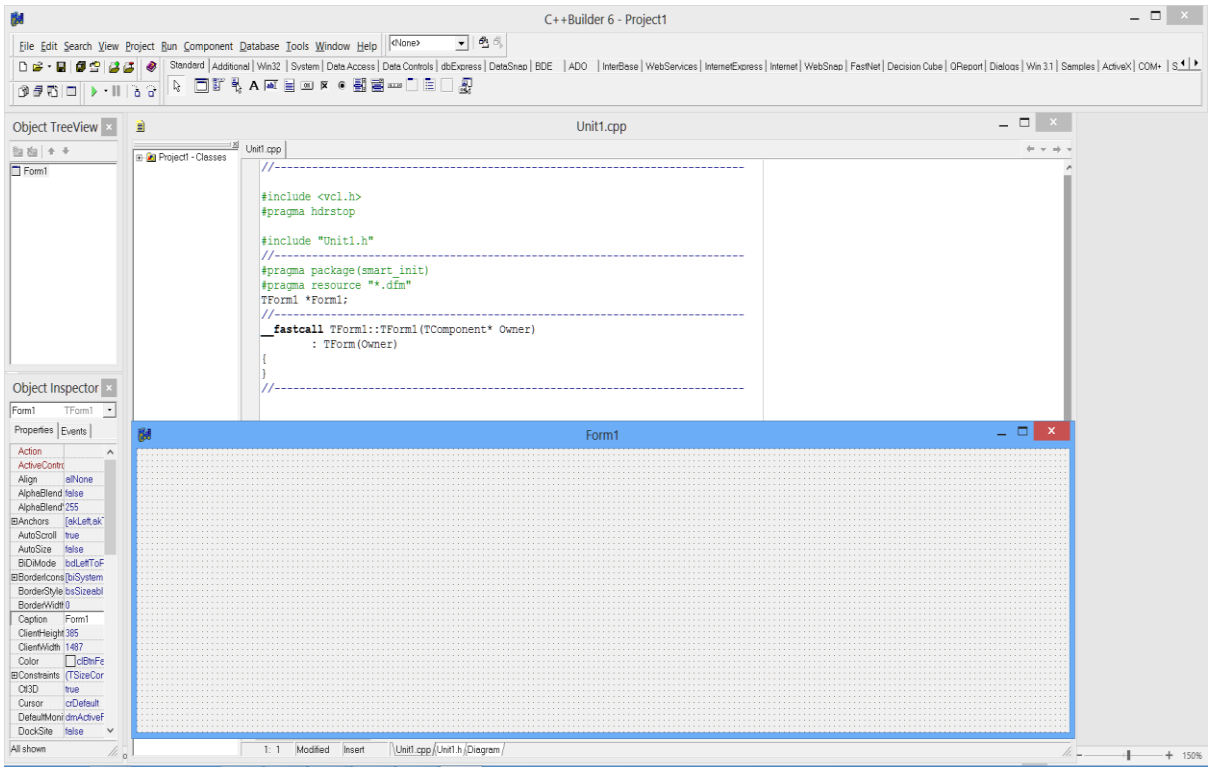


Рис. 3.6. Перейти к коду программы.

Borland начинается с окна формы программирования C ++ Builder 6, то есть компоненты формы устанавливаются, а дизайн формируется. Затем пишется код модуля для управления компонентами этой формы. Когда код модуля написан, система Borland C ++ Builder 6 создает базовое программное обеспечение для управления проектом для этого модуля в файле «Project1.cpp». Программа проекта немного отличается от основной программы консольного приложения. Ниже мы рассмотрим приложение «Hello World» для Form1. Для этого введите следующий текст программы:

```
//---------------------------------------------------------
```

```
#include <vcl.h>
#pragma hdrstop
#include “Unit1.h”

//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource “*.dfm”
TForm1 *Form1;
//---------------------------------------------------------
__fastcall Tform1::Tform1(Tcomponent* Owner)
  : Tform(Owner)
{
}

//---------------------------------------------------------
void __fastcall Tform1::Button1Click(Tobject *Sender)
{
Label1->Caption=”Hello World”;
}
//---------------------------------------------------------
```

В результате появится следующая версия программы:

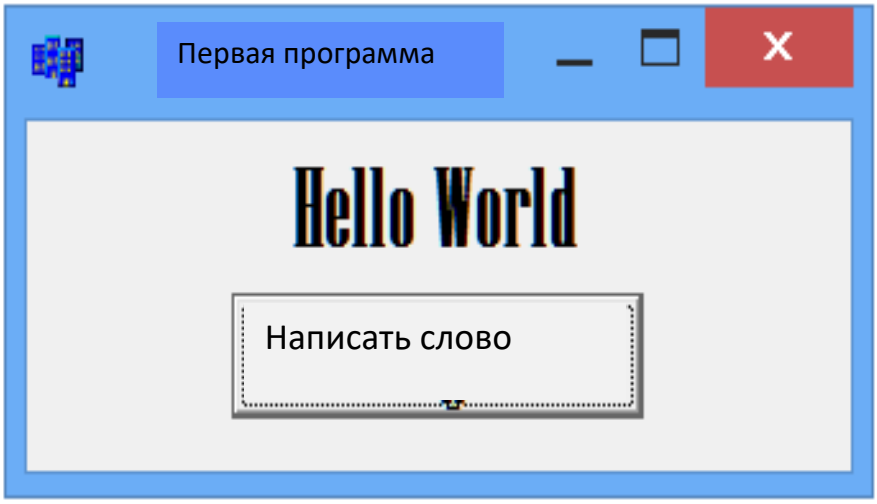


Рис. 3.7. Окно результатов.

Не рекомендуется менять базовое программное обеспечение проекта, все действия, выполняемые над объектами приложения, приведены в модуле.

Как только код модуля будет сохранен, он запустится, что означает, что программа на C ++ пройдет три основных этапа:

- сборник;

- установка;

- производительность.

На этапе компиляции текст программы переводится в машинный код. В начале фазы компиляции необходимые дополнения и компоненты объединяются.

При компиляции программы будут создаваться файлы с расширением * .cpp, * .dfm, * .res, * .obj, * .h, * .bpr, * .exe (рис. 3.8). В файле расширения * .cpp хранится текст программы, Введеноный в окне записи кода, органайзер окна формы * .dfm.

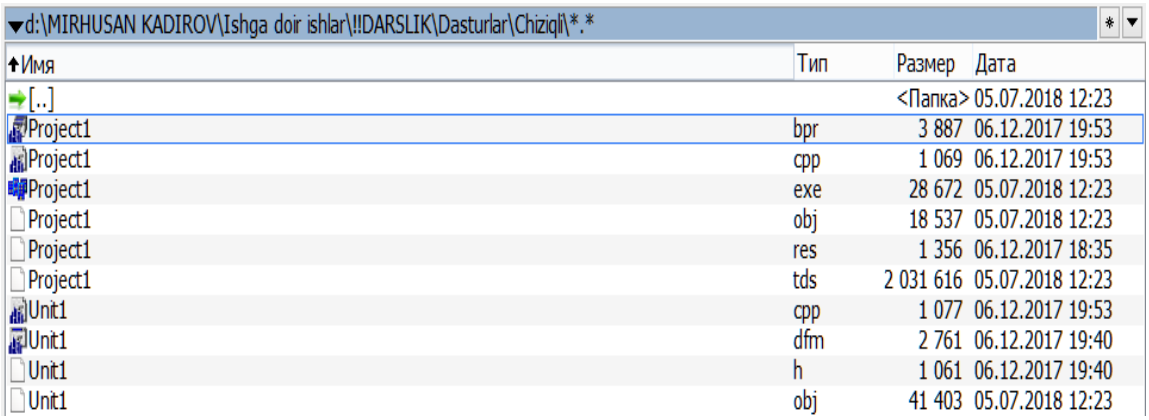


Рис. 3.8. Расширения файлов, созданные в C ++ Builder 6.

Реализация этого вида окна и компоненты, которые установлены для запуска в результате программа может быть достигнута. Это похоже на другие окна операционной системы Windows. Вы можете переместить его, изменить его размер и закрыть его. Это облегчит программирование.

Вопросы по главе 3

1. C ++ программирование объединяет технологии?

2. Каковы основные составляющие C ++?

3. Что такое переменные C ++?

4. Какие категории данных доступны в C ++?

5. Файлы проекта вместе?

Тестовые вопросы:

1. Основная функция файла project.exe?

а) File Launcher;

б) Файл, в котором хранятся компоненты проекта и значки;

в) текстовый файл программного обеспечения;

г) Файл, в котором хранятся текст и компоненты программы;

2. Какие переменные?

а) переменные а, б, х, у;

б) Значения, которые нельзя изменить в ходе выполнения программы;

в) Значения, которые могут изменить значение программы во время выполнения;

г) Значения, которые могут и не изменяют свое значение во время выполнения программы;

3. Каким оператором объявляются переменные?

а) int;

б) float;

в) char;

г) const;

4. Выберите правильный ряд категорий?

а) Bool, char, int, float;

б) Long, Bool, int, float;

в) Long, char, int, float;

г) Bool, int, float, signed;

5. Какому типу относиться оператор bool?

а) весь

б) логичный

в) символ

г) настоящий

Глава 4. ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ ПРОЦЕССОВ В СРЕДЕ BORLAND C++ BUILDER 6

Ключевые слова: операторы, оператор обучения, генерация значений, ввод и вывод данных, консольная среда, визуальная среда, компиляция программ, оператор контента, модуль, стандартный поток.

4.1. Понятие оператора. Классификация операторов языка C++

Операторы - это выражения команд в алгоритме решения проблем, которые соответствуют синтаксису конкретного языка программирования.

В C ++ операторы подразделяются на исполняемые и неработающие, простые и составные операторы. Исполняемый файл выполняет некоторые действия с данными, неисполняемый файл используется для описания данных, например int a; - вся категория a - служит оператором описания переменной.

Простые операторы включают в себя операторы ввода и вывода, операторы мастеринга.

Составной оператор или блок - это группа операторов с "{}" скобками.

Кроме того, существуют категории управляющих операторов, к которым относятся:

- операторы выбора – if и swich;

- операторы итератора - while, do while, for;

- операторы перехода - break, goto, return.

4.2. Операторы присваивание

Процесс решения проблемы делится на ряд этапов. На каждом из этих этапов новые значения рассчитываются (рассчитываются) на основе известных значений. Некоторые из этих определенных значений являются конечными значениями, некоторые являются промежуточными значениями и являются начальными значениями для последующих шагов.

Выражение используется для определения новых значений, причем каждое выражение определяет одно правило определения значения.

Чтобы использовать вычисленное значение на следующем шаге процесса вычисления, необходимо помнить, что это делается путем присвоения вычисленного значения известной переменной. Это делается с помощью оператора значения, который является одним из ключевых операторов, то есть главного оператора.

Главный оператор определяется в синтаксисе следующим образом:

<главный оператор> :: = <переменная> = <ifact>;

Здесь главный герой с 1 символом читает «=» «ассимилировать».

При выполнении главного оператора значение выражения справа от знака «=» вычисляется по переменной слева от символа. Вместо выражения справа от оператора может также использоваться переменная, переменная с заданным значением.

Например,

y = sin(x + 6,78) + cos(x + 5,889); x = 6; s = «Информатика»;

Размеры слева и справа от оператора должны быть объявлены в одной категории. По мере выполнения главного оператора определенные переменные будут принимать новое значение, которое может быть использовано позже.

Выражение справа от оператора также можно использовать вместо оператора выражения, например, a = b = c;

Справа и слева от главного оператора + =, * =, / = и так далее. Его также можно использовать, например, a + = b или b = a ++. Мы рассмотрим эти действия более подробно позже в этом разделе.

4.3. Ввод и вывод данных

Процесс ввода - это инициализация переменных, используемых в этой программе.

Не существует специальных инструментов для ввода и экспорта данных в C ++ с использованием стандартных библиотечных функций, категорий и объектов:

iostream.h - ввод данных из библиотечного файла с потоков клавиатуры «cin» и «cout», которые определяются:

1) << - записать данные в поток;

2) >> - Чтение данных из потока.

Например:

```
#include <iostream.h>;

. . . . . .

cout << “n Значение переменной = ”;

cin >> n;
```

Выходные данные записываются как «cout <<», а входные данные - «demon >>». cout - console output и cin - console input.

Имена переменных даны после слова «jin» на входе. Если их более одного, между каждой переменной вставляется>>.

Например:

```
cin >> a >> b;

cin >>max >>min >> a2;
```

Числовые значения этих переменных задаются клавиатурой после компиляции программы. Вы можете вводить цифры через пробел или клавишу Enter. С помощью оператора «cin» можно вводить только числовые значения, а выражение нельзя перезаписать.

Оператор cout <<используется для извлечения данных. Здесь вы можете вводить имена переменных, любые слова или фразы в кавычках. Если есть несколько переменных, значение может быть извлечено отдельно или последовательно. Например:

cout << a; cout << x << y;

cout << “y=” << y;

cout << “оператор значение=” << fax;

Когда отображаются несколько результатов, они появляются последовательно на экране. Это может вызвать дискомфорт при чтении результатов. Поэтому, когда вставляется cout << в конце строки <endl (end line-конец строки, курсор перемещается на следующую строку и отображается следующий результат. Например:

cout << “max=”<<max<<endl;

cout << “min=”<<min<<endl;

На экране появятся следующие ответы:

max=9

min=2

cout << Вы также можете использовать арифметические операции в операторе:

cout <<”значение выражения=” << t*2+sin(t))<<endl;

Результаты также могут быть суммированы по восьми и шестнадцати пунктам. Для этого используются слова oct (восемь) и hex (шестнадцать). Например:

cin >> a;

cout << a << endl;

cout << oct << a << endl;

cout << hex << a << endl;

Следующие символы также могут использоваться с оператором «cout» (они также называются ESC-символами):

\ n - новая строка. Курсор перемещается в начало новой строки;

\ t горизонтальная вкладка, где курсор перемещается на несколько букв вправо;

\ v - вертикальная вкладка, где курсор проходит несколько строк;

\ r - возврат, то есть курсор возвращается в начало той же строки и не перемещается на новую строку;

\ a - компьютер производит динамический звук;

Есть разница между \ n и endl.

При отображении реальных чисел вы можете отформатировать их, то есть контролировать, сколько очков комната получает после данной точки. Для этого

cout. precision(n); - используется функция, где n - показатель точности.

Например: cout.precision (4);

cout << y;

Этой функции достаточно написать только один раз в программе.

Язык C ++ использует ряд стандартных функций для ввода и извлечения символов:

- getch (arg) - ожидание ввода одного символа. Введеноные символы не отображаются на мониторе. Если вы используете эту функцию в конце программы без аргументов, монитор может считывать данные до тех пор, пока не будет нажата клавиша;

- putch (arg) - используется для возврата одного символа в поток по умолчанию. Символ не отображается на мониторе;

- getchar (arg) - дождаться ввода одного символа. Символ входа отображается на мониторе. Если вы используете эту функцию в конце программы без аргументов, монитор может считывать данные до тех пор, пока не будет нажата клавиша.

- putchar (arg) - используется для возврата одного символа в поток по умолчанию. Символ отображается на мониторе. Эти функции находятся в модуле iostream.h.

Ввод и вывод символов в программе выглядит следующим образом:

```
c=getchar();

putchar(c);

c=getch();

putchar();

getch();
```

Введите или отобразите данные программы форматирования. Для этого используйте следующие функции:

1. Функция Scanf находится в модуле iostream.h со следующим обзором:

```
scanf (control, arg1, arg2, ...)
```

Функция будет читать символы из потока по умолчанию, форматировать их на основе массива управления и записывать в соответствующие параметры. Параметр должен быть индексом. Массив контроллера содержит следующие спецификации модификации: пробел, табуляция и следующий символ.

1. Простые символы (кроме%) должны соответствовать следующим символам во входном потоке;

символы спецификации, начинающиеся с символа% 2;

3. символ *, запрещающий начинать значение с%

4. Максимальное количество полей, начинающихся с символа%;

Можно использовать следующие символы спецификации:

1. d - ожидается целое число

2. o –число которое начинается или не начинается с ожидается 8

3. x - 0 x шестнадцатеричный или не шестнадцатеричный.

4. h - ожидается десятичное число.

5. c - ожидается один символ.

6. Дождитесь строки s.

7. Ожидаемое количество категорий f - float. Вы можете указать целое число Введеноного вами числа и количество десятичных знаков после точки, а также число чисел мантиссы после “E” или “e”.

2. Функция Printf используется для сброса указанных параметров в поток по умолчанию. Функция находится в модуле iostream.h и имеет следующий обзор:

printf (control, arg1, arg2, ...);

-control управления называется массивом элементов управления и состоит из двух типов символов: простые выходные символы и спецификации, которые изменяют следующий параметр. Каждая спецификация начинается с символа% и заканчивается символом, который представляет тип изменения. Следующие символы могут быть вставлены между знаком% и изменяющим символом:

- знак минус, указывающий, что аргумент слева должен быть выровнен - "-";

- строка чисел с указанием минимальной длины поля – “n”;

- точка, отделяющая длину поля от следующего числа - “.” .

Последовательность чисел, указывающая, сколько символов следует исключить из одной строки, и сколько десятичных цифр будет напечатано после точки с плавающей запятой или двузначных цифр.

Маркер длины указывает, что номер, который будет выдан, относится к длинной категории l.

Символы модификации включают в себя:

d - параметр преобразуется в целое число;

o - параметр преобразуется в восемь цифр без значения и первая цифра 0;

x - параметр преобразуется в шестнадцатеричное без 0 и 0x;

h - параметр преобразуется в шестнадцатеричное;

c - параметр рассматривается как один символ;

s - Символы строки параметров нажимаются до тех пор, пока не отобразится нулевой символ или указанное количество символов;

e - Параметр считается числом с плавающей или двойной категориями и представлен десятичным числом, представленным символом m, nnnnnnE + -xx (2.33333E + 21);

f - параметр обрабатывается как число с плавающей запятой или двойное число и преобразуется в шестнадцатеричное в форме mnnnnnn;

g - используется как% e или% f.

Если символ ниже% не является символом замены, он будет напечатан. Вам нужно набрать %%, чтобы напечатать символ%.

4.4. Программирование линейных процессов в консольном приложении

Мы рассмотрим некоторые простые линейные технологические решения для консольных сред.

Пример 1 Рассчитайте радиус сферы по следующей формуле:

$$V = \frac{4}{3}\pi R^2$$

Программа выглядит так.

```
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>
#include <vcl.h>
#pragma hdrstop

//---------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{ float pi=3.14;
 float r,v;
cout << " Введите значение радиуса R :"<<"\n";
cin >>r;
v=4/3*(pi*pow(r,2));
cout << " результат: "<<"\n";
cout <<" Размер сферы x="<<v;
```

```
  getch();
  return 0;
}
//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

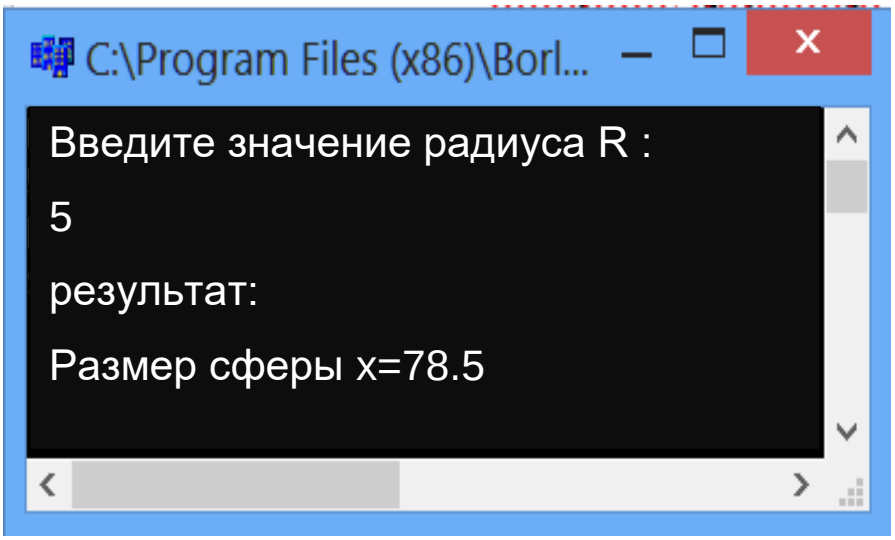


Рис. 4.1. Окно результатов.

Пример 2. Рассчитать общее сопротивление цепи, состоящей из следующих резисторов: последовательное сопротивление Rket = R1 + R2, параллельное сопротивление Rpar = R1 * R2 / (R1 + R2). Обозначим R1 как R1, R2 R2, Rket как RKET, Rpar как RPAR. Создайте программу сопротивления цепи:

```
//---------------------------------------------------------
 #include <stdio.h>
#include<iostream.h>
#include<conio.h>
```

```
#include <math.h>
#include <vcl.h>
#pragma hdrstop

//---------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
float R1, R2,RKET,RPAR;
cout<<" Введите значения R1: "<<"\n";
cin>>R1;
cout<<" Введите значения R2: "<<"\n";
cin>>R2;
RKET=R1+R2;
RPAR=R1*R2/(R1+R2);
cout<<" Последовательно соединенная цепь, R="<<RKET<<"\n";
cout<<" Параллельно соединенная цепь, R="<<RPAR<<"\n";
  getch();   return 0;
}
//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

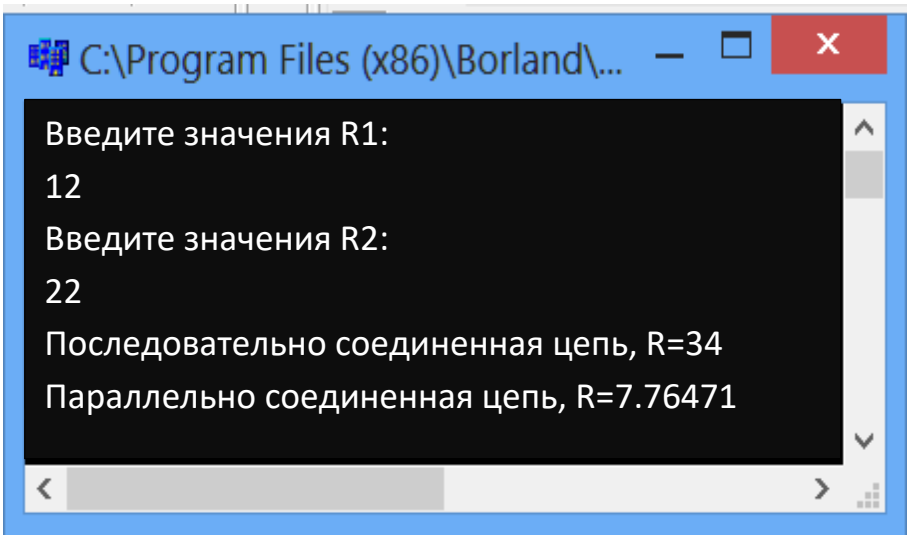


Рис. 4.2. Окно результатов.

Пример 3 Декартова область содержит 3 точки с координатами (x1, u1, x2, u2, x3, u3). Найти расстояние между точками R1, R2, R3.

```
//---------------------------------------------------------

#include <stdio.h>

#include<iostream.h>

#include<conio.h>

#include <math.h>

#include <vcl.h>

#pragma hdrstop


//---------------------------------------------------------

#pragma argsused

int main(int argc, char* argv[])

{
```

```
float  x1, y1, x2, y2, x3, y3, r1, r2, r3;
 cout<<" Введите значения X1,Y1"<<"\n"; cin>>x1>>y1;
 cout<<" Введите значения X2,Y2 "<<"\n"; cin>>x2>>y2;
 cout<<" Введите значения X3,Y3 "<<"\n"; cin>>x3>> y3;
    r1= sqrt(pow(x2-x1,2)+pow(y2-y1,2));
    r2= sqrt(pow(x3-x2,2)+pow(y3-y2,2));
    r3= sqrt(pow(x3-x1,2)+pow(y3-y1,2));
 cout<<"R1= "<<r1<<"   R2= "<<r2<<"   R3= "<<r3;
  getch();
return 0;
}
//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

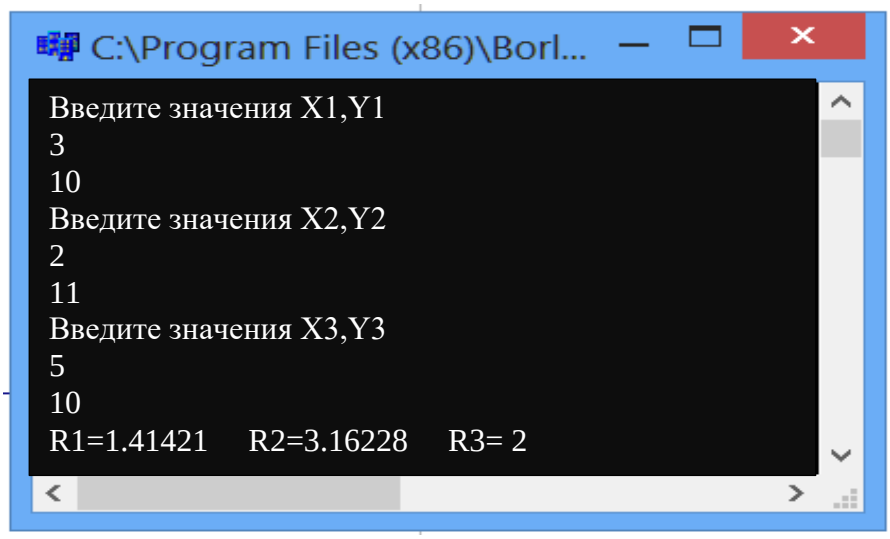


Рис.4.3. Окно результатов.

Пример 4. Даны выражения $A = \frac{\sin x^2 + e^{y+1}}{\sqrt{x+y}}$ va $B = tgx^3 + \ln y + |x+y|$. Здесь x = 3,1, y = 1,2. Создайте программу для расчета этих выражений.

```
//---------------------------------------------------------
 #include <stdio.h>
#include<iostream.h>
#include<conio.h>
#include <math.h>
#include <vcl.h>
#pragma hdrstop


//---------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
float a,b,x,y;
 cout<<" Введите значение x"<<"\n"; cin>>x;
 cout<<" Введите значение y"<<"\n"; cin>>y;
 a=sin(pow(x,2))+exp(y+1);
   b=tan(pow(x,3))+log(y)+fabs(x+y);
   cout<<"A= "<<a<<"   B= "<<b;
  getch();
```

```
 return 0;
}
//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

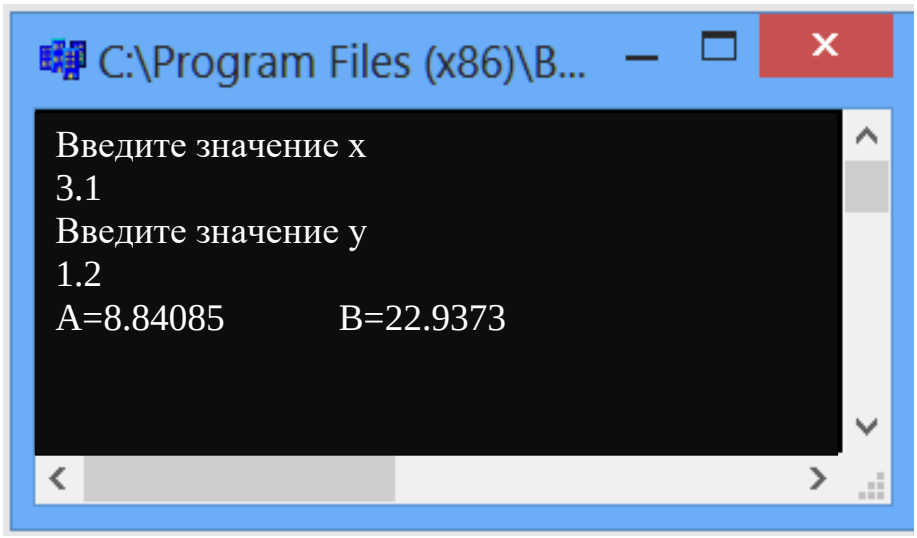


Рис. 4.4. Окно результатов.

4.5. Программирование линейных процессов в приложении «Форма»

В приложении Form линейное программирование процессов выполняется с использованием визуальных компонентов. Добавление значений в форму, вывод результатов в приложение и выполнение значений для переменных выполняются с помощью специальных компонентов. В среде визуального программирования эти типы проблем используются для добавления значения в переменные формы: редактирование компонента, текста в метку, кнопки или BitBit для запуска программы.

Пример 1. Даны выражения $A = x^2 + \sqrt{y}$ va $B = \sin x + tgy^2$. Здесь x = 2,1, y = 4.2. Создайте программу для расчета этих выражений.

Алгоритм решения этого примера следующий:

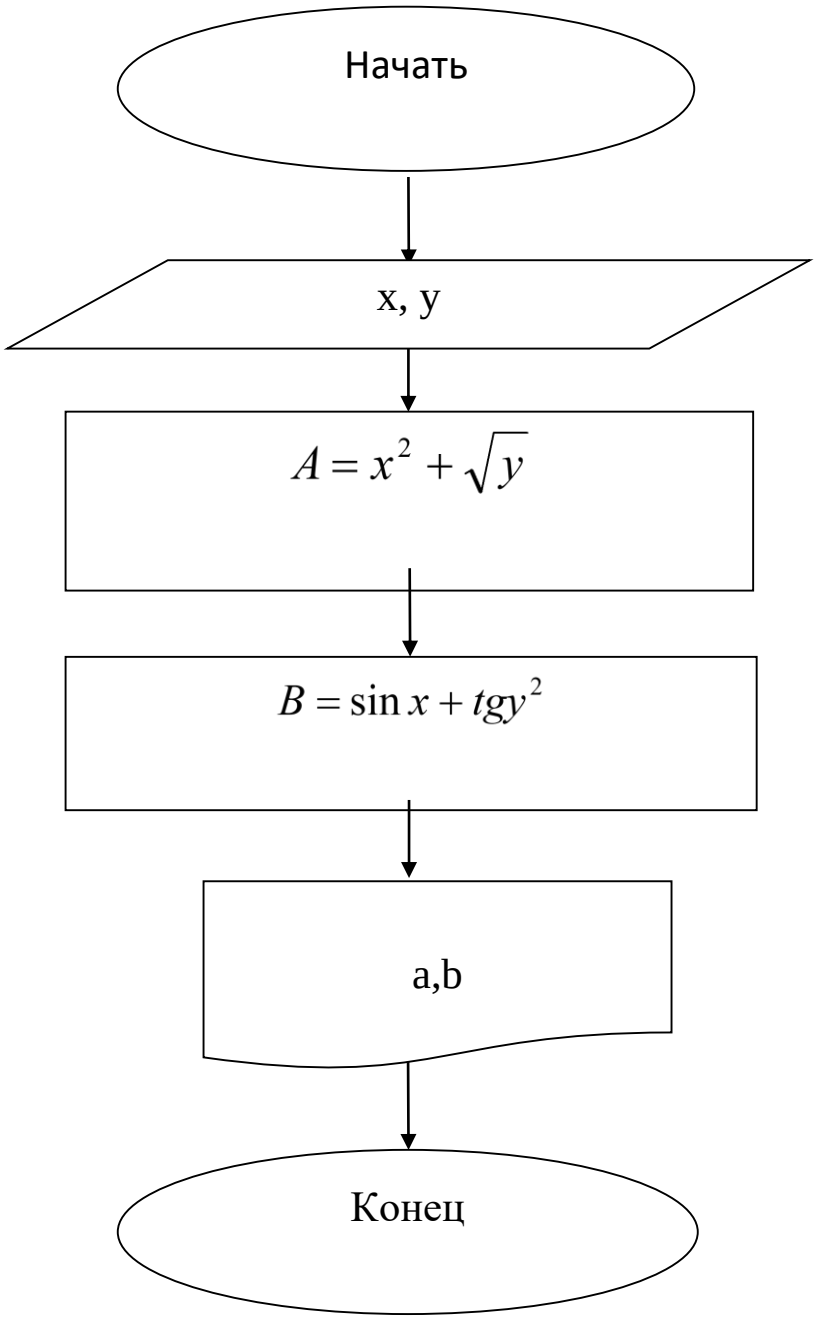


Рис. 4.5. Схема программного блока.

Вам понадобятся 4 компонента Labels, 2 Edit и 2 Button для программирования вашего примера в визуальной среде.

После того, как необходимые компоненты установлены в окне формы, внешний вид программы будет выглядеть так:

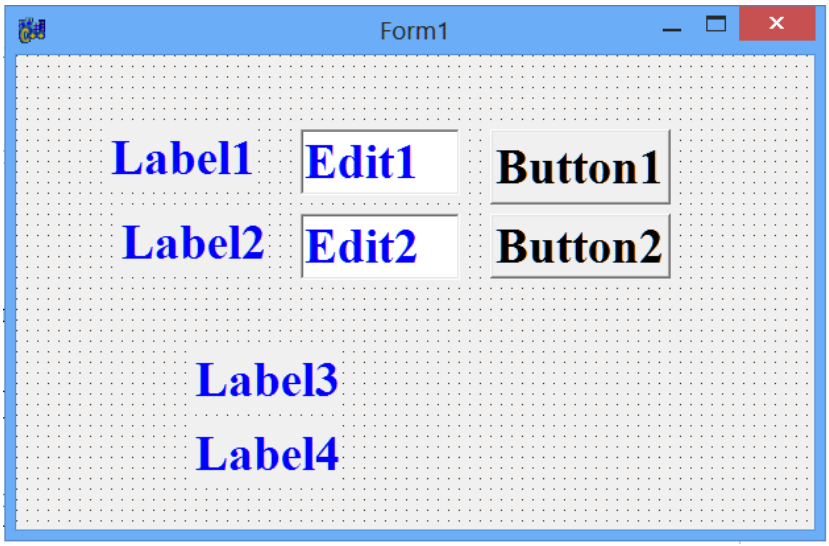


Рис. 4.6. Просмотр приложения.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 4.1. Ввод свойств компонента

Имя компонента	Имя свойства (Состояние окна Object Inspector)	Процесс, который должен быть выполнен
Form1	Caption (Properties)	Вводится слово “Программирование линейного процесса”.
Label1	Caption (Properties)	Вводится слово “значение X”
Label2	Caption (Properties)	Вводится слово “значение Y”
Label3	Caption (Properties)	Слово “A =” вводится.
Label4	Caption (Properties)	Слово “B =” вводится.
Edit1	Text (Properties)	Удалить слово “Edit1”.
Edit2	Text (Properties)	Удалить слово “Edit2”.
Button1	Caption (Properties)	Вводится слово “Результат”
	OnClick (Events)	Введется текст программы.
Button2	Caption (Properties)	Введется слово “Выход”
	OnClick (Events)	Введётся Close();

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

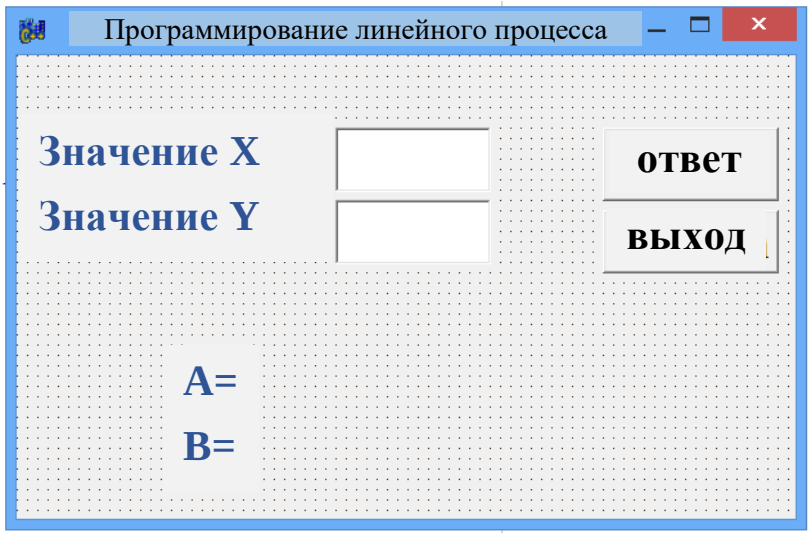


Рис. 4.7. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------

#include <vcl.h>

 #include <math.h>

#pragma hdrstop

#include "Unit1.h"


//---------------------------------------------------------

#pragma package(smart_init)

#pragma resource "*.dfm"

TForm1 *Form1;

//---------------------------------------------------------
```

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}


//---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
float  a,b,x,y;
 x=StrToFloat(Edit1->Text);
 y=StrToFloat(Edit2->Text);
 a=pow(x,2)+sqrt(y);
  b=sin(x)+tan(pow(y,2));
  Label3->Caption= "A="+FloatToStrF(a,ffFixed,6,4);
   Label4->Caption= "B="+FloatToStrF(b,ffFixed,6,4);
}
//---------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
Close();
}
//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

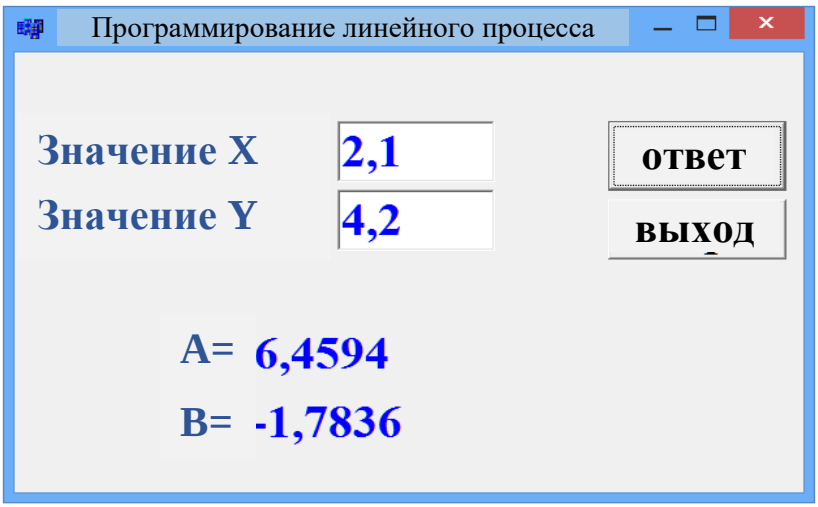


Рис. 4.8. Окно результатов.

Пример 2. Мы рассмотрим, как реализовать программу вычисления цепей в визуальной среде, состоящей из вышеуказанного сопротивления. Мы создадим программу расчета сопротивления цепи.

Вам понадобятся 4 компонента Labels, 2 Edit и 2 Button для программирования вашего примера в визуальной среде.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 4.2. Ввод свойств компонента

Имя компонента	Имя свойства (Состояние окна Object Inspector)	Процесс, который должен быть выполнен
Form1	Caption (Properties)	Вводится слово “Программирование линейного процесса”.
Label1	Caption (Properties)	Вводится слово “ Введите R1 ”
Label2	Caption (Properties)	Вводится слово “ Введите R1 ”
Label3	Caption (Properties)	Вводится слово “последовательное сопротивление цепи =”
Label4	Caption (Properties)	Вводится слово “параллельное сопротивление =”.

Продолжение Таб. 4.2

Edit1	Text (Properties)	Удалить слово “Edit1”.
Edit2	Text (Properties)	Удалить слово “Edit2”.
Button1	Caption (Properties)	Введется слово “результат”.
	OnClick (Events)	Введется текст программы.
Button2	Caption (Properties)	Введется слово “Выход”
	OnClick (Events)	Введётся Close();

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

Рис. 4.9. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------

#include <vcl.h>

 #include <math.h>

#pragma hdrstop

#include "Unit1.h"
```

```
//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
float  a,b,x,y;
 x=StrToFloat(Edit1->Text);
 y=StrToFloat(Edit2->Text);
 a=pow(x,2)+sqrt(y);
  b=sin(x)+tan(pow(y,2));
  Label3->Caption= "A="+FloatToStrF(a,ffFixed,6,4);
   Label4->Caption= "B="+FloatToStrF(b,ffFixed,6,4);
}
//---------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
```

{

Close();

}

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

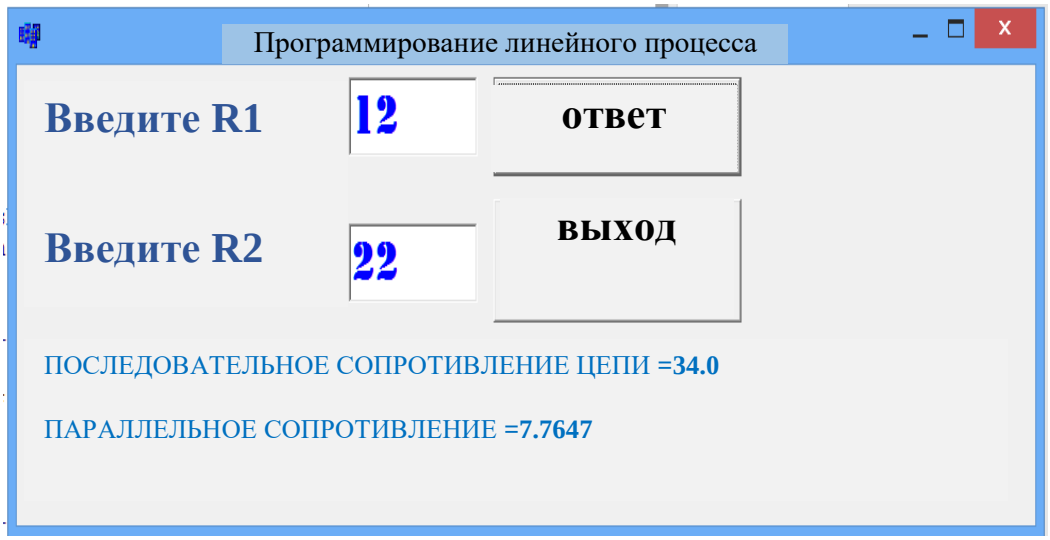


Рис. 4.10. Окно результатов.

Вопросы по главе 4

1. Что такое «линейный процесс»?

2. Опишите процесс приложения в консольном приложении Borland C ++ Builder 6.

3. Какие компоненты используются в приложении Form в Borland C ++ Builder 6?

Тестовые вопросы:

1. Укажите правую объявленную строку переменных.

а) Xx,y,a,b float;

б) x,y,a,b int;

в) integer x,y,a,b;

г) int x,y,a,b;

2. Введите правильную объявленную строку.

а) a= StrToFloat(Edit1->Caption);

б) a:= StrToFloat(Edit1.Text);

в) a= StrToFloat(Edit1->Text);

г) b= strtofloat(Edit1->Text);

3. Пожалуйста, укажите правильную позицию?

а) Label5.Caption: = "Y="+FloatToStr(y);

б) Label1->Caption = "Y="+FloatToStr(y);

в) Label5->Text = "Y="+FloatToStr(y);

г) Label5->Add = "Y="+FloatToStr(y);

4. Какой компонент используется для отображения результата в визуальной среде?

а) Label;

б) MainMenu;

в) Button;

г) BitBtn;

5. Какая библиотека используется для использования стандартных математических функций?

а) # include <math.h>;

б) # include <vcl.h>;

в) # include <fstream.h>;

г) # include < conio.h>;

Глава 5. ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЯЮЩИХСЯ ПРОЦЕССОВ В СИСТЕМЕ ПРОГРАММИРОВАНИЯ C++ BUILDER 6

Ключевые слова: ветвящийся процесс, оператор условного перехода, оператор выбора, оператор безусловного перехода, логические выражения, символы оператора, простые и сложные сетевые процессы.

Во многих случаях некоторые процессы выполняются в отношении определенных условий или условий, т.е. другие процессы или действия выбираются в зависимости от того, выполнено ли условие. Такие процессы называются «сетевыми процессами».

Сетевые вычисления могут быть простыми и сложными. Это зависит от количества сетей в процессе. В конкретном сетевом процессе может быть больше сетевых подключений. Вычислительные процессы в таких сетях называются сложными вычислительными процессами.

Безусловные, условные операторы перехода и выбора используются для программирования сетевых процессов на C ++.

5.1. Оператор безусловного перехода

В некоторых случаях программе может потребоваться передать управление непосредственно любому оператору, то есть прервать последовательность выполнения. Это делается с помощью оператора безусловного перехода.

Общий вид оператора безусловного перехода выглядит следующим образом:

goto <символ оператора>;

Символ оператора здесь является символом передаваемого оператора. Символы могут использоваться между 0-9999, натуральными числами, символами (символами) и смесью символов.

Например:

int matrix [n][m];

```
int value;
....
for (int i=0; i<n; i++)
for (int j=0; j<m; j++)
if (matrix [i][j]==value)
{pintf ("value %d found in cell (%d, %d)\n", value,i,j);
goto end_loop;
}
printf("value %d not found \n", value);
end_loop: ;
```

Оператор будет иметь «:» между символом и оператором. Неправильное применение этого оператора может помешать выполнению программы. Поэтому желательно, чтобы в программе было меньше операторов.

5.2. Оператор условного перехода

В программе управление конкретной сетью (известная последовательность процессов) обеспечивается оператором условного переключения. Оператор условной прокрутки можно использовать двумя способами: полный и короткий.

Давайте посмотрим на полный вид оператора условного перехода. Его появление в программе выглядит следующим образом:

if (<mantiqiy ifoda>) <operator –1>; else <operator–2>;

здесь: if-если, else- иначе используемые слова, оператор-1 и оператор -2 являются необязательными операторами.

Логическое выражение в операторе указывает условие для передачи управления.

Режим оператора выглядит следующим образом: если заданное логическое выражение истинно, то есть если условие выполняется; выполняется 1 оператор, в противном случае выполняется 2 оператор else .

Отношения, логические действия могут быть использованы вместо логических выражений. Например,

a>b, a=b, x<4. 55, 2+z>0, x+y<=1 va h. k.

Условия могут быть простыми и сложными.

Если логическое выражение задается одним отношением, оно представляет «нормальное состояние».

Термины для взрослых называются «Сложные условия», когда они состоят из нескольких отношений, связанных с признаками «и», «или», «не» (в C ++ &&, || ,!). Иконки отношений показаны в Таб 5.1.

Таб. 5.1. Иконки отношений

Знак действия отношения	Название	Вид в примере
==	равный	2=2;
!=	Не равно	5!=4;
<	маленький	2<3;
>	большой	5>4;
>=	Большой или равный	1<=Z;
<=	Маленький или равный	1>=Z;

Например, сложные логические выражения (условия) записываются в программе:

- 1 <x <= 4 записывается как логическое выражение (x> 1 && x <= 4).

- логическое выражение a = b = 0 (a = 0 && b = 0);

- 6 <= x <10 записывается как логическое выражение (x> = 6 && x <10).

Примеры включают использование оператора условного перехода.

1) if (u>0) d=sqrt(y); else d=u;

Если условный оператор равен u> 0, оператор d = sqrt (y), в противном случае d = u.

2) if (x=0 && x>0) x=sqrt (x); else x=pow(x,2);

В результате этого оператора, если x равен 0 и положителен, его значение вычитается из корня, в противном случае возводится в квадрат.

Пример 1 Создайте программу для определения значения функции y. Выполните в консольной среде.

$$y=\begin{cases}\dfrac{4r+3m^2}{r-m}, & agar \quad r \geq m+1 \\ |r-m|, & agar \quad r < m+1\end{cases}$$

Текст программы в среде консоли выглядит так:

```
//---------------------------------------------------------

 #include <stdio.h>

#include<iostream.h>

#include<conio.h>

#include <math.h>

#include <vcl.h>

#pragma hdrstop


//---------------------------------------------------------
```

```
#pragma argsused
int main(int argc, char* argv[])
{
float  y, r, m;
cout<<" Введите значение r "<<"\n"; cin>>r;
cout<<" Введите значение m "<<"\n"; cin>>m;
   if (r>=m+1)
        {
y=(4*r+3*pow(m,2))/(r-m);
cout<<" Функция определяется при условии 1, y="<<y;
}
else
{
y=fabs(r-m);
     cout<<" Функция определяется при условии 1, y="<<y;
   }
getch();
 return 0;
 }
//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

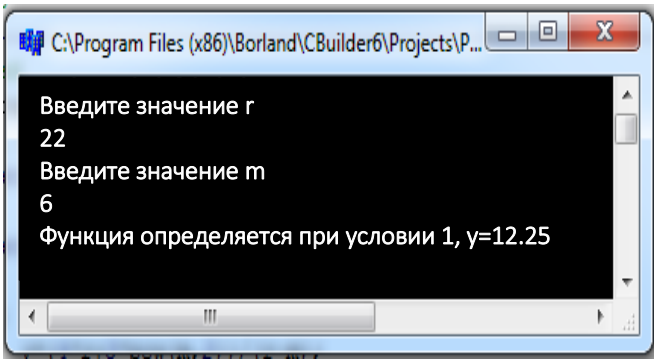


Рис. 5.1. Окно результатов.

Пример 2. Создайте программу для определения значения функции y. Выполните в визуальной среде.

$$y = \begin{cases} 3x^2 - 3abx \quad agar \quad a > 0 \\ 13a - b^2 x \quad agar \quad a \le 0 \end{cases}$$

Алгоритм решения этого примера следующий:

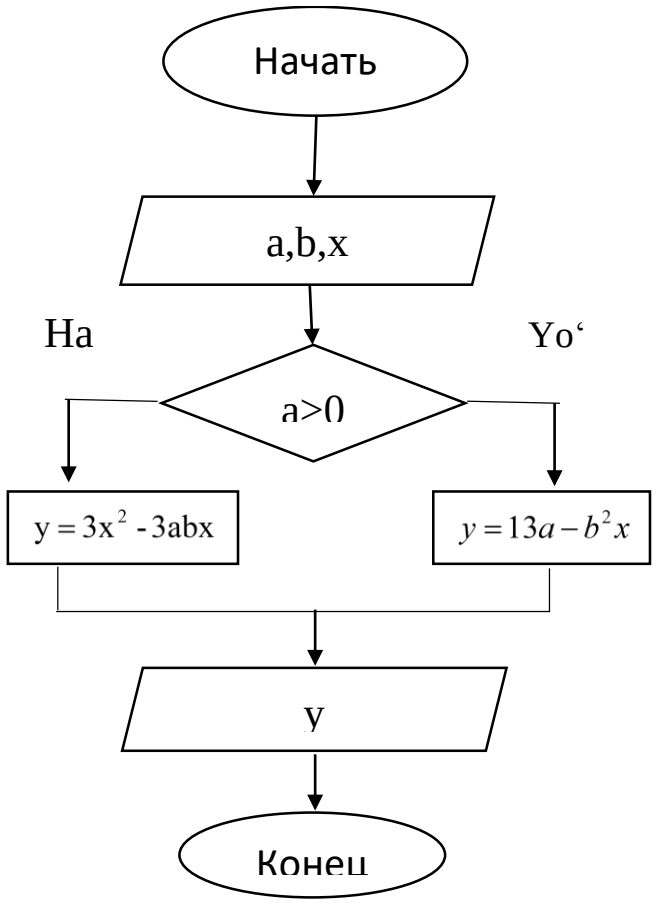


Рис. 5.2. Схема программного блока.

Для программирования в визуальной среде требуются 4 компонента Label, 2 Edit, 2 BitBtn и RadioGroup.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 5.2. Ввод свойств компонента

Имя компонента	Название недвижимости (Состояние окна Object Inspector)	Процесс должен быть реализован
Form1	Caption (Properties)	Вводится слово “сетевой процесс” вводится.
Label1	Caption (Properties)	Вводится слово “Введите значение A”.
Label2	Caption (Properties)	Вводится слово “Введите значение B”.
Label3	Caption (Properties)	Вводится слово “Введите значение X” вводится.
Label4	Caption (Properties)	Вводится слово “Y=”.
Edit1	Text (Properties)	Удалите слово “Edit1”.
Edit2	Text (Properties)	Удалите слово “Edit2”.
BitBtn1	Kind (Properties)	Выбирается свойство “bkOK”.
	Caption (Properties)	Вводится слово “Результат”.
	OnClick (Events)	Текст программы введен.
BitBtn2	Kind (Properties)	Выбирается свойство “bkClose”.
	Caption(Properties)	Выбирается слово “Выход”.
	OnClick (Events)	Вводится Close();
RadioGroup1	Caption (Properties)	Вводится слово “Проверка состояния”.
	Items (Properties)	Слово “A> 0 встречается, а A = <0 встречается”.

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

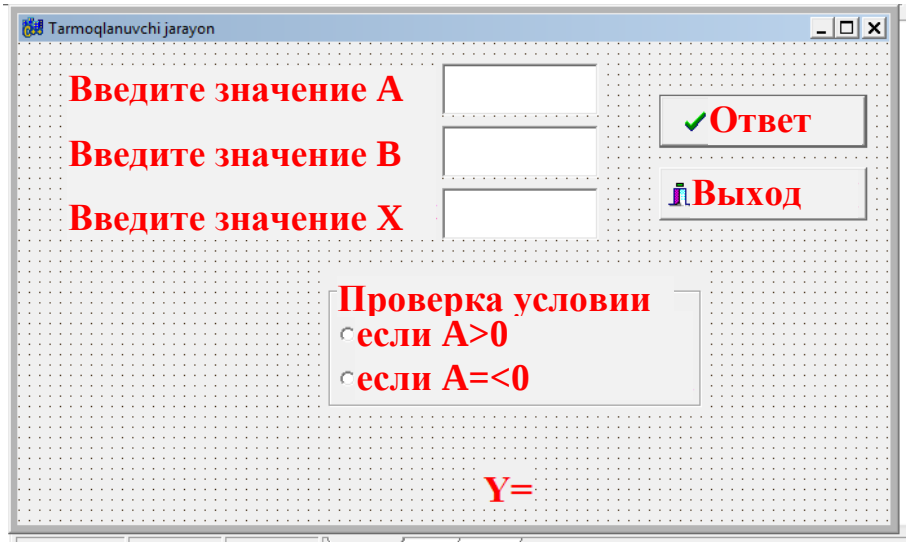


Рис. 5.3. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------
#include <vcl.h>
 #include <math.h>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
```

```
{

}


//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
float  y,x,a,b;
a=StrToFloat(Edit1->Text);
b=StrToFloat(Edit2->Text);
x=StrToFloat(Edit3->Text);
if (a>0)
{
   y= 3*pow(x,2)-3*a*b*x;
   RadioGroup1->ItemIndex=0;
 Label4->Caption ="y="+FloatToStrF(y,ffFixed,6,4);
        }
   else
{
   y=13*a-pow(b,2)*x;
   RadioGroup1->ItemIndex=1;
 Label4->Caption ="y="+FloatToStrF(y,ffFixed,6,4);
}
```

```
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
Close();
}
//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

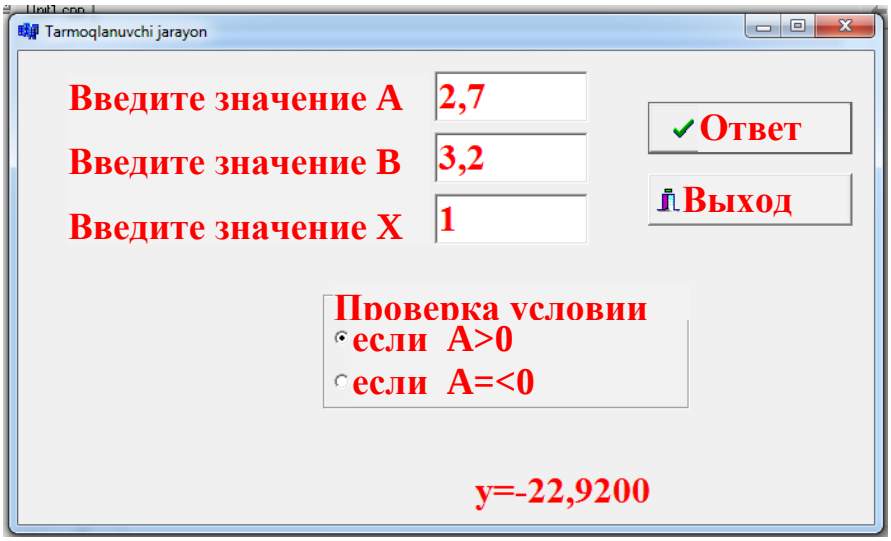


Рис. 5.4. Окно результатов.

В некоторых алгоритмах это иногда имеет место, когда некоторые вычисления выполняются в процессе вычисления. В противном случае никаких действий предприниматься не будет. В этом случае оператор условного перехода можно кратко суммировать. Его написание выглядит следующим образом:

if (<mantiqiy ifoda>) <operator>;

Процедура для оператора следующая: если логическое выражение принимает значение TRUE (истина), оператор выполняется, иначе выполняется оператор, который стоит после IF;

Например:

if (x<0) t=x*x;

Вместо оператора в метаданных оператора условного оператора можно использовать полное и краткое представление оператора условного перехода. Например,

1) if (b1) {if (b2) a};

Здесь: b1, b2- логическое выражение, a – оператор.

В результате этого оператор b1 является значением логического выражения, если TRUE примет значения, b2 является значением логического выражения, если и это будет истиной выполняется оператор a. Если логические выражения b1 или b2 являются ложными (false), условный оператор выполняется после условного оператора.

Пример 3. Если переменная «a» является отрицательным числом $y = \sin(a) + \cos(b)$ создает программу значения функции. Выполните в консольной среде.

Текст программы будет выглядеть так:

```
//---------------------------------------------------------

 #include <stdio.h>

#include<iostream.h>

#include<conio.h>

#include <math.h>

#include <vcl.h>
```

```
#pragma hdrstop
//---------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{  float  y, a, b;
cout<<" Введите значение a "<<"\n";
cin>>a;
cout<<" Введите значение b"<<"\n";
cin>>b;
   if (a<0)
        { y=sin(a)+cos(b);
        cout<<" Значение a является отрицательным, y="<<y; }
    getch();
       return 0;  }
//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

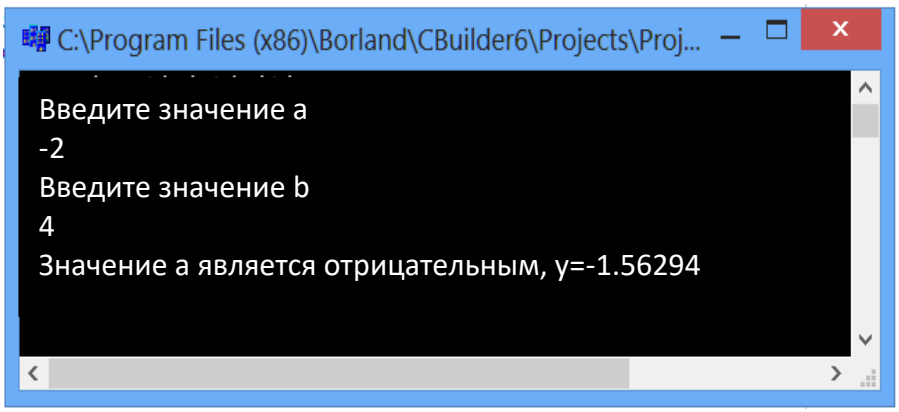


Рис. 5.5. Окно результатов.

Пример 4. Если переменная «a» является положительным целым числом, создайте программу для определения значения функции. Реализуйте программу в визуальной среде.

Для программирования примера в визуальной среде вам понадобятся 3 компонента Labels, 2 Edit, 2 BitBtn.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 5.3. Ввод свойств компонента

Имя компонента	Название недвижимости (Состояние окна Object Inspector)	Процесс должен быть реализован
Form1	Caption (Properties)	Вводится слово “if оператор”.
	Color (Properties)	Выбрано “clAppWorkSpace”.
Label1	Caption (Properties)	Вводится слово “Введите значение A”.
Label2	Caption (Properties)	Вводится слово “Введите значение B”.
Label3	Caption (Properties)	Введется слово “Label3”.
Edit1	Text (Properties)	Удалите слово «Edit1».
Edit2	Text (Properties)	Удалите слово «Edit1».
BitBtn1	Kind (Properties)	Выбирается свойство “bkOK”.
	Caption (Properties)	Вводится слово “Результат”.
	OnClick (Events)	Текст программы введен.
BitBtn2	Kind (Properties)	Выбирается свойство “bkClose”.
	Caption (Properties)	Вводится слово “Выход”.
	OnClick (Events)	Вводится Close();

Мы добавим необходимые компоненты в последовательности. После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

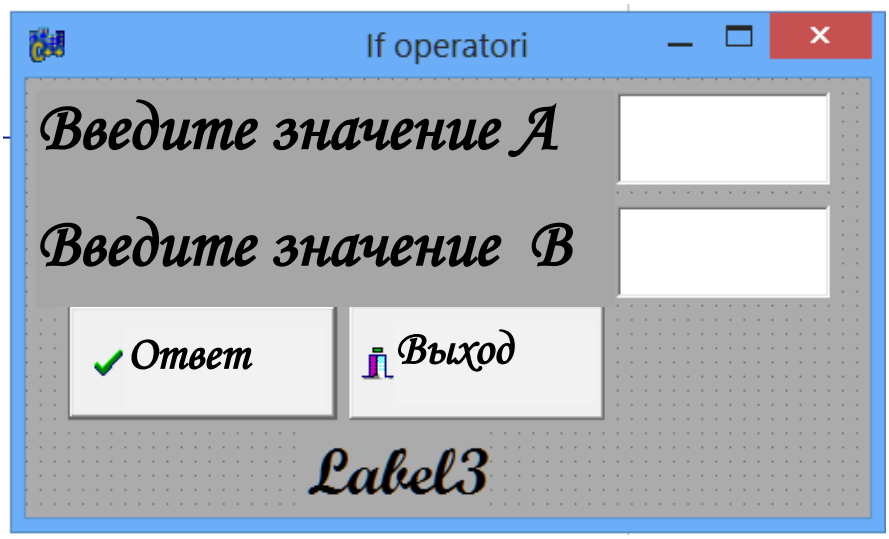


Рис. 5.6. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------

#include <vcl.h>

 #include <math.h>

#pragma hdrstop

#include "Unit1.h"


//---------------------------------------------------------

#pragma package(smart_init)

#pragma resource "*.dfm"

TForm1 *Form1;


//---------------------------------------------------------
```

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}


//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
float  y,x,a,b;
a=StrToFloat(Edit1->Text);
b=StrToFloat(Edit2->Text);
if (a<0)
{
y= y= pow(a,2)+pow(b,2);
Label3->Caption ="значения а отрицательное y="+ FloatToStrF (y, ffFixed,
6,4);
}
}
//---------------------------------------------------------


void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
Close();
```

```
}
//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

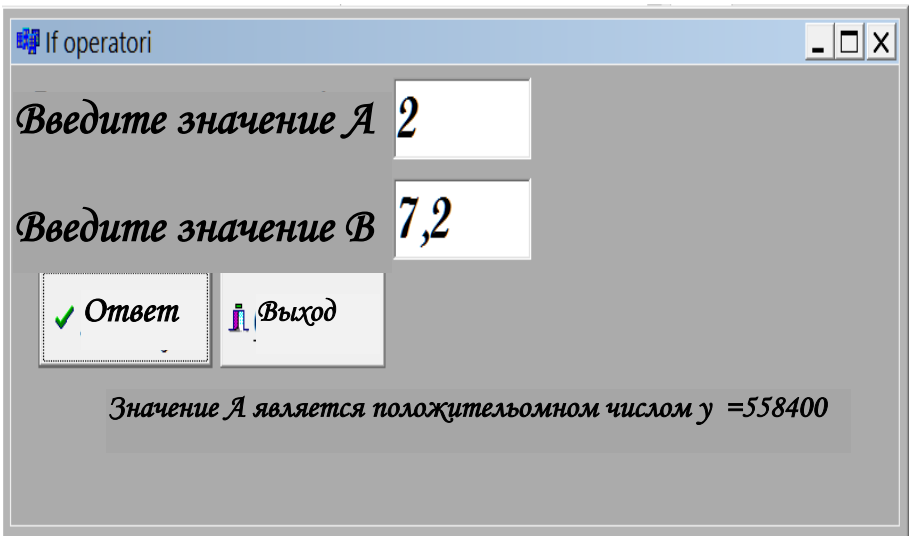


Рис. 5.7. Окно результатов.

5.3. Оператор варианта

В большинстве сетевых процессов сеть разделяется на две или более сетей. В общем, это можно сделать с помощью оператора условного перехода, который мы знаем:

```
if (b1) a1; else
        if (b2) a2; else
                .. .. .. .. .. .. .. .. .. ..
                if (bk) ak;
```

Но в этом случае написание операторов условного перехода неудобно.

В большинстве случаев программисту легко использовать обобщенное представление условного оператора - оператора выбора. Общий вид оператора выбора выглядит следующим образом:

switch (ifoda yoki o‘zgaruvchi – selektor)

{

case <1 значение >: <1- оператор(ы)>; break;

case < 2 значение >: < оператор(ы)>; break;

....

case < n значение> :< оператор(ы)>; break;

default: < в противном случае оператор (ы>;

}

здесь: switch (выбрать или переключить) является служебным словом.

Switch - оператор выполняет сетевой процесс, выбирая одного из нескольких операторов. Все операторы в операторе выбора явно перечислены, включая оператор, выбранный для выполнения (последовательность данных операторов ограничена). Оператор или последовательность операторов, которая должна быть выполнена, определяется значением селектора оператора. Если опция недоступна, будет выполнен оператор по умолчанию. Оператор break используется для выхода и выхода из каждого экземпляра, а вместо него можно использовать оператор return.

Селектор оператора может использоваться с любым недопустимым скалярным выражением или переменной. В операции оператора каждый оператор, содержащийся в нем, представлен так называемым «знаком выбора». Этот символ представляет собой совокупность, соответствующую описанию селектора, которая принимает специфическое для селектора значение, необходимое для выполнения оператором. Чтобы оператор мог

работать с несколькими доступными значениями, необходим список критериев выбора.

Значение селектора изначально рассчитывается оператором. Затем за указанным оператором, который соответствует значению селектора, следует оператор, который соответствует значению после слова «case». Если в последовательности операторов такой оператор не найден, в программе записывается ошибка. Поэтому программа должна состоять из специального оператора или оператора, который соответствует значению селектора во время выполнения. При этом символы, назначенные оператору выбора, не объявляются.

Выполнение оператора выбора приводит к выполнению одного оператора в последовательности операторов, содержащихся в нем.

Например:

```
#include <iostream.h>
#include<conio.h>
#include <vcl.h>
#pragma hdrstop

//---------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
int оценка;
cin>> оценка;
switch(оценка)
```

```
{
 case 2: cout <<"\n неудовлетворительный "; break;
 case 3: cout <<"\n  удовлетворительный "; break;
 case 4: cout <<"\n  хорошо" ;break;
 case 5: cout <<"\n  отлично"; break;
 default: cout <<"\n оценка Введеноо неправильно ";
}        getch();
        return 0;
 }
//------------------------------------------------------------
```

В результате этого оператора, если Введена «оценка» - значение переменной - его значение сравнивается со значениями, заданными в опциях регистра. Если значение равно 2 «Плохо», 3 - «Удовлетворительно» и т.д. слова отображаются на экране, в противном случае отображается «неправильный рейтинг».

Пример 5 Разработайте программу выравнивания функций, основанную на количестве клавиатур. Выполните в консольной среде.

Текст программы будет выглядеть так:

```
//------------------------------------------------------------
#include <iostream.h>
#include <math.h>
#include<conio.h>
#include <vcl.h>
#pragma hdrstop
```

```
//---------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
int n,y;
cout<<"введите значение n"<<"\n";
cin>> n;
switch(n)
{
 case 2: y=pow (2,2); cout <<"y="<<y; break;
 case 3: y=pow (2,3); cout <<"y="<<y; break;
 case 4: y=pow (2,4); cout <<"y="<<y; break;
 case 5: y=pow (2,5); cout <<"y="<<y; break;
 case 6: y=pow (2,6); cout <<"y="<<y; break;
 case 7: y=pow (2,7); cout <<"y="<<y; break;
 case 8: y=pow (2,8); cout <<"y="<<y; break;
 case 9: y=pow (2,9); cout <<"y="<<y; break;
 case 10: y=pow (2,10); cout <<"y="<<y; break;
 default: cout <<"n Введено неверно";
  }
getch();
```

```
    return 0;
}
//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

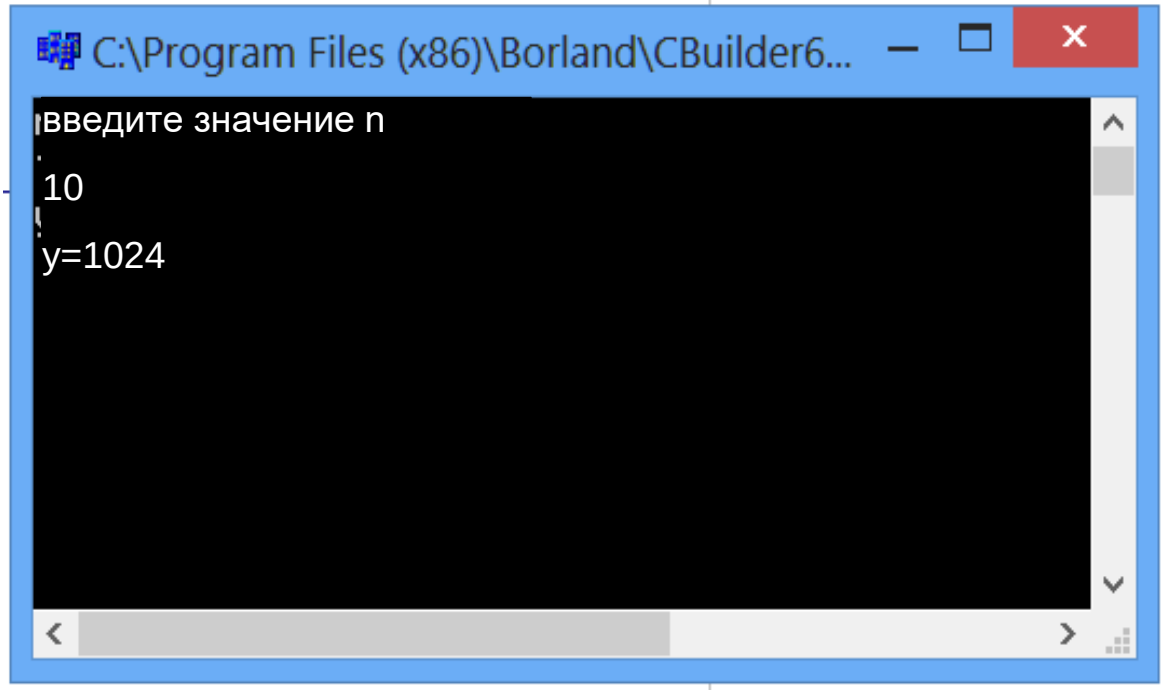


Рис. 5.8. Окно результатов.

Пример 6. Создайте программу для указания нечетного или четного числа. Реализуйте программу в визуальной среде.

Вам понадобятся 3 компонента Labels, 1 Edit, 2 BitBtn для программирования вашего примера в визуальной среде.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 5.4. Ввод свойств компонента

Имя компонента	Имя свойства (Состояние окна Object Inspector)	Процесс, который должен быть выполнен
Form1	Caption (Properties)	Вводится слово “Оператор выбора”.
Label1	Caption (Properties)	Вводится слово “Введите цифру”.
Label2	Caption (Properties)	Вводится слово “Введеноная цифра:”.
Label3	Caption (Properties)	Удалите слово “Label3”.
Edit1	Text (Properties)	Удалите слово “Edit1”.
BitBtn1	Kind (Properties)	Выбирается свойство “bkOK”.
	Caption (Properties)	Вводится слово “Результат”.
	OnClick (Events)	Текст программы введен.
BitBtn2	Kind (Properties)	Выбирается свойство “bkClose”.
	Caption (Properties)	Вводится слово “Выход”.
	OnClick (Events)	Вводится Close();.

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

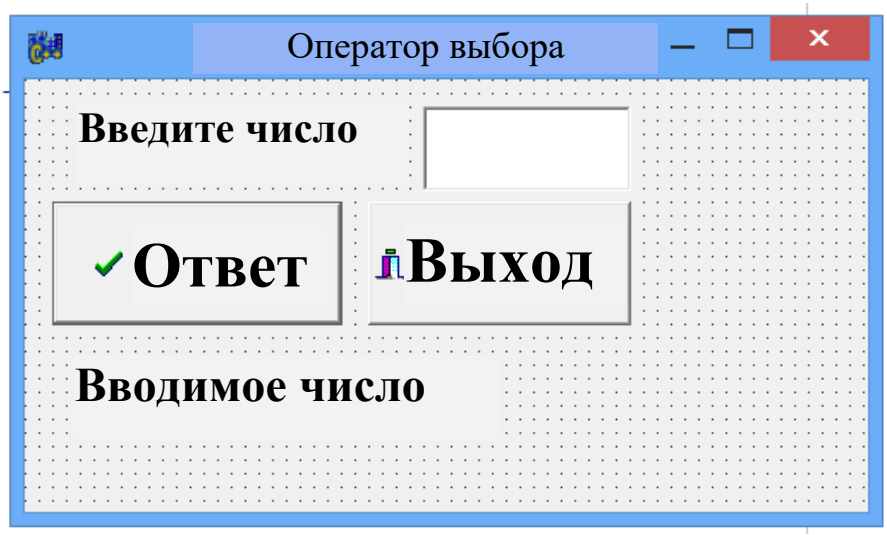


Рис. 5.9. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{ }
//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{ int n;
n=StrToFloat(Edit1->Text);
switch (n)
{ case 1:
case 3:
case 5:
case 7:
case 9: Label3->Caption=("Toq son"); break;
```

```
case 2:
case 4:
case 6:
case 8:
case 10: Label3->Caption=(" четное число "); break;
default: Label3->Caption=("Введите цифры от 1 до 10!"); }}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{Close();       }
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

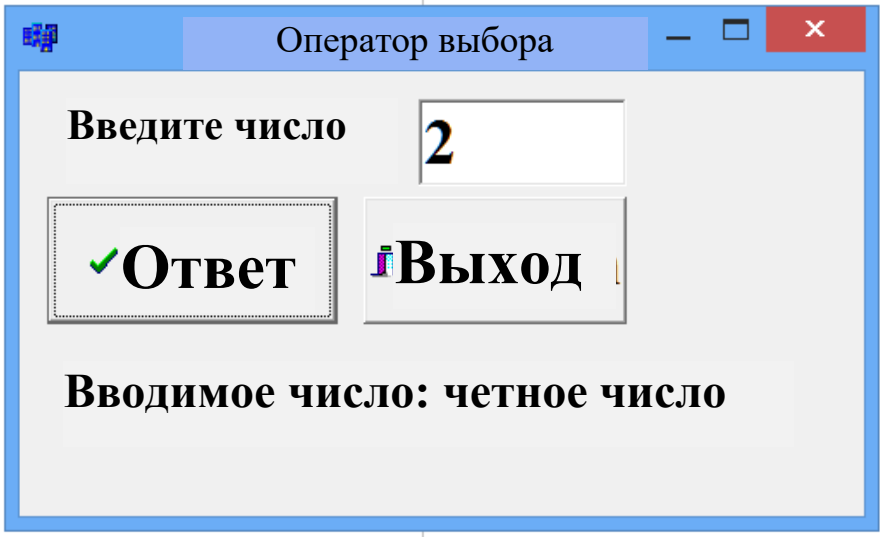


Рис. 5.10. Окно результатов.

Пример 7. Создать программу-калькулятор. Реализуйте программу в визуальной среде.

Для программирования в визуальной среде требуется 1 компонент Edit и 16 компонентов BitBtn.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 5.5. Ввод свойств компонента

Имя компонента	Имя свойства (Состояние окна Object Inspector)	Процесс, который должен быть выполнен
Form1	Caption (Properties)	Вводится слово “Calc”.
Edit1	Text (Properties)	Удалите слово “Edit1”.
BitBtn1	Caption (Properties)	Вводится слово “1”.
	OnClick (Events)	Текст программы введен.
BitBtn2	Caption (Properties)	Вводится слово “2”.
	OnClick (Events)	Текст программы введен.
BitBtn3	Caption (Properties)	Вводится слово “3”.
	OnClick (Events)	Текст программы введен.
BitBtn4	Caption (Properties)	Вводится слово “4”.
	OnClick (Events)	Текст программы введен.
BitBtn5	Caption (Properties)	Вводится слово “5”.
	OnClick (Events)	Текст программы введен.
BitBtn6	Caption (Properties)	Вводится слово “6”.
	OnClick (Events)	Текст программы введен.
BitBtn7	Caption (Properties)	Вводится слово “7”.
	OnClick (Events)	Текст программы введен.
BitBtn8	Caption (Properties)	Вводится слово “8”.
	OnClick (Events)	Текст программы введен.
BitBtn9	Caption (Properties)	Вводится слово “9”.
	OnClick (Events)	Текст программы введен.
BitBtn10	Caption (Properties)	Вводится слово “0”.
	OnClick (Events)	Текст программы введен.
BitBtn11	Caption (Properties)	Вводится слово “+”.
	OnClick (Events)	Текст программы введен.
BitBtn12	Caption (Properties)	Вводится слово “-”.
	OnClick (Events)	Текст программы введен.
BitBtn13	Caption (Properties)	Вводится слово “*”.
	OnClick (Events)	Текст программы введен.
BitBtn14	Caption (Properties)	Вводится слово “/”.
	OnClick (Events)	Текст программы введен.
BitBtn15	Caption (Properties)	Вводится слово “C”.
	OnClick (Events)	Текст программы введен.
BitBtn16	Caption (Properties)	Вводится слово “=”.
	OnClick (Events)	Текст программы введен.

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

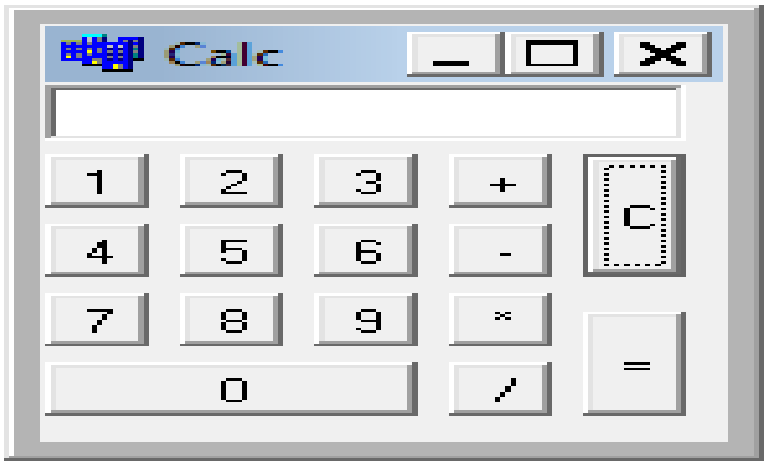


Рис. 5.11. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------
#include <vcl.h>
#include <math.h>
#pragma hdrstop
#include "Unit1.h"

//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//
```

```
float a,b,c;
char k;

//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}

//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
Edit1->Text=Edit1->Text+"1";
b=StrToFloat(Edit1->Text);
}

//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
Edit1->Text=Edit1->Text+"2";
b=StrToFloat(Edit1->Text);
 }
```

```
//---------------------------------------------------------
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
Edit1->Text=Edit1->Text+"3";
b=StrToFloat(Edit1->Text);
 }

//---------------------------------------------------------
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{
Edit1->Text=Edit1->Text+"4";
b=StrToFloat(Edit1->Text);
}

//---------------------------------------------------------
void __fastcall TForm1::BitBtn5Click(TObject *Sender)
{
Edit1->Text=Edit1->Text+"5";
b=StrToFloat(Edit1->Text);
}
```

```
//---------------------------------------------------------
void __fastcall TForm1::BitBtn6Click(TObject *Sender)
{
Edit1->Text=Edit1->Text+"6";
b=StrToFloat(Edit1->Text);
}

//---------------------------------------------------------
void __fastcall TForm1::BitBtn7Click(TObject *Sender)
{
Edit1->Text=Edit1->Text+"7";
b=StrToFloat(Edit1->Text);
}

//---------------------------------------------------------
void __fastcall TForm1::BitBtn8Click(TObject *Sender)
{
Edit1->Text=Edit1->Text+"8";
b=StrToFloat(Edit1->Text);
}

//---------------------------------------------------------
```

```
void __fastcall TForm1::BitBtn9Click(TObject *Sender)
{
Edit1->Text=Edit1->Text+"9";
b=StrToFloat(Edit1->Text);
}

//---------------------------------------------------------
void __fastcall TForm1::BitBtn10Click(TObject *Sender)
{
Edit1->Text=Edit1->Text+"0";
b=StrToFloat(Edit1->Text);
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn11Click(TObject *Sender)
{
a=StrToFloat(Edit1->Text);
k='+';
Edit1->Text="";
}

//---------------------------------------------------------
void __fastcall TForm1::BitBtn12Click(TObject *Sender)
```

```
{
a=StrToFloat(Edit1->Text);
k='-';
Edit1->Text="";
}

//---------------------------------------------------------
void __fastcall TForm1::BitBtn13Click(TObject *Sender)
{
a=StrToFloat(Edit1->Text);
k='*';
Edit1->Text="";
}

//---------------------------------------------------------
void __fastcall TForm1::BitBtn14Click(TObject *Sender)
{
a=StrToFloat(Edit1->Text);
k='/';
Edit1->Text="";
}
```

```
//---------------------------------------------------------
void __fastcall TForm1::BitBtn16Click(TObject *Sender)
{
switch(k)
{
   case '+':a=a+b;
            Edit1->Text=FloatToStr(a);
break;
   case '-':a=a-b;
            Edit1->Text=FloatToStr(a);
break;
   case '*':a=a*b;
            Edit1->Text=FloatToStr(a);
break;
   case '/':a=a/b;
}
}

//---------------------------------------------------------
void __fastcall TForm1::BitBtn15Click(TObject *Sender)
{
Edit1->Text="";
```

}

//---

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

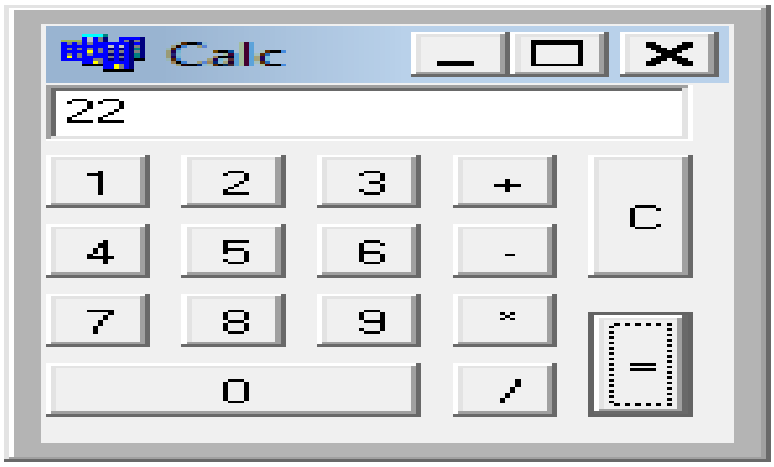


Рис. 5.12. Окно результатов.

Вопросы по главе 5

1. Опишите сетевой вычислительный процесс.

2. Оператор безусловного перехода и его использование.

3. Полный обзор оператора условного перехода.

4. Краткий обзор оператора условного перехода.

5. Выбор оператора и его применение в программе.

Тестовые вопросы:

1. Выберите правильную строку для типов ветвления C ++.

а) условный переход, безусловный, параметр;

б) условный переход, безусловный, обязательный;

в) условный переход, безусловный отбор;

г) условное прохождение, безусловный выезд;

2. Выберите поле, в котором показан общий вид оператора безусловного перехода.

а) Case <символ оператора>;

б) Goto <символ оператора>;

в) If <значок оператора>;

г) While <значок оператора>;

3. Пожалуйста, укажите правильную позицию?

а) if (x>0) { y =x*x*x; Label3->Caption=(“y=”+FloatTo Str(y)); }

б) x>0 if { y =x*x*x;label3->Caption->(“y=”+floattostr (y)); }

в) if x>0 { y =x*x*x; Label3->Caption->(“y=”+FloatTo Str(y)); }

г) if (x>0) { y =x*x*x; label3->Caption->(“y=”+Floatto Str(y)); }

4. Как 6 <= X <10 выражено в C ++?

а) (X<=6) and (X>10) ;

б) (X<=6) and (X<10) ;

в) (X>=6) and (X>10) ;

г) (X>=6) and (X<10) +;

5. Выберите правильную строку оператора безусловного перехода

а) case <1 –значение>: <1 – оператор(ы)>; break;

б) if (x=0) {y=x*x;}

в) goto 25; .. 25: y=x*x;

г) if U>0 D=sqrt (U); else D=U;

Глава 6. ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ ПРОЦЕССОВ В СИСТЕМЕ ПРОГРАММИРОВАНИЯ C++ BUILDER 6

Ключевые слова: итерационный процесс, операторы цикла, тело цикла, параметр цикла, прямое повторение, условное повторение, множественные циклы, процесс повторения параметра.

6.1. Организация циклических процессов

На практике сложные процессы программирования требуют многократного выполнения определенных команд на определенных условиях. Повторяющийся процесс (цикл) – это процесс повторения конкретной системы команд с разными значениями и разными значениями.

Повторяющаяся вычислительная часть повторяющегося процесса называется «Повторить тело».

Переменная со значением, которое повторяется в повторении, называется «Повторяющаяся переменная» или «Повторяющаяся переменная (параметр цикла)».

Итерационный алгоритм, как правило, должен включать:

1. Подготовка повторения - начальные значения переменных, участвующих в итерации, или значение заголовка переменной повторения устанавливаются до начала повторения, и указывается шаг переменной повторения.

2. Повторите тело - Повторите последовательность для различных значений переменных повторения.

3. Присвойте новое значение переменной повтора. Перед каждым повторением переменной присваивается новое значение в соответствии с шагом изменения.

4. Управление повторениями - отображается условие проверки повторения, начало повторения.

В алгоритмическом языке С ++ возможно создать повторяющийся вычислительный процесс в трех типах, и есть специальные операторы для программирования этих процессов:

- итеративный процесс повторения (предварительное условие «TOKI») выполняется специальным оператором while;

- Повторяемое условие («to ...»), за которым следует специальный оператор «Do while»;

- параметрическая итерация, которая выполняется специальным оператором For.

6.2. Программирование циклических процессов с предусловием

Этот тип повторяющегося процесса используется, когда количество повторений неизвестно, т.е. когда процесс повторения является специфическим. В этом процессе повторения условие повторения проверяется перед выполнением повторяющегося тела (рисунок 6.1).

Рис. 6.1. Алгоритм, для которого это в первую очередь проверить.

Общий вид этого оператора выглядит следующим образом:

<Условный предварительно проверенный оператор> (логическое выражение) <оператор>;

или в то время как (м) О;

где, в то время как - «m» является логическим выражением, «о» - это группа операторов или операторов, которые задают тело повторения. Тело повторения может иметь один или несколько операторов. В этом случае группа операторов должна быть записана между {и}.

Производительность оператора выглядит следующим образом:

«M» - это значение логического выражения. Если логическое выражение «m» истинно, выполняется «o», и оператор повторяется до тех пор, пока значение логического выражения «m» не станет ложным.

Если значение логического выражения «m» является ложным, когда первая проверка ложна, оператор «o» никогда не выполняется, и управление передается следующему оператору во время выполнения оператора.

Если «m» является логическим выражением и компьютеру по какой-то причине требуется выполнить «o» во время выполнения оператора «o», то он выполняется оператором break.

Пример 1. Пусть m будет действительным числом. Требуется, чтобы наименьшее целое число «k» было равно 3k <m.

Чтобы создать программу задачи, нам нужно добавить дополнительную переменную, которая хранит значение выражения 3k. Если мы установим это значение с идентификатором "y", то мы заменим k на формулу y = y * 3 при k = 0 и изменим его один шаг за шагом. В этом случае условие повторяющегося списания составляет y> m.

Используя вышеупомянутый оператор, мы создаем программу для этого примера в консольных средах.

```
//---------------------------------------------------------

#include <stdio.h>

#include<iostream.h>

#include<conio.h>

#include <math.h>

#include <vcl.h>

#pragma hdrstop
```

```
//---------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
float y, m; int k;
     y=1; k=0; m=27;
while (y<m)
   {
y=y*3; k=k+1;
     cout<< "k= " << k<< " y="<< y <<endl;
    }
getch();
 return 0;
}
```

После того, как Вводится текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

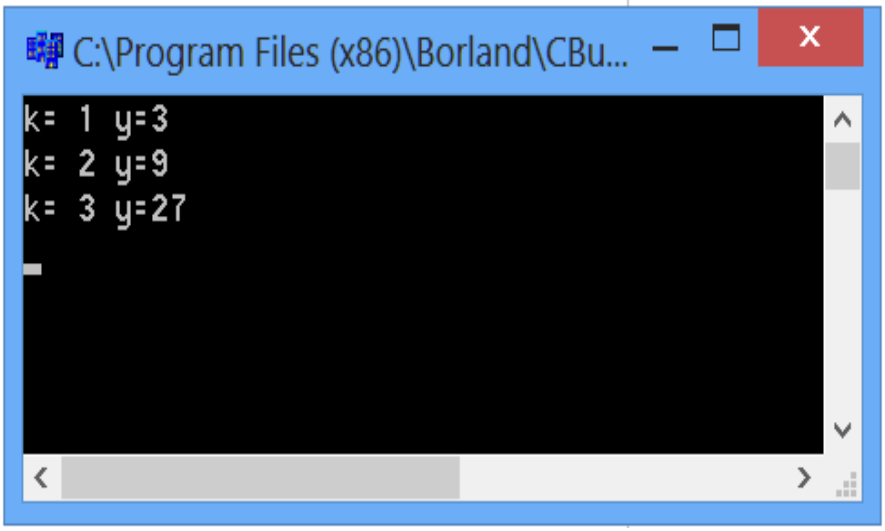


Рис. 6.2. Окно результатов.

Пример 2. Создайте программу для расчета значений f1 и f2. Пусть x изменяет значения переменной x с шага a по b на h.

f1= $\sqrt{x^3 + e^{-x}}$ va f2= $1 + 2\sin x$.

Вид программы в режиме консоли выглядит следующим образом:

```
//---------------------------------------------------
 #include <math.h>
#include<iostream.h>
#include<conio.h>
#include <vcl.h>
#pragma hdrstop


//---------------------------------------------------
int main(int argc, char* argv[])
{
```

```
float a, b, h, f1, f2, x;
   cout<<" Введите значения A, B, H" <<endl;
   cin>>a>>b>>h;
   x=a;
   while (x<=b)
   {
f1=sqrt(pow(x,2))*x+1*exp(-x);
     f2=1+2*sin(x);
    cout<<"x= "<<x<<" f1="<<f1<<"   f2="<<f2<< endl;
    x=x+h;
   }
getch();
return 0;
}
//---------------------------------------------------
```

После того, как Вводится текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

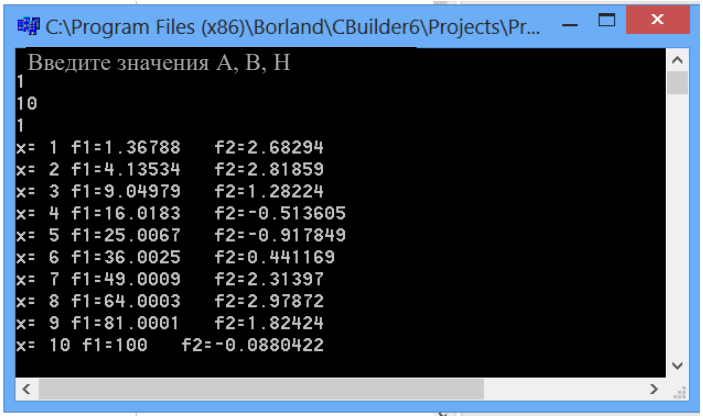


Рис. 6.3. Окно результатов.

Пример 3 Создайте программу для расчета значения функции $y = x^2 + \sin x + e^x$. Пусть x изменяет значения переменной x с шага a по b на h. Реализуйте программу в визуальной среде.

Алгоритм решения этого примера следующий:

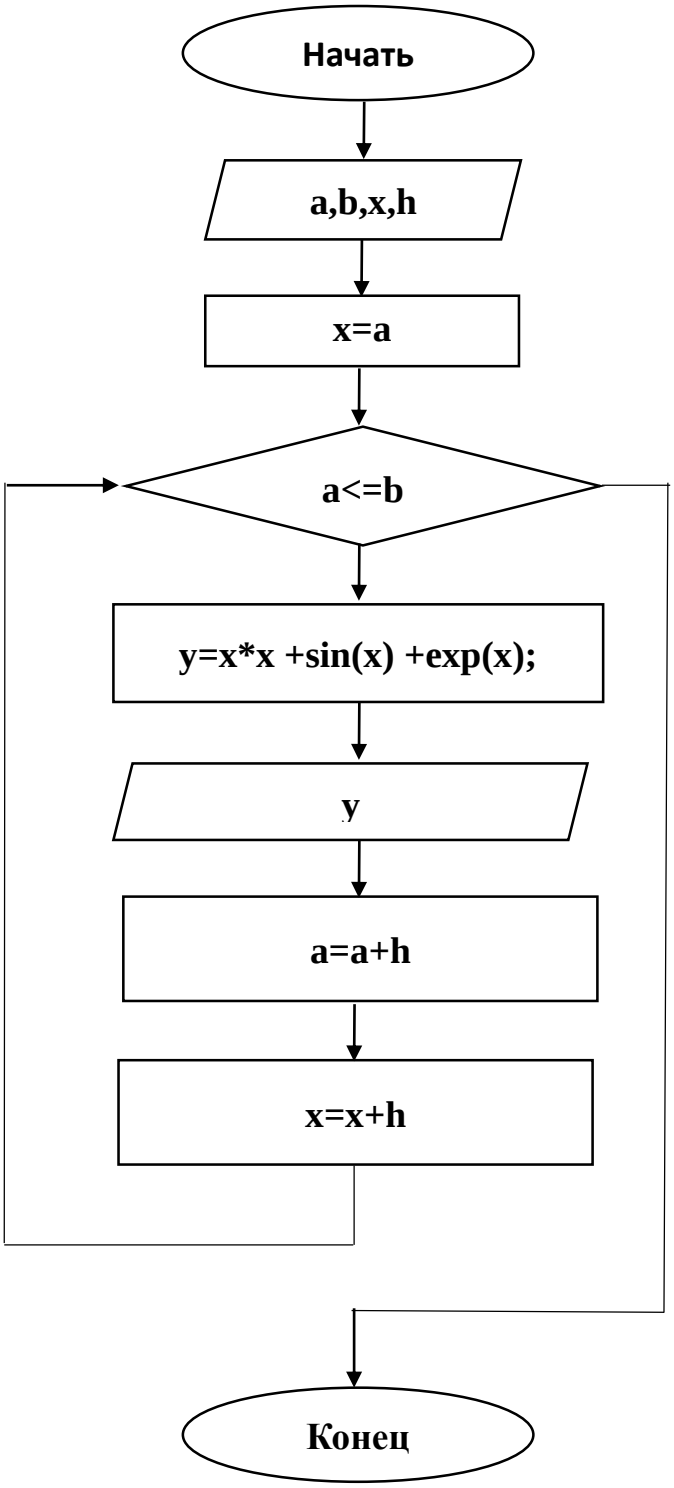


Рис. 6.4. Схема программного блока.

Для программирования в визуальной среде требуются 4 компонента Label, 3 Edit, 3 BitBtn и 1 Memo.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 6.1. Ввод свойств компонента

Имя компонента	Название недвижимости (Состояние окна Object Inspector)	Процесс должен быть реализован
Form1	Caption (Properties)	Вводится слово “Программирование повторяющихся процессов”.
Label1	Caption (Properties)	Вводится слово “Начальное значение X”.
Label2	Caption (Properties)	Вводится слово “Конечное значение X”.
Label3	Caption (Properties)	Вводится слово “h количество шагов”.
Label4	Caption (Properties)	Вводится слово “Окно результатов”.
Edit1	Text (Properties)	Удалить слово “Edit1”.
Edit2	Text (Properties)	Удалить слово “Edit2”.
Edit3	Text (Properties)	Удалить слово “Edit3”.
BitBtn1	Kind (Properties)	Выбрано свойство “bkOK”.
	Caption(Properties)	Вводится слово “ответ”.
	OnClick (Events)	Текст программы введен.
BitBtn2	Kind (Properties)	Выбрать свойство “bkRetry”.
	Caption(Properties)	Вводится слово “убрать”.
	OnClick (Events)	Текст программы введен.
BitBtn3	Kind (Properties)	Выбрать свойство “bkClose”.
	Caption (Properties)	Вводится слово “Выход”.
	OnClick (Events)	Вводится Close();
Memo1	Lines (Properties)	Удалите слово “Memo1”.

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

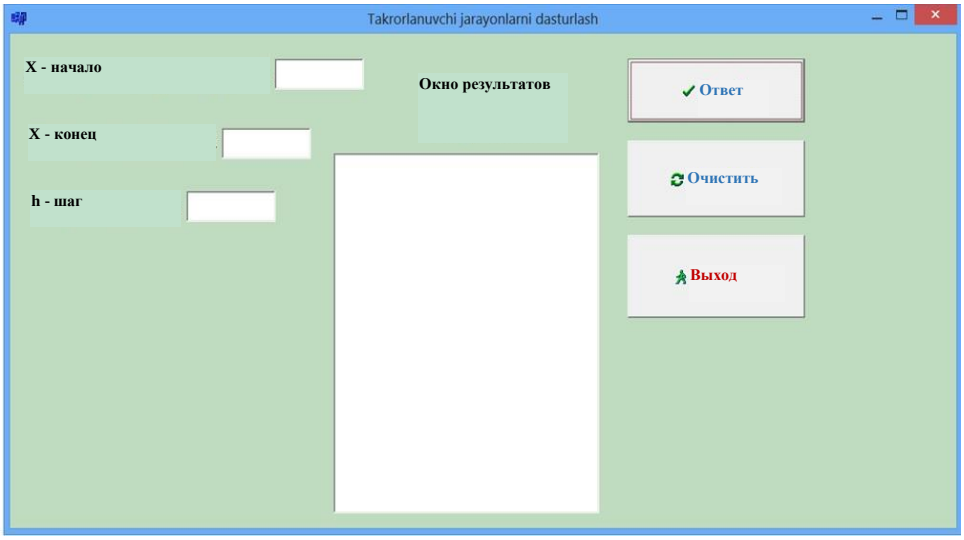


Рис. 6.5. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------

#include <vcl.h>

#include <math.h>

#pragma hdrstop

#include "Unit1.h"

//---------------------------------------------------------

#pragma package(smart_init)

#pragma resource "*.dfm"

TForm1 *Form1;

//---------------------------------------------------------
```

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
float x,y,a,b,h;
  a=StrToFloat(Edit1->Text);
  b=StrToFloat(Edit2->Text);
  h=StrToFloat(Edit3->Text);
  Memo1->Clear();
    x=a;
  while (a<=b)
{
    y=x*x +sin(x) +exp(x);
Memo1->Lines->Add("x=("+FloatToStr(a)+")Y="+FloatTo StrF (y,
ffFixed,6,2));
    x=x+h;
    a=a+h;
 }
 }
//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click (TObject *Sender)
```

```
{
//Tozalash amalga oshiriladi
Edit1->Clear();
Edit2->Clear();
Edit3->Clear();
Memo1->Clear();
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click (TObject *Sender)
{
Close();
}
//---------------------------------------------------------
```

После того, как вводится текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

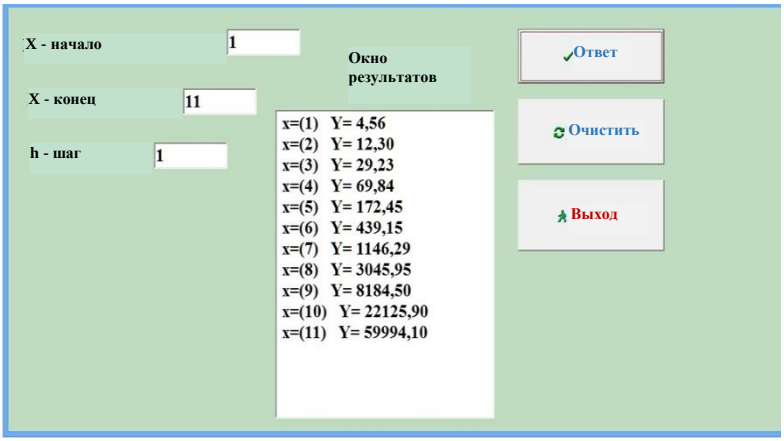


Рис. 6.6. Окно результатов.

6.3. Программирование циклических процессов с постусловием

Этот шаблон повторений также используется, когда количество повторений неизвестно. В этом процессе команда или система команд повторяются до тех пор, пока не будет выполнено указанное условие. Это отличается от процесса повторения, где тело повторения выполняется по меньшей мере один раз, так как условие повторения проверяется после выполнения тела повторения.

Специальный оператор do while используется для программирования этого процесса повторения.

Общий вид этого оператора выглядит следующим образом:

<Оператор проверки состояния> = do <группа операторов> while <логическое выражение>;

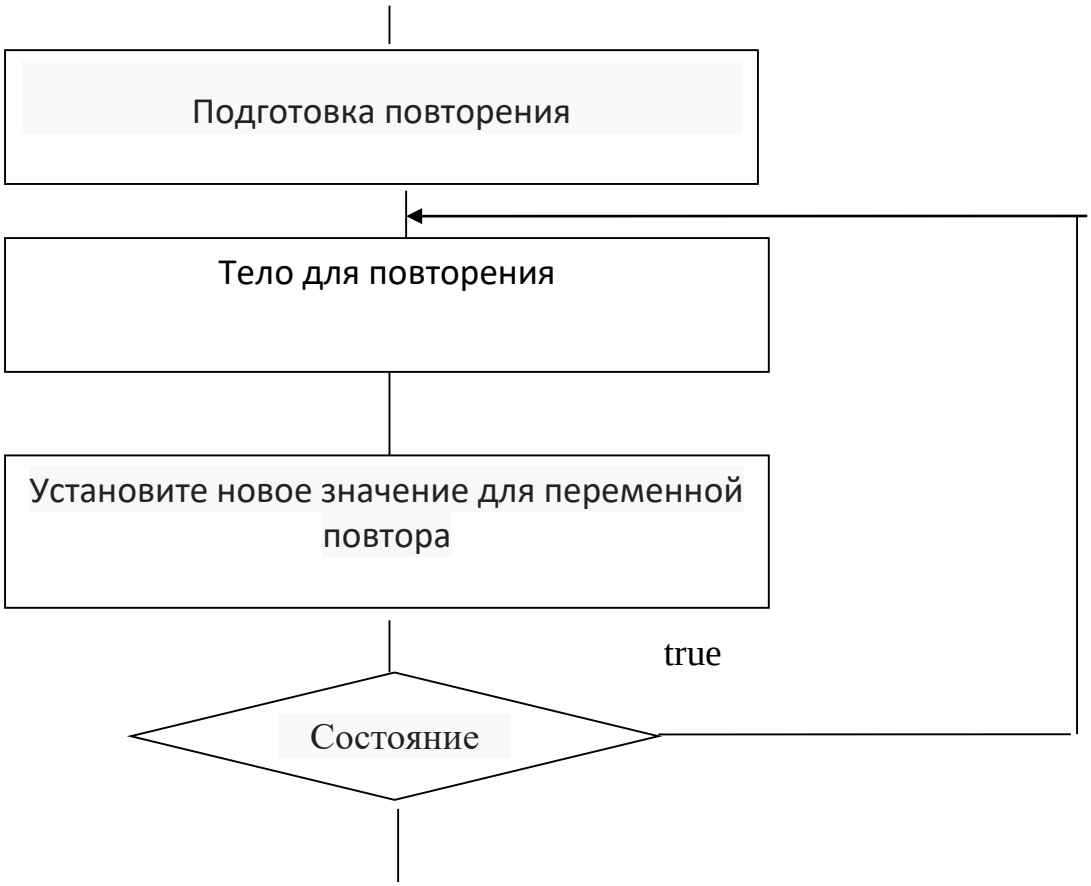
или сделать L в то время как (M);

где: do - do, a - слова слуги, которые означают «к»; L - повторяющееся тело, которое может иметь один оператор или группу операторов; M логическое выражение.

Оператор выполняет в следующем порядке:

- Операторы, входящие в повторяющееся тело, выполняются один за другим. Затем M находит значение логического выражения, то есть условие проверяется. Если это условие не выполняется (значение M равно False), элемент управления выходит за пределы дубликата и передает оператор while оператору. В противном случае повторение будет продолжено.

Этот образец повторения отличается от приведенного выше повторения, при котором тело повторяется по крайней мере один раз.



Пример 1. Создайте программу для расчета значения функции $y = x^2 + 1$ false. Пусть x изменяет значения переменной x с шага a по b на h.

Вид программы в режиме консоли выглядит следующим образом:

```
#include <vcl.h>

#include <iostream.h>

#include <math.h>

#include <conio.h>

#pragma hdrstop


//---------------------------------------------------------

#pragma argsused
```

```
int main(int argc, char* argv[])
{
float a,b,x,y,h;
cout<<"a=";
cin>>a;
cout<<"b=";
cin>>b;
cout<<"h=";
cin>>h;
x=a;
do
{
y=pow(x,2)+1;
cout <<"x=" <<x<<endl;
cout <<"y=" <<y<<endl;
x=x+h;
}
while(x<=b);
getch();
return 0;
}
//-------------------------------------------------------
```

После того, как Вводится текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

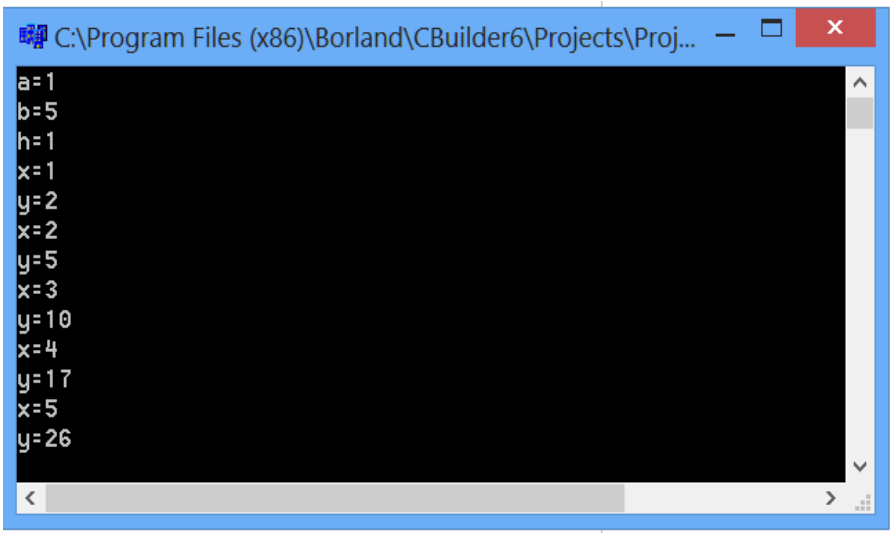


Рис. 6.8. Окно результатов.

Значение функции рассчитывается в заданном диапазоне.

Пример 2. Создайте программу для расчета значения функции $y = \sin x + \cos x$. Пусть x изменит значение переменной x с шага a по b на h. Реализуйте программу в визуальной среде.

Чтобы решить этот пример, сначала мы рассмотрим блок-схему. Алгоритм решения этого примера следующий:

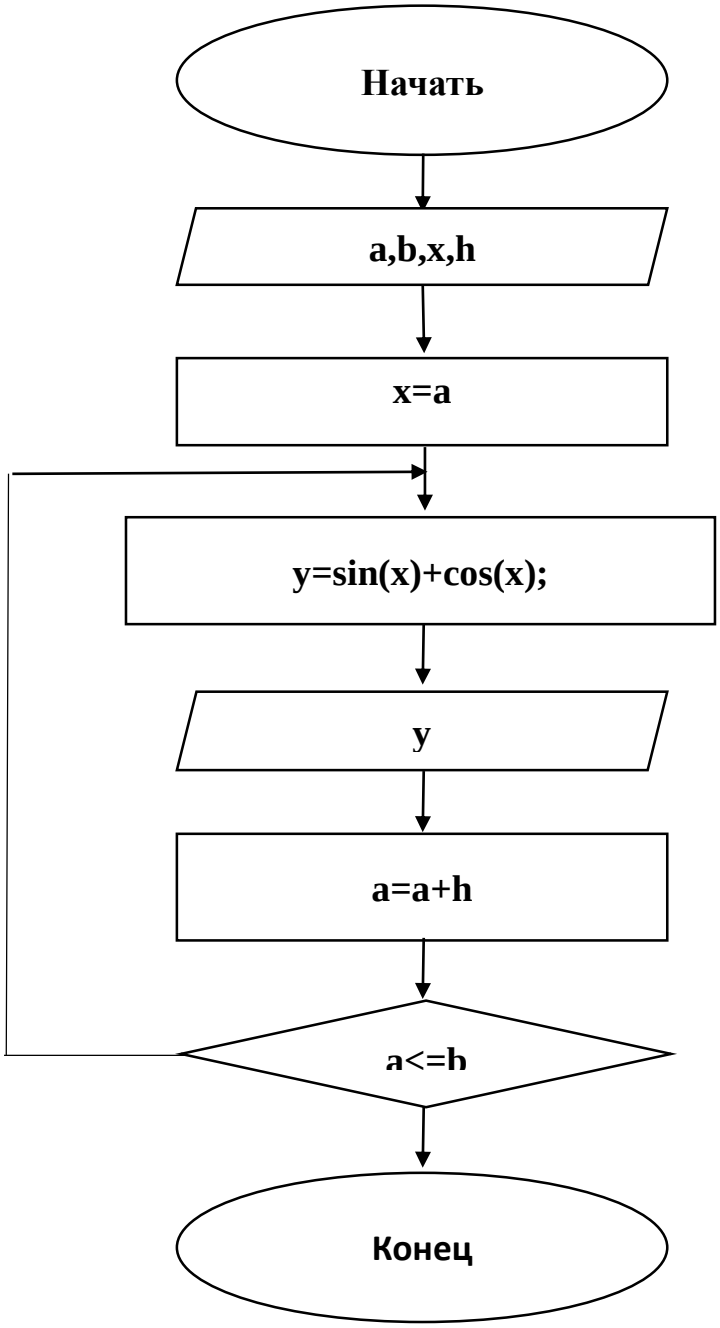


Рис. 6.9. Схема программного блока.

Для программирования в визуальной среде требуются 4 компонента Label, 3 Edit, 3 BitBtn и 1 Memo.

Укажите свойства компонента следующим образом:

Таб. 6.2. Ввод свойств компонента

Имя компонента	Название недвижимости (Состояние окна Object Inspector)	Процесс должен быть реализован
Form1	Caption (Properties)	Вводится слово“do while”.
Label1	Caption (Properties)	Вводится слово “Начальное значение”.
Label2	Caption (Properties)	Вводится слово “Конечное значение”.
Label3	Caption (Properties)	Вводится слово “H количество шагов”.
Label4	Caption (Properties)	Вводится слово “окно результатов” .
Edit1	Text (Properties)	Удалить слово “Edit1”.
Edit2	Text (Properties)	Удалить слово “Edit2”.
Edit3	Text (Properties)	Удалить слово “Edit3”.
BitBtn1	Kind (Properties)	Выбрать свойство “bkOK”.
	Caption (Properties)	Вводится слово “Ответ”.
	OnClick (Events)	Текст программы введен.
BitBtn2	Kind (Properties)	Выбрать свойство “bkRetry”.
	Caption (Properties)	Вводится слово “Убрать”.
	OnClick (Events)	Текст программы введен.
BitBtn3	Kind (Properties)	Выбрать свойство “bkClose” .
	Caption (Properties)	Вводится слово“Выход”.
	OnClick (Events)	ВводитсяClose();

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

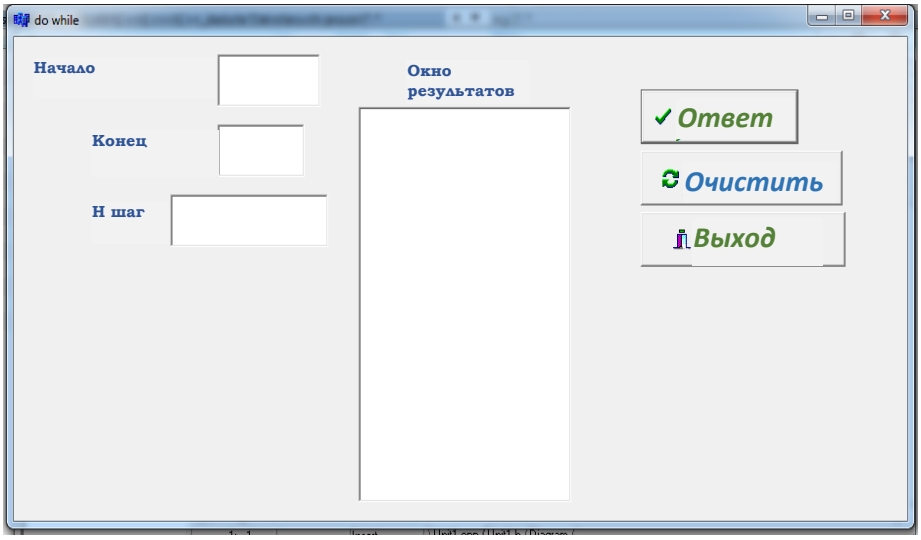


Рис. 6.10. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------
#include <vcl.h>
#include <math.h>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
//takrorlanish tanasi
float a,b,h,y,x;
  h=StrToFloat(Edit3->Text);
```

```
  a=StrToFloat(Edit1->Text);
  b=StrToFloat(Edit2->Text);
  Memo1->Clear();
 do
{
 x=a;
    y=sin(x) +cos(x);
Memo1->Lines->Add("x=("+FloatToStr(a)+") Y= " + FloatToStrF (y,ffFixed,
6,4));
    a=a+h;
}
  while (a<=b);
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click (TObject *Sender)
{
// tozalash jarayoni
Edit1->Clear();
Edit2->Clear();
Edit3->Clear();
Memo1->Clear();
}
//---------------------------------------------------------
```

```
void __fastcall TForm1::BitBtn2Click (TObject *Sender)
{
// dasturdan chiqish
Close();
}
//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

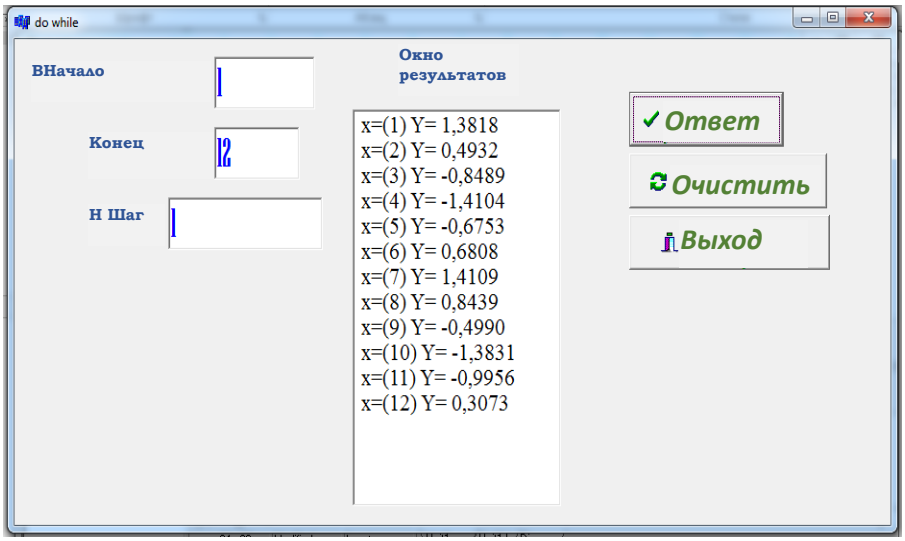


Рис. 6.11. Окно результатов.

6.4 Циклический процесс с параметром

Желательно использовать вышеуказанные операторы цикла в тех случаях, когда количество повторений неизвестно. Если число повторений известно до его выполнения, лучше всего использовать оператор параметра повторения. Этот итерационный алгоритм имеет следующее содержание:

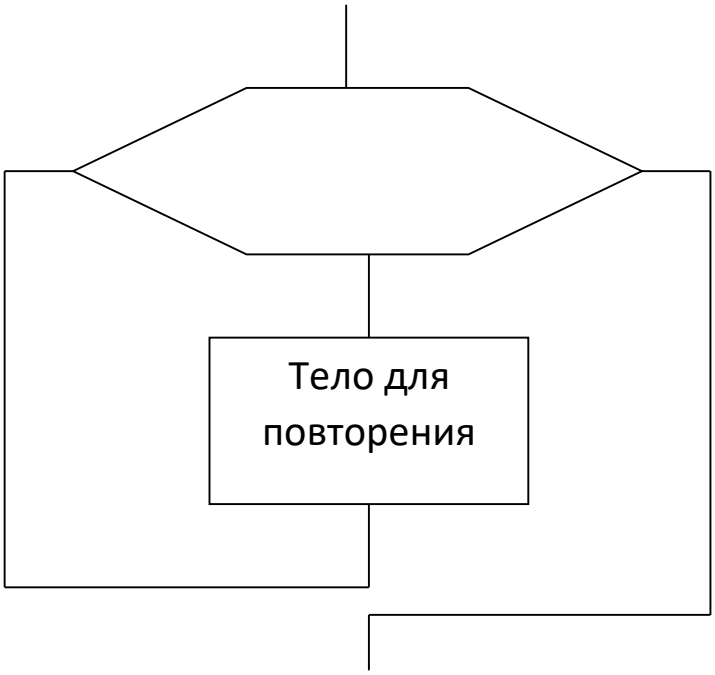


Рис. 6.12. Алгоритм параметрического повторения.

Общая структура записи этого оператора выглядит следующим образом: <Оператор репликации параметров> = для (<Инициализация управляющей переменной, дублированное условие, изменение управляющей переменной) {<группа операторов>};

Режим оператора выглядит следующим образом:

Повторяющееся тело повторяется снова и снова для всех значений переменной менеджера (от начального значения до последнего значения). Например, эти операторы могут быть записаны следующим образом:

1) for (x=a; x<=b; x++) y=m;

2) for (x=b; x<=a; x--) y=m;

3) for (x=a + b; x<=c*k; x++) y=k;

4) for (int i =1, s = 0; i<=100; i++)   s += i;

Пример 1. Создайте программу для выражения выражений $y = \sum_{i=1}^{30} i^3 + 1$, где i - натуральное число. Запустите программу в режиме консоли.

Вид программы в режиме консоли выглядит следующим образом:

```
//---------------------------------------------------------

#include <math.h>

#include <vcl.h>

#include<iostream.h>

#include<conio.h>

#pragma hdrstop

//---------------------------------------------------------

#pragma argsused

int main(int argc, char* argv[])

{ float y;

  y=0;

for (int i=1;i<=30;i++)

y=y+pow(i,3)+1;

cout<<"y="<<y<<endl;

getch();

return 0; }

//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

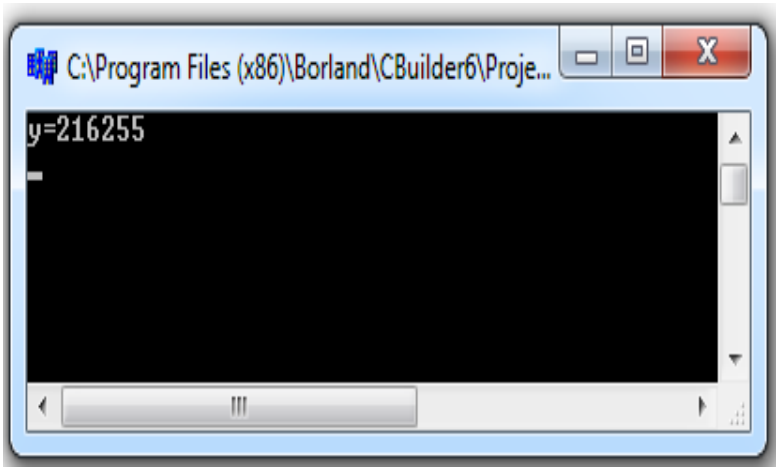


Рис. 6.13. Окно результатов.

Пример 2. Создать программу для вычисления выражения $S = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{50}$. Реализуйте программу в визуальной среде.

Вам понадобятся 1 компонент Label и 2 компонента BitBtn для программирования вашего примера в визуальной среде.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 6.3. Ввод свойств компонента

Имя компонента	Название недвижимости (Состояние окна Object Inspector)	Процесс должен быть реализован
Form1	Caption (Properties)	Вводится слово “Оператор повторения параметров”.
Label1	Caption (Properties)	Вводится слово “результат”.
BitBtn1	Kind (Properties)	Выбрать свойство “bkOK”.
	Caption (Properties)	Вводится слово “расчет”.
	OnClick (Events)	Введен текст программы.
BitBtn2	Kind (Properties)	Выбрать свойство “bkClose” .
	Caption (Properties)	Вводится слово“Выход”.
	OnClick (Events)	ВводитсяClose();

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

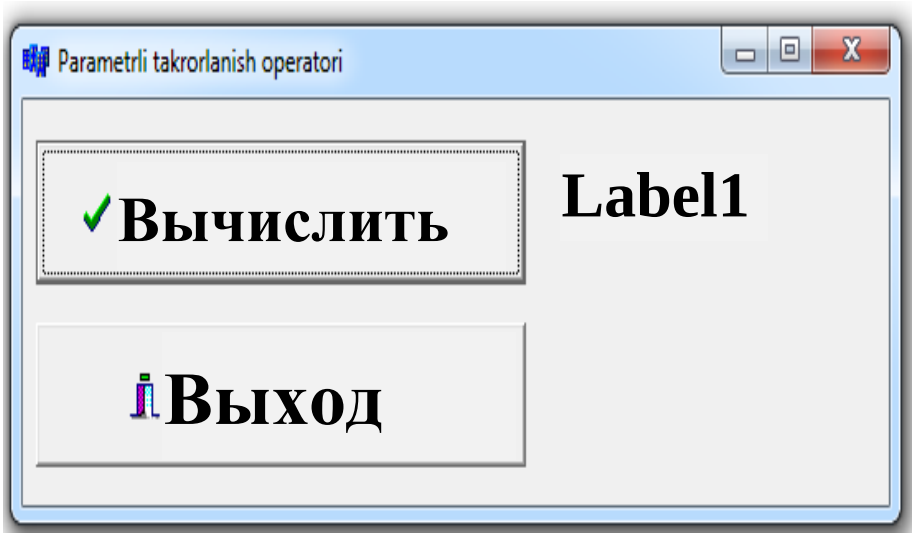


Рис. 6.14. Просмотр приложения.

После настройки компонентов окна формы вводится следующий текст программы:

```
//---------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
```

```
        : TForm(Owner)
{
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
int i;
float s;
s=0;
for (i=1;i<=50; i++)
{
s=s+(float)1/i;
}
Label1->Caption= FloatToStrF(s,ffFixed,6,4);
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
Close();
}
//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

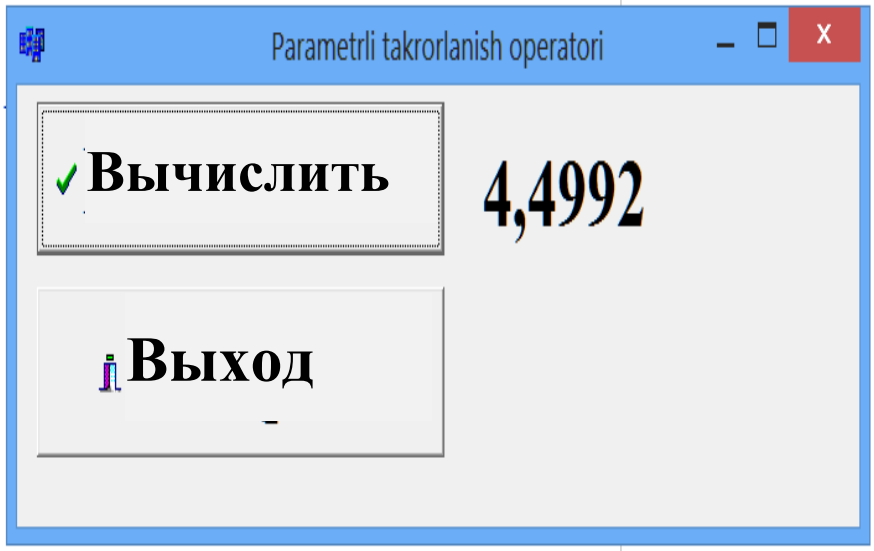


Рис. 6.15. Окно результатов.

Пример 3. Создайте программу для расчета значения функции $y = \sin(x) + 1$. Измените значения переменной X в шагах с 1 по a. Реализуйте программу в визуальной среде.

Алгоритм решения этого примера показан на рисунке 6.16.

Для программирования вашего примера в визуальной среде вам понадобятся 2 Label, 2 Edit, 2 BitBtn и 1 Memo.

Мы организуем необходимые компоненты в последовательности.

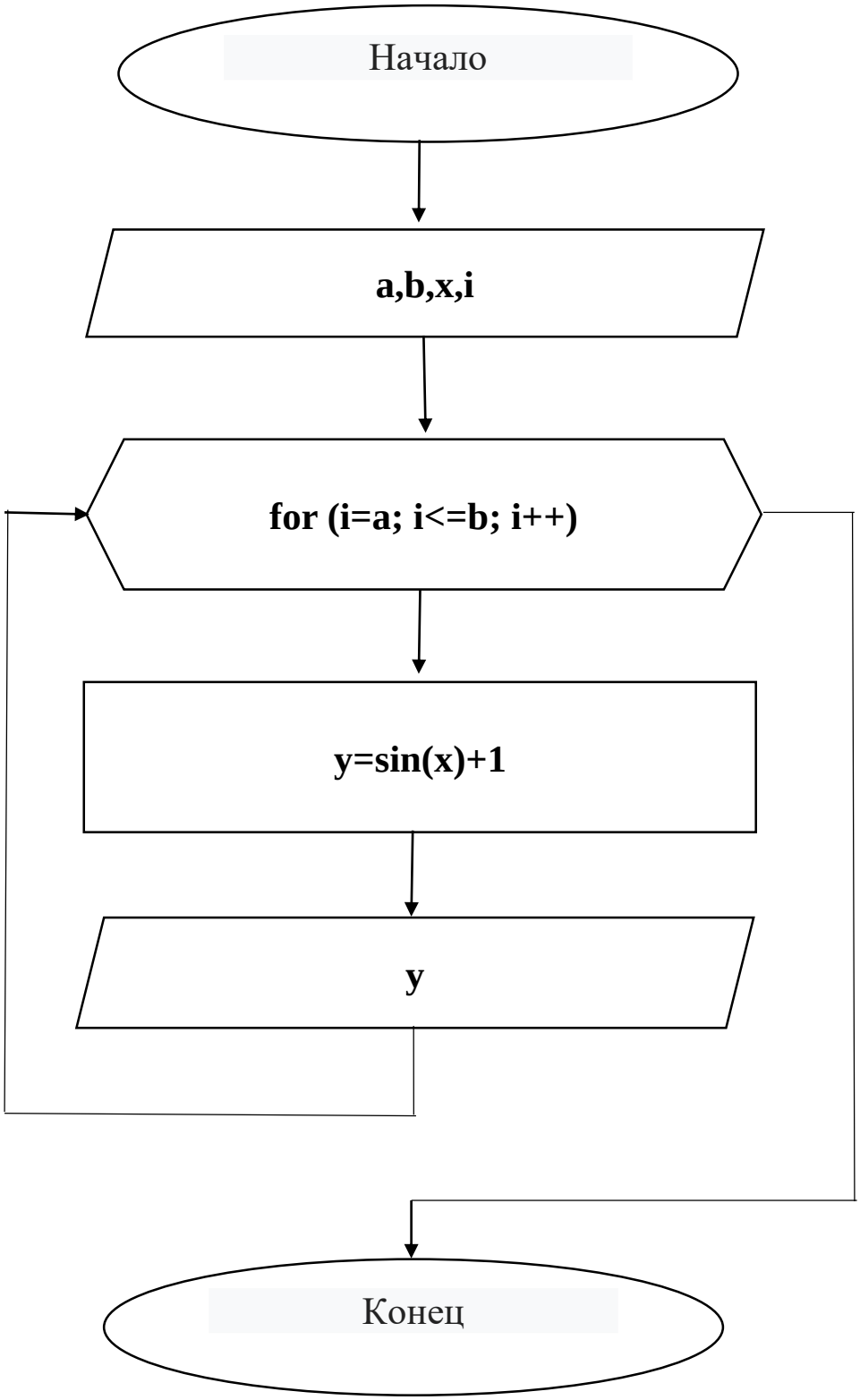


Рис. 6.16. Схема программного блока.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 6.3. Ввод свойств компонента

Имя компонента	Название недвижимости (Состояние окна Object Inspector)	Процесс должен быть реализован
Form1	Caption (Properties)	Вводится слово “Оператор повторения параметров”.
Label1	Caption (Properties)	Вводится слово “Начальное значение”.
Label2	Caption (Properties)	Вводится слово “Конечное значение”.
Edit1	Text (Properties)	Удалить слово “Edit1”.
Edit2	Text (Properties)	Удалить слово “Edit2”.
BitBtn1	Kind (Properties)	Выбрать свойство “bkOK”.
	Caption (Properties)	Вводится слово “Ответ”.
	OnClick (Events)	Текст программы введен.
BitBtn2	Kind (Properties)	Выбрать свойство “bkClose”.
	Caption (Properties)	Вводится слово “Выход” .
	OnClick (Events)	Вводится Close();
Memo1	Lines (Properties)	Удалите слово “Memo1”.

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

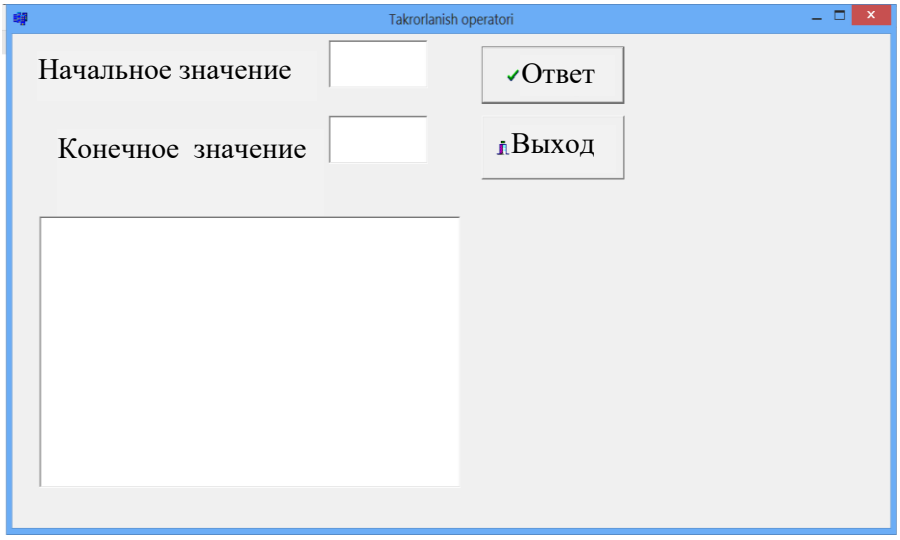


Рис. 6.17. Просмотр приложения.

```
//---------------------------------------------------------
#include <vcl.h>
#include <math.h>
#pragma hdrstop
#include "Unit1.h"

//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{

}

//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
float x,y;
```

```
int a,b;
a=StrToFloat(Edit1->Text);
  b=StrToFloat(Edit2->Text);
        Memo1->Clear();
  for (x=a; x<=b; x++)
{
y=sin(x)+1;
Memo1->Lines->Add("x=("+FloatToStr(x)+")Y="+     FloatToStrF(y,ffFixed,
6,2));
}
}

//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
Close();
}
//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

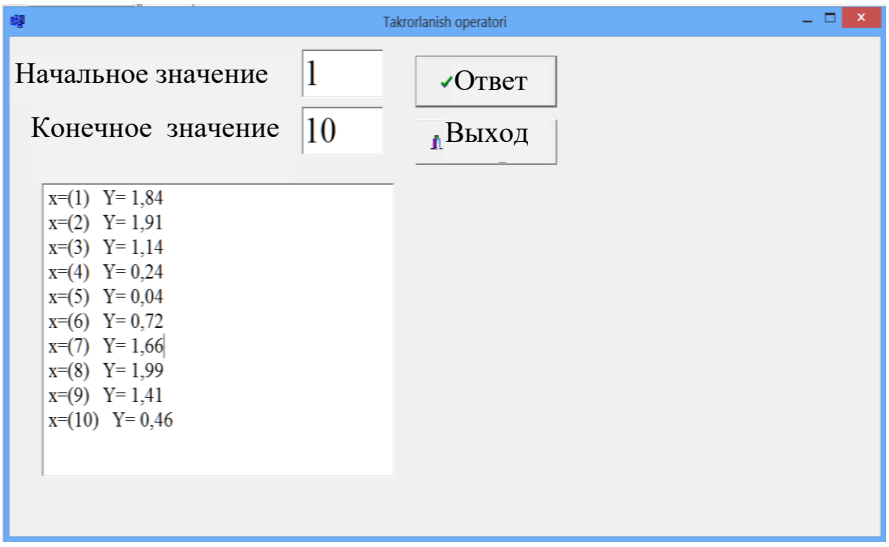


Рис. 6.18. Окно результатов.

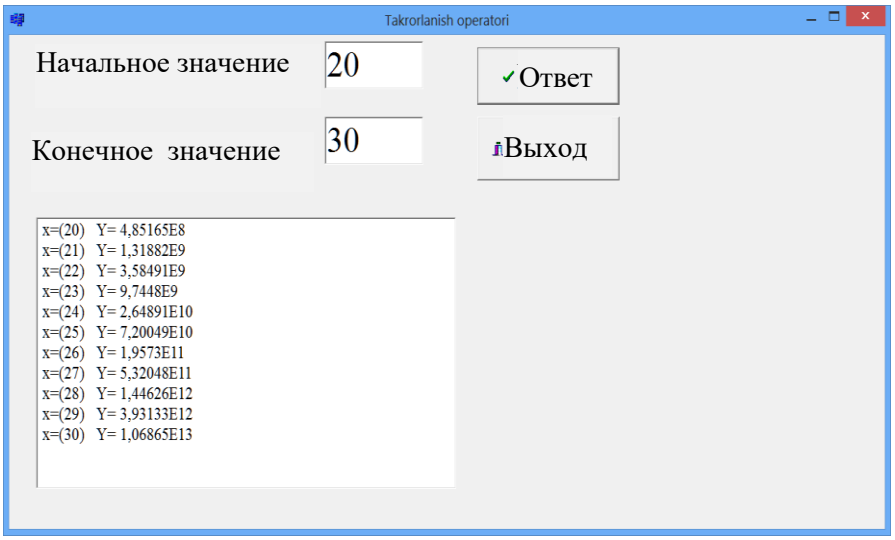


Рис. 6.19. Окно результатов.

6.5. Вложенные циклические процессы

На основании вышеперечисленных операторов можно делать сложные повторения. Если тело повторяющихся процессов состоит из повторяющихся структур, такое повторение называется «внутренним или сложным», иными словами, сложными повторяющимися программами, если одно или несколько

повторений включены в одно повторение. Этот сложный процесс показан на рисунке 6.19.

Повторение, которое включает в себя другие процессы повторения, называется «Внешнее повторение».

Повторение внутри процесса повторения называется «Внутреннее повторение».

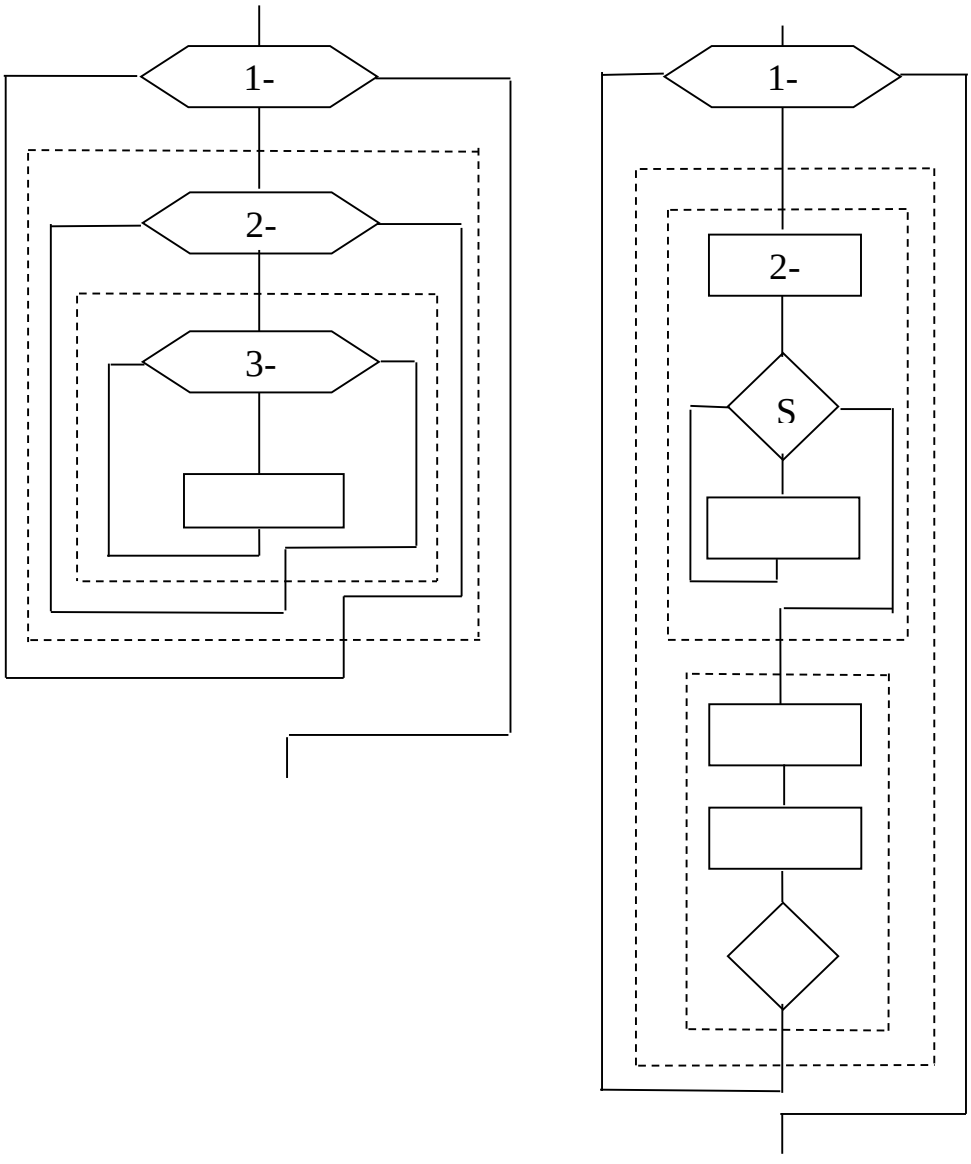


Рис. 6.19. Процесс повторяющийся рецептуры.

Вопросы по главе 6

1. Что такое процесс, называемый повторяющимися вычислениями?

2. Каковы компоненты повторяющегося вычислительного алгоритма?

3. Дайте представление о теле повторяющегося процесса и переменной повторения.

4. Сначала организуйте процесс повторения (условие, повторение, повторение).

5. Организовать процесс повторения, который проверяется условно (do while, итерация, подсчет повторений).

Тестовые вопросы:

1.Какие операторы используются для генерации дублирования в C ++?

а) for, while, do while, repeat;

б) for, while, do while;

в) for, while, repeat;

г) if else, goto, case;

2.C ++ отображается в строке, в которой сначала должен проверяться оператор повторения.

а) for;

б) do while;

в) while;

г) goto;

3. в то время как (L) M; в итерационном операторе (L) и что такое M?

а) L - уравнения, M - группа операторов или операторов;

б) формула L, M группа операторов или операторов;

в) L - оператор или группа операторов, M - логическое выражение;

г) L - логическое выражение, M - группа операторов или операторов;

4. Общий вид оператора повторения параметров?

a) for (i=1; i<=n; i++);

b) if (M) else (O);

d) while (L) M;

e) Do M While (L);

5. Итератор для C ++, который затем можно проверить

а) goto;

б) do while;

в) for;

г) while;

Глава 7. РАБОТА С МАССИВАМИ В СИСТЕМЕ ПРОГРАММИРОВАНИЯ C++ BUILDER 6

Ключевые слова: массив, элемент массива, одномерный, многомерный, память массива, матрица, компонент StringGrid.

7.1. Понятие массива и их типы

Данные, используемые в C ++, разбиты на простые и сложные категории. Основной отличительной чертой данных общей категории от других категорий является их порядок и целостность, то есть, например, произвольное целое число представляет собой целое число, которое не является одним целым числом. Комплексная информация представляет собой совокупность нескольких размеров. Эти категории поэтому называются составными категориями.

Класс массива C ++ используется почти каждой программой. Массив - это отсортированная последовательность данных той же категории с общим именем. Обычно удобно представлять список данных, элементов таблиц различных типов в виде массивов. Массивы служат для выделения места для хранения. Например, вы должны сохранить 5 цифр в категории int. Для этого вам необходимо ввести следующую строку:

int myMassiv [5];

Компилятор выделяет место для хранения массива. Поскольку вам требуется 4 МБ данных Int, вы можете выделить 20 МБ общего пространства. Вы можете увидеть это на картинке ниже.

myMassiv [0]	myMassiv [1]	myMassiv [2]	myMassiv [3]	myMassiv [4]
baseAddr	baseAddr+4	baseAddr+8	baseAddr+12	baseAddr+16

Рис. 7.1. Выделение памяти для массива.

Понятие массы может быть объяснено на примере числового вектора с общим именем и ряда фиксированных переменных в одной категории: A (5) = (a1, a2, a3, a4, a5), где a1, a2 .... массивные элементы. При их выражении используются индексы. Из математического курса индекс указывает положение переменных в порядке их появления.

7.2. Описание массивов на языке C++

Если программа используется массово, она должна быть объявлена перед использованием. Массив объявлен следующим образом:

<тип> <имя массива> [<количество элементов>] = {начальные значения};

где <имя массива> - необязательный идентификатор, <количество элементов> - выражение указателя, которое указывает количество элементов массива и отображает символы, используемые для записи индексов, поэтому в этой категории могут использоваться все обычные категории, кроме категорий «истина» и «неограниченная», <тип> - это категория элементов массива, которая может использоваться всеми категориями, кроме категорий файлов и коллекций.

Ниже приведены некоторые примеры правильно описанных массивов:

- float a_massiv [4];

- char year [10];

- int butun [22];

- bool mantiq [44].

Массив должен описывать категорию, имя и количество элементов.

В этом примере a_massive представляет собой массив из 4 элементов с действительными числами. Индексы имеют номера от 0 до 3 и могут быть описаны как:

float a_массив [4];				
Элементы массива	a_massiv [0]	a_ массив [1]	a_ массив [2]	a_ массив [3]
Элементы массива	2.2	6	1.3	6.8

Рис. 7.2. Элемент массива и его значение.

Существует различие между понятием индикатора и категорией индекса, а категория индекса обозначает количество и порядок элементов массива, используемых в разделе дескриптора массива. Указатель устанавливает серийный номер элемента массива и используется только в разделе оператора. Если вы используете полное имя, то есть имя переменной, для ссылки на массив, переменная-указатель используется для ссылки на конкретный элемент массива.

Переменные выражения могут быть представлены как слева, так и справа от оператора значения, и могут включать в себя сравнения, сортировку, арифметические операции, минимальные и максимальные значения, то есть все действия, которые могут быть выполнены в базовой категории. , Например, если базовой категорией является int, то можно использовать все действия, которые можно выполнить для всего класса, даже функции по умолчанию.

До сих пор мы рассматривали только одномерные массивы, т.е. одномерные массивы. Элементы массива языка программирования C ++ обеспечивают возможность хранения значений через строки и столбцы в виде двумерных переменных без каких-либо ограничений, за исключением того,

что все они должны принадлежать к одной и той же категории. Эти массивы образуют многомерные массивы.

Многомерные массивы описаны в программе следующим образом:

<тип> <имя массива> [<количество строк>] [<количество столбцов>];

В отличие от одномерных массивов, двумерные массивы представляют два значения в скобках [] после имени. Первый - это количество строк, а второй - количество столбцов. То есть элемент двумерного массива представлен двумя индексами. Например, двумерные массивы могут использоваться в качестве матриц. Объявление двумерного массива выглядит следующим образом:

int a [3][3];

float b [2][4].

Элементы матрицы A имеют следующий вид:

$$a = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

Матрица имеет 3 строки и 3 столбца. Массив, содержащий m строк и n столбцов, называется m * n. Если количество строк и столбцов равно, квадрат называется массивом.

7.3. Ввод элементов массива в память

Чтобы использовать массивы в программе, значения элементов массива должны храниться в памяти. Вы можете использовать операторы ввода или выравнивания данных, чтобы присвоить значение элементам массива.

Пример 1. Создайте программу для вставки и отображения массивов из n элементов, состоящих из целых чисел и n элементов.

Вид программы в режиме консоли выглядит следующим образом:

```
//---------------------------------------------------------
#include <vcl.h>
#include <iostream.h>
#include <conio.h>
#pragma hdrstop
//---------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
int a[10]={0};
int n;
cout << "n="; cin>>n;
for (int i=0; i<n; i++)
{
cout <<"a["<<i<<"]=";
cin>> a[i];}
for (int i=0; i<n; i++)
cout<<a[i]<<"";
  getch ();
  return 0;}
//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

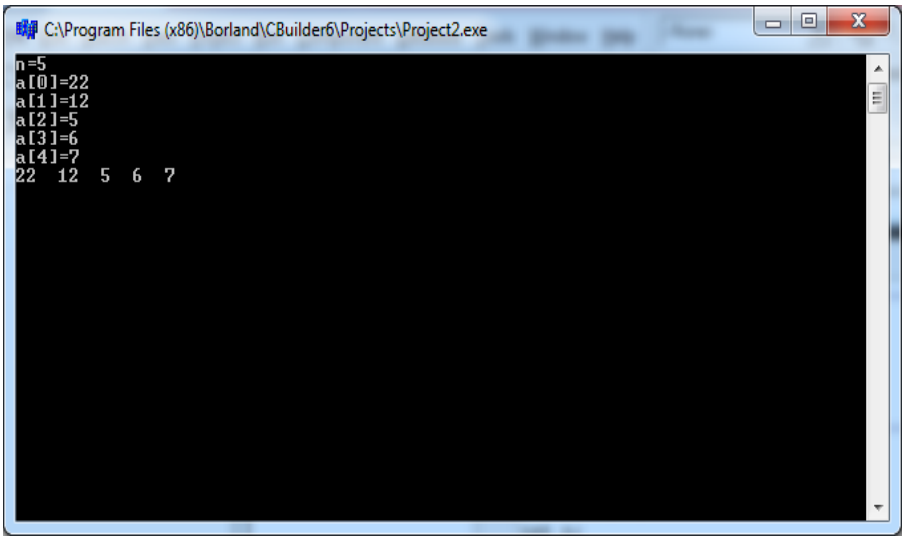


Рис.7.3. Окно результатов.

Пример 2. Дан массив, состоящий из N элементов. Создайте программу для расчета суммы этих элементов. Вид программы в режиме консоли выглядит следующим образом:

```
//---------------------------------------------------------

#include <vcl.h>

#include <iostream.h>

#include <conio.h>

#pragma hdrstop

//---------------------------------------------------------

#pragma argsused

int main(int argc, char* argv[])

{  int a[10]={0};

int n;

int s=0;
```

```
cout << "n="; cin>>n;
for (int i=0; i<n;i++)
{cout <<"a["<<i<<"]=";
cin>> a[i];
s+=a[i];}
cout<<" Сумма элементов массива ="<<s<<endl;
    getch ();
    return 0;}
//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

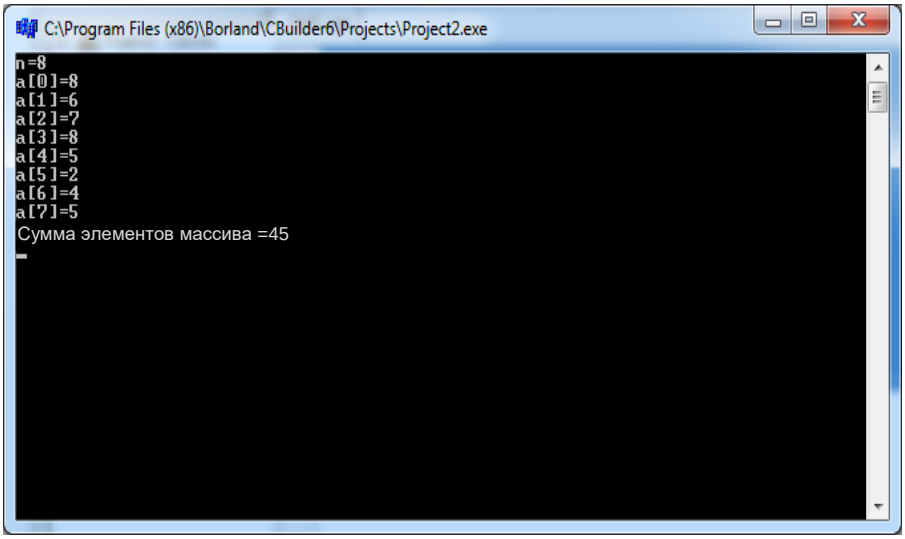


Рис. 7.4. Окно результатов.

Когда описательные массивы могут быть инициализированы, то есть заданы начальные значения. Например:

```
float C[]={1,-1,2,10,-12.5};
```

В этом примере граница массива определяется автоматически. Если для массивной инициализации указана граница массива, количество элементов в списке может быть меньше этого порога, но не может быть больше. Например:

int a[5]={2,-2}.

В этом случае значения a [0] и a [1] определены, а 2 и –2 соответственно, а остальные элементы предполагаются равными 0.

В следующем примере описывается двумерный массив m и d, а также значения:

```
const int строчка = 3, столбец= 4;

int m[строчка][ столбец] = {{1, 3, 3, 6}, {1, 1, 2, 2}, {3, 3, 2, 0}};

float d[2][3]={(1,-2.5,10),(-5.3,2,14)};
```

Когда инициализация определяется инициализацией, первая граница индекса массива не требуется, но остальные индексы должны быть указаны.

Например:

```
double x[][2]={(1.1,1.5),(-1.6,2.5),(3,-4)} ;

double x[][2]={(1.1,1.5),(-1.6,2.5),(3,-4)} ;
```

7.4. Методы работы с массивами в приложении «Форма»

Компонент таблицы StringGrid очень удобен для работы с массивами в приложении Form. Компонент StringGrid находится в дополнительной палитре компонентов и используется для отображения, например, значения элементов матрицы на экране в виде таблицы, ввода и редактирования их значения. Номера строк и столбцов таблицы начинаются с нуля. Количество столбцов и строк может быть изменено соответствующим образом. Это определяется его собственностью. Каждый столбец пересечения и строка таблицы называется ячейкой, а Введеноные данные определяются как строка символов.

Данные отображаются именно на ячейках. Основные свойства компонента таблицы StringGrid обсуждаются в таб. 7.1.

Таб.7.1 Основные свойства компонента StringGrid

№	Свойства	Задача
1.	Name	Наименование компонента. Используется для доступа к свойствам компонента
2.	ColCount	Определяет количество столбцов в таблице
3.	RowCount	Определяет количество строк в таблице
4.	Cells	Указывает ячейку таблицы, номер столбца столбца и номера строк
5.	FixedCols	Определяет количество фиксированных столбцов
6.	FixedRows	Определяет количество фиксированных линий
7.	Options -> goEditing	Указывает состояние таблицы (обнаружение выполняется на основе ее параметров; например, если параметр GoEditing имеет значение true, ячейка может быть отредактирована, в противном случае это невозможно)
8.	Options -> goTab	Определяет использование клавиши <Tab> для перехода к следующей ячейке таблицы
9.	DefaultColWidth	Определяет ширину основных столбцов таблицы
10.	DefaultRowHeight	Определяет высоту основного столбца таблицы
11.	GridLineWidth	Определяет ширину линии границы в ячейках таблицы
12.	Left	Устанавливает расстояние от левой границы формы до левого края таблицы
13.	Top	Установить расстояние от верхней части таблицы до верхней части формы
14.	Height	Устанавливает высоту табличного пространства
15.	Width	Устанавливает ширину табличного пространства
16.	Font	Указать шрифт для ячеек таблицы

Пример 3. Создайте массив из 10 элементов. Создайте программу для заполнения и умножения 10 элементов массива на разные числа.

Реализуйте программу в визуальной среде. Вам понадобятся 2 компонента Labels, 2 StringGrid, 3 BitBtn для программирования вашего примера в визуальной среде.

Мы определяем свойства компонента окна формы следующим образом:

Таб.7.2. Ввод свойств компонента

Имя компонента	Имя свойства (Состояние окна Object Inspector)	Процесс, который должен быть выполнен
Form1	Caption (Properties)	Вводится слово “работа с массивами”.
Label1	Caption (Properties)	Вводится слово “всплывающее окно”.
Label2	Caption (Properties)	Вводится слово “Выбранные элементы массива”.
BitBtn1	Kind (Properties)	Выбираем свойство “bkOK”.
	Caption (Properties)	Вводится слово “заполнить массив”
	OnClick (Events)	Введем текст программы.
BitBtn2	Kind (Properties)	Выбираем свойство “bkAll”.
	Caption (Properties)	Вводится слово "сортировка массива".
	OnClick (Events)	Введем текст программы.
BitBtn3	Kind (Properties)	Выбираем свойство “bkClose”.
	Caption (Properties)	Вводится слово “Выход”.
	OnClick (Events)	Вводится Close();
StringGrid1	FixedCols (Properties)	Вводим цифру 0
	FixedRows (Properties)	Вводим цифру 0
	ColCount (Properties)	Вводим цифру 10
	RowCount (Properties)	Вводим цифру 1
StringGrid2	FixedCols (Properties)	Вводим цифру 0
	FixedRows (Properties)	Вводим цифру 0
	ColCount (Properties)	Вводим цифру 10
	RowCount (Properties)	Вводим цифру 1

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------

#include <vcl.h>

#pragma hdrstop

#include "Unit1.h"


//---------------------------------------------------------

#pragma package(smart_init)

#pragma resource "*.dfm"

TForm1 *Form1;

int  a[10][1];

int i,j;
```

```
//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{randomize();
for(i=0; i<=9; i++)
{ a[i][0]=random(10);
StringGrid1->Cells[i][0]=IntToStr(a[i][0]);}}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
int k;
        for(j=0;j<=9;j++)
         {
          for(i=0;i<=8;i++)
                if (a[i][0]>a[i+1][0])
                 {
                  k=a[i][0];
                  a[i][0]=a[i+1][0];
```

```
                a[i+1][0]=k;
}
            StringGrid2->Cells[j][0]=IntToStr(a[j][0]);
    }
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
Close ();
}
//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

Рис. 7.6. Окно результатов.

Пример 4. Создайте массив из 4 строк и 4 столбцов. Создайте программу для заполнения 4 * 4 элементов массива различными числами и вычисления суммы элементов. Реализуйте программу в визуальной среде.

Программирование в визуальной среде требует 1 компонентов Label, 1 StringGrid, 3 BitBtn.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 7.3. Ввод свойств компонента

Имя компонента	Имя свойства (Состояние окна Object Inspector)	Процесс, который должен быть выполнен
Form1	Caption (Properties)	Вводится слово “работа с массивами”.
Label1	Caption (Properties)	Вводится слово “накопленный”.
BitBtn1	Kind (Properties)	Выбирается свойство “bkOK”.
	Caption (Properties)	Вводится слово “заполнить массив”.
	OnClick (Events)	Введем текст программы.
BitBtn2	Kind (Properties)	Выбирается свойство “bkOK”.
	Caption (Properties)	Вводится слово “подсчет суммы”.
	OnClick (Events)	Введем текст программы.
BitBtn3	Kind (Properties)	Выбирается свойство “bkClose”.
	Caption (Properties)	Вводится слово “Выход”.
	OnClick (Events)	Вводится Close();
StringGrid1	FixedCols (Properties)	Вводится цифра 0
	FixedRows (Properties)	Вводится цифра 0
	ColCount (Properties)	Вводится цифра 4
	RowCount (Properties)	4 sonini kiritamiz

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

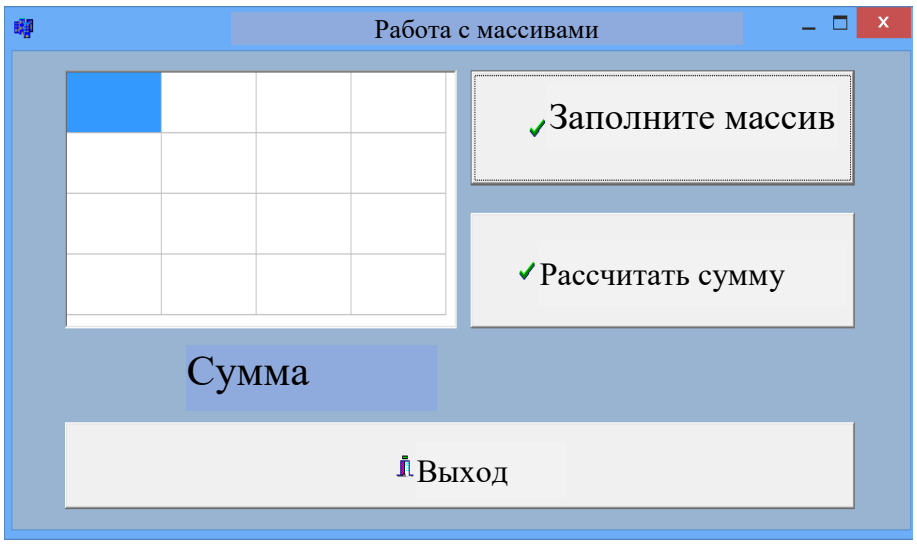


Рис.7.7. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
#include <vcl.h>

#pragma hdrstop

#include "Unit1.h"

//---------------------------------------------------------

#pragma package(smart_init)

#pragma resource "*.dfm"

TForm1 *Form1;

int  sum=0;

//---------------------------------------------------------

void __fastcall TForm1::BitBtn1Click(TObject *Sender)

{int i,j, a[4][4];
```

```
randomize();
for(i=0;i<4;i++)
for(j=0;j<4;j++)
{ a[i][j]=random(10);
StringGrid1->Cells[i][j]=IntToStr(a[i][j]);
sum+=a[i][j];}}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{Label1->Caption="Yig‘indi=" +IntToStr(sum);}
//---------------------------------------------------------
```

После того, как Введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

Рис. 7.8. Окно результатов.

Пример 5. Создайте программу для заполнения элементов массива различными числами и расчета максимального, минимального количества элементов массива, а также суммы элементов.

Реализуйте программу в визуальной среде. Для программирования в визуальной среде требуется 1 компонент Label, 1 StringGrid, 4 BitBtn и 1 Edit.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 7.4. Ввод свойств компонента

Имя компонента	Имя свойства (Состояние окна Object Inspector)	Процесс, который должен быть выполнен
Form1	Caption (Properties)	Вводится слово “Работа с массивами”.
Label1	Caption (Properties)	Вводится слово “Результат”.
Edit1	Text (Properties)	удалите слово “Edit1”.
BitBtn1	Kind (Properties)	Выбираем свойство “bkOK”.
	Caption (Properties)	Вводится слово “Заполните массив”.
	OnClick (Events)	Введем текст программы.
BitBtn2	Kind (Properties)	Выбираем свойство “bkOK”.
	Caption (Properties)	Вводится слово “Наименьший элемент массива”.
	OnClick (Events)	Введем текст программы.
BitBtn3	Kind (Properties)	Выбираем свойство “bkOK”.
	Caption (Properties)	Вводится слово “Самый большой элемент массива”.
	OnClick (Events)	Введем текст программы.
BitBtn4	Kind (Properties)	Выбираем свойство “bkOK”.
	Caption (Properties)	Вводится слово “Сумма элементов массива“.
	OnClick (Events)	Введем текст программы.
StringGrid1	FixedCols (Properties)	Вводится цифра 0
	FixedRows (Properties)	Вводится цифра 0
	ColCount (Properties)	Вводится цифра 5
	RowCount (Properties)	Вводится цифра 5

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

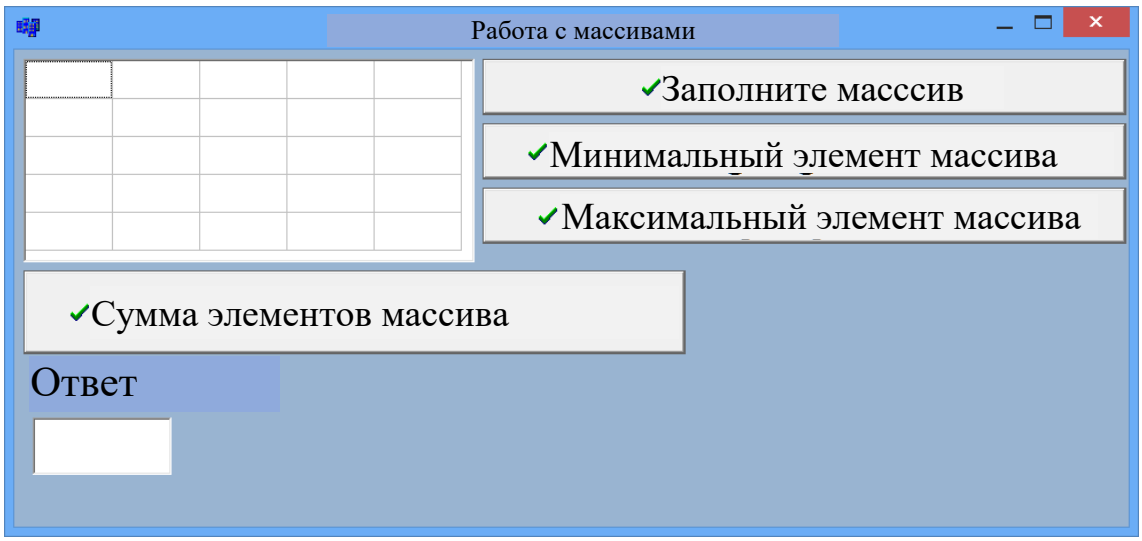


Рис. 7.9. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------

#include <vcl.h>

#pragma hdrstop

#include "Unit1.h"

//---------------------------------------------------------

#pragma package(smart_init)

#pragma resource "*.dfm"

TForm1 *Form1;

 int  a[5][5];

 int i,j,s,max,min;

//---------------------------------------------------------

__fastcall TForm1::TForm1(TComponent* Owner)
```

```
        : TForm(Owner)
{
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
for (i=0;i<=4;i++)
  for (j=0;j<=4;j++)
  {
   a[i][j]=random(50);
   StringGrid1->Cells[i][j]=IntToStr(a[i][j]);
   }
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
min=a[0][0];
 for (i=0;i<=4;i++)
  for (j=0;j<=4;j++)
   if (a[i][j]<min) min=a[i][j];
 Edit1->Text=IntToStr(min);
}
```

```
//---------------------------------------------------------
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
max=a[0][0];
 for (i=0;i<=4;i++)
  for (j=0;j<=4;j++)
   if (a[i][j]>max) max=a[i][j];
 Edit1->Text=IntToStr(max);
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{
s=0;
for (i=0;i<=4;i++)
  for (j=0;j<=4;j++)
   s=s+a[i][j];
 Edit1->Text=IntToStr(s);
}
//---------------------------------------------------------
```

После того, как Введено текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

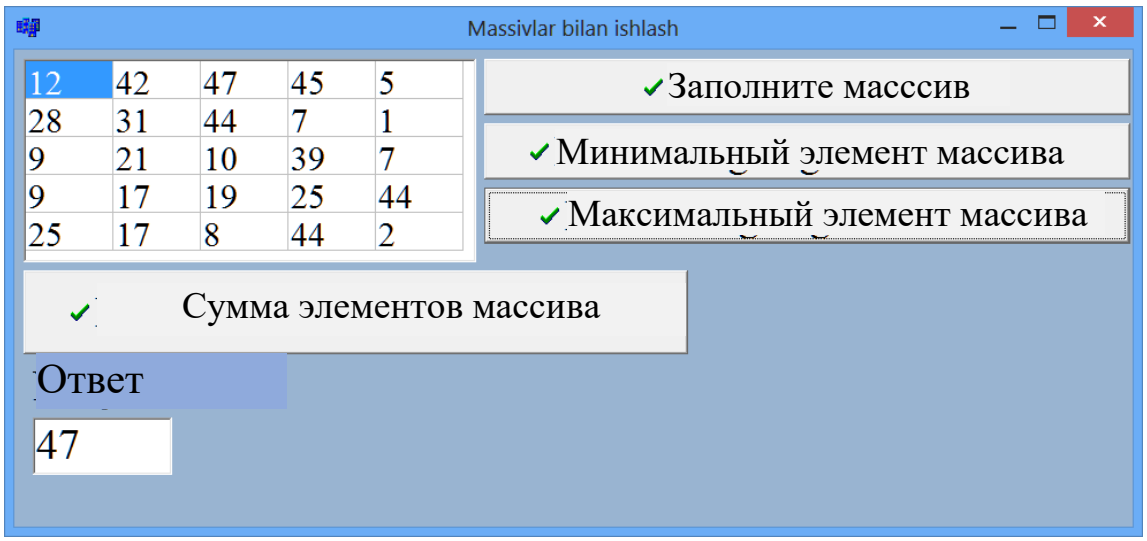


Рис. 7.10. Окно результатов.

Вопросы для повторения

1. Опишите массив и зачем его использовать?

2. Массив элементов и действий над ними.

3. Что такое одномерные и многомерные массивы?

4. Как массивы вводятся в память?

5. Как работать с массивами в визуальной среде?

Тестовые вопросы:

1. Сколько элементов в следующем массиве? Int a [10] [15];

а) 45;

б) 25;

в) 35;

г) 150;

2. Какой оператор используется для хранения элементов массива?

а) повторение;

б) Сеть;

в) смешанная категория;

г) оператор условного перехода;

3. Что означает свойство ColCount компонента StringGrid?

а) Определяет количество столбцов и строк в таблице;

б) Определяет количество строк в таблице;

в) Определяет количество столбцов в таблице;

г) Определяет количество ячеек в таблице;

4. Выберите правильный многомерный массив;

a) int matrix[3] [5];

b) float a[1..4];

d) massiv Int [4 5];

e) massiv of [4] [5];

5. Что означает свойство FixedRows компонента StringGrid?

а) Определяет количество строк в таблице ;

б) Определяет количество столбцов, которые будут исправлены ;

в) Определяет количество ячеек;

г) выполняет фиксацию;

Глава 8. СТРУКТУРНЫЙ ТИП НА ЯЗЫКЕ С++

Ключевые слова: структуры, структурные элементы, структурная привязка, смешанная категория, индикаторы смешанной категории.

8.1. Структурный тип на языке С++ и его описание

Определенные типы документов, каталогов, списков используются для решения экономических и информационных проблем. Например, студенческие анкеты: фамилия, имя, отчество, место жительства, год рождения, специальность, номер группы и т. Д. В этих случаях необходимо объединить разные категории данных в одну группу. В нашем примере мы можем объединить эти данные со студенческой группой. Как видите, категории данных в этой группе разные: фамилия, имя (символ), год рождения, номер группы - все категории.

В С ++ такие данные можно представить с помощью структурированной категории. Структура - это объединение данных в единый набор именованных элементов. Структурные элементы (поля) могут быть разных типов и должны иметь разные имена.

Структура определяется следующим образом:

struct {<список описания>}

Структура должна иметь один компонент. Переменная структурного типа определяется следующим образом:

<имя_структуры><переменная>;

Структура может быть описана в программе несколькими способами:

1. структура <имя категории>

{

<категория><1 элемент >;

<категория><2 элемент >;

. . .

};

Например:

```
structДата
{
int day;
int month;
intyear;
};structДата Категория ;   //Дата Категория Переменнаяstruct
{
<категория><1 элемент>;
<категория><2 элемент >;
. . .
};< Список переменных>;
```

Например:

```
struct
{
  int min;
   int sec;
   intmsec;
}  время;
```

3. Категория структуры также может быть описана словом typedef.

Например:

```
typedef struct
{
 float re;
 float im;
} Complex;
Complexa[100];
```

Предметы названы в честь символического имени. Поскольку элементы относятся к разным категориям, их описания категорий приведены отдельно. Он содержит ряд структурных полей с названием предмета и его описанием. Следовательно, конструкции могут состоять из нескольких областей.

Как и у каждого объекта, структура и ее элементы являются символическими.

8.2. Элемент структуры и процедуры с элементами

Элементы, принадлежащие одной и той же структуре, должны называться по-разному. Однако разные элементы могут иметь разные имена, так как на каждый предмет ссылается имя записи, к которой он принадлежит.

Структурные элементы представлены в программе:

<имя структуры><имя элемента>

Пример: stud.nomgr, stud.nom [2], stud.fam

Вот имя переменной структуры студии (название смешанной категории), nomgr, цена [2], имена fam - элементов.

Как упоминалось ранее, категория предмета может быть разной. Категория элемента может быть определена либо непосредственно в записи, либо в разделе описания категории. Категория товара, в свою очередь, может содержать запись. В этом случае записи составляют сложную структуру.

На структурных элементах вы можете выполнять действия, которые можно выполнять с данными указанной категории.

В программе входные значения для элементов категории структуры могут быть введены с использованием последовательности jin или инициализации. Например:

```
struct студент
{charФамилия[15];
Int курс;
floatотценка;};
talabas={“Абдуллаев”, 2, 5.0};
```

Пример 1. Книги предоставляются. На основании этих данных определите книги, опубликованные в 2017 году или позже.

Вид программы в режиме консоли выглядит следующим образом:

```
//---------------------------------------------------------
#include <iostream.h>
#include <conio.h>
#include <vcl.h>
#pragma hdrstop

//---------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{typedef struct
{
char   title [40];
char   author [20];
int    entry;} book;
int sum=0;
book  k;
book b[10];
int  i;
for (i=1; i<=5; i++)
{
cout<<"названиекниги"<<endl;
cin>>b[i].title;
cout<<"автор"<<endl;
cin>>b[i].author;
cout<<"дата"<<endl;
cin>>b[i].entry;
}
for (i=1; i<=3; i++)
   if (b[i].entry<=2017)  sum=sum+1;
cout<<"кол-во книг ="<<sum;
```

```
getch();
return 0;
}
//---------------------------------------------------------
```

После того, как введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

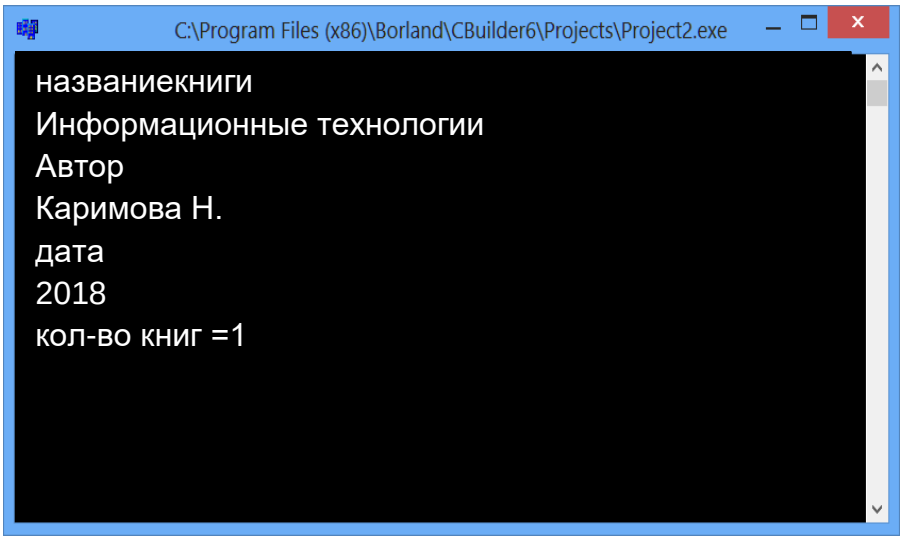


Рис. 8.1. Окно результатов.

8.3. Применение комбинированного типа на языке C++

Смешанный тип широко используется в языке программирования C++. Элементы смешанной категории -это разные категории, поэтому с ними легко работать. В качестве примера рассмотрим несколько программ в визуальной среде C ++ Builder.

Пример 2. Составьте список с именем студента и номером курса. Сделайте программу отбора на основе курса студента.

Для программирования в визуальной среде требуются 2 компонента Label, 2 компонента Edit, 7 BitBtn и 2 Memo.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 8.1 Ввод свойств компонента

Имя компонента	**Имя свойства (Состояние окна ObjectInspector)**	**Процесс, который должен быть выполнен**
Label1	Caption (Properties)	Введите слово“Введите фамилию ”.
Label2	Caption (Properties)	Введите слово“Введите курс”.
Memo1	Lines (Properties)	Слово “Memo1” будет очищено.
Memo2	Lines (Properties)	Слово “Memo2” будет очищено.
Edit1	Text (Properties)	Удаляем слово“Edit1”
Edit2	Text (Properties)	Удаляем слово“Edit2”
BitBtn1	Kind (Properties)	Выбираем функцию“bkOK”.
	Caption (Properties)	Введите слово“Вводить заметку”.
	OnClick (Events)	Вводится текст программы.
BitBtn2	Kind (Properties)	Выбирается функция “bkOK”.
	Caption (Properties)	Вводится слово “Список всех студентов”.
	OnClick (Events)	Вводится текст программы.
BitBtn3	Kind (Properties)	Выбирается функция “bkOK”.
	Caption (Properties)	Вводится слово “студенты 1 курса”.
	OnClick (Events)	Вводится текст программы.
BitBtn4	Kind (Properties)	Выбрать свойства“bkOK”
	Caption (Properties)	Вводится слово “студенты 2 курса”.
	OnClick (Events)	Вводится текст программы.
BitBtn5	Kind (Properties)	Выбирается функция “bkOK”.
	Caption (Properties)	Вводится слово “студенты 3 курса”.
	OnClick (Events)	Вводится текст программы.
BitBtn6	Kind (Properties)	Выбирается функция “bkOK”.
	Caption (Properties)	Вводится слово “студенты 4 курса”.
	OnClick (Events)	Вводится текст программы.
BitBtn7	Kind (Properties)	Выбирается функция “bkClose”.
	Caption (Properties)	Вводится слово “Выход”.
	OnClick (Events)	Вводится Close();

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

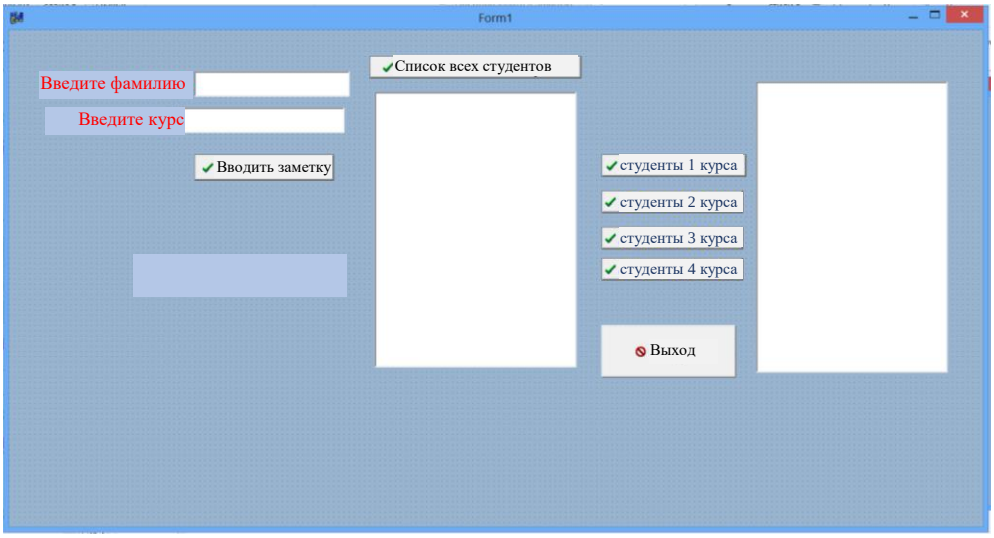


Рис. 8.2. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
TForm1 *Form1;
typedef struct
{String fam;
int    kurs;} talaba;
talabaa[10];
int n;

//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
```

```
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
n=n+1;
a[n].fam=Edit1->Text;
a[n].kurs=StrToInt(Edit2->Text);
Edit1->Text="";
Edit2->Text="" ;
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
int i;
for (i=1;i<=n; i++)
{
  Memo1->Lines->Add(a[i].fam);
  Memo1->Lines->Add(IntToStr(a[i].kurs));
}
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
int i;
Memo2->Clear();
for (i=1;i<=n; i++)
{
if (a[i].kurs==1)
{
Memo2->Lines->Add(a[i].fam);
```

```
Memo2->Lines->Add(IntToStr(a[i].kurs));
}
}
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{
int i;
Memo2->Clear();
for (i=1;i<=n; i++)
{
if (a[i].kurs==2)
{
Memo2->Lines->Add(a[i].fam);
Memo2->Lines->Add(IntToStr(a[i].kurs));
}
}
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn5Click(TObject *Sender)
{
int i;
Memo2->Clear();
for (i=1; i<=n; i++)
{
if (a[i].kurs==3)
{
Memo2->Lines->Add(a[i].fam);
Memo2->Lines->Add(IntToStr(a[i].kurs));
}
```

```
}}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn6Click(TObject *Sender)
{int i;
Memo2->Clear();
for (i=1;i<=n; i++)
{
if (a[i].kurs==4){
  Memo2->Lines->Add(a[i].fam);
  Memo2->Lines->Add(IntToStr(a[i].kurs));
}}}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn7Click(TObject *Sender)
{
Close ();
}
//---------------------------------------------------------
```

После того, как введен текст программы, будет нажата клавиша F9, что приведет к следующему виду программы:

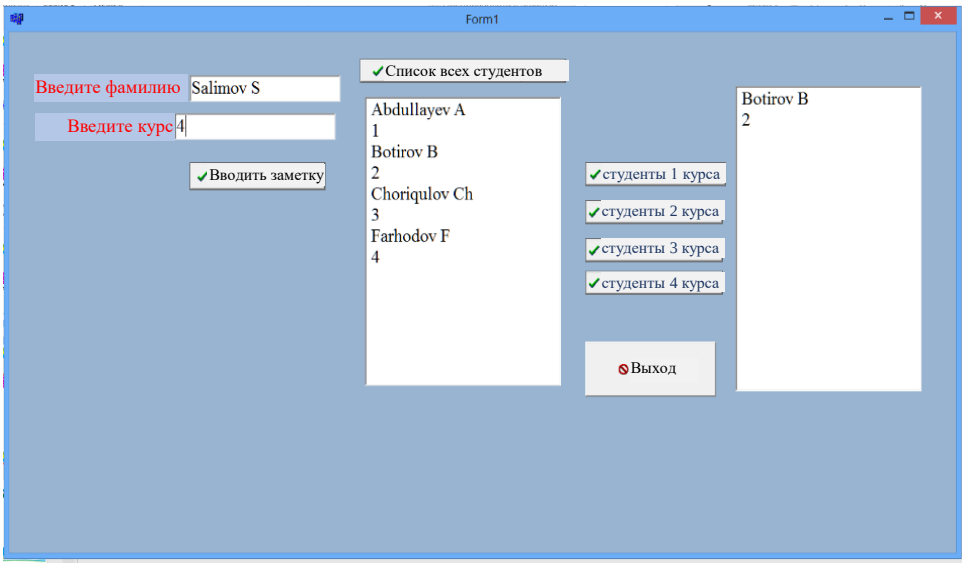


Рис. 8.3. Окно результатов.

Пример 3. Фамилия, имя, лист ответов и оценка приведены. Создать программу для подсчета очков.

Для программирования в визуальной среде требуется 4 компонента Labels, 4 Edit, 3 BitBtn и 2 Memo.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 8.2 Ввод свойств компонента

Имя компонента	**Имя свойства (Состояние окна ObjectInspector)**	**Процесс, который должен быть выполнен**
Form1	Caption (Properties)	Вводится слово “Абитуриент”.
Label1	Caption (Properties)	Вводится слово “Фамилия абитуриента”.
Label2	Caption (Properties)	Вводится слово “Имя абитуриента”.
Label3	Caption (Properties)	Вводится слово“Лист ответов”.
Label4	Caption (Properties)	Вводится слово “Набранныебалы”.
Edit1	Text (Properties)	Удаляем слово “Edit1” .
Edit2	Text (Properties)	Удаляем слово “Edit2” .
Edit3	Text (Properties)	Удаляем слово “Edit3” .
Edit4	Text (Properties)	Удаляем слово “Edit4” .
Memo1	Lines (Properties)	Удаляемслово“Memo1”.
BitBtn1	Kind (Properties)	Выбираем функцию “bkOK”.
	Caption (Properties)	Вводится слово “Введем запись ”.
	OnClick (Events)	Вводится текст программы.
BitBtn2	Kind (Properties)	Выбираем функцию “bkOK”.
	Caption (Properties)	Вводится слово “Сортировка”.
	OnClick (Events)	Вводится текст программы.
BitBtn3	Kind (Properties)	Выбираем функцию “bkIgnore”.
	Caption (Properties)	Вводится слово “Выход”.
	OnClick (Events)	Вводится Close();

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

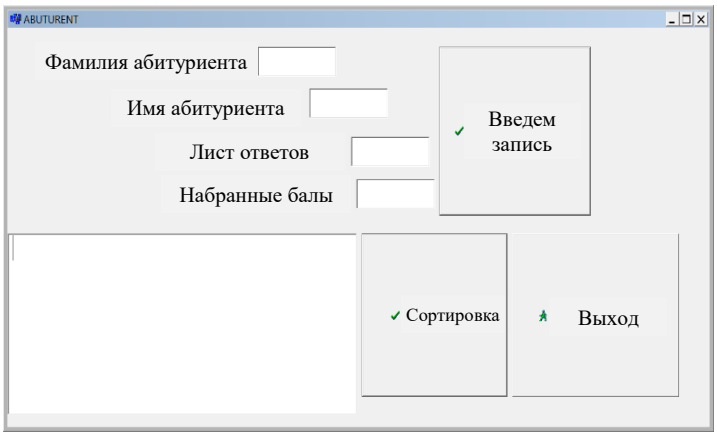


Рис. 8.4. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
typedef struct
{String fam;
String ism;
String jav;
int ball;} abuturent;
abuturent ab [10];
int i,j,m; abuturent k; int n;
//---------------------------------------------------------
```

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
n=n+1;
ab[n].fam=Edit1->Text;
ab[n].ism=Edit2->Text;
ab[n].jav=Edit3->Text;
ab[n].ball=StrToInt(Edit4->Text);
Edit1->Text="";
Edit2->Text="" ;
Edit3->Text="" ;
Edit4->Text="" ;
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
 Memo1->Clear();
 for (i=1;i<=n-1; i++)
 for (j=i+1;j<=n;j++)
  {
 if  (ab[i].ball<ab[j].ball)
   {
k=ab[i];
 ab[i]=ab[j];
 ab[j]=k;
}}
```

```
    for (i=1;i<=n; i++)
    {
Memo1->Lines->Add(IntToStr(i) + ". " + ab[i].fam + " " + ab[i].ism + "  " +
ab[i].jav + "  " + IntToStr(ab[i].ball));
}}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{Close ();}
//---------------------------------------------------------
```

После ввода текста программы нажмите F9, и появится следующая версия программы:

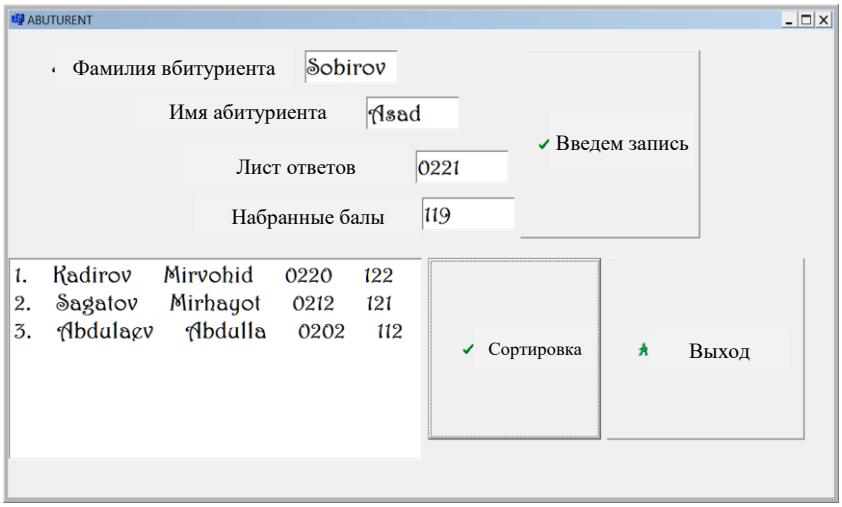


Рис. 8.5. Окно результатов.

Пример 4. Студентам предоставляется следующая информация: фамилия, имя, информационные технологии, физика и математика.

Создайте отличную программу отбора студентов, основанную на баллах студентов.

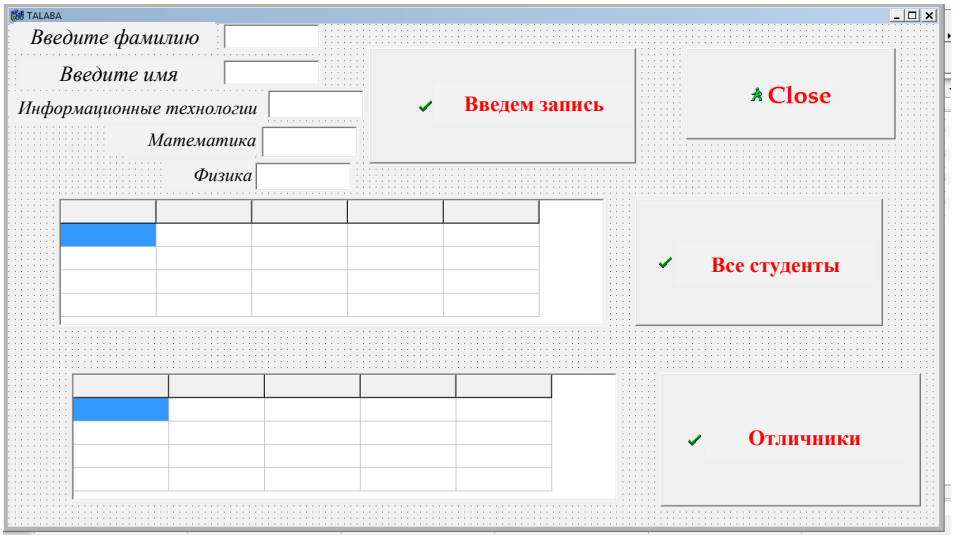
Вам понадобится 5 компонентов Labels, 2 StringGrid, 4 BitBtn и 5 Edit для программирования вашего примера в визуальной среде.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 8.3 Ввод свойств компонента

Имя компонента	**Имя свойства (Состояние окна ObjectInspector)**	**Процесс, который должен быть выполнен**
Form1	Caption (Properties)	Вводится слово “Студент”.
	OnCreate (Events)	Вводится текст программы.
	OnCloseQuery(Events)	Вводится текст программы.
Label1	Caption (Properties)	Вводится слово “Введите фамилию”.
Label2	Caption (Properties)	Вводится слово “Введите имя”.
Label3	Caption (Properties)	Вводится слово “Информационные технологии”.
Label4	Caption (Properties)	Вводится слово “Математика”.
Label5	Caption (Properties)	Вводится слово “Физика”.
Edit1	Text (Properties)	Удаляется запись“Edit1”
Edit2	Text (Properties)	Удаляется запись “Edit2”
Edit3	Text (Properties)	Удаляется запись “Edit3”
Edit4	Text (Properties)	Удаляется запись “Edit4”
Edit5	Text (Properties)	Удаляется запись “Edit5”
BitBtn1	Kind (Properties)	Выбирается свойство “bkOK”.
	Caption (Properties)	Вводится слово “Види запись”
	OnClick (Events)	Текст программы введен.
BitBtn2	Kind (Properties)	Выбирается свойство “bkOK”.
	Caption (Properties)	Вводится слово “Все студенты”.
	OnClick (Events)	Текст программы введен.
BitBtn3	Kind (Properties)	Выбирается свойство “bkOK”.
	Caption (Properties)	Вводится слово “Отличники”.
	OnClick (Events)	Вводиться текст программы
BitBtn4	Kind (Properties)	Выбирается свойство “bkClose”
	Caption (Properties)	Вводится слово “Выход”.
	OnClick (Events)	Вводится Close();.
StringGrid1	FixedCols (Properties)	Введем цифру 0
	FixedRows(Properties)	Введем цифру 1
	ColCount (Properties)	Введем цифру 5
	RowCount (Properties)	Введем цифру 5
StringGrid2	FixedCols	Введем цифру 0
	FixedRows	Введем цифру 1
	ColCount (Properties)	Введем цифру 5
	RowCount	Введем цифру 5

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:



Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"

//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//уровень
typedef struct {
    String наука;
    String Имя;
    int информация;
    int мат;
    intфиз; }студент ;
```

```
студента[30];
int i,j,n;

//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
StringGrid1->Cells[0][0]="Фамилия";
 StringGrid1->Cells[1][0]="Имя";
StringGrid1->Cells[2][0]="Математика";
StringGrid1->Cells[3][0]="Информационные технологии";
StringGrid1->Cells[4][0]="Физика";
 StringGrid2->Cells[0][0]="Фамилия";
 StringGrid2->Cells[1][0]="Имя";
 StringGrid2->Cells[2][0]="Математика";
 StringGrid2->Cells[3][0]="Информационныетехнологии ";
 StringGrid2->Cells[4][0]="Физика";
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
n=n+1;
a[n].fam=Edit1->Text;
a[n].ism=Edit2->Text;
a[n].axborot=StrToInt(Edit3->Text);
a[n].mat=StrToInt(Edit4->Text);
```

```
a[n].fiz=StrToInt(Edit5->Text);
Edit1->Text="";
Edit2->Text="" ;
Edit3->Text="" ;
Edit4->Text="" ;
Edit5->Text="" ;
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
j=0;
for (i=1;i<=n; i++)
 {
StringGrid1->Cells[j][i]=a[i].наука;
 StringGrid1->Cells[j+1][i]=a[i].Имя;
 StringGrid1->Cells[j+2][i]=IntToStr(a[i].информация);
 StringGrid1->Cells[j+3][i]=IntToStr(a[i].мат);
 StringGrid1->Cells[j+4][i]=IntToStr(a[i].физ);
}
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
j=0;
for (i=1;i<=n; i++)
   if (a[i].информация>85 && a[i].мат>85 && a[i].физ>85)
 {
StringGrid2->Cells[j][i]=a[i].наука;
 StringGrid2->Cells[j+1][i]=a[i].Имя;
 StringGrid2->Cells[j+2][i]=IntToStr(a[i].информация);
```

```
 StringGrid2->Cells[j+3][i]=IntToStr(a[i].мат);
 StringGrid2->Cells[j+4][i]=IntToStr(a[i].физ);
}
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{
Close();
}
//---------------------------------------------------------
void__fastcall TForm1::FormCloseQuery (TObject *Sender, bool &CanClose)
{
if(MessageDlg(“Выйтиизпрограммы", mtInformation, TMsgDlgButtons()
<<mbYes<<mbNo,0)==mrYes)
{
CanClose=true;
}
else
{
CanClose=false;
}
}
//---------------------------------------------------------
```

После ввода текста программы нажмите F9, и появится следующая версия программы:

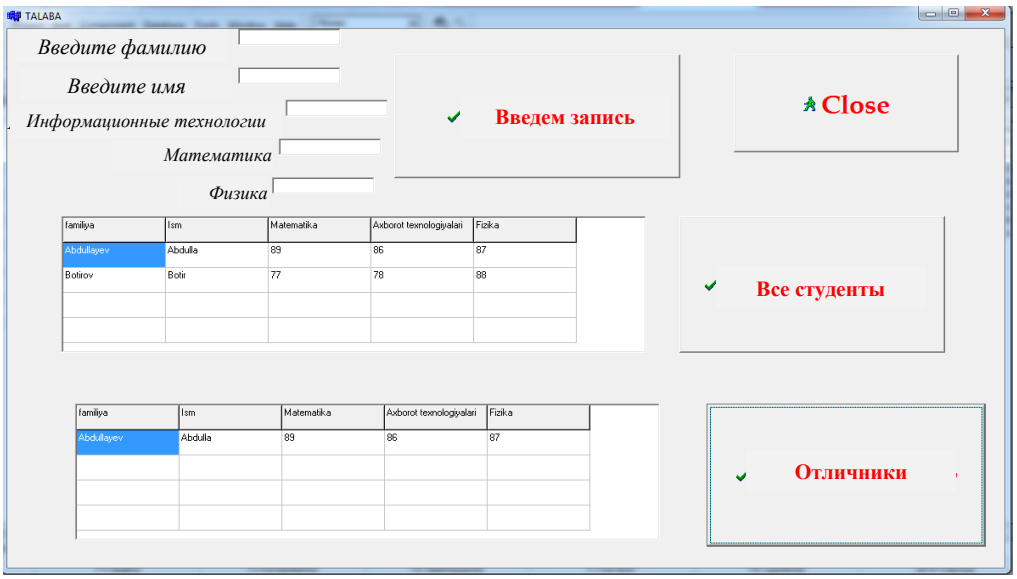


Рис. 8.7. Окно результатов.

Вопросы по главе 8

1. В чем причина использования структурной категории данных в программе?

2. Каковы методы описания смешанной категории в программе?

3. Элемент структуры, какой может быть его категория?

4. Действия над элементом структуры?

5. Как ввести данные смешанной категории?

Тестовые вопросы:

1. Что такое структура в C ++ ?

а) Ограниченный набор данных разных категорий;

б) Одноименные данные, включая функции;

в) это серия именованных типов данных;

г) Управляет свойствами компонента, установленного в форме;

2. Какой сотрудник начинает с описания структуры?

a) typedef struct;

b) void fastcall;

в) include<fstream>;

г) Typef структура;

3. Сколько разных категорий может быть объявлено в смешанной категории?

а) только одна категория ;

б) Разные категории ;

в) многомерные массивы ;

г) Только категории Int и float;

4. Какая строка категории структуры (составная категория) правильно указана?

a) {float re;floatim;}Complex;Complex a[100];

b) typedef struct {float re;floatim;} Complex;

d) structure {float re; float im;} студент;

e) студент {float re;floatim;} typedef struct;

5. struct {String наука; intфиз; } студент; что это значит?

а) Функция (многопараметрическая) ;

б) Массивный (многомерный) ;

г) структура (смешанная категория) ;

д) Процедура (названа в честь студента).

Глава 9. ФУНКЦИИ И ПРОЦЕДУРЫ С++. ПРИМЕНЕНИЕ МЕТОДОВ СТРУКТУРНОГО ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ С++

Ключевые слова: программы обработки детали, локальные переменные, глобальные переменные, функции, функции по умолчанию, категория функции, ссылка на программу обработки детали, имя функции, официальные и истинные параметры.

9.1. Функции на языке С++ и их использование

На практике существуют некоторые проблемы, которые необходимо повторить несколько раз при разных значениях переменных, которые изменяют не только одну функцию, но и несколько функций.

Реплицирующие команды могут быть записаны один раз в программе и обращаться к ним можно по мере необходимости, чтобы упростить программу для решения таких проблем.

Набор операторов, которые могут выполняться много раз в результате применения из разных частей программы, называется «процедурой» (программой обработки деталей), а программа, которая содержит процедуру, называется «основной программой».

В С ++ программы обработки деталей могут быть организованы в два типа: процедура и функция. Программы в этом представлении содержатся в основной программе, и они также имеют свои собственные заголовки и содержимое.

Функция является логически завершенной программной частью. Это поможет им избежать необходимости писать большие и сложные вычисления и упростит реализацию программы. Его можно настроить один раз и запустить, и к нему можно получить доступ из любой точки программы. При организации функции предоставляется информация о категории функции, ее

названии и ее параметрах. Эти параметры называются формальными параметрами, значение которых определяется временем вызова функции.

Функция обычно записывается следующим образом:

Категория функции Имя функции (официальные настройки)

```
{
    функциональное тело;
}
```

Имя функции может быть необязательно латинским словом, а официальные параметры - необязательными переменными.

Перед применением к функции необходимо определить фактические параметры, которые заменят официальные параметры. Применяя это следующим образом:

переменная = имя функции (фактические параметры);

Количество формальных и фактических параметров, их классификация и последовательность прибытия должны быть согласованы. Имена формальных и фактических параметров могут быть одинаковыми. При объявлении функции в главной функции можно перечислить только их категории без указания фактических имен параметров, например:floatmax (float, float);

Переменные, которые могут использоваться в основной программе, также называются глобальными переменными. Глобальные переменные должны быть объявлены в основной программе. Только переменные, которые могут использоваться внутри функции, называются локальными переменными. Они объявлены в функции.

Функции могут быть определены до и после функции main (). Если он определен перед основной функцией, его не нужно объявлять отдельно в функции main (); Например, в следующей программе функция funk была объявлена и определена после основной функции:

```
#include <vcl.h>
#pragma hdrstop
#include <math.h>
```

```
# include <iostream.h>
# include <conio.h>

//---------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{ int k, n, funk (int n);
cin>>n;
  k=funk(n);
cout<< "k="<<k<<endl;
getch( );
} intfunk(inta)   // определение функции

{ int c;
    c=1+sin(a)+cos(a);
    returnc; // вернуть результат в функцию
}
```

Тело функции, упомянутой выше, также может быть записано:

```
int funk(int a)
{
return 1+sin(a)+cos(a);
}
```

Если функция объявлена перед основной функцией, программу можно написать следующим образом:

```
//---------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <math.h>
# include <iostream.h>
# include <conio.h>
```

```
//---------------------------------------------------------
#pragma argsused
 int funk (int a)
  {    return 1+sin(a)+cos(a);   }
   int main(int argc, char* argv[])
{ int k, n ;
cin>>n;
    k=funk(n);
cout<< "k="<<k<<endl;
getch( );
return 0;   }
  //---------------------------------------------------------
```

Это означает, что при доступе к функциям операции выполняются следующим образом:

1. При выполнении функции память резервируется для формальных параметров, то есть они преобразуются во внутренние параметры функции. Это изменит категорию параметров: категория с плавающей запятой будет двойной, а категории char и shortint будут преобразованы в int.

2. Фактические значения параметров даны или рассчитаны.

3. Фактические параметры записываются в память, разделенную на формальные параметры.

4. Тело функции выполняется с использованием внутренних параметров, и значение отправляется в место возврата.

5. При выходе из функции память, выделенная для формальных настроек, освобождается.

Пример 1. Установите функцию, чтобы найти наибольшее из двух произвольных чисел. Вид программы в режиме консоли выглядит следующим образом:

```
//---------------------------------------------------------
```

```
#include<vcl.h>
#pragma hdrstop
#include <math.h>
# include <iostream.h>
# include <conio.h>
//---------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{ float x=20, b=40, c, max(float x, float y);
    c = max(x, b);
cout<< "c="<<c<<endl;
getch( );
}  float max(float x, float y)
{ if  (x > y)   return x;
            else
return;
return 0;     }
//---------------------------------------------------------
```

После ввода текста программы нажмите F9, и появится следующая версия программы:

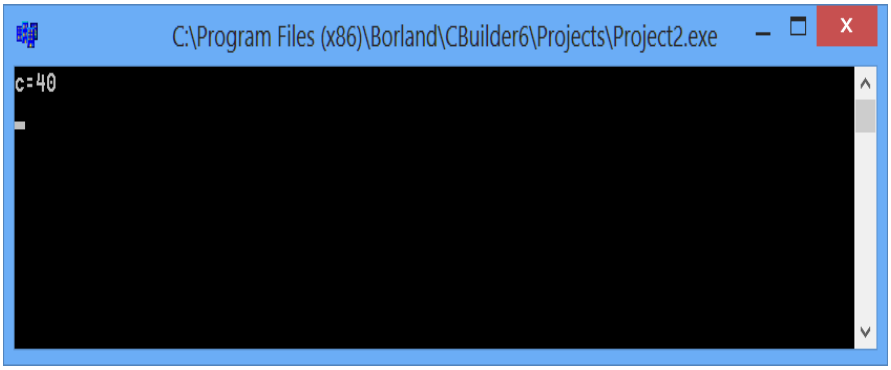


Рис. 9.1. Окно результатов.

В этом примере при объявлении функции max формальные параметры не указываются, тело функции использует 2 оператора возврата.

9.2 Процедуры и их организация

В некоторых случаях при работе с функцией необходимо изменить фактические значения параметров в теле функции, то есть результат должен быть больше одного. Желательно выполнять такой процесс с использованием процедурных функций. При указании функции имена результатов отображаются вдоль линии с формальными параметрами. Поэтому при работе с процедурами желательно использовать класс функции как пустой (void), но без использования оператора return.

Следующий программный код является примером:

```
double FSum(double X1,double X2, int A);
void SPrint(AnsiString S);
void F1(void);
void SPrint(AnsiString S)
{ if (S != "")
ShowMessage(S);}
```

Его легко использовать в программировании, и достаточно просто обратиться к его названию. Например:

```
voidshowmes()
{ShowMessage(“Применить функции и процедуры");}
void __fastcallTForm1::Button1Click(TObject *Sender)
{showmes;}
//---------------------------------------------------------
```

Пример 2. Создайте программу под названием «Рассчитать». Создайте программу для определения чисел и вычисления среднего арифметического программы. Реализуйте программу в визуальной среде.

Вам понадобятся 1 компонент Label и 4 компонента BitBtn для программирования вашего примера в визуальной среде.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 9.1 Ввод свойств компонента

Имя компонента	**Имя свойства (Состояние окна ObjectInspector)**	**Процесс, который должен быть выполнен**
Form1	Caption (Properties)	Вводится слово “Работа с функциями и процедурами”.
	FormCloseQuery (Events)	Вводится текст программы.
Label1	Caption (Properties)	Вводится слово“Результат”.
BitBtn1	Kind (Properties)	Выбираем свойство“bkOK”.
	Caption (Properties)	Вводится слово “Нахождение количества и средней арифметической элементов”.
	OnClick (Events)	Вводится текст программы.
BitBtn2	Kind (Properties)	Выбираемсвойство“bkOK”.
	Caption (Properties)	Вводится слово “Нахождение количества и средней арифметической элементов”.
	OnClick (Events)	Вводится текст программы.
BitBtn3	Kind (Properties)	Выбираем свойство“bkOK”.
	Caption (Properties)	Вводится слово “Нахождение количества и средней арифметической элементов”.
	OnClick (Events)	Вводится текст программы.
BitBtn4	Kind (Properties)	Выбираем свойство “bkClose”.
	Caption (Properties)	Вводится слово“Выход”.
	OnClick (Events)	Вводится Close();

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

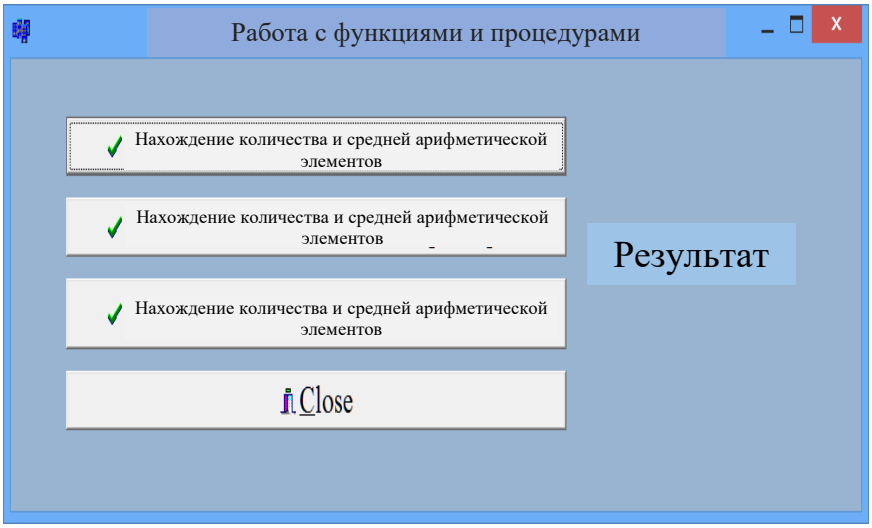


Рис. 9.2. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"

//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

//---------------------------------------------------------
#pragma package(smart_init)
void расчет(AnsiString mess, int N,...)
{
double A = 0;
```

```
va_list ap;
va_start(ap, N);
for(int i = 0; i< N; i++)
A += va_arg(ap,int);
Form1->Label1->Caption = mess + "N = " +IntToStr(N) + ",средный = " +
FloatTo Str(A/N);
va_end(ap);
}
//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{}
//---------------------------------------------------------
void __fastcall TForm1::FormCloseQuery(TObject *Sender, bool &CanClose)
{
if(MessageDlg("Выйтиизпрограммы", mtInformation, TMsgDlgButtons()
<<mbYes<<mbNo,0)==mrYes)
{CanClose=true;}
else
{
CanClose=false;
}}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
Расчет ("результат: ",5,4,2,3,5,4);
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
```

```
Расчет ("результат: ",6,4,2,3,5,4,10);
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
Расчет ("результат: ",8,4,2,3,5,4,8,11,12);
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{Close();}
//---------------------------------------------------------
```

После ввода текста программы нажмите F9, и появится следующая версия программы:

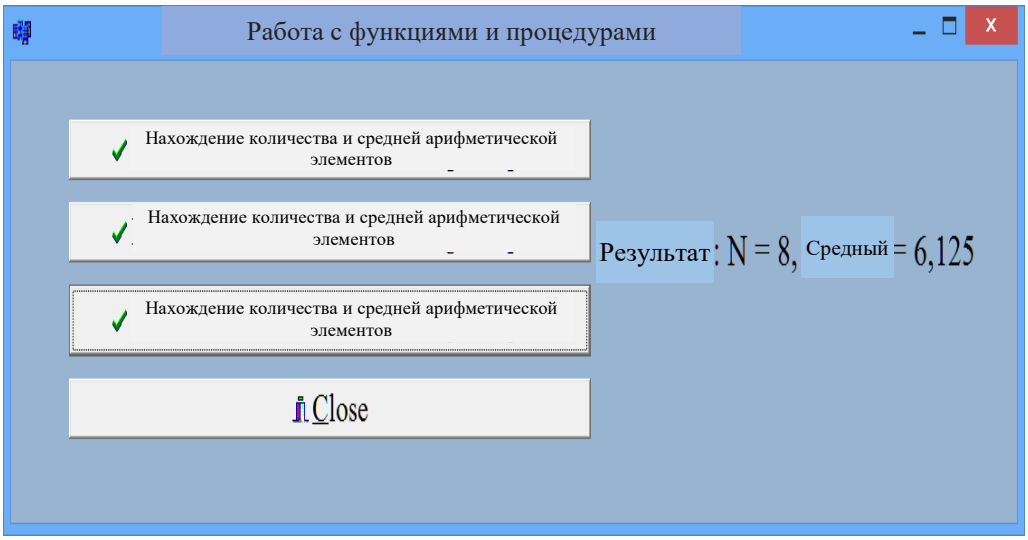


Рис. 9.3. Окно результатов.

9.3 Использование массивов в функциях

Массивы также могут быть использованы в качестве официальных параметров. Он указывает тип массива, имя и скобки [], а также размер массива. Например: voidmas(inta[], intn);

В программе при вызове функции указывается только имя массива, а скобки [] не нужны:

```
int num=0;
int first[6][9], second[6][9], third[6][9];
void Mass(int zero[6][9])
{for(int i=0;i<6;i++)
for(int j=0;j<9;j++)
      zero[i][j]=num;
   num++;}
```

Функция выполняется следующим образом:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{Mass(first);
Mass(second);
Mass(third);}
```

Когда мы передаем массивы в функцию, одна из проблем заключается в назначении количества элементов в массиве. В этом случае один из оптимальных вариантов - объявить размер массива в функции через дополнительный формальный параметр.

Пример 3. Рассчитайте следующую функцию:

$$y = \frac{\sqrt{AXT^2 + 1.3}}{b * e^{-xc}} * \sin t$$

Сдесьт = $2x^2$; (t = $2x^2$)

X: уравнение X2 + 2X + 4 = 0 [1; 2] найти приблизительный корень ошибки с помощью итерационного метода

$$c = \int_{0.6}^{1.5} \frac{dx}{\sqrt{x^2 + 1}}, n = 16$$

Вычислить интеграл, используя метод Симпсона.

A: Max$\{a_i\}$, $i \in 1 \div 20$; $a_i \in [1 \div 100]$, A(20)

B: Max$\{b_i\}$ $i \in 1 \div 20$; $b_i \in [1 \div 50]$, B(20)

Вычислите максимальное значение членов массивов A и B, используя программы разбиения.

Вам понадобятся 14 компонентов Labels, 2 Button, 2 StringGrid и 1 image для программирования вашего примера в визуальной среде.

Мы определяем свойства компонента окна формы следующим образом:

Таб.9.2 Ввод свойств компонента

Имя компонента	Имя свойства (Состояние окна ObjectInspector)	Процесс, который должен быть выполнен
Form1	Caption (Properties)	Вводится слово “Функция и процедура”.
Label1	Caption (Properties)	Вводится слово “Значение A”.
Label2	Caption (Properties)	Вводится слово “Значение T”.
Label3	Caption (Properties)	Вводится слово “Значение C”.
Label4	Caption (Properties)	Останется слово“Label 4”.
Label5	Caption (Properties)	Останется слово“Label 5”.
Label6	Caption (Properties)	Останется слово “Label 6”.
Label7	Caption (Properties)	Вводится слово “Значение B”.
Label8	Caption (Properties)	Вводится слово “Значение X”.
Label9	Caption (Properties)	Останется слово “Label 9”.
Label10	Caption (Properties)	Останется слово “Label 10”.
Label11	Caption (Properties)	Вводится слово “Значение функции”.
Label12	Caption (Properties)	Останется слово“Label 12”.
Label13	Caption (Properties)	Вводится слово“a[i]=”.
Label14	Caption (Properties)	Вводитсяслово“b[i]=”.
BitBtn1	Kind (Properties)	Выбираемсвойство“bkOK”.
	Caption (Properties)	Вводится слово “расчет”.
	OnClick (Events)	Вводится текст программы.
StringGrid1	FixedCols (Properties)	Вводим цифру 0
	FixedRows (Properties)	Вводим цифру 0
	ColCount(Properties)	Вводим цифру 20
	RowCount (Properties)	Вводим цифру 1
StringGrid2	FixedCols (Properties)	Вводим цифру 0
	FixedRows (Properties)	Вводим цифру 0
	ColCount (Properties)	Вводим цифру 20
	RowCount (Properties)	Вводим цифру 1

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

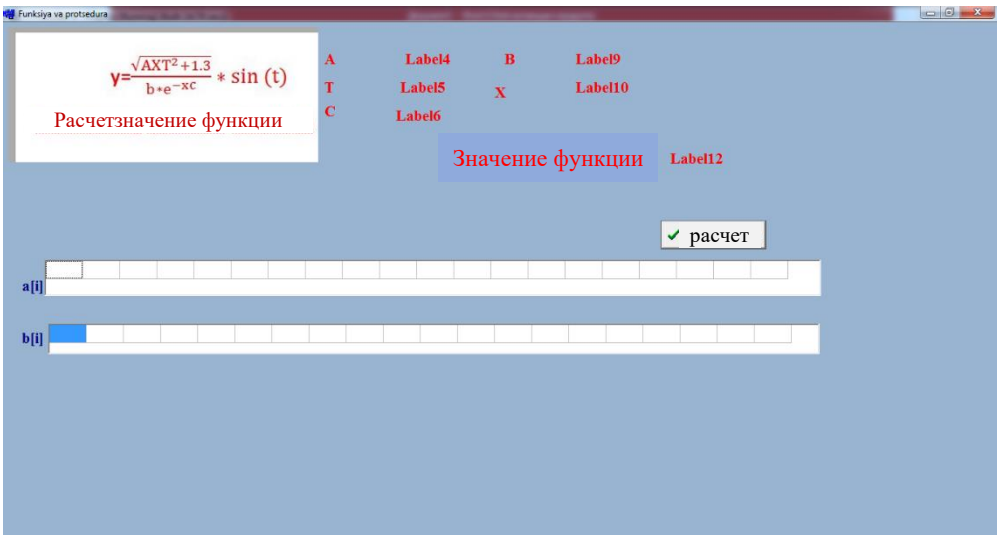


Рис. 9.4. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------
 #include <iostream.h>
#include <math.h>
#include <stdio.h>
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"

//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
// Рассчитатьфункцию A
int  значение_A(int );
int значение_A(int a[20])
{
int max;
```

```
max=a[0];
for(int i=1; i<20; i++)
if(a[i]>max) max=a[i];
returnmax;
}

//-----------------------------------------------------
// Рассчитать функцию X
floatзначение_X();
floatзначение_X();
{
floaty,y0=1;
y=-(y0*y0+4)/2;
if(fabs(y-y0)>0.0004)
{
y0=y;
y=-(y0*y0+4)/2;
}
return 1.5;
}
//-----------------------------------------------------
// Рассчитать функцию T
floatзначение_C();
floatзначение_C();
{
floatt;
t=2*pow(значение_X(),2);
return t;
}
//-----------------------------------------------------
```

```
// Рассчитать функцию C
floatзначение_C();
floatзначение_C();
{
float c=0,a,b,h;
a=1.5;
b=0.6;
h=(a-b)/16;
for(int i=0;i<16; i++)
{
c=c+(b+h)/sqrt(pow(значение_X(),2)+1);
b=b+h;
}
return c;
}
//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
float y;
int a[20], b[20];
for(int i=0; i<20; i++)
{
a[i]=random(100);
StringGrid1->Cells[i][0]=a[i];
b[i]=random(50);
```

```
StringGrid2->Cells[i][0]=b[i];
}
Label4->Caption=FloatToStr(значение_A(a));
//
Label9->Caption=FloatToStr(значение_A(b));
//
Label5->Caption=FloatToStr(значение_T());
//
Label10->Caption=FloatToStr(значение_X());
//
Label6->Caption=FloatToStr(значение_C());
//
y=sqrt(значение_A(a)*значение_X()*значение_X()*значение_T()+1.3)*sin(
значение_T())/(значение_A(b)*pow(2.7, значение_Xi ()* значение_C()));
Label12->Caption=FloatToStr(y);
}
//---------------------------------------------------------
```

После ввода текста программы нажмите F9, и появится следующая версия программы:

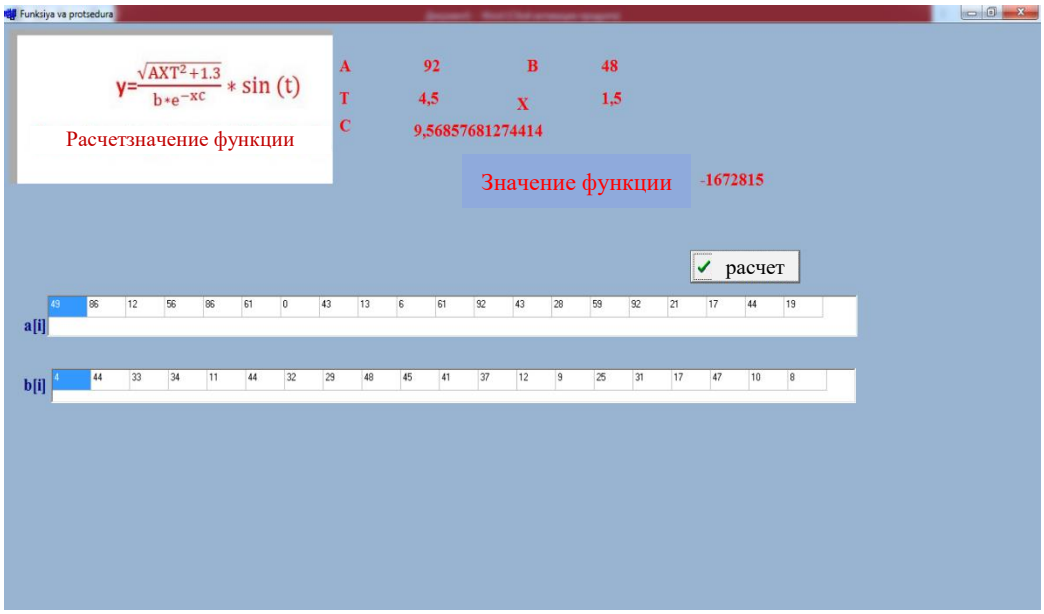


Рис. 9.5. Окно результатов.

Вопросы поглаве 9

1. Частичное программирование на C ++ и его применение.
2. Роль функции в программе.
3. Как применять функции?
4. Реальные параметры и их использование.
5. Функция и ее особенности.

Тестовые вопросы:

1. В C ++ функция

а) Компонент, который хранит эти компоненты;

б) это ключевая часть программы;

в) часть, которая содержит математические функции;

г) логически завершенный программный компонент;

2. Выберитеправуюстрокуфункции.

а) Function (float, int, char, string);

б) float max(float , float);

в) Max int (matn);

г) Funktion max(int);

3. Функция создается и записывается один раз, и к ней можно получить доступ из любой точки программы. Как подать заявку?

а) через категорию функции;

б) через ложные параметры;

в) по названию функции;

г) через его фактические параметры;

4. Каковы параметры функции?

а) Программное и аппаратное обеспечение;

б) Формальный и действительный;

в) Maxint (текст);

г) имя и категория;

5. В какой форме объявляются массивы в функции?

а) abc int [2];

б) typedef struct matrix (int a[11], int n);

в) a int [2][3];

г) void mas (int a[4], int n);

Глава10. ФАЙЛОВЫЕ И ССЫЛОЧНЫЕ ТИПЫ ДАННЫХ В СИСТЕМЕ ПРОГРАММИРОВАНИЯ С++

Ключевые слова: файл, категория файла, индекс файла, элемент файла, динамическая память, текстовые файлы, двоичные файлы, стандартные потоки, потоки ввода / вывода, справочная категория, индексные функции.

10.1. Файловый тип данных

До сих пор мы сталкивались с несколькими категориями языка программирования С ++. Это масштабируемые, простые и сложные категории. При решении проблем с этой категорией информации исходные данные вводятся с клавиатуры в оперативную память и отображается результат. Их нельзя использовать в других программах, поскольку они никогда не сохраняются после выхода из системы, и они также содержат заранее определенное количество конечных элементов. Даже если это обнаруживается динамически, эти данные состоят из нисходящих элементов из-за ограниченного объема оперативной памяти. Многие практические вопросы требуют постоянного хранения данных. Например, данные о персонале должны быть постоянными. С ++ имеет категорию файлов для хранения этих данных. Категория файлов занимает особое место. При работе с категорией файлов вам необходимо освоить определенные понятия.

Файл является именованной частью внешнего хранилища для хранения данных. Такие файлы называются физическими файлами.

Другим важным понятием является понятие индекса файла. Индекс файла - указывает, где вы в данный момент читаете или записываете в файл (место записи), то есть вы можете прочитать одну запись из файла индекса файла или поместить туда новую запись.

Ссылка на записи файла производится последовательно: для чтения n-записи необходимо прочитать n-1. Следует отметить, что процесс чтения из

файла был частично «автоматизирован», так что после чтения i-записи индекс отображается в начале следующей записи i + 1, так что чтение может быть продолжено (так как индекс не нужно увеличивать). Файл является местом хранения данных и не может быть напрямую доступен через его записи. Чтобы выполнить файл, значение записи должно быть прочитано в переменную типа, соответствующего операционной памяти. Впоследствии над этой переменной будут выполнены необходимые действия, и результаты при необходимости могут быть снова записаны в файл.

C ++ - это логическая концепция файла, которая относится к переменному типу файла. Переменные типа файла не могут быть назначены оператору типа значения, как другие переменные. Другими словами, никакие действия с переменными типа файла не определены. Все выполняемые над ними функции выполняются через функции. Работа с файлами включает следующие шаги:

Definitely Чейнджер файлов определенно связан с файлом на диске; Opens файл открывается;

– операции записи или чтения выполняются над файлом;

– файл закрыт;

– Вы можете переименовать или удалить файл с диска.

Текстовые и бинарные файлы. C ++ наследует C-библиотеку по умолчанию, стандартную библиотеку функций для чтения и записи. Эти функции объявлены в заголовочном файле <stdio.h>.

Операции чтения-записи выполняются с файлами. Файл может быть, как текстовым, так и двоичным. Текстовый файл представляет собой набор символов в коде ASCII. Последовательность символов подразделяется на строки и представляет собой пару символов CR ('\ r') и LF ('\ n') в качестве конца строки. При чтении данных из текстового файла эта пара символов заменяется одним символом CR, и наоборот, символ CR заменяется двумя символами CR и LF. Конец файла помечен # 26 (^ Z).

Существует также другое определение для текстового файла. Если файл можно отобразить и прочитать в текстовом редакторе, это текстовый файл.

Другими словами, все данные, отображаемые программой, можно просматривать как текстовый файл stdout. Точно так же любые данные, считанные с клавиатуры, считаются считанными из текстового файла. Компоненты текстовых файлов называются строками.

Строки могут быть расположены непрерывно, различной длины и пространства.

Двоичные файлы - это просто последовательность байтов. Обычно нет необходимости «напрямую» просматривать содержимое двоичных файлов пользователем. При чтении из бинарных файлов преобразование байтов не производится.

10.2. Функции чтения-записи из файла

Вам нужно открыть поток файлов для операции чтения / записи потока файлов. Прототип этого действия выглядит следующим образом:

FILE * fopen (constchar * имя файла, режим constchar *);

Второй параметр mode в списке параметров определяет режим открытия файла. Значения, которые он может принять, приведены в таблице 10.1.

Таб. 10.1 Режимы открытия файлов

Значение режима	Описание состояния открытия файла
R	Файл только для чтения
W	Открывается для записи файла. Если такой файл существует, он будет перезаписан.
A	Как добавить файл в файл. Если файл существует, файл будет открыт в конце файла для записи, в противном случае новый файл будет создан и открыт в режиме записи.
r+	Существующий файл открывается для модификации (чтение и запись).
w+	Новый файл создается и открывается для редактирования (чтение и запись). Если файл существует, предыдущие записи в нем будут удалены, и они будут перезаписаны.
a+	Как добавить файл в файл. Если файл существует, он открывается в конце файла (после EOF) для записи (чтения), в противном случае новый файл будет создан и открыт в режиме записи.

Это делается функцией fopen (). Функция открывает файл с именем файла, связывает с ним поток и возвращает идентификатор потока. Fopen () возвращает нулевое значение, которое файл не удалось открыть.

Вам нужно набрать 't' в строке режима открытия файла, чтобы указать, что текстовый файл открывается. Например, вы должны набрать «rt +», чтобы указать, что текст открыт для редактирования (чтения и записи) файлов. Вы также должны использовать 'b' для работы с двоичными файлами. Например, «wb +» открытия файла означает обновление двоичного файла.

Когда файл открывается для редактирования (чтение-запись), можно считывать данные из потока, а также записывать в поток. Тем не менее, он не может быть прочитан сразу после операции записи, так как функция fseek () или rewind () вызывается перед операцией чтения.

Предположим, текстовый файл с именем «C: \ Users \ admin \ Desktop \ Request \ group.txt» должен быть открыт для чтения. Это требование выполняется путем написания следующего заявления:

FILE * f = fopen ("C: \ Users \ admin \ Desktop \ Request \ group.txt", "r +");

В результате файл на диске понимается точно так же, как переменная f в программе. Другими словами, все действия, выполняемые над «f» позже в программе, будут происходить с файлом «band.txt» на диске.

После завершения потока файлов его следует закрыть. Вы можете использовать функцию fclose () для этого. Прототип функции выглядит так:

int fclose(FILE * stream);

Функция fclose () очищает связанные с потоком буферы (например, при копировании инструкций в файл копирует данные, собранные в буфере, в файл на диске) и закрывает файл. Если закрытие файла вызывает ошибку, функция возвращает значение EOF, равное 0.

Вот некоторые функции для чтения и записи данных из файла:

1. Функцияfgetc () выглядиттак:

int fgetc(FILE *stream);

Прототип функции, как определено в приведенной выше форме, считывает символ из потока файлов. Если чтение успешно, функция преобразует прочитанный символ в целое число без типа int. Если была сделана попытка прочитать конец файла или произошла ошибка, функция возвращает EOF.

Как видите, функции getc () и fgetc () работают почти одинаково, различие в том, что функция getc () читает символ из потока по умолчанию. Другими словами, функция getc () - это макрос, определенный функцией fgetc (), которая является стандартным устройством для обработки файлов.

2. Функцияfputc () выглядиттак:

int fputc(int c, FILE *stream);

Функция fputc () записывает (выводит) файл, указанный в аргументе, в поток файлов, и он аналогичен функции putc ().

3. Чтобы прочитать строку из файлового потока, используйте следующую функцию:

char * fgets(char * s, int n, FILE *stream)

Функция fgets () считывает последовательность символов из потока файлов в строку «s». Функция прекращает чтение после чтения знака n-1 из потока или после пропуска следующего символа ('\ n'). В последнем случае символ '\ n' также добавляется к строке 's'. В конце строки 's' символ '\ 0' добавляется в конец строки. Если прочитанная строка успешна, функция возвращает строку аргумента, в противном случае она получает нулевое значение.

4. Строка может быть выведена в файловый поток с помощью функции fputs (). Эта функция определяется следующим образом.

int fputs (const char *s, FILE *stream);

Переход на новую строку в конце строки не происходит. Если вывод успешен, функция возвращает неотрицательное число, в противном случае выполняется EOF.

5. Функция fef () на самом деле является макросом, который указывает, встречается ли конец файла во время операций чтения-записи файла. Функция выглядит так:

int feof(FILE *stream);

Он возвращает ненулевое число, если встречается конец файла, в противном случае он возвращает 0.

10.3. Ввод – вывод файловых объектов

В С ++ есть классы потоков ввода-вывода, которые являются объектно-ориентированным эквивалентом стандартной библиотеки ввода-вывода. К ним относятся следующие директивы:

•istream - входной поток

•ostream - поток нагнетания

•iostream - поток ввода / вывода

Следующие глаголы используются для ввода и вывода данных из строковых буферов, хранящихся в памяти:

• ifstream - запись файла

• ofstream - вывод файла

• fstream - ввод / вывод файла

Обычно эти потоки записываются в производные <...>.

Потоки ifstream, ofstream и fstream используются для генерации файлов в программе и доступа к содержащимся в них данным. Их использование заключается в следующем:

имя потока ("путь \ имя_файла");

Эта функция позволяет вам создать файл данных, например, базу данных. Имя здесь - это произвольное имя (латинское) - имя потока. Это имя будет использовано позже для записи или чтения данных в файле.

Например, ofstreammk ("C: \ Users \ admin \ Desktop \ file.txt");

«File.txt» - это имя файла, который мы создали, который связан с именами потоков.

Теперь мы хотим открыть его, чтобы использовать сгенерированные данные. Вам необходимо ввести следующий оператор:

ifstreammk ("C: \ Users \ admin \ Desktop \ file.txt");

Вы должны всегда закрывать файлы, которые вы открыли. Этот процесс выполняется следующим образом:

name.close();

Например, mk.close ();

Это означает, что имена файлов и file.txt с mc позволяют обмениваться данными.

Пример 1 Создайте файл, содержащий сумму 2 целых чисел, и создайте программу, которая использует результат.

Вид программы в режиме консоли выглядит следующим образом:

```
//---------------------------------------------------------
#include <vcl.h>
# include <iostream.h>
# include <fstream.h>
# include <conio.h>
# include <math.h>
#pragma hdrstop

//---------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{int a=12, b=10;
int summa;
float s1;
ofstreammk ("hujjat.txt");
summa = a + b;
cout<<"summa=" << summa <<endl;
mk<< summa <<endl;
mk.close ( );
ifstream  mk1 ("hujjat.txt");
```

```
mk1 >>summa;
s1 = sin (summa);
cout<<"s1="<<s1<<endl;
mk1.close( );
getch( );
return 0;
}
//---------------------------------------------------------
```

После ввода текста программы нажмите F9, и появится следующая версия программы:

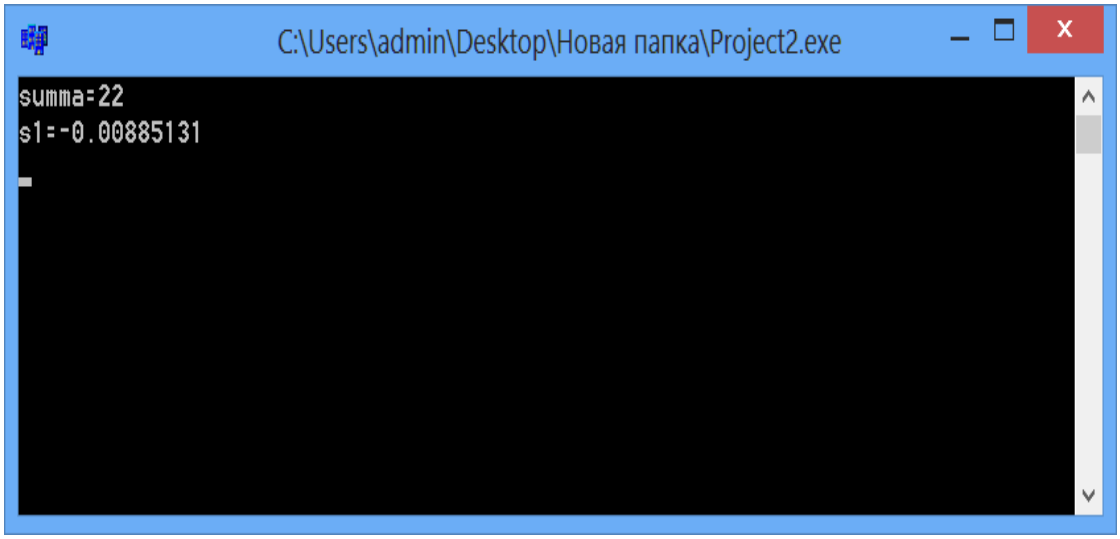


Рис. 10.1. Окно результатов.

Пример 2. Матрица и векторы приведены. Числовые значения не являются обязательными. Используя эти значения, создайте программу для умножения матрицы на вектор, вычисления трассы матрицы и вычисления суммы вектора.

Вид программы в режиме консоли выглядит следующим образом:

```
//---------------------------------------------------------
#include <vcl.h>
# include <iostream.h>
# include <fstream.h>
```

```
# include <conio.h>
# include <math.h>
#pragma hdrstop

//---------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
int  a [3][3], b[3], c[3];
int i, j;
int s1=0, s2=0;
ofstream said ("mk.txt");
for ( i=0; i<3; i++)
{
for (j=0; j<3; j++)
{
a[i][j] = random (9);
said <<a[i][j]<<endl;
}
}
for (i=0; i<3; i++)
{
b[i] = random (11);
said << b[i]<<endl;
}
said.close ( );
ifstream said1 ("mk.txt");
for ( i=0; i<3; i++)
for (j=0; j<3; j++)
said1 >>a[i][j];
```

```
for ( i=0; i<3; i++)
said1 >> b[i];
for ( i=0; i<3; i++)
{
c[i] = 0;
for (j=0; j<3; j++)
c[i] = c[i] + a[i][j] * b[i];
cout<< "c="<<c[i]<<endl;
}
for ( i=0; i<3; i++)
s1 = s1 + a[i][i];
for ( i=0; i<3; i++)
s2=s2 + b[i];
cout<< "s1="<<s1<<endl;
cout<<"s2="<<s2<<endl;
getch( );
return 0;
}
//---------------------------------------------------------
```

После ввода текста программы нажмите F9, и появится следующая версия программы:

Рис. 10.2. Окно результатов.

Пример 3. Создать программу для записи текстовых сообщений во внешнее хранилище. Вам понадобятся 1 Memo и 2 компонента BitBtn для программирования вашего примера в визуальной среде.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 10.2 Ввод свойств компонента

Имя компонента	**Имя свойства (Состояние окна**	**Процесс, который должен быть выполнен**
Form1	Caption (Properties)	Вводится слово “Работа с файлами”.
	FormCloseQuery (Events)	Текст программы введен.
Memo1	Lines (Properties)	Вводится слово “Memo1”.
BitBtn1	Kind (Properties)	Выбирается свойство “bkOK”.
	Caption (Properties)	Вводится слово “Написать в файл”.
	OnClick (Events)	Текст программы введен.
BitBtn2	Kind (Properties)	Выбирается свойство“bkClose”.
	Caption (Properties)	Вводится слово “Выход”.
	OnClick (Events)	Вводится Close();

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

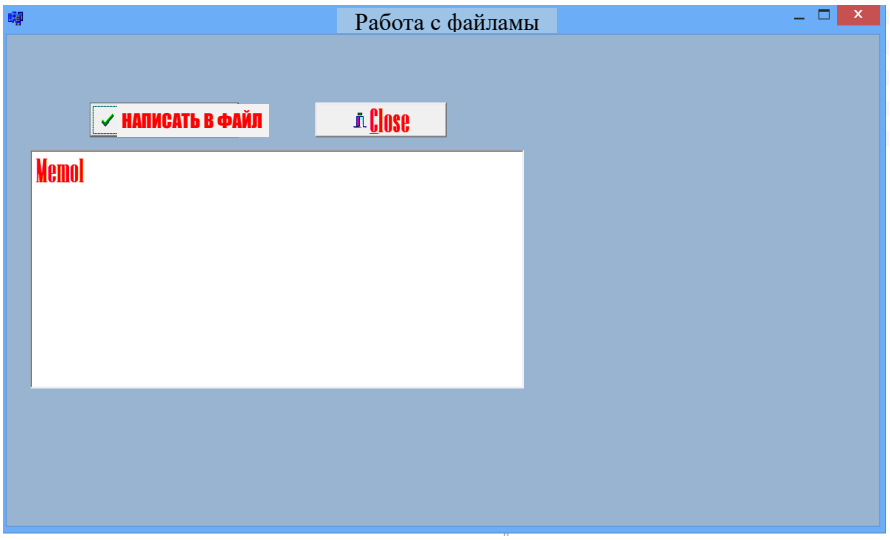


Рис. 10.3. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------
#include <vcl.h>
#include <fstream.h>
#pragma hdrstop
#include "Unit1.h"

//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
```

```
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
char sin[100];
ofstreamoutfile("Test.txt");
if(!outfile)
{ ShowMessage("Faylyaratilmadi");
return;}
outfile<<"Абдуллаев"<<"Абдулла"  <<"Абдуллаевич"  <<"1-17"  <<"5"
<<endl;
outfile<<"Ботыров"<<"Ботырали" <<"Собирович" <<"2-16" <<"4" <<endl;
outfile<< "Саматова  "<<"Саида" <<"Саматовна" <<"3-15  " <<"3" <<endl;
outfile<<"Мирполатов"<<"Миролим" <<"Мирхайдарович" <<"77-17" <<"5
" <<endl;
outfile<<"Мирполатова" <<"Зиеда" <<"Мирхайдаровна" <<"22-17" <<"5 "
<<endl;
outfile<< "Толаганова" <<"Зохида" <<"Алымовна" <<"12-15" <<"5" <<endl;
outfile.close();
ifstreaminfile("Test.txt");
if(!infile)
{ShowMessage("Файлнеудалился");
return;}
  Memo1->Clear();
while (!infile.eof())
{infile.getline(sin,100);
Memo1->Lines->Add(AnsiString(sin));
}
infile.close(); }
//---------------------------------------------------------
```

```
void __fastcall TForm1::FormCloseQuery(TObject *Sender, bool &CanClose)
{if(MessageDlg("Dasturdanchiqasmi", mtInformation, TMsgDlgButtons()
<<mbYes<<mbNo,0)==mrYes){
CanClose=true;}
else
{CanClose=false;}}
//---------------------------------------------------------
```

После ввода текста программы нажмите F9, и появится следующая версия программы:

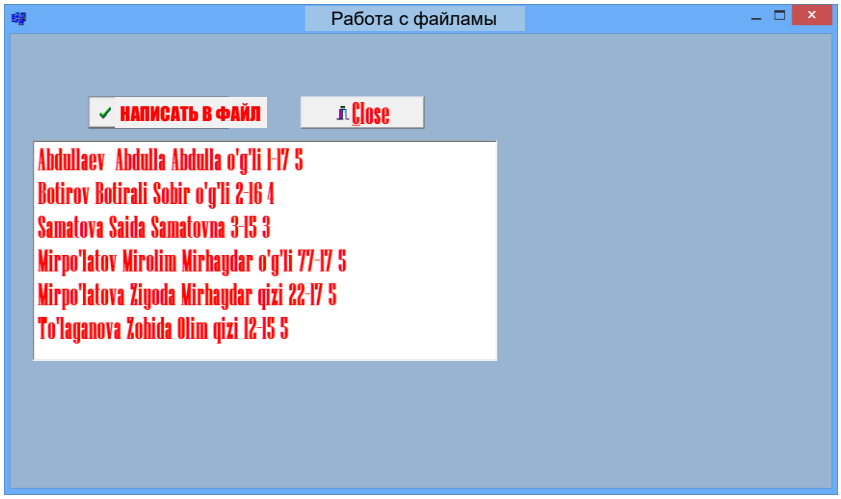


Рис.10.4. Окно результатов.

В результате выполнения программы во внешней памяти будет создан файл с именем test.txt, который будет выглядеть следующим образом:

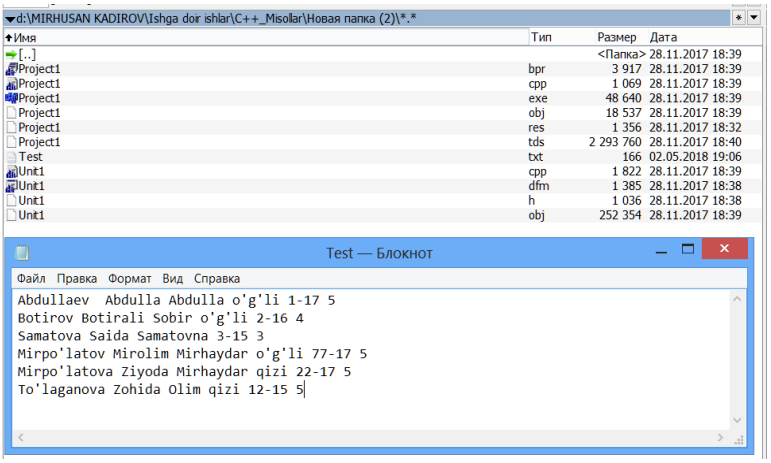


Рис. 10.5. Окно результатов.

Пример 4. Создать текстовый редактор для ввода и обработки текстовых файлов. Программа должна делать следующее: сохранять файлы, открывать сохраненные файлы, распечатывать.

Программирование в визуальной среде требует 1 Memo, 1 MainMenu, 1 OpenDialog, 1 SaveDialog, 1 FontDialog, 1 PrintDialog, 1 FindDialog и 1 StatusBar.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 10.3 Ввод свойств компонента

Имя компонента	**Имя свойства (**	**Процесс, который должен быть выполнен**
Form1	Caption (Properties)	Вводится слово “Блокнот Уз”.
	FormCloseQuery (Events)	Вводится текст программы.
Memo1	Lines (Properties)	Удаляется слово “Memo1”.
	Align (Properties)	Выбирается “alClient”.
MainMenu	Items (Properties)	“Файл”, “Редактировать”, “Помошь” имена главного меню будут введены. Каждое главное меню состоит из нескольких разделов, поэтому разделы включены в отдельное свойство “Caption”.
	OnClick (Events)	Все созданные вами разделы будут запрограммированы.
OpenDialog	Filter (Properties)	“Филтер Name” текст будет содержать слова “Текстовые файлы (* .txt) ” и свойство “Фильтр”“*.txt”.
SaveDialog	Filter (Properties)	“Филтер Name” текст будет содержать слова “Текстовые файлы (* .txt) ” и свойство “Фильтр”“*.txt”.

Окно создания главного окна программы выглядит следующим образом:

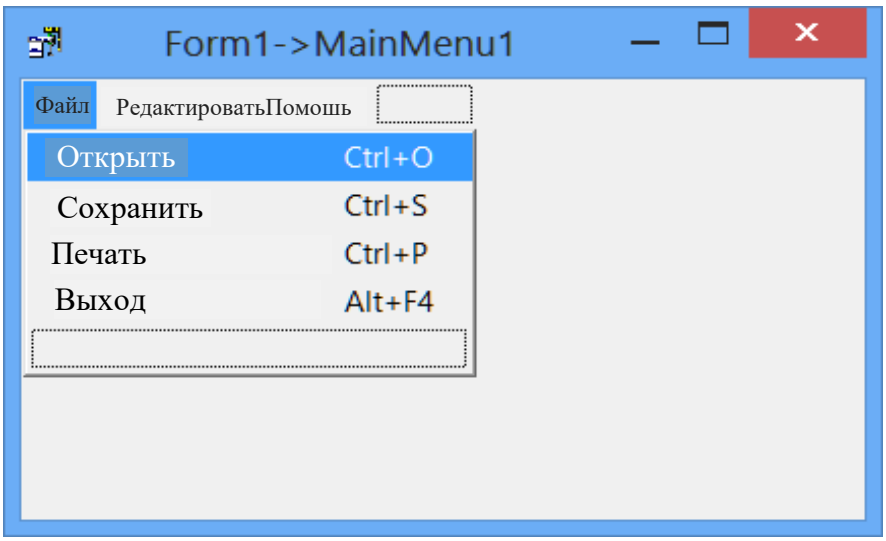


Рис. 10.6. Создание главного меню программы.

В приведенном выше окне перечислены все основные пункты меню и их разделы. После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

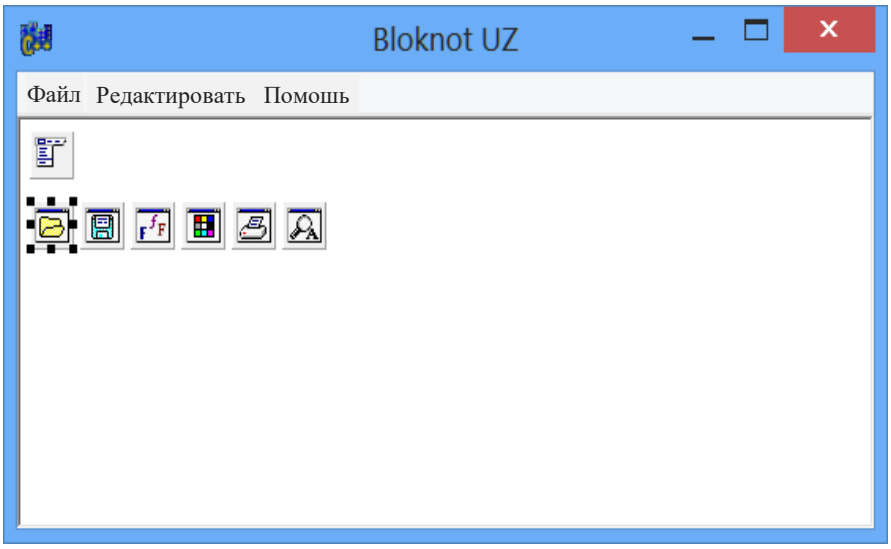


Рис. 10.7. Просмотр приложения.

Главное меню состоит из трех разделов, которые включают файл, редактирование и помощь.

Файловое меню включает в себя разделы открытия, сохранения, печати и вывода.

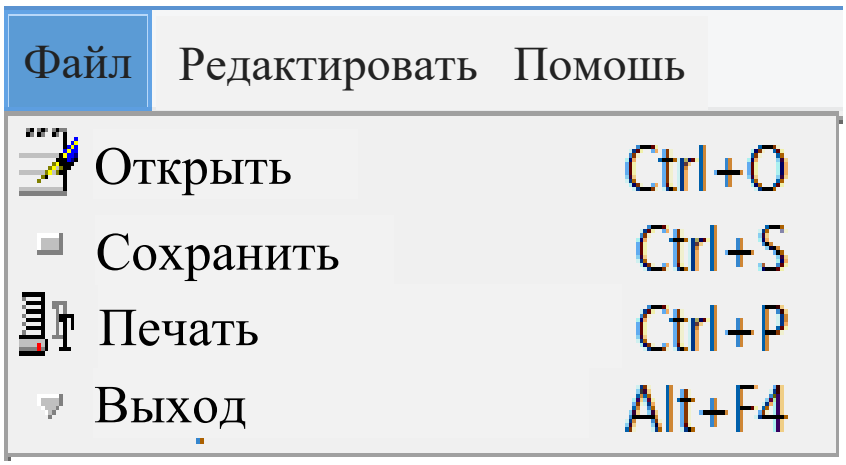


Рис.10.8. Файловое меню.

Меню Правка включает в себя выбор шрифта, выбор фона и поиск. На рисунке 10.9 показаны все разделы меню редактирования. Разделы позволяют выполнять необходимые действия.

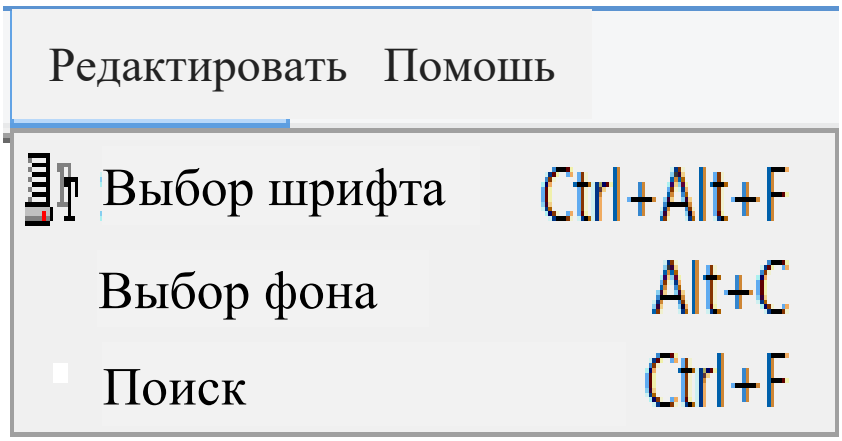


Рис. 10.9. Редактировать меню.

Меню Справка содержит информацию о программе и разделы для программиста.

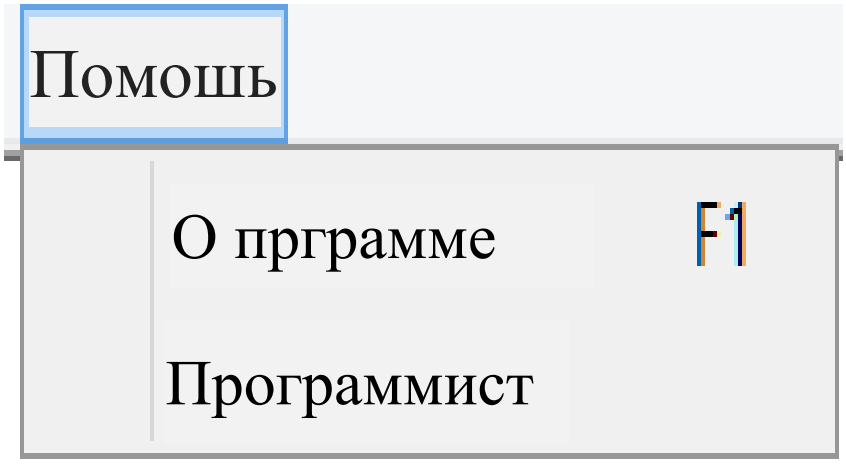


Рис. 10.10. Справочное меню.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include "Unit2.h"

//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}

//---------------------------------------------------------
```

```
void __fastcall TForm1::Открыть1Click(TObject *Sender)
{
//
if (Form1->OpenDialog1->Execute ())
Form1->Memo1->Lines->LoadFromFile(Form1->OpenDialog1->FileName);
}
//---------------------------------------------------------

void __fastcall TForm1::Сохранить1Click(TObject *Sender)
{
if(Form1->SaveDialog1->Execute () )
Form1->Memo1->Lines->SaveToFile(Form1->  SaveDialog1->FileName  +
".txt" );
}

//---------------------------------------------------------
void __fastcall TForm1::Распечатать1Click(TObject *Sender)
{
PrintDialog1->Execute() ;
}

//---------------------------------------------------------
void __fastcall TForm1::Рисовать1Click(TObject *Sender)
{
Close();
}

//---------------------------------------------------------
void __fastcall TForm1::Выборшрифта1Click(TObject *Sender)
{
```

```
if (FontDialog1->Execute())
Memo1->Font->Assign(FontDialog1->Font) ;
}

//---------------------------------------------------------
void __fastcall TForm1::Выборфона1Click(TObject *Sender)
{
if (ColorDialog1->Execute ())
Memo1->Color=ColorDialog1->Color;
}

//---------------------------------------------------------
void __fastcall TForm1::Поиск1Click(TObject *Sender)
{
FindDialog1->FindText=Memo1->SelText;

FindDialog1->Execute() ;
}

//---------------------------------------------------------
void __fastcall TForm1::Опрограмме1Click(TObject *Sender)
{
```

ShowMessage («Данное программное обеспечение является узбекским NotebookNotepad UZ. Позволяет вводить и обрабатывать текстовую информацию»);

```
}

//---------------------------------------------------------
void __fastcallTForm1::FormCloseQuery(TObject *Sender, bool&CanClose)
//
```

```
    {
    if(MessageDlg("Выходизпрограммы", mtInformation, TMsgDlgButtons()
<<mbYes<<mbNo,0)==mrYes)
    //
    {CanClose=true;
    }
    else
    {
    CanClose=false;
    }
    }

    //---------------------------------------------------------
    void __fastcall TForm1::Программист1Click(TObject *Sender)
    {
    //
    AboutBox->ShowModal();
    }
    //---------------------------------------------------------
```

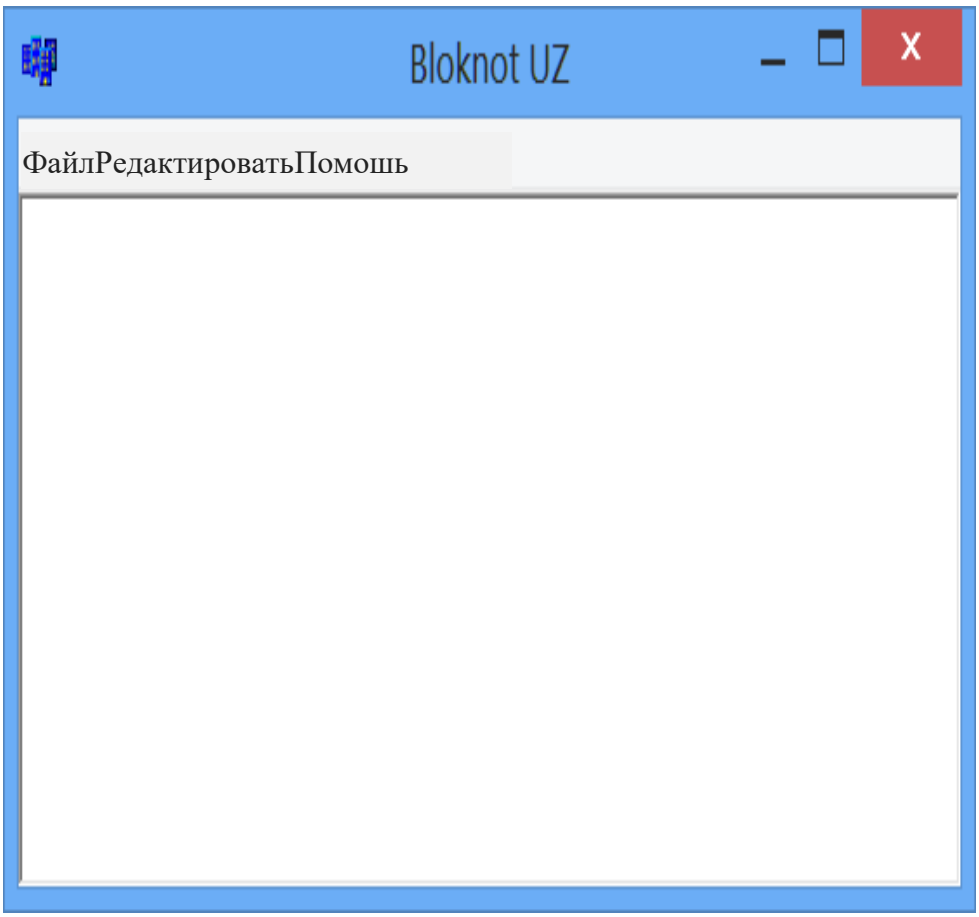


Рис. 10.11. Окно результатов.

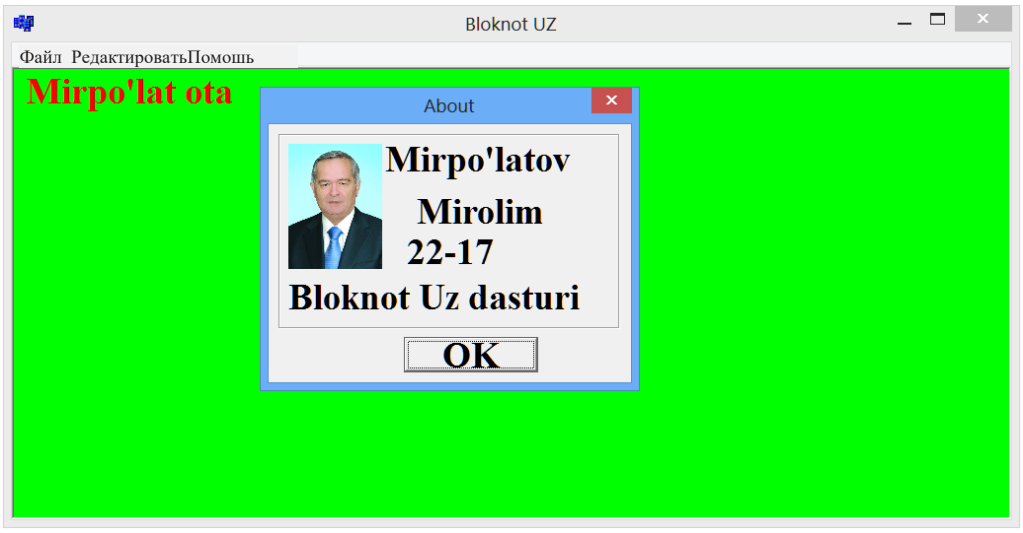


Рис. 10.12. Справка Меню программиста раздела.

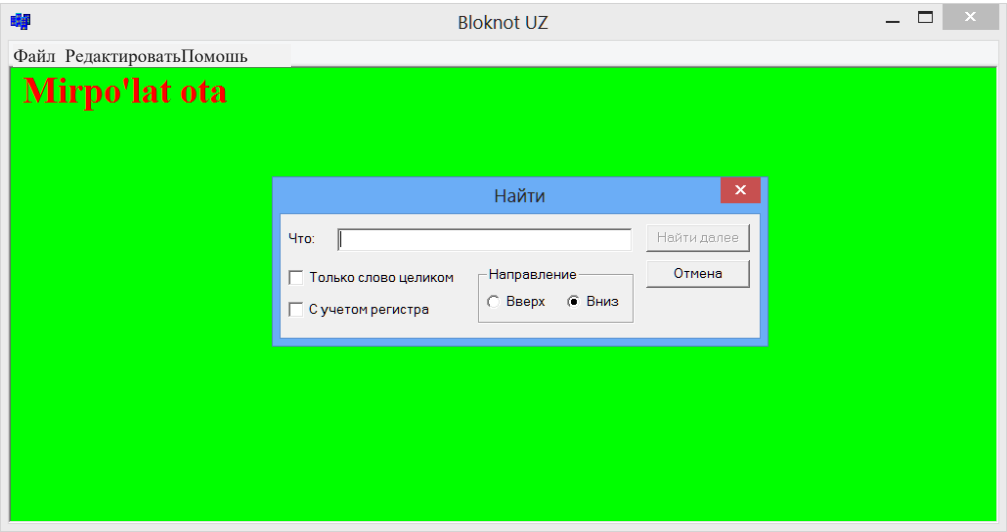


Рис. 10.13. Изменить меню поиска в разделе.

10.4. Работы с указателями на языке C++

Переменные, которые отображают (сохраняют) память как ее значение, называются переменными указателя.

Например, значение указателя:

1) 0x22ff40

2) 0x22ff33

3) может быть точной частью памяти, как упомянуто выше.

Как и другие переменные, необходимо объявить и определить категорию для использования индикаторов. Показатель дается следующим образом:

```
int *countPtr, count;
```

где countPtr - указатель на объект класса int, а count - переменная в обычной целой (int) категории. При объявлении показателей каждой переменной должна предшествовать *.

Пример 5. Создайте программу для поиска значения адреса, указанного указателем.

Вид программы в режиме консоли выглядит следующим образом:

```
//---------------------------------------------------------
#include <vcl.h>
```

```
#include <iostream.h>
#include <conio.h>
#pragma hdrstop

//---------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
int n = 6;
int * nPtr;
// &адресдействийnPtr = &n;
cout<< "n=" << n <<endl;
*nPtr = 22;
cout<< "n=" <<n<<endl;
cout<< "\ значение указателя n,\n";
cout<< " Адрес, на который указывает указатель=" <<nPtr<<endl;
cout<< " Значение адреса, указанное указателем="
<<*nPtr<<endl;
getch();
return 0;
}
//---------------------------------------------------------
```

После ввода текста программы нажмите F9, и появится следующая версия программы:

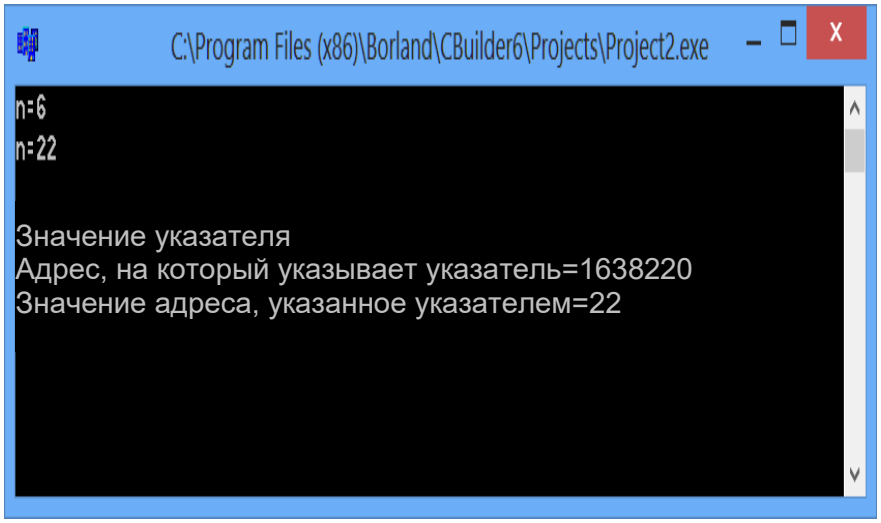


Рис. 10.14. Окно результатов

Мы реализуем эту программу в визуальной среде. Вам понадобятся 3 компонента Labels, 3 Edit и 2 компонента BitBtn. Мы определяем свойства компонента окна формы следующим образом:

Таб. 10.4 Ввод свойств компонента

Имя компонента	**Имя свойства (Состояние окна ObjectInspector)**	**Процесс, который должен быть выполнен**
Form1	Caption (Properties)	Вводится слово“Работа с индикаторами”.
Label1	Caption (Properties)	Вводится слово “значение указателя n”.
Label2	Caption (Properties)	Вводится слово “Адрес, на который указывает указатель=”.
Label3	Caption (Properties)	Вводится слово “Значение адреса, указанное указателем=”.
Edit1	Text (Properties)	Удаляется слово “Edit1”
Edit2	Text (Properties)	Удаляется слово “Edit2”
Edit3	Text (Properties)	Удаляется слово “Edit3”
BitBtn1	Kind (Properties)	Выбираем свойство “bkOK”.
	Caption (Properties)	Вводится слово “расчет”.
	OnClick (Events)	Текст программы введен.
BitBtn2	Kind (Properties)	Выбираем свойство “bkClose”.
	Caption (Properties)	Вводится слово “Выход”.
	OnClick (Events)	Вводим Close();.

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

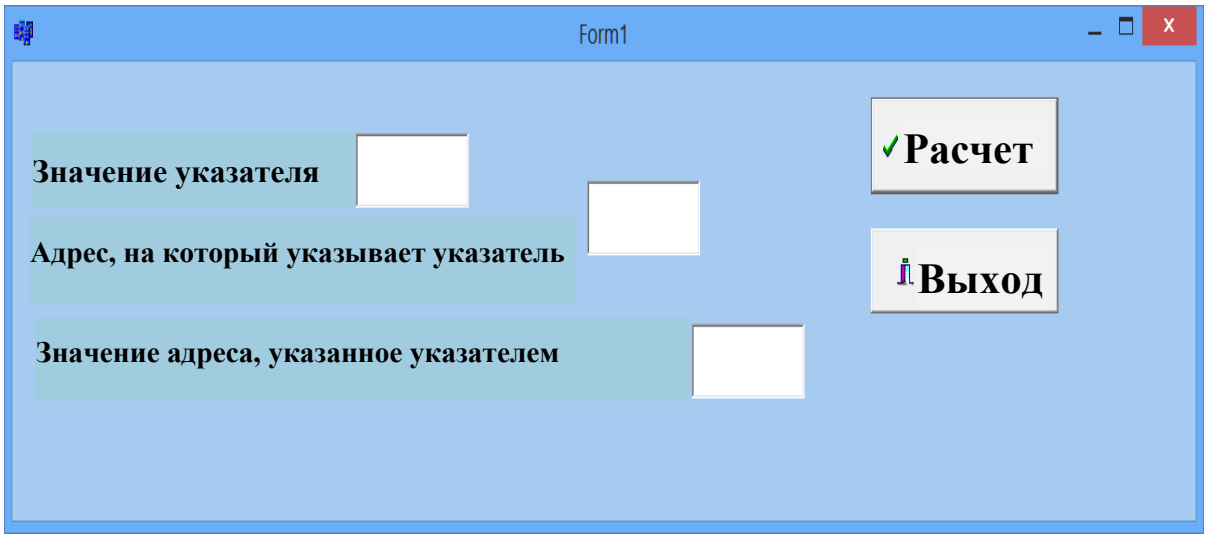


Рис. 10.15. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{int n = 5;
```

```
int * nPtr;
nPtr = &n;
*nPtr = 15;
Edit1->Text=IntToStr(n);
Edit2->Text=IntToStr(nPtr);
Edit3->Text=IntToStr(*nPtr);}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{Close();}
//---------------------------------------------------------
```

Рис. 10.16. Окно результатов.

Пример 6. Создать указатель местоположения программы. Вид программы в режиме консоли выглядит следующим образом:

```
//---------------------------------------------------------
#include <vcl.h>
#include <iostream.h>
#include <conio.h>
#pragma hdrstop
//---------------------------------------------------------
#pragma argsused
```

```
int main(int argc, char* argv[])
{double n = 5;
double *kPtr;
kPtr = &n;
cout<< " Значение переменных" <<endl;
cout<< "n=" <<n<<endl;
cout<< "*kPtr=" << *kPtr<<endl;
cout<< "\nxotiramanzilii" <<endl;
cout<< " n - адрес переменной. &n=" <<&n<<endl;
cout<< "Адрес, на который указывает указатель.kPtr="<<kPtr<<endl;
cout<< " Указатель - местоположение.  &kPtr=" <<&kPtr<<endl;
cout<< "\ n Размер памяти, используемой переменными" <<endl;
cout<< "n=" <<sizeof(n) <<endl;
cout<< "*kPtr=" <<sizeof(kPtr) <<endl;
getch();
return 0;}
//---------------------------------------------------------
```

После ввода текста программы нажмите F9, и появится следующая версия программы:

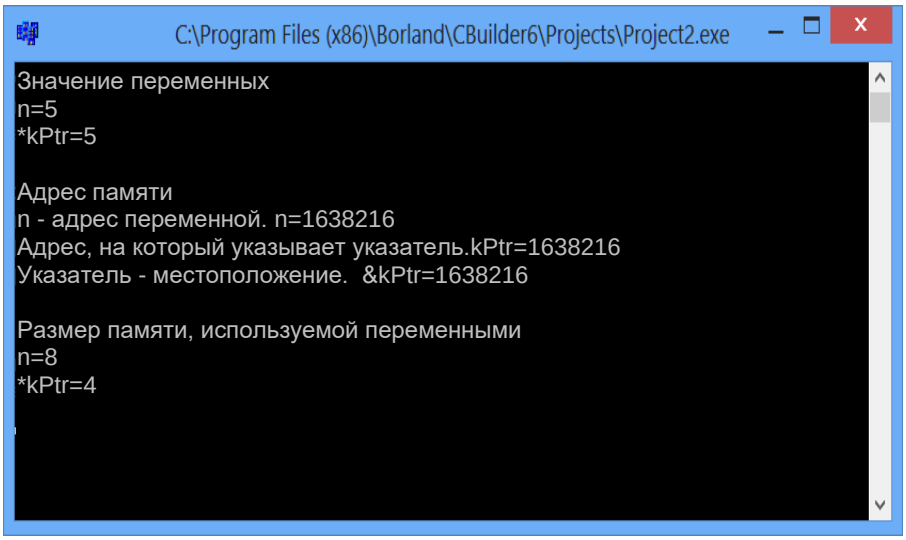


Рис. 10.17. Окно результатов.

Мы реализуем эту программу в визуальной среде. Вам понадобятся 4 компонента Labels, 4 Edit и 2 компонента BitBtn.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 10.4

Ввод свойств компонента

Имя компонента	**Имя свойства (Состояние окна ObjectInspector)**	**Процесс, который должен быть выполнен**
Form1	Caption (Properties)	Вводится слово “Работа с индикаторами”.
Label1	Caption (Properties)	Вводится слово “ n значение индикатора”.
Label2	Caption (Properties)	Вводится слово “Адрес, на который указывает указатель”.
Label3	Caption (Properties)	Вводится слово “ Значение адреса, указанное указателем” .
Label4	Caption (Properties)	Вводится слово “Расположение указателя”.
Edit1	Text (Properties)	Удаляем слово “Edit1”
Edit2	Text (Properties)	Удаляем слово “Edit2”
Edit3	Text (Properties)	Удаляем слово “Edit3”
Edit4	Text (Properties)	Удаляем слово “Edit4”
BitBtn1	Kind (Properties)	Выбираем свойство “bkOK”.
	Caption (Properties)	Вводится слово “Расчет”.
	OnClick (Events)	Текст программы введен.
BitBtn2	Kind (Properties)	Выбираем свойство “bkClose”.
	Caption (Properties)	Вводится слово “Выход”.
	OnClick (Events)	Вводится Close();

После ввода свойств существующих компонентов дизайн программы выглядит следующим образом:

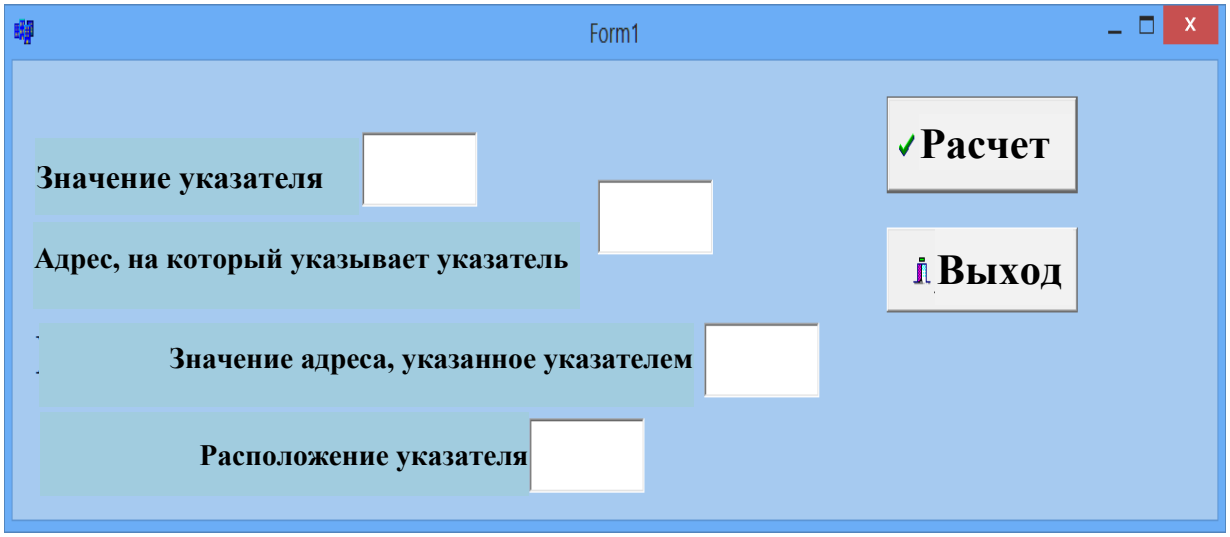


Рис. 10.18. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
```

```
int n = 5;
int * nPtr;
nPtr = &n;
Edit1->Text=IntToStr(n);
Edit2->Text=IntToStr(&n);
Edit3->Text=IntToStr(nPtr);
Edit4->Text=IntToStr(*nPtr);
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
Close();
}
//---------------------------------------------------------
```

Рис. 10.19. Окно результатов.

Ссылки используются как синоним имени, отображаемого в объявлении, что означает, что на одну переменную могут ссылаться разные имена. Ссылка может рассматриваться как постоянное значение. Обращение объявляется следующим образом:

<toifa>&<nomi>;

Где <тип> является оператором ссылочного значения, обозначая знак «&», за которым следует тип <имя> ссылочной категории.

Другими словами, знак «&» называется действием для получения адреса.

Например:

```
intk;
int&p = k;
```

Ссылка используется в основном как функция функции, которая передается через адрес.

10.5. Функции обращения к текстовым данным

Программа C ++ имеет следующие функции указателя для доступа и выполнения текстовых данных:

- ✓ strcat - объединить строки;
- ✓ strcmp - сравнение строк;
- ✓ strcpy - копировать из одного ряда в другой;
- ✓ strstr - поиск по строке;
- ✓ strlen - определить длину строки;
- ✓ тренд - перейти в конец стрелки;
- ✓ strpos - определить статус индекса

Например:

```
char str [100];
strcpy (str, "2");
strcat (str, "+");
strcat (str, "2");
strcat (str, "= ");
strcat (str, "4" );
std::cout<< str << std::endl;
```

-misol. Создайте программу, используя функцию указателя для ссылки на тексты и выполнения строк. Вид программы в режиме консоли выглядит следующим образом:

```
//---------------------------------------------------------
#include<vcl.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#pragma hdrstop

//---------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
char str[100];
strcpy( str, "адрес");
strcat( str, "категория ");
strcat( str, "данные ");
strcat( str, "снова");
strcat( str, "обработка." );
std::cout<< str << std::endl;
getch ();
return 0;
}
//---------------------------------------------------------
```

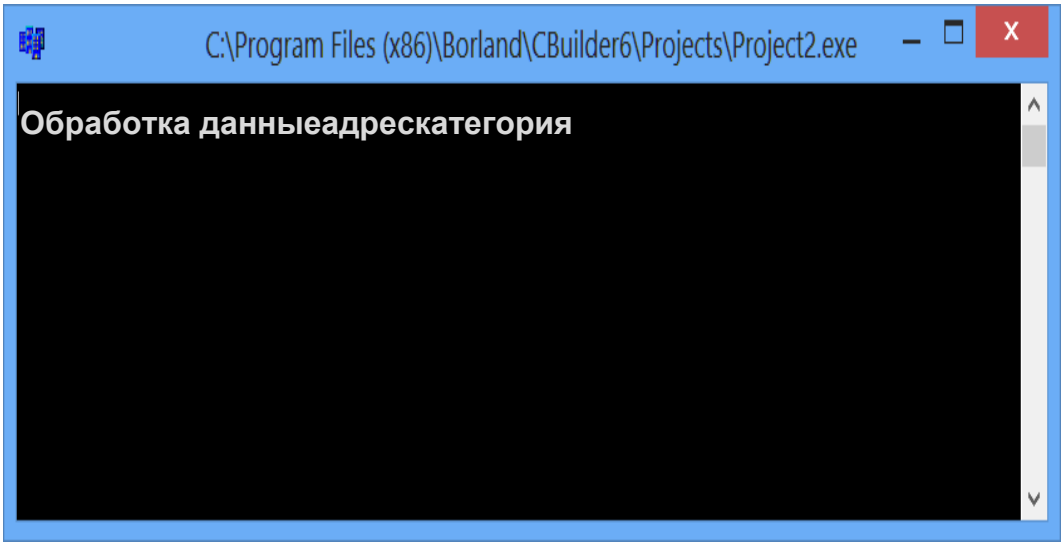


Рис. 10.20. Окно результатов.

Пример 8. Создайте программу, используя функции указателя для доступа и применения текстов.

Вам понадобится 5 компонентов Labels и 4 компонента BitBtn для программирования вашего примера в визуальной среде.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 10.5 Ввод свойств компонента

Имя компонента	**Имя свойства (Состояние окна ObjectInspector)**	**Процесс, который должен быть выполнен**
Form1	Caption (Properties)	Вводится слово “Работа с индикаторами”.
BitBtn1	Kind (Properties)	Выбираем свойство “bkOK”.
	Caption (Properties)	Вводится слово “объединение”.
	OnClick (Events)	Текст программы введен.
BitBtn2	Kind (Properties)	Выбираем свойство “bkOK”.
	Caption (Properties)	Вводится слово “Сравнение”.
	OnClick (Events)	Текст программы введен.
BitBtn3	Kind (Properties)	Выбираемсвойство “bkOK”.
	Caption (Properties)	Вводится слово“Поиск”.
	OnClick (Events)	Текст программы введен.
BitBtn4	Kind (Properties)	Выбираем свойство “bkOK”.
	Caption (Properties)	Вводится слово “Создать”.
	OnClick (Events)	Текст программы введен.

После того, как существующие компоненты компонента введены, дизайн программного обеспечения выглядит следующим образом:

Рис. 10.21. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------
#include <vcl.h>
#include <string.h>
#pragma hdrstop
#include "Unit1.h"

//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
```

```
{
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
char S1[20] = "Elektronikava ", S2[10] = "avtomatika";
Label1->Caption=strcat(S1,S2);
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
char S1[20] = "ener", S2[10] = "getika";
Label2->Caption=strcmp(S1,S2);
}

//---------------------------------------------------------
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
char S1[20] = "Elektronika", S2[10] = "E";
Label3->Caption=strstr(S1,S2);
char S3[20] = "Elektronika", S4[10] = "y";
Label4->Caption=strstr(S3,S4);}

//---------------------------------------------------------
void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{
char S1[20]="ilektronika", S2[20]="i", S3[20]="E", S[60], *St;
St = strstr(S1,S2);
if(St)
{
```

```
*St = 0;
St += strlen(S2);
Label5->Caption = strcat(strcat(strcpy(S,S1),S3),St);
} else Label5->Caption = "текстненайден";
}
//---------------------------------------------------------
```

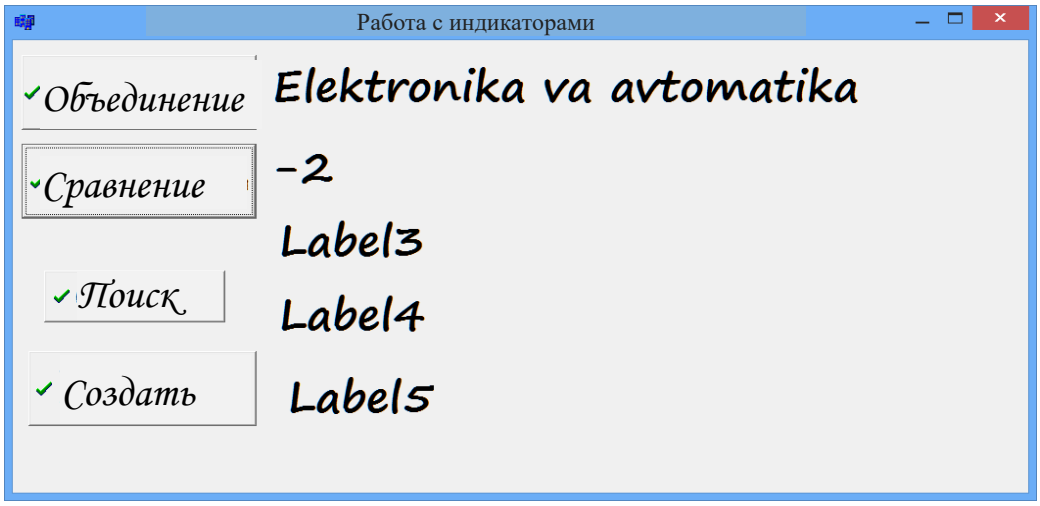


Рис. 10.22. Окно результатов.

Рис. 10.23. Окно результатов.

Вопросы по главе 10

1. Основы работы с файлами.
2. Функции чтения-записи-файла.

3. Функции управления индексами файлов.

4. Какие функции используются для работы с файлом?

5. Перечислите функции доступа к текстовым данным.

Тестовые вопросы:

1. Для чего используетсяifstream?

а) Ввод файла;

б) Редактировать файл;

г) вывод файла;

д) запись в файл;

2. Это функция, которая определяет конец файла

а) EEF;

б) EOF;

в) END;

г) EOA;

3. Двоичные файлы –это ….

а) Именованная часть внешнего хранилища для хранения этих данных;

б) Конкретные структурированные, категоризированные файлы включены в языки программирования для обработки этих физических файлов;

в) Это просто последовательность байтов;

г) поддерживает логическую модель данных, не имея дело с этой физической памятью.

4. Почему используется ofstream?

а) Ввод файла;

б) Редактировать файл;

в) запись в файл;

г) вывод файла;

5. Strcat (); что делает функция указателя?

а) Объединить строки;

б) Сравнение строк;

в) копирование из одного ряда в другой;

г) поиск строки;

Глава11. КЛАССЫ И ОБЪЕКТЫ В СИСТЕМЕ ПРОГРАММИРОВАНИЯ BORLAND C++ BUILDER 6

Ключевые слова: класс, объект, объектно-ориентированный подход, атрибуты, инкапсуляция, абстракция, наследование, полиморфизм.

11.1. Классы

Объектно-ориентированный подход основан на систематическом использовании моделей для создания программных систем, независимых от языка программирования. Не каждая модель может представлять все особенности объекта, который она представляет, но только некоторые очень важные особенности. В объектно-ориентированном программировании программа позволяет описывать объекты и их свойства (атрибуты) и классы, которые их объединяют.

Можно создавать базовые классы и их производные на основе изучения атрибутов и методов.

Другой теоретически важной и необходимой особенностью объектно-ориентированного программирования является механизм обработки событий, с помощью которого можно изменять значения объектов. Объектно-ориентированное программирование экономит значительную работу по объектно-ориентированному программированию благодаря использованию ранее созданной объектной библиотеки и методов.

Объекты, классы и методы могут быть полиморфизмами, обеспечивая универсальность и универсальность программного обеспечения.

Термин «класс» определяет тип объекта. Каждый класс ,(копия) класса называется объектом. У каждого объекта свое состояние. Статус актива определяется по текущей стоимости его участников. Задача класса определяется способностью его членов выполнять действия над объектами класса. Класс не является физическим по своей природе, и объявление структуры является его ближайшим аналогом. Только память используется,

когда класс используется для создания объекта. Этот процесс также называется созданием копии класса.

Каждый объект языка C ++ имеет такие же атрибуты, как и функциональность с другими объектами этого класса. Программист несет полную ответственность за создание собственных классов и поведение объектов этих классов. При работе в среде программист имеет доступ к большой библиотеке классов (например, библиотеке визуальных компонентов C ++ Builder).

Абстракция данных - это возможность создавать новый тип данных, и он может работать с тем же типом, что и базовый тип данных. Новые виды часто называют абстрактными типами данных, хотя их можно проще назвать «определяемыми пользователем видами».

Инкапсуляция - это механизм объединения данных и кода их обработки. Инкапсуляция позволяет защитить код и код от внешнего воздействия.

Абстракция - это процесс упрощения сложной проблемы. Вступая в конкретную проблему, вы не будете пытаться учесть все детали, а выберите те, которые облегчат решение.

Допустим, вам нужно создать модель дороги. Очевидно, что вместо светофоров, автомобилей, автомагистралей, улиц с односторонним и двусторонним движением, погодных условий и так далее. вы создаете классы. Каждый из этих элементов влияет на трафик. Однако, даже если на дороге могут появиться насекомые и птицы, вы не создадите свою собственную модель. Вы упрощаете реальный мир и используете только его основные элементы. Автомобиль является важной деталью модели, но это BMW или любой другой автомобиль, а детали дорожной модели являются дополнительными.

У абстракции есть два преимущества. Во-первых, это упрощает решение проблем. Что еще более важно, из-за абстракции программные компоненты могут использоваться повторно. Они часто специализируются на создании повторно используемых компонентов. То есть, поскольку компоненты

предназначены для решения конкретной проблемы, они также взаимосвязаны. Фрагмент приложения становится трудно использовать в другом месте. По возможности старайтесь создавать объекты, которые решают ряд проблем. Абстракция позволяет использовать одно решение для решения других проблем в этой области.

При написании классов мы следуем правилам написания функций. Первая строка класса открывает ключевое слово class и имя класса, а затем новая строка содержит фигурные скобки, которые содержат методы и атрибуты класса.

Класс может иметь следующие разделы:

1. частный (частный, внутренний).

2. Защищенный (защищенный, защищенный раздел).

3. публичный (public, public).

Теперь вы можете написать общий синтаксис базового класса следующим образом:

class className {private:

<private членыданных><private конструкторы><private методы>

protected:

< Защищенные участники><Защищенныеконструкторы>

< Защищенные методы>

public:

< Общие ссылки>< Члены с общей ссылкой>

< Генеральные проектировщики и деструкторы><Методы общего права>

}

Ниже приведены права на основные классы C ++:

1. Частная часть - определяет ссылки только на методы этого класса. Частные методы не допускаются для производных классов.

2. Защищенные имена дают только ссылку на методы этого класса и методы класса этого класса.

3. Общие публичные имена обеспечивают доступ к методам всех классов.

Основные правила использования разделов в определении класса:

1. Разделы могут рекламироваться в любом порядке, и даже может произойти редизайн.

2. Если раздел не назван, то компилятор принимает последние имена классов, определенные как личные данные.

Если мы хотим ограничить доступ к заявкам на членство, мы не должны размещать их в открытом доступе.

11.2. Абстракции

Абстракция — это любое выражение любого языка программирования, отличное от дескрипторов.

В объектно-ориентированном программировании каждый объект имеет принципиальную динамическую природу, то есть он меняется со временем и подвержен влиянию внешних факторов. Другими словами, объект ведет себя в некоторой степени. В объектно-ориентированном программировании абстракция - это модель объектно-ориентированного программирования. Класс объединяет объекты, которые имеют общие черты и поведение. Объекты, принадлежащие к одному и тому же классу, имеют одинаковые свойства и демонстрируют одинаковое поведение.

Классы похожи на шаблоны: они используются для создания копий объектов. Персонажи являются внешними чертами класса. Объект может отображать свои свойства только в том случае, если он обеспечивает прямой доступ к внутренней переменной или возвращает значение с помощью метода.

Поведение - действия, выполняемые объектом в ответ на сообщение или изменение статуса. Это обозначает, что делает объект.

Один объект может действовать на другой и влиять на его поведение. Термин «поведение» используется вместо «вызвать метод», «вызвать функцию» или «передать сообщение».

Связь между объектами является важной составляющей объектно-ориентированного программирования. Есть два основных способа взаимодействия с объектами.

Первый способ заключается в том, что объекты могут существовать независимо от других. Если отдельные объекты нуждаются во взаимодействии, они отправляют сообщения друг другу.

Объекты общаются друг с другом через сообщения. Объект, который получает сообщение, выполняет определенные действия.

Передача аналогична вызову метода или использованию одной из поведенческих моделей для изменения состояния объекта.

Второй способ: объект может содержать другие объекты. Как и в случае объектно-ориентированного программирования, объекты могут быть объединены из других объектов, в свою очередь, так как программа состоит из объектов. Каждый из этих объектов имеет интерфейс со стилем и символами.

Сообщение является важной концепцией для объектно-ориентированного подхода. Благодаря механизму обмена сообщениями объекты могут сохранять свою независимость.

Для объекта, который отправляет сообщение другому объекту, не имеет значения, как объект, получающий сообщение, выполняет требуемое действие. Важно, чтобы действие было предпринято.

11.3. Наследование

Последовательность позволяет вам создавать новый класс на основе существующего определения класса. Когда новый класс создается на другой основе, его описание автоматически наследует все атрибуты, поведение и реализацию существующего класса. Все методы и свойства существующего интерфейса класса автоматически появляются в интерфейсе наследования. Наследование позволяет предвидеть поведение, несовместимое с унаследованным классом. Эта полезная функция позволяет настраивать

программное обеспечение для изменения требований. Если вам нужно внести некоторые изменения, новый класс, который наследует функции старого класса, будет перезаписан. Функции, которые необходимо изменить, затем переопределяются и добавляются новые функции. Суть этой замены заключается в том, что она позволяет вам изменить работу объекта, не меняя исходного описания класса. Ведь не стоит трогать основные классы, которые были тщательно протестированы при повторном тестировании. Если вы решили использовать наследников для многократного использования или для каких-либо других целей, сначала посмотрите, совместимы ли класс наследия и виды класса, дающего наследие.

В последовательности видов часто совместимы Класс-преемник другого класса, который должен обрабатывать класс-преемник таким образом, чтобы результирующее отношение имело смысл, то есть классификацию преемственности.

Класс наследования может иметь три типа методов и свойств:

- заменить: не только усвоить метод или черту предков нового класса, но и дать ему новое определение;
- новый: новый класс добавляет совершенно новые методы или функции;
- рекурсивный: новый класс сохраняет свои наследственные методы или черты напрямую.

Большинство объектно-ориентированных языков ищут описание в передаваемом объекте. Если там невозможно найти описание, поиск будет двигаться вверх, пока описание не будет найдено. Именно так осуществляется управление данными, и поэтому можно сделать заказ.

Последующие классы могут получить доступ к методам и функциям, которые имеют безопасный уровень доступа. Доступ к базовому классу может быть назначен только защищенным методам, которые могут использоваться поколениями. В других случаях следует использовать частный или публичный

доступ. Этот подход обеспечивает более надежную конструкцию, чем доступ ко всем классам, включая сетевые классы.

Наследование применяется в трех основных случаях:

В работе многократного использования;

- выделяться;
- Types для переключения типов.

Использование одних видов наследования предпочтительнее других. Наследование позволяет новому классу использовать старый класс много раз. Вместо очистки или ввода кода преемник обеспечивает автоматический доступ к коду, то есть при вводе кода он рассматривается как часть нового класса. Когда вы используете наследование для многократного использования, вы связаны унаследованной реализацией. Этот тип наследования следует использовать с осторожностью.

Дифференциальное наследование позволяет только программировать различия между классом поколения и классом предка. Разностное программирование - хороший инструмент. Небольшое количество кодирования и простое управление кодом облегчают проектирование проекта. В этом случае вам придется писать меньше строк кода, что также уменьшит количество добавляемых ошибок.

Наследование является важной особенностью объектно-ориентированного программирования. Например, чтобы определить наследование класса B класса A (состоящего из класса A), в классе D после имени класса вставляются две точки, за которыми следуют унаследованные классы:

```
class A {
public: A (); A ();
MethodA ();
};
Class V: public A {
public: V();
....};
```

11.4. Полиморфизм

Если инкапсуляция и наследование могут рассматриваться как полезные инструменты объектно-ориентированного подхода, полиморфизм является наиболее универсальным и радикальным инструментом. Подход к инкапсуляции и последовательности, основанный на полиморфизме, без полиморфизма не может быть эффективным. Полиморфизм - это центральное понятие в парадигме объектно-ориентированного подхода. Без полиморфизма объектно-ориентированный подход не может быть эффективно использован.

Полиморфизм - это состояние, в котором что-то принимает разные формы. В языке программирования «несколько форм» означает, что одно имя представляет имена различных кодов, которые автоматически выбираются механизмом. Таким образом, используя полиморфизм, одно имя может означать другое поведение.

Наследование необходимо для использования определенных типов полиморфизма. Можно использовать полиморфизм из-за возможности сходства. С полиморфизмом можно добавлять дополнительные функции, когда он совместим с системой. При написании программы вы можете даже добавить новые классы с непредсказуемым функционалом, и все это можно сделать без изменения исходной программы. Это инструменты, которые можно легко адаптировать к новым требованиям.

Существует три основных типа полиморфизма:

- Полиморфизм присоединения;
- параметрический полиморфизм;
- перегрузка;

Полиморфизм сложения иногда называют чистым полиморфизмом. Полиморфизм объединения интересен тем, что версии сетевых классов могут вести себя по-разному. Используя полиморфизм соединения, можно изменить поведение системы путем введения новых сетевых классов. Его главное

преимущество в том, что можно создавать новое поведение, не меняя исходную программу.

Из-за полиморфизма нет необходимости указывать последовательное использование репликации. Вместо этого следует использовать преемственность в первую очередь для достижения полиморфного поведения посредством взаимосвязей. Если взаимность взаимности установлена правильно, то произойдет повторное использование. Полиморфизм сложения может использоваться повторно от базового класса, к любому поколению и к методам, используемым базовым классом.

Используя параметрический полиморфизм, можно создавать связанные методы и связанные типы. Связанные методы и типы позволяют вам написать программу, которая может обрабатывать множество типов доказательств. Если использование объединяющего полиморфизма влияет на восприятие объекта, использование параметрического полиморфизма повлияет на используемые методы. Используя параметрический полиморфизм, можно создавать связанные методы, не объявляя тип параметра до времени выполнения. Как и в случае параметрических параметров метода, сам вид может быть параметрическим. Однако этот тип полиморфизма встречается не во всех языках.

При перегрузке одно имя может обозначать разные методы. Методы отличаются только количествами и типами параметров. Перегрузка полезна, когда метод не зависит от своих аргументов. Способ не ограничен определенными типами параметров, но применяется к различным типам параметров. Например, рассмотрим метод max. Макс - это связанная концепция, которая принимает два конкретных параметра и сообщает им, какой из них больше. Описание не изменяется при сравнении целых чисел или чисел с плавающей точкой.

Первым шагом к эффективному использованию полиморфизма является эффективное использование инкапсуляции и наследования. Без программы инкапсуляции — это может легко зависеть от реализации в классе. Если

приложение связано с одним из аспектов реализации классной комнаты, это невозможно исправить в сетевом классе.

Наследование является важной составляющей полиморфизма соединения. Вы всегда должны пытаться установить замещающие отношения, пытаясь программировать как можно ближе к базовому классу. Этот метод увеличивает количество типов объектов, обрабатываемых в программе.

Хорошо продуманная стратификация помогает строить отношения. Необходимо привести общие части в абстрактные классы и программировать объекты таким образом, чтобы они были запрограммированы, а не специализированными копиями. Это позволит любому классу-преемнику примениться в программе.

Однако во многих случаях неопытные дизайнеры пытаются поднять свое поведение до очень высокого уровня, чтобы воспроизвести полиморфизм. В этом случае любое поколение может выдержать такое поведение. Важно помнить, что поколения не могут получить функции своих предков. Нет необходимости разбивать тщательно спланированные слои преемственности, чтобы сделать программу более полиморфной.

Вопросы по главе 10

1. Что такое инкапсуляция?
2. Понятие о полиморфизме.
3. Использование преемственности.
4. Свойства и методы класса?
5. Что такое абстракция?

Тестовые вопросы:

1. Инкапсуляция –это..

а) Процесс копирования заданных объектов;

б) Механизм объединения данных и кода обработки;

в) объектно-ориентированный язык программирования;

г) Статус актива - это текущая стоимость его членов;

2. Абстракция–это…

а) Механизм объединения данных и обработки кода;

б) процесс копирования заданных объектов;

в) Любой язык программирования, отличный от идентификатора.

г) Статус актива - это текущая стоимость его членов;

3. Какую возможность имеет преемник программы?

а) Любой язык программирования, отличный от дескрипторов;

б) Создает новый класс на основе существующего определения класса;

в) скопировать предоставленные объекты;

г) обработка функций;

4. Сколько существует основных полиморфизмов?

а) 5;

б) 2;

в) 3;

г) 4;

5. Как называется каждый представитель класса (копия)?

а) C ++ Builder;

б) наследование;

в) абстракция ;

г) объект;

Глава 12. ГРАФИЧЕСКИЕ ВОЗМОЖНОСТИ BORLAND C++ BUILDER 6

Ключевые слова: интерфейс, графика, холст, изображение объекта, координаты, пиксели, карандаши, скульптуры, изображения.

12.1. Работа с готовыми изображениями

Borland C ++ Builder позволяет разработчику создавать схемы, диаграммы и графики с помощью графического программного обеспечения. В среде программирования Borland C ++ Builder доступно 2 различных типа графики.

1. Готовые картинки.

2. Созданные пользователем изображения.

Дополнительным компонентом является компонент изображения, используемый для работы с готовыми изображениями. После того, как компонент установлен в окне формы, он выглядит следующим образом:

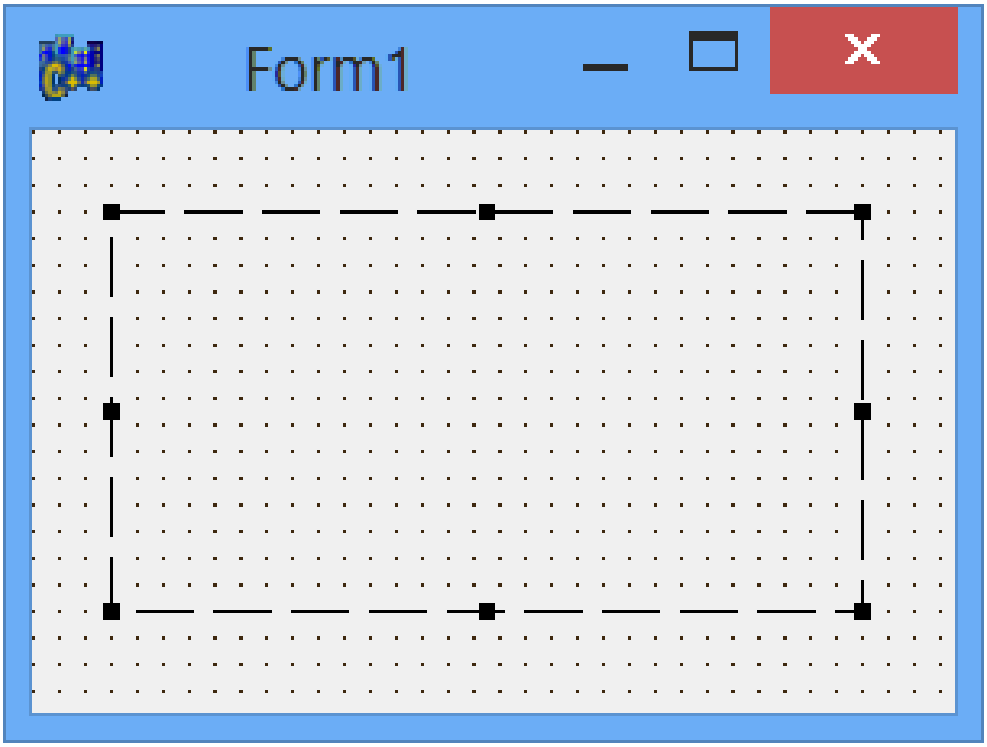


Рис. 12.1. Компонент изображения.

Размеры компонентов изображения корректируются с помощью свойства height и width. Компонент изображения компонента изображения выбирает желаемое изображение. Если щелкнуть свойство Picture, появится следующее окно:

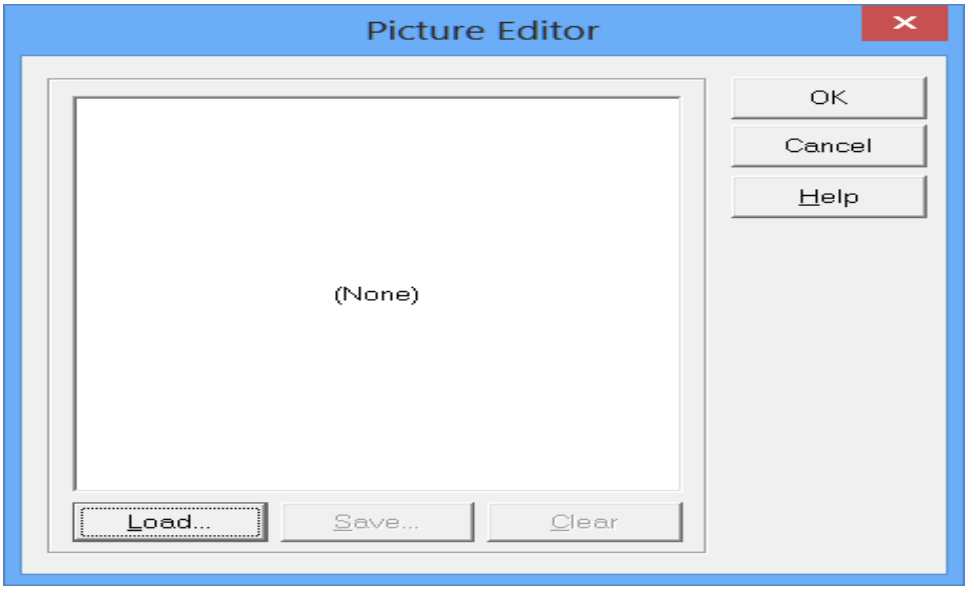


Рис. 12.2. Компонент изображения Спецификация изображения.

Во всплывающем окне нажмите кнопку загрузки и выберите путь к изображению.

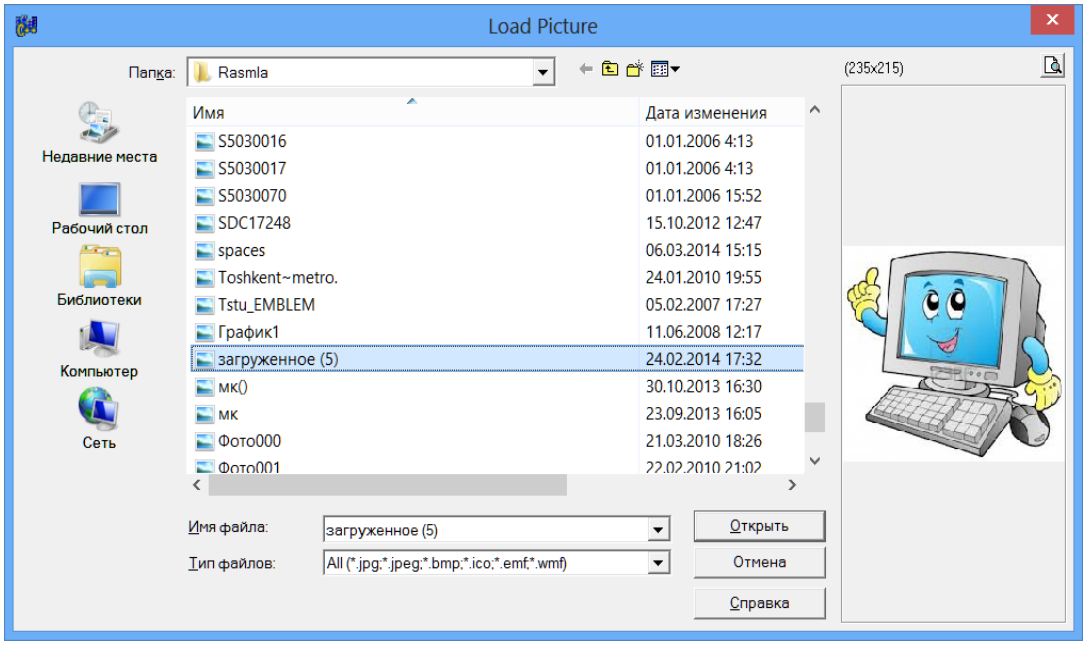


Рис. 12.3. Вкладка «Выбор изображения».

После выбора соответствующего файла вид программы будет выглядеть так:

Рис. 12.4. Окно результатов.

12.2. Карандаш и кисть Графические функции Borland C ++ Builder также позволяют использовать карандаши и шаблоны. Карандаш используется для рисования линий и контуров, а кисть - для рисования поверхности, очерченной контуром.

Карандаш и кисть характерны для пера (ручки) и кисти соответственно при рисовании на поверхности для рисования. Pen.

Карандаш, точка, геометрические фигуры: прямоугольник, круг, эллипс и т. Д. используется как оружие для рисования Свойство объекта TPen показано в таблице 12.1. Таблица

12.2. Карандаш и кисть

Свойство	Задача
Color	Цвет линии (контура)
Width	Толщина линии
Style	Вид линии
Mode	Режим изображения

Свойство Color объекта TPen устанавливает цвет карандаша для рисования. Ниже перечислены особенности PenColor:

Таб. 12.2 Особенность PenColor

Константа	Цвет	Константа	Цвет
clBlack	черный	clSilver	серебряный цвет
clMaroon	каштановый цвет	clRed	красный
clGreen	зеленый	clLime	Светло зеленый
clFuchsia	розовый	clBlue	голубой
clNavy	синий	clYellow	желтый
clPurple	цвет роз	clMedGray	светло серый
clGray	серый	clWhite	белый

Свойство width объекта TPen определяет толщину пера для рисования (в пикселях).

Например, Canvas->Pen->Width=2 имеет толщину 2 пикселя.

Свойство style объекта TPen определяет тип рисованной линии. Свойства стиля объекта TPen показаны в таблице 12.3.

Таб. 12.3 Особенности стиля

Константа		Вид
psSolid	Прямая линия	————
psDash	Длинная пунктирная линия	— — — — -
psDot	Короткая пунктирная линия	.....
psDashDot	Длинная пунктирная линия	– · – · – · –
PsDashDotDot	Одна длинная и две короткие пунктирные линии	— · · — · · —
PsClear	Невидимая линия	

2. Окраска. Кисть используется для заполнения закрытых областей, таких как геометрическое формирование и так далее. Скульптура как объект обладает двумя свойствами:

• Цвет - цвет окрашенной области;

• Стиль - это поле зрения.

Например, внутренний контур контура может быть окрашенным или полосатым. Вы можете использовать все переменные в качестве функции цвета.

Особенности стиля показаны в таблице12.4.

Таб. 12.4 Свойство стиля

Константа	Тип покраски
bsSolid	Сплошное раскрашивание
bsClear	Окрашивание без объекта
bsHorizontal	Горизонтальная штриховка
bsVertical	Вертикальная штриховка
bsFDiagonal	Диагональная штриховка с изгибом вперед
bsBDiagonal	Диагональная штриховка с изгибом назад
bsCross	Горизонтально-вертикальнаяштриховка в виде сети
bsDiagCross	Диагональная штриховка, в виде сети

Пример 1. Создайте программу, показывающую все возможности цветовой схемы. В этом

Вам понадобится 1 компонент BitBtn и 1 Label для программирования примера в визуальной среде.

Мы определяем свойства компонентов окна формы следующим образом:

Таб. 12.5 Ввод свойств компонента

Имя компонента	**Имя свойства (Состояние окна ObjectInspector)**	**Процесс, который должен быть выполнен**
Form1	Caption (Properties)	Вводится слово “Карандаш и кисть”.
Label1	Caption (Properties)	Вводится слово “Особенности покраски”.
BitBtn1	Kind (Properties)	Выбираем свойсво “bkOK”.
	Caption (Properties)	Вводится слово “кисть”.
	OnClick (Events)	Текст программы введен.

После того, как существующие компоненты компонента введены, дизайн программы показан на рисунке 12.5.

```
//---------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
```

```
Canvas->Pen->Width=8;
Canvas->Brush-> Style=bsClear;
Canvas->Pen->Color=clRed;
Canvas->Rectangle(50,150,150,250);
//---------------------------------------------------
Canvas->Brush-> Style=bsSolid;
Canvas->Pen->Color=clMedGray;
Canvas->Brush-> Color=clMedGray;
Canvas->Rectangle(170,150,270,250);
//---------------------------------------------------
Canvas->Brush-> Style=bsHorizontal;
Canvas->Pen->Color=clBlack;
Canvas->Brush-> Color=clBlack;
Canvas->Rectangle(290,150,390,250);
Canvas->Brush-> Style=bsVertical;
Canvas->Pen->Color=clBlue;
Canvas->Brush-> Color=clBlue;
Canvas->Rectangle(410,150,510,250);
//---------------------------------------------------
Canvas->Brush-> Style=bsFiagonal;
Canvas->Pen->Color=clMaroon;
Canvas->Brush-> Color=clMaroon;
Canvas->Rectangle(530,150,630,250);
//---------------------------------------------------
Canvas->Brush-> Style=bsBDiagonal;
Canvas->Pen->Color=clNavy;
Canvas->Brush-> Color=clNavy;
Canvas->Rectangle(650,150,750,250);
//---------------------------------------------------
Canvas->Brush-> Style=bsCross;
```

```
Canvas->Pen->Color=clPurple;
Canvas->Brush-> Color=clPurple;
Canvas->Rectangle(770,150,870,250);

//---------------------------------------------------------
Canvas->Brush-> Style=bsDiagCross;
Canvas->Pen->Color=clFuchsia;
Canvas->Brush-> Color=clFuchsia;
Canvas->Rectangle(890,150,990,250);
}
//---------------------------------------------------------
```

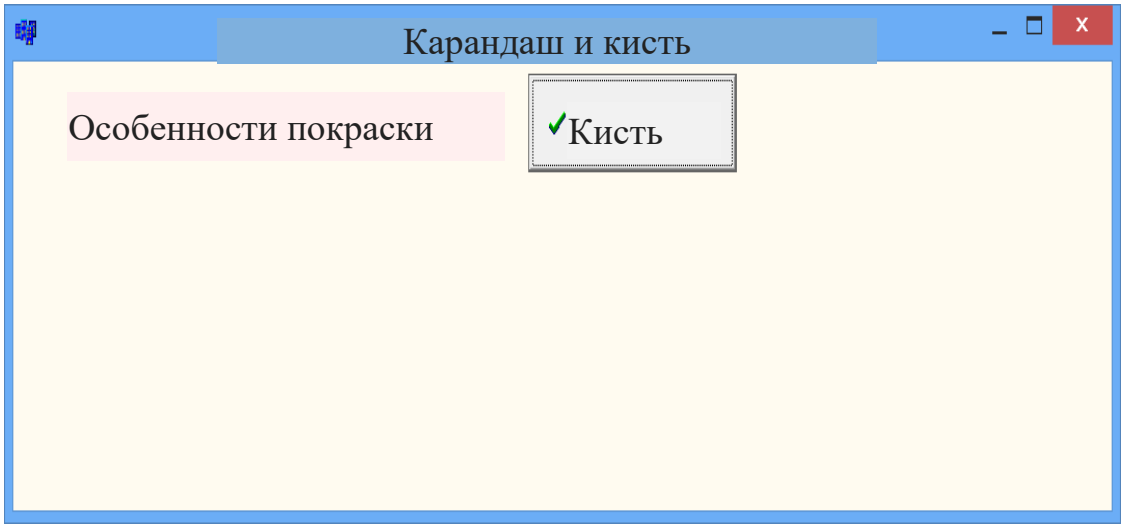


Рис. 12.5. Просмотр приложения.

Цвет программы отличается по цвету. После ввода текста программы нажмите F9, и появится следующая версия программы:

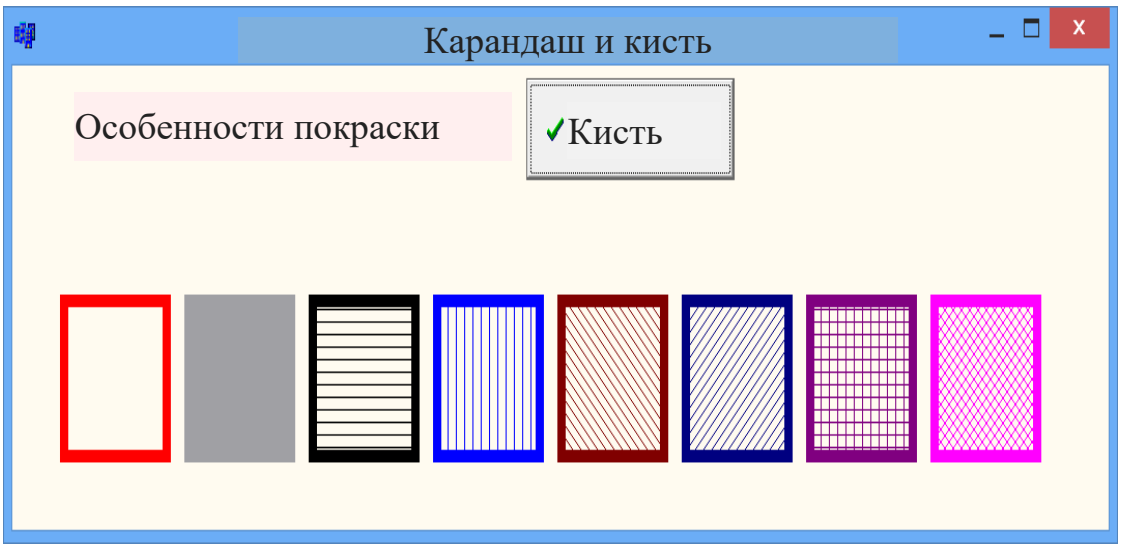


Рис. 12.6. Окно результатов.

12.3. Рисование простых фигур

Свойства холста используются для создания графических элементов (линии, круга, прямоугольника и т. Д.) На поверхности объекта.

Borland может создавать следующие графические элементы в C ++ Builder:

1. Прямая Borland использует LineTo для создания прямой линии в C ++. Его написание выглядит следующим образом:

Canvas->LineTo (x, y);

LineTo рисует линию прямо от точки, где карандаш (стрелка) указывает на x, y -. Поэтому начальная точка линии должна быть отрегулирована в правильное положение. Здесь мы ссылаемся на MoveTo:

Canvas->MoveTo (x0, y0);

Форма (цвет, толщина и тип) линии представлена объектом Pen.

2. Круг и эллипс. Метод эллипса используется для рисования эллипсов и кругов. Формат письма Эллипса следующий:

Canvas->Ellipse (x1, y1, x2, y2);

где x1, y1, x2, y2 - координаты верхней левой и нижней правой сторон прямоугольника, образованного кругом или эллипсом (рис. 12.7).

Форма (цвет, толщина и тип) линии представлена объектом Pen.

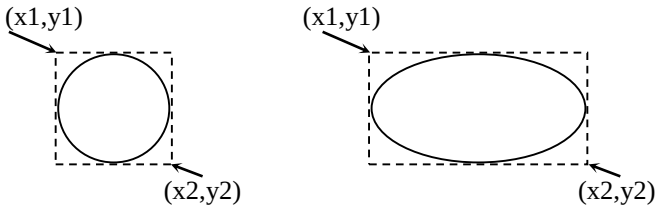


Рисунок 12.7. Присвоение значения кругу и эллипсу.

3. Стрелка. Метод дуги используется для создания дуг. Его формат выглядит следующим образом:

Canvas-> Arc (x1, y1, x2, y2, x3, y3, x4, y4);

где x1, y1, x2, y2 - координаты прямоугольника, обращенного наружу к эллипсу, который следует за результирующей дугой; x3, y3 - начальная точка дуги; x4, y4 - конечная точка дуги.

Следует отметить, что дуга рисуется в направлении, противоположном точке часов (рисунок 12.8).

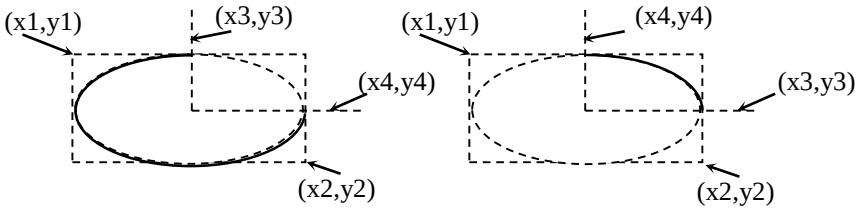


Рис. 12.8. Присвойте значение дуги

Форма (цвет, толщина и тип) линии представлена объектом Pen.

4. Правильный прямоугольник. Метод Rectangle используется для создания правильного прямоугольника. Его формат выглядит следующим образом:

Canvas->Rectangle (x1, y1, x2, y2);

где x1, y2, x2, y2 - правая верхняя правая и нижняя правая координаты прямоугольника.

Стиль RoundRect также рисует правый прямоугольник, за исключением Rectangle, углы которого круглые (гладкие). Формат подписки:

Canvas->RoundRect (x1, y1, x2, y2, x3, y3)

где x1, y2, x2, y2 - правая верхняя правая и нижняя правая координаты прямоугольника; x3, y3 - размеры эллипса, используемые для формирования круга (рисунок 12.9).

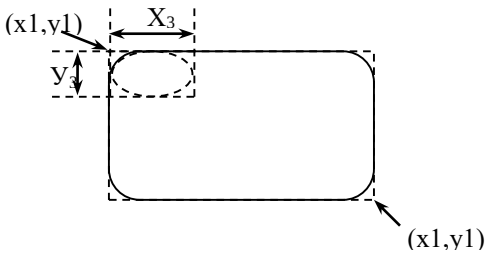


Рис. 12.9. Присвойте значение прямоугольнику с закругленными краями

5. Полигон. Можно нарисовать многоугольник, используя специфику многоугольника. В качестве параметра полигон принимает массив типов TPoint. Каждый элемент массива содержит координаты (x, y) одного многоугольника. Характеристика многоугольника объединяет эти точки с последовательными прямыми линиями. Формат подписки выглядит следующим образом:

Canvas->Polygon (p, n);

Здесь p - набор массивов типа TPoint, содержащий координаты многоугольника. n - количество углов многоугольника.

6. Сектор. Метод Pie используется для создания сектора эллипса или круга. Общий формат стиля пирога:

Canvas->Pie (x1, y1, x2, y2, x3, y3, x4, y4)

где x1, y1, x2, y2 - координаты правого прямоугольника, нарисованного на эллипсе (окружности) путем продолжения результирующего сектора; x3, y3 - начальная точка сектора; x4, y4 - конечная точка сектора (рисунок 12.10).

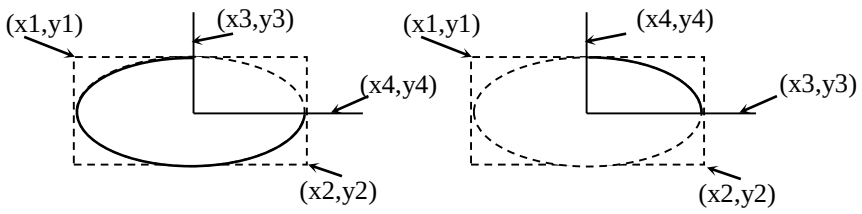


Рисунок 12.10. Придавать значение сектору.

7. Сгенерируйте текст. Используйте метод TextOut для генерации текста на графической поверхности. Формат записи TextOut выглядит следующим образом:

Canvas->TextOut (x, y, Text);

где x, y - координатная отправная точка текста; Текст - это общий текст или строковая переменная (рисунок 12.11).

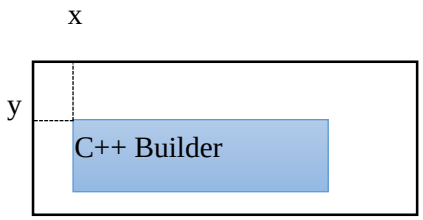


Рисунок 12.11. Координирует область, в которой генерируется текст.

Результирующие текстовые символы представлены свойством Font, соответствующим объекту Canvas. Свойство font принадлежит объекту TFont, и в таблице 12.1 перечислены свойства символов и используемые методы.

Таб. 12.1 Свойства объекта шрифта

Особенность	Идентифицированный
Name	Шрифт для использования. Имя шрифта указывается в качестве значения. (Например, TimesNewRoman)
Size	Размер шрифта, выраженный в пунктах. Это единица измерения, используемая в пунктуации, которая составляет приблизительно 1772 дюйма.
Style	Способ написания символа может быть следующим: обычный, жирный, курсив, подчеркнутый, с наложением. Это делается с использованием следующих констант: fsBold (полужирный), fsItalic (курсив), fsUnderline (подчеркнутый) и fsStrikeOut.
Color	Цвет знака. Константы TColor могут быть использованы в качестве значений.

Пример 2. Создайте графические объекты, используя вышеуказанные методы.

Вам понадобятся компоненты 1 Label, 1 Image и 2 BitBtn для программирования вашего примера в визуальной среде.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 12.2 Ввод свойств компонента

Имя компонента	**Имя свойства (Состояние окна ObjectInspector)**	**Процесс, который должен быть выполнен**
Form1	Caption (Properties)	Вводится слово “Работа с фигурами”.
Label1	Caption (Properties)	Вводится слово “Окно результатов”.
BitBtn1	Kind (Properties)	Выбираем свойство “bkOK”.
	Caption (Properties)	Вводится слово“Результат”.
	OnClick (Events)	Текст программы введен.
BitBtn2	Kind (Properties)	Выбираем свойство“bkClose”.
	Caption (Properties)	Вводится слово “Выход”.
	OnClick (Events)	Вводится Close();

После того, как существующие компоненты компонента введены, дизайн программного обеспечения выглядит следующим образом:

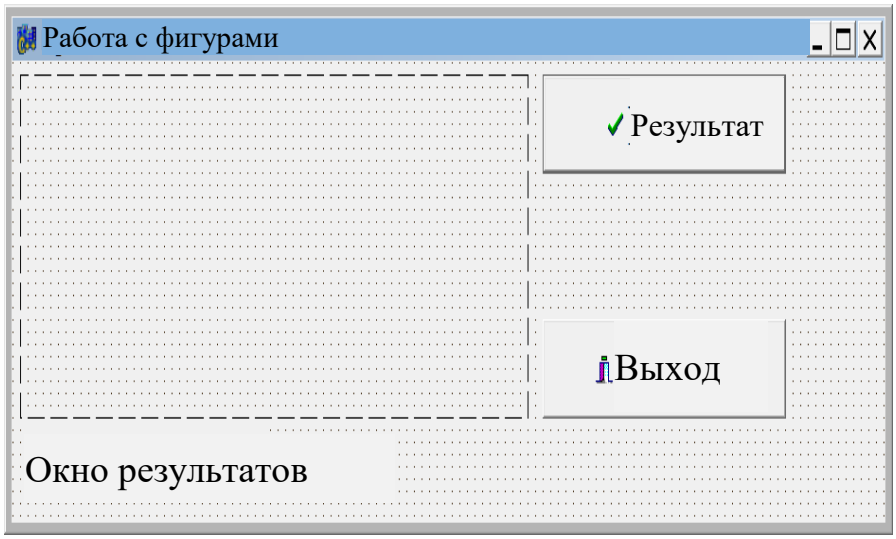


Рис. 12.12. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"

//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
```

```
{
Image1->Canvas->Font->Style << fsBold;
Image1->Canvas->Pen->Width=5;
 Image1->Canvas->Arc(10,10,90,90,90,50,10,50);
 Image1->Canvas->TextOut(40,60,"Arc");
//---------------------------------------------------------
 Image1->Canvas->Chord(110,10,190,90,190,50,110,50);
 Image1->Canvas->TextOut(135,60,"Chord");
//---------------------------------------------------------
 Image1->Canvas->Ellipse(210,10,290,50);
 Image1->Canvas->TextOut(230,60,"Ellipse");

//---------------------------------------------------------
 Image1->Canvas->Pie(310,10,390,90,390,30,310,30);
 Image1->Canvas->TextOut(340,60,"Pie");
 TPoint points[5];
 points[0] = Point(30,150);
 points[1] = Point(40,130);
 points[2] = Point(50,140);
 points[3] = Point(60,130);
 points[4] = Point(70,150);
 Image1->Canvas->Polygon(points,4);
 Image1->Canvas->TextOut(30,170,"Polygon");
//---------------------------------------------------------
 points[0].x += 100;
 points[1].x += 100;
 points[2].x += 100;
 points[3].x += 100;
 points[4].x += 100;
 Image1->Canvas->Polyline(points,4);
```

```
 Image1->Canvas->TextOut(130,170,"Polyline");
//---------------------------------------------------------
 Image1->Canvas->Rectangle(230,120,280,160);
 Image1->Canvas->TextOut(230,170,"Rectangle");
//---------------------------------------------------------
 Image1->Canvas->RoundRect(330,120,380,160,20,20);
 Image1->Canvas->TextOut(325,170,"RoundRect");
//---------------------------------------------------------
 Image1->Canvas->MoveTo(430,160);
 Image1->Canvas->LineTo(470,10);
 Image1->Canvas->TextOut(430,170,"Chiziq");
}
//---------------------------------------------------------
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
Close();
}
//---------------------------------------------------------
```

После ввода текста программы нажмите F9, и появится следующая версия программы:

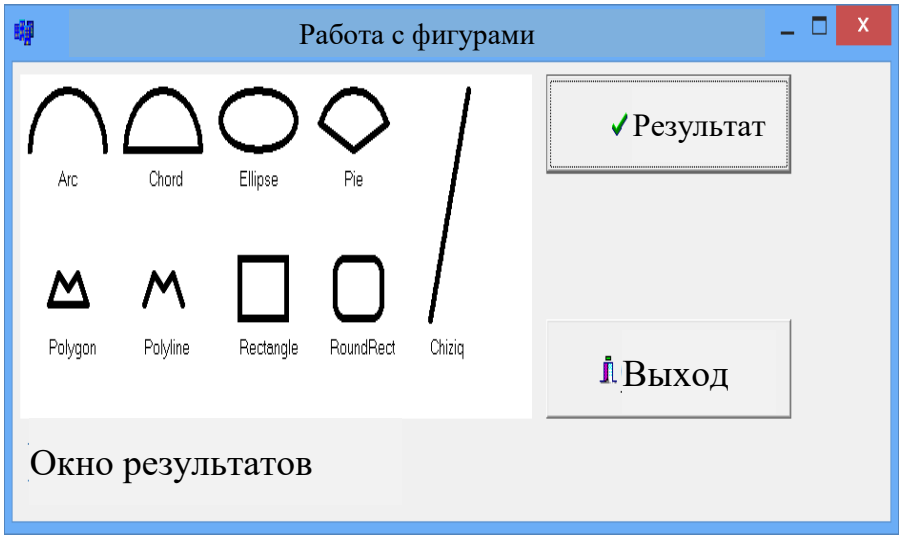


Рис. 12.13. Окно результатов.

Пример 3. Создать программу для рисования олимпийских флагов. Создать программу для визуальной среды.

Вам потребуется 1 компонент BitBtn для программирования примера в визуальной среде.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 12.3 Ввод свойств компонента

Имя компонента	**Имя свойства (Состояние окна ObjectInspector)**	**Процесс, который должен быть выполнен**
Form1	Caption (Properties)	Вводится слово “Графика”.
BitBtn1	Kind (Properties)	Выбираем свойство “bkOK”.
	Caption (Properties)	Вводитсяслово“Результат”.
	OnClick (Events)	Текст программы введен.

После того, как существующие компоненты компонента введены, дизайн программного обеспечения выглядит следующим образом:

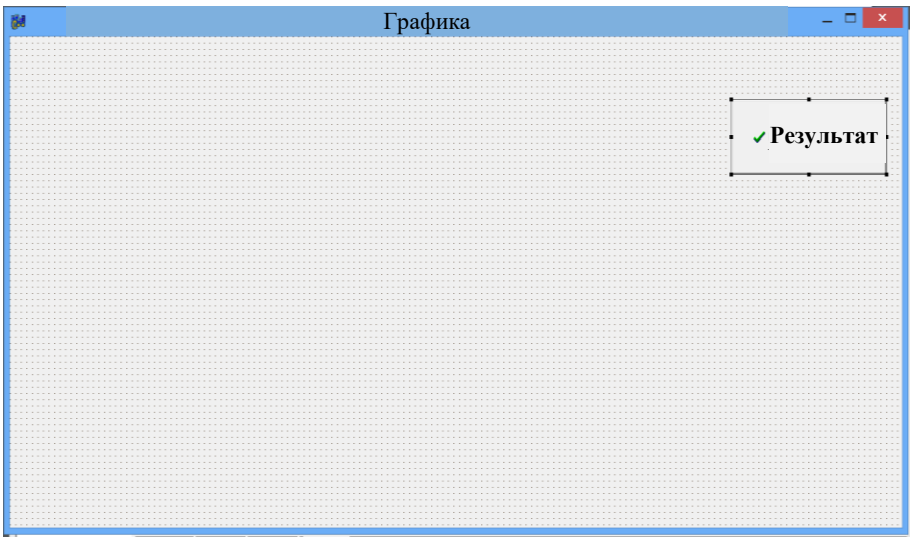


Рис. 12.14. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"

//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}

//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
Canvas->Pen->Width=1;
 Canvas->Pen->Color=clBlack;
 Canvas->Brush->Color =clCream;
 Canvas->Rectangle(30,30,700,500);
//
Canvas->Pen->Width=5;
Canvas->Brush-> Style=bsClear;
Canvas->Pen->Color=clBlue;
Canvas->Ellipse(150,140,310,300);
//
Canvas->Pen->Color=clBlack;
```

```
Canvas->Ellipse(270,140,430,300);
//
Canvas->Pen->Color=clRed;
Canvas->Ellipse(380,140,540,300);
//
 Canvas->Pen->Color=clYellow;
 Canvas->Ellipse(210,230,370,390);
//
 Canvas->Pen->Color=clGreen;
 Canvas->Ellipse(330,230,490,390);
}
//---------------------------------------------------------
```

После ввода текста программы нажмите F9, и появится следующая версия программы:

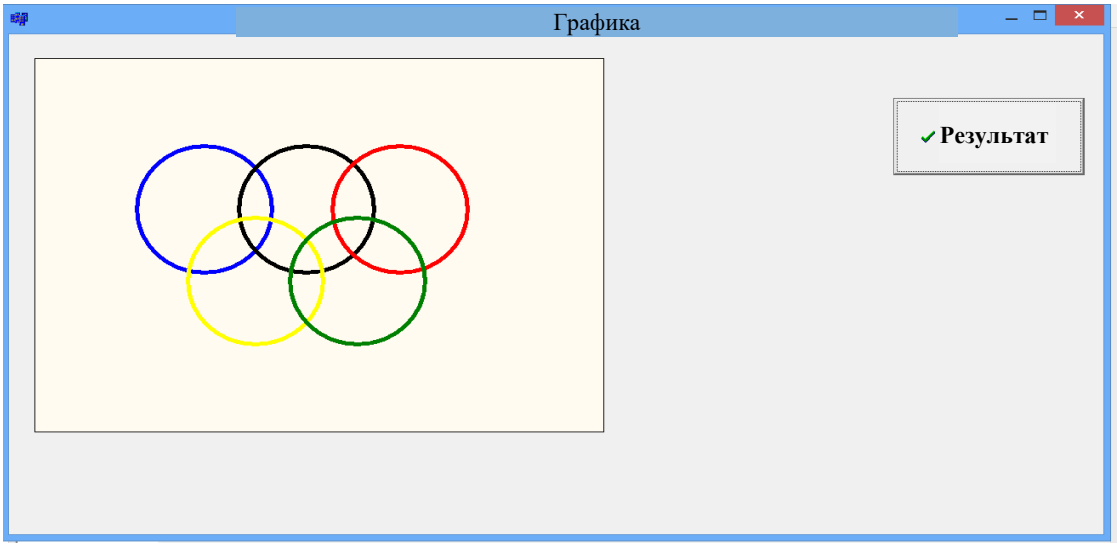


Рис. 12.15. Окно результатов.

Пример 4 Создать функцию графической программы.

Вам потребуется 1 компонент BitBtn для программирования примера в визуальной среде.

Мы определяем свойства компонента окна формы следующим образом:

Таб. 12.4 Ввод свойств компонента

Имя компонента	**Имя свойства (Состояние окна ObjectInspector)**	**Процесс, который должен быть выполнен**
Form1	Caption (Properties)	Вводитсяслово “Функциональный график”.
BitBtn1	Kind (Properties)	Выбираем свойство “bkOK”.
	Caption (Properties)	Вводится слово “Результат”.
	OnClick (Events)	Текст программы введен.

После того, как существующие компоненты компонента введены, дизайн программного обеспечения выглядит следующим образом:

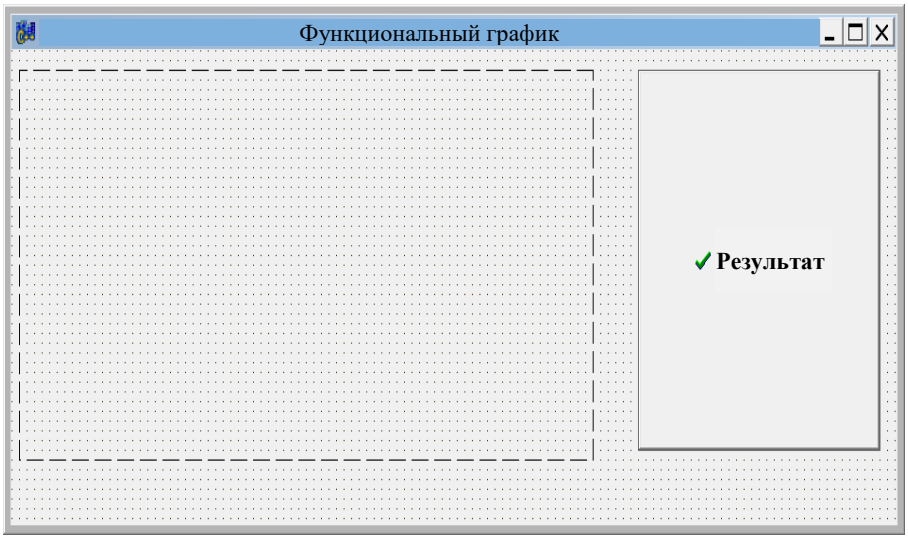


Рис. 12.16. Просмотр приложения.

Как только дизайн программы будет готов, будет добавлен следующий текст программы:

```
//---------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
```

```
#include "Unit1.h"

//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}

//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
float y,x,e;
        int z,w;
    Image1->Canvas->MoveTo(320,0);
        Image1->Canvas->LineTo(320,480);
    Image1->Canvas->MoveTo(0,240);
        Image1->Canvas->LineTo(640,240);
        z=320;
        w=240;
    for(x=0;x<=3*3.14;x+=3.14/60)
        {
            y=cos(x);
        Image1->Canvas->MoveTo(z,w);
        Image1->Canvas->LineTo(320+x*50,240-y*50);
        z=320+x*50;
```

```
        w=240-y*50;
        Image1->Canvas->TextOut(50,50,"u=cos(x) Функциональныйграфик ");
      }
    }
//---------------------------------------------------------
```

После ввода текста программы нажмите F9, и появится следующая версия программы:

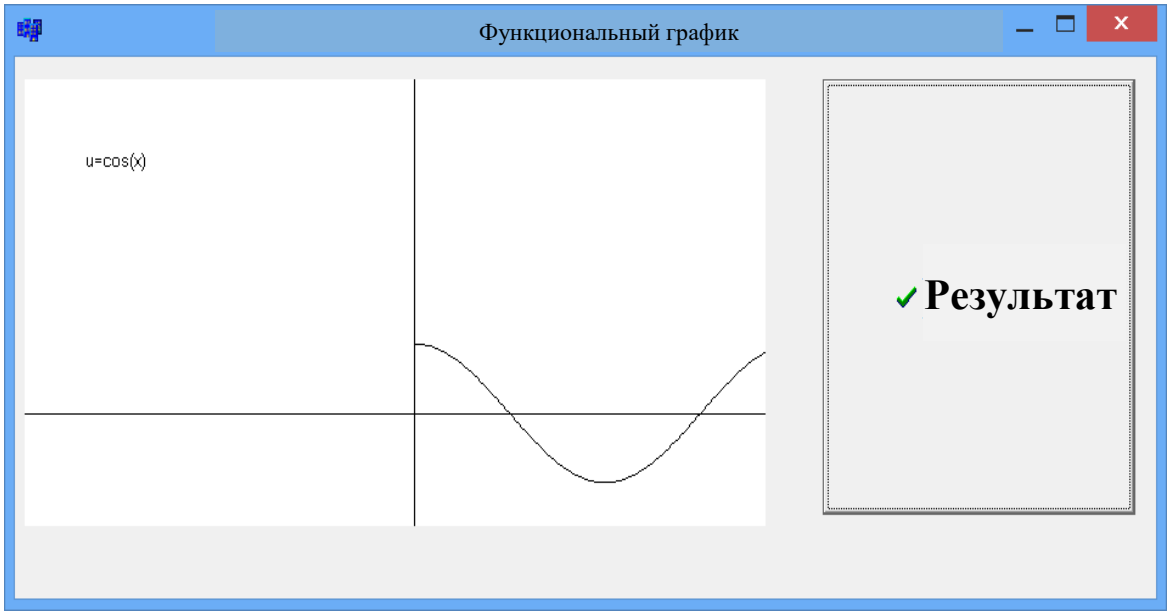


Рис. 12.17. Окно результатов.

Вопросы по главе 12.

1. C ++ графические особенности.
2. Функции создания изображений.
3. Особенности компонента изображения.
4. Карандаш и цветовые варианты для рисования графических изображений.
5. Функции ввода текста в графические изображения.

Тестовые вопросы:

1. Какой метод рисует дугу?

а) Arc(x1,y1,x2,y2,x3,y3,x4,y4);

б) Arc(x1,y1,x2,y2,x3,y3);

в) Ellipse(x1,y1,x2,u2,x3,u3,x4,u4);

г) Circle(x1,y1,x2,u2,x3,u3,x4,u4);

2. Характеристики пера (Pen) в графике

а) bsBDiagonal, color, style;

б) Canvas, image, pen;

в) Color, Width, Style;

г) Brush, color, style;

3. Какие методы используются для рисования линии в C ++?

а) Canvas->Ellipse(x1,y1,x2,y2);

б) MoveTo(x0,y0), LineTo(x,y);

в) Canvas->Arc(x1,y1,x2,y2,x3,y3,x4,y4);

г) Canvas->Rectangle(x1,y1,x2,y2);

4. Цвет объекта представлен каким свойством?

а) Pen;

б) Style;

в) Color;

г) Width;

5. Какой метод используется в C ++ для рисования круга?

а) Ellipse;

б) Circle;

в) Rectangle;

г) Arc;

Глава 13. ОБРАБОТКА БАЗЫ ДАННЫХ В ИНТЕГРИРОВАННОЙ СРЕДЕ BORLAND C++ BUILDER 6

Ключевые слова: база данных, системы управления базами данных, поле, строка, столбец, реляционная модель, сетевая модель, запрос, SQL, MicrosoftAccess, таблица.

13.1. Создание базы данных и их управление

Технология баз данных минимизирует многие проблемы создания традиционных файлов. База данных - это набор данных, который можно использовать для управления несколькими приложениями путем эффективного контроля и централизации данных. Это сохраняет все пользовательские данные в централизованном месте вместо хранения данных для каждого приложения в отдельных файлах.

База данных (БД) - это взаимосвязанная и организованная база данных, которая описывает свойства, условия и отношения между объектами в определенной области.

Создание баз данных для хранения, передачи и обработки данных, а также их широкое использование становятся сегодня все более актуальными. Управление большими объемами данных, извлечение их в любой форме по запросу, резервное копирование данных, сжатие больших объемов данных, организация управления базой данных в удобном интерфейсе, генерация отчетов и кроме этого, есть программные комплексы, которые делают много данных. Эти приложения известны как системы управления базами данных.

Системы управления базами данных (СУБД) - это программное приложение, которое позволяет централизовать данные организации, чтобы она могла эффективно управлять данными и использовать данные, хранящиеся в прикладном программном обеспечении. МВВТ действует как интерфейс между прикладным программным обеспечением и физическими

файлами данных. Используя традиционную файловую систему данных, программист должен определить размер и формат каждого элемента данных, используемого в программе, а затем указать местоположение на компьютере.

МВВТ - это набор программ, которые нужны многим пользователям для создания, добавления и обмена МБ. Основным компонентом СУБД являются данные.

Система управления базами данных является общей концепцией, которая включает в себя базу данных. Например, машиностроительный завод - это МВВТ, а машина - это база данных. МВВТ - это распространенная программа, которая управляет базой данных.

МВВТ минимизирует избыточность и минимизирует дублирующиеся данные. МВВТ разделяет программы и данные, что позволяет независимо использовать данные. Расширение доступа к данным, снижение затрат на разработку и поддержку программного обеспечения, а пользователи и программисты могут выполнять конкретные запросы к базе данных. СУБД позволяет организации централизовать управление данными и повысить безопасность информации.

Примерами СУБД являются: Oracle МВВТ, MySQL МВВТ, MS SQL Server МВВТ, MS Access МВВТ, которые могут создавать базы данных.

В то время как техническая часть содержит дополнительную внешнюю память, программная часть организует связь между МБ и пользователем. Структура БД зависит от внешнего вида, значения, структуры и размеров изучаемого объекта.

В настоящее время одним из самых популярных типов МВВТ для ПК и серверных компьютеров является реляционная модель. В реляционных моделях объекты и их взаимосвязи представлены в виде двумерных таблиц. Этот способ описания данных дает четкую картину взаимосвязи между объектами.

Таблица 13.1 дает пример.

Таб. 13.1 Реляционная модель

	Фамилия	Имя	Дата рождения	Группа	Место жительство
	Кадир	Мирза	22.05.19	117	Мирзо Улугбек 20дом
	Кабул	Фарход	12.02.19	118	Ибн Сино 14 дом
	Амин	Санат	14.05.19	11-	Алмазарсий район 15 дом
	Толип	Жасур	15.03.19	22-	Улица Беруний22 дом

13.2 Базы данных и их обработка в Borland C++ Builder 6

В этом разделе мы рассмотрим основы работы с базами данных в Borland C ++ Builder 6. Используя Borland C ++ Builder 6, вы можете создавать однопользовательские базы данных, а также серверные приложения, такие как Oracle, Sybase, Informix, Interbase, MS SQL Server, DB2 и ODBC MBBT. Borland C ++ Builder 6 обладает широким спектром возможностей, связанных с созданием приложений, использующих базы данных.

Borland C ++ Builder 6 имеет много компонентов базы данных. Вот некоторые из компонентов:

•  - Объект TTable используется для связи с существующими таблицами в базе данных. TTable может напрямую ссылаться на каждое поле и поле любого типа базы данных (FoxPro, ODBC, SQL и т. Д.). Этот компонент также может взаимодействовать с отдельными отчетами.

Перед использованием объекта TTable необходимо подключиться к базе данных, то есть в списке в свойстве DatabaseName этого компонента выберите нужную таблицу и в списке в свойстве TableName. Вот некоторые особенности объекта TTable:

✓ Активен - принимает два значения. Если «true» равно true, таблица активна, а «false» означает, что таблица закрыта.

✓ DatabaseName- это имя каталога, в котором находится требуемая таблица, или псевдоним удаленной базы данных в псевдониме. Надстройку

Alias можно установить с помощью конфигурации BDE или проводника SQL, расположенного в главном меню Database / Explore. Эту функцию можно изменить, только если таблица закрыта (ее свойство 'active' имеет значение false), например:

```
Table1->Active = false;
Table1->DatabaseName = "BCDEMOS"
Table1->Active = true;
```

- ✓ TableName- это имя таблицы.
- ✓ Исключительно - если для этого свойства установлено значение true, никакой другой пользователь не сможет открыть таблицу, если она открыта текущим приложением. Если для этого свойства установлено значение false, другие пользователи смогут открыть его.
- ✓ IndexName - указывает вторичный индекс для таблицы. Эту функцию нельзя изменить, когда таблица открыта.
- ✓ MasterFields - указывает имя поля для создания ссылки на другую таблицу.
- ✓ MasterSource- это имя компонента TDataSource, которое позволяет извлекать данные из таблицы, с которой связан компонент TTable.
- ✓ Поля - TField является узбекским массивом, который можно использовать для ссылки на номер поля. Например:

```
Edit1-> Text = Table1-> Fields [2] ->AsString;
```

•  Объект TADOTable- это то же самое, что и объект TTable для связи и ссылки на таблицу в базе данных. Этот объект в первую очередь предназначен для работы с базой данных, созданной системой управления базами данных MSAccess. Этот объект в основном используется с объектом TADOconnection и подключается к базе данных TADOConnection. Затем один или несколько объектов TADOTable подключаются к TADOConnection с помощью свойства Connection, а свойство tableName связывается с таблицей. Чтобы активировать

актив, значение свойства Active должно быть установлено в true. С помощью этого объекта вы можете извлекать данные из базы данных с помощью одного фильтра.

Вот некоторые свойства объекта TADOTable:

✓ Активен - принимает два значения. Если «true» равно true, таблица активна, а «false» означает, что таблица закрыта.

✓ ReadOnly - режим «Только для чтения» включается, если значение установлено в true. Свойство ReadOnly нельзя изменить, пока таблица открыта.

✓ TableName- это имя таблицы.

✓ Связь - для связи с базой данных в таблице.

✓ Фильтр - позволяет сортировать базу данных в таблице. Например, вы можете ввести следующий программный код, чтобы сделать выбор:

```
{ADOTable1->Filtered = false;
ADOTable1->Filter = "Name like ‘C*’";
ADOTable1->Filtered = true;
}
```

✓ Name - это имя компонента.

Объект TDataSource напрямую соединяется с TTable или TAdoTable, что позволяет редактировать и ссылаться на записи базы данных. Для этого выберите нужный элемент таблицы в списке свойств компонента DataSet, чтобы два объекта были связаны друг с другом. Объект TDataSource может подключаться к одной таблице в одной базе данных.

Все три из вышеперечисленных объектов являются невидимыми объектами во время выполнения приложения и не могут быть изменены, когда конструктор форм смотрит на них. Они могут быть связаны с базой данных как в режиме конструктора форм, так и при выполнении программы.

Для этого введите следующие коды:

```
{
Table1->DatabaseName= “DBDEMOS”;
 Table1->TableName= “animals.dbf”;
 Table1->Active=True;
 DataSource1->DataSet=Table1;
 DBGrid1->DataSource = DataSource1;
}
```

 Объект TDBgrid используется для отображения отчетов базы данных, таблиц и запросов в табличном представлении. Этот объект может отображаться, редактироваться и изменяться в базе данных. Внесенные вами изменения перезапишут текущую запись и будут сохранены только при переключении на другую запись или при закрытии приложения. Объект TDBgrid напрямую связан с объектом TDataSource с использованием свойства DataSource, тем самым отображая данные.

 Объект TDBNavigator (QDBCtrls) используется в тот момент, когда приложение обращается к записям базы данных с использованием компонентов TDBgrid или TDBedit. TDBNavigator используется для редактирования или просмотра записей базы данных пользователя. Когда пользователь нажимает на одну из кнопок TDBNavigator, действие, связанное с этим ключом, выполняется в программе.

В таблице 13.2 ниже показаны кнопки объекта TDBNavigator и их действия.

Таб. 13.2 Кнопка для объекта TDBNavigator

Кнопка	Выполняемые функции
First	Активировать начальную запись в базе данных. Работает только тогда, когда текущая запись не является исходной записью.
Prior	Включить запись перед текущей записью в базе данных. Работает только тогда, когда текущая запись не является исходной записью.
Next	Включить пост-текущую запись в базе данных. Это работает только тогда, когда текущая запись не является последней записью.
Last	Включить последнюю запись в базе данных. Это работает только тогда, когда текущая запись не является последней записью.
Insert	Добавьте новую строку данных в таблицу. Это сохранит изменения при вводе данных в произвольное поле.
Delete	Удалить текущую запись. Это потребует удаления записи, и удаленная запись не будет восстановлена.
Edit	Измените текущую запись так, чтобы она была редактируемой.
Post	Сохраните сделанные изменения в памяти. Это сохранит все изменения, которые были сделаны в текущем поле.
Cancel	Отменить изменения, внесенные в текущую запись. Это действие можно использовать до тех пор, пока текущая запись не будет изменена.
Refresh	Обновленные данные введены.

13.3 Обработка базы данных в технических системах

На основе вышеуказанных компонентов мы рассмотрим обработку базы данных в технических системах.

Пример 1. Используйте базу данных, созданную в MS Access, чтобы создать программу обработки групповой базы данных для окна формы.

Для программирования примера в визуальной среде вам понадобятся 1 ADOConnection, 1 ADOTable, 1 DataSourse, 1 DBNavigator, 1 DbGrid, 3 Label, 2 Edit и 2 BitBtn компонентов. Программа реализована в 2 вкладках формы.

Мы определяем свойства компонента окна Form1 следующим образом:

Таб. 13.3 Ввод свойств компонента

Имя компонента	**Имя свойства (Состояние окна ObjectInspector)**	**Процесс, который должен быть выполнен**
Form1	Caption (Properties)	Вводится слово “База данных технических систем”.
Label1	Caption (Properties)	Вводится слово “ Введите ваше имя пользователя”.
Label2	Caption (Properties)	Вводитсяслово“Введите пароль”.
Edit1	Text (Properties)	Удалитеслово “Edit1”.
Edit2	Text (Properties)	Удалитеслово “Edit2”.
BitBtn1	Kind (Properties)	Выбираем свойсво“bkOK”.
	Caption (Properties)	Вводится слово“Активировать”.
	OnClick (Events)	Текст программы введен.
BitBtn2	Kind (Properties)	Выбираем свойство “bkClose”.
	Caption (Properties)	Вводится слово “Выход из системы”.
	OnClick (Events)	Вводится Close();

После того, как существующие компоненты компонента введены, дизайн программного обеспечения выглядит следующим образом:

Рис. 13.1. Просмотр приложения.

В этой программе мы используем 2 окна формы. Мы определяем свойства компонентов в окне Form2 следующим образом:

Таб. 13.4 Ввод свойств компонента

Имя компонента	**Имя свойства (Состояние окна ObjectInspector)**	**Процесс, который должен быть выполнен**
Form1	Caption (Properties)	Вводится слово “Базы данных”.
Label1	Caption (Properties)	Вводится слово “ Обработка базы данных”.
AdoConnection	ConnectionString (Properties)	Use Connection String→Build→ MicrosoftJet 4.0 OLE DB Провайдер→далее→ База.mdb→Ok
	LoginPrompt(Properties)	false
ADOTable	Connection(Properties)	Connection1
	TableName(Properties)	Таблица
	Active(Properties)	True
DataSourse	Dataset(Properties)	ADOTablel1
DBGrid	DataSourse(Properties)	DataSourse1
DBNavigator	DataSourse(Properties)	DataSourse1

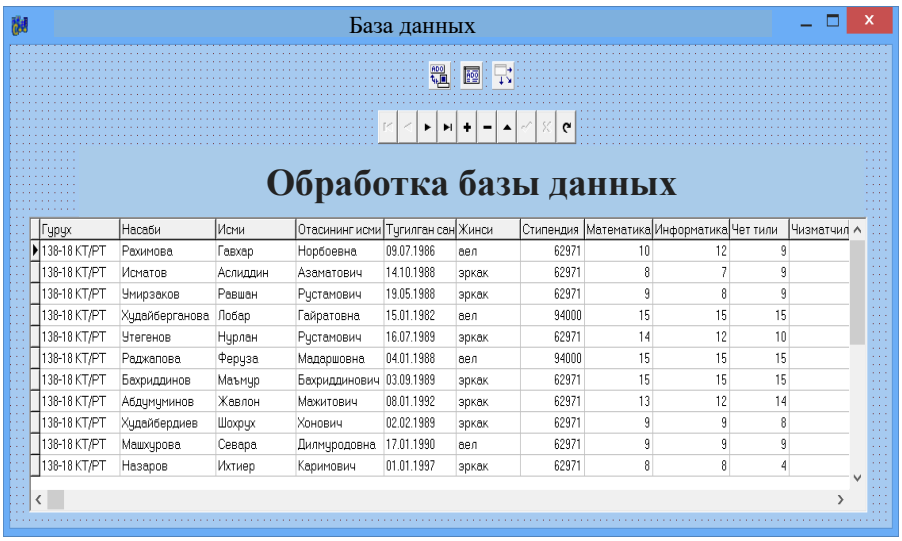


Рис. 13.2. Просмотр приложения.

После развертывания необходимых компонентов мы сделаем следующее:

AdoConnection получает доступ к свойству ConnectionString и ссылается на существующую базу данных. Для этого мы создаем таблицу с именем «Base.mdb» в MicrosoftAccess. Мы сохраняем файл в текущем каталоге, где был создан проект. Связывание файлов выполняется следующим образом (рисунок 13.3):

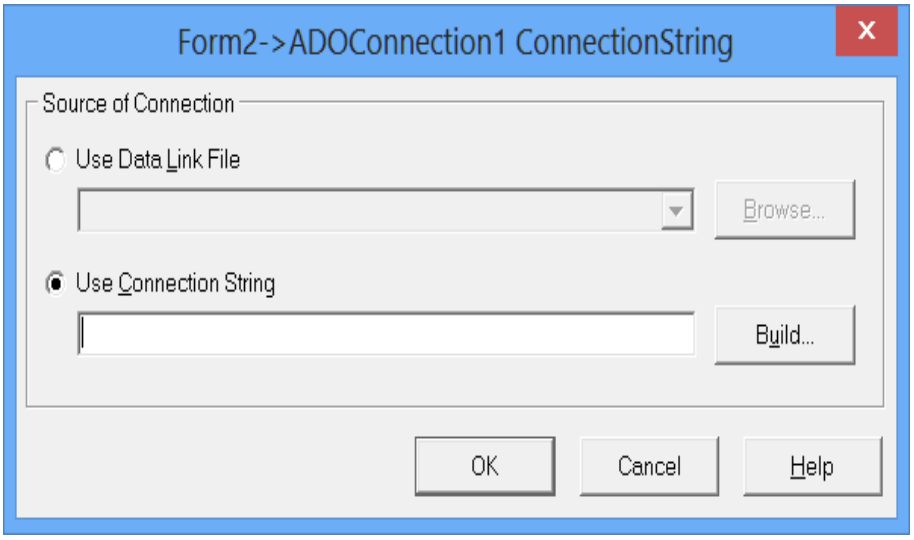


Рис. 13.3. Свойство ConnectionString.

Нажмите кнопку «Построить» во всплывающем окне. В результате появится список, позволяющий подключиться. Из списка выберите

MicrosoftJet OLEDB 4.0 Provider, на котором запущено программное обеспечение MS Aceess (рисунок 13.4):

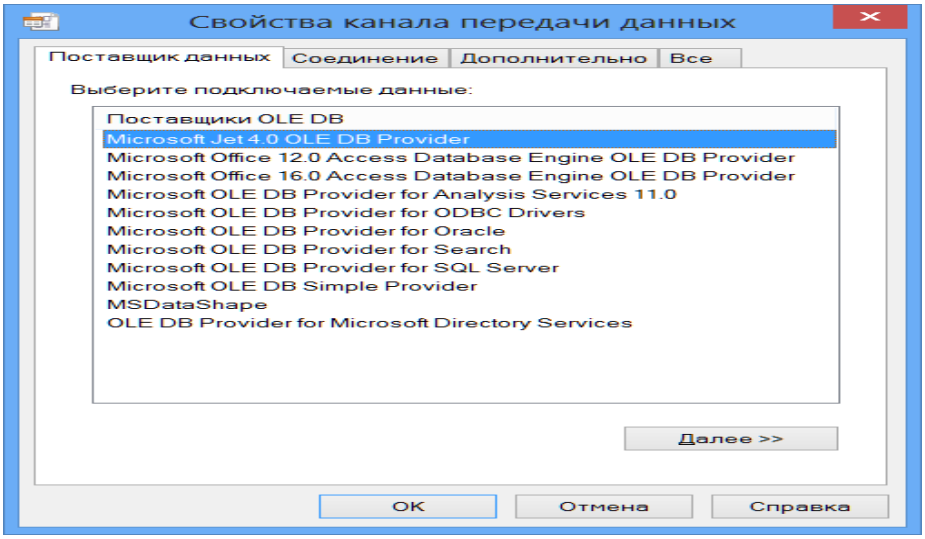


Рис. 13.4. Окно выбора поставщика MicrosoftJet OLE DB 4.0.

Когда выбран поставщик MicrosoftJet OLE DB 4.0, нажмите кнопку «Далее». В сгенерированном окне можно установить соединение с базой данных. Нажмите эту кнопку, чтобы показать файл, созданный в MS Acces.

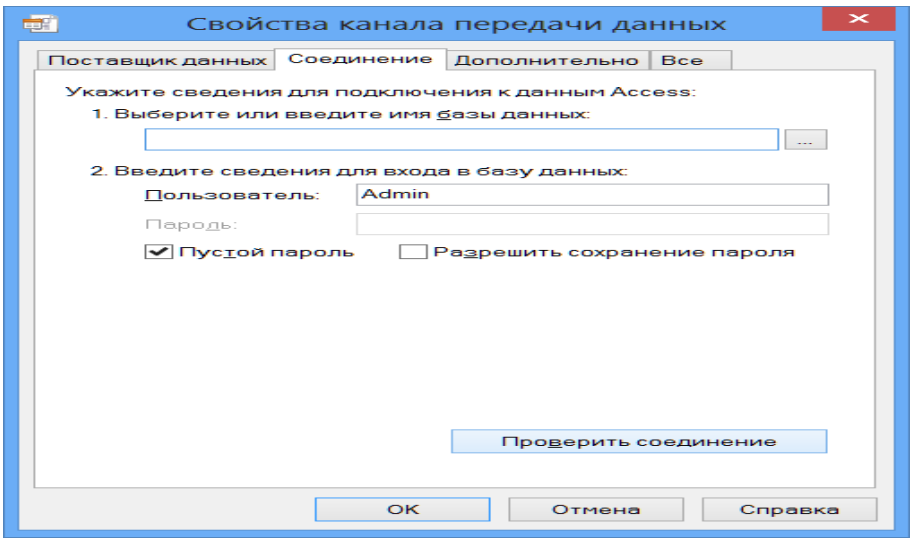


Рис. 13.5. Окно передачи данных.

Используйте окно по умолчанию в операционной системе Windows, чтобы выбрать файл базы данных.

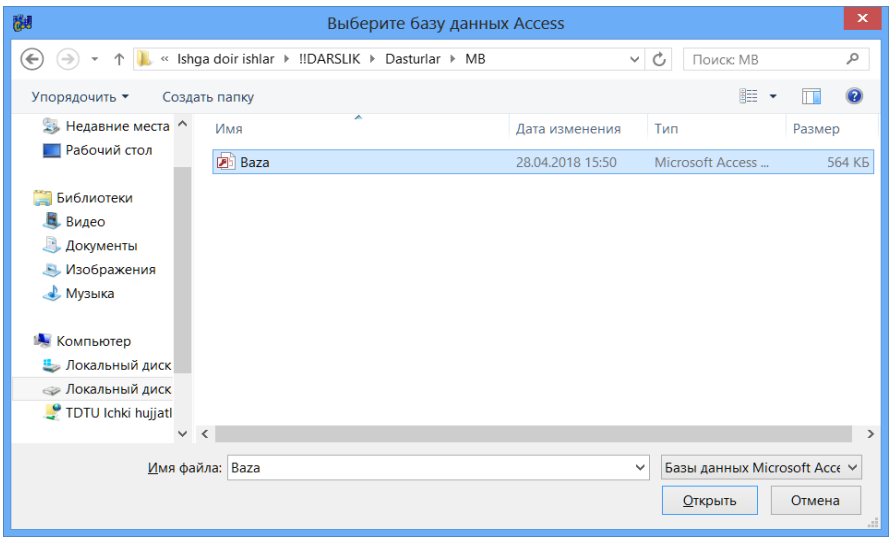


Рис. 13.6. Окно открытия файла.

После выбора файла он будет проверен кнопкой «Вкладка Безопасность». Если в результате «Привязка проверки» приложение считается подключенным к базе данных. Случай на рисунке 13.7 связан.

После того, как все действие выполнено, нажмите кнопку «ОК». Свойства всех других компонентов настраиваются, как показано в таблицах 13.3 и 13.4.

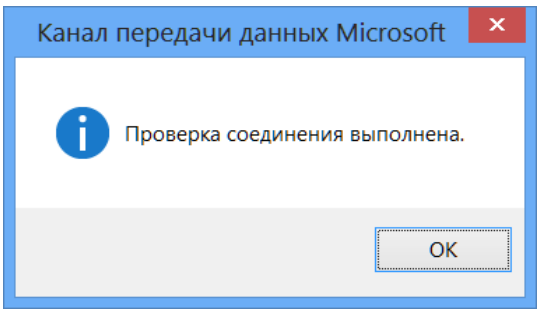


Рис. 13.7. Контрольный список подключения к данным.

Когда проект программы готов, вставляется следующий текст программы (для окна Form1):

```
//---------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include "Unit2.h"
```

```
//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}

//---------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
if (Edit1->Text == "MK")
{
      if (Edit2->Text == "12345")
{
            Form2->ShowModal();
     } else
{
ShowMessage (“неправильныйпароль”);
}
} else
{
ShowMessage (“неправильныйлогин”);
}
}
//---------------------------------------------------------
```

Текстокнаформы Form2:

```
//---------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit2.h"

//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;

//---------------------------------------------------------
__fastcall TForm2::TForm2(TComponent* Owner)
        : TForm(Owner)
{
}

//---------------------------------------------------------
void __fastcall TForm2::DBGrid1TitleClick(TColumn *Column)
{
if (ADOTable1->Active)
 if ((ADOTable1->Sort.Pos(Column->FieldName) > 0) && (ADOTable1->Sort.Pos("ASC") > 0))
{
ADOTable1->Sort = Column->FieldName + " DESC";
}
else
{
ADOTable1->Sort = Column->FieldName + " ASC";
```

```
}
}
//---------------------------------------------------------
```

После ввода текста программы нажмите F9, и появится следующая версия программы:

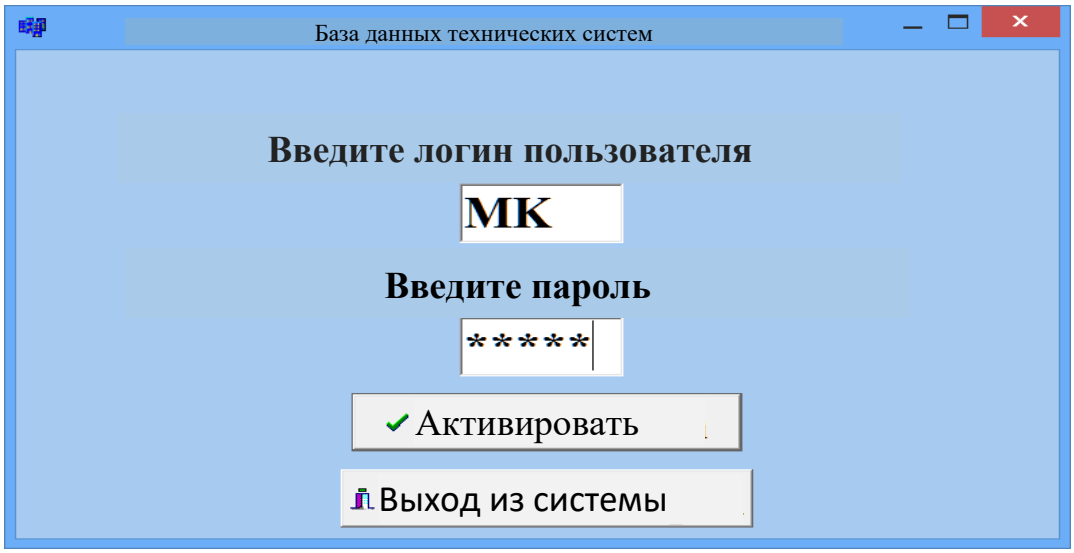
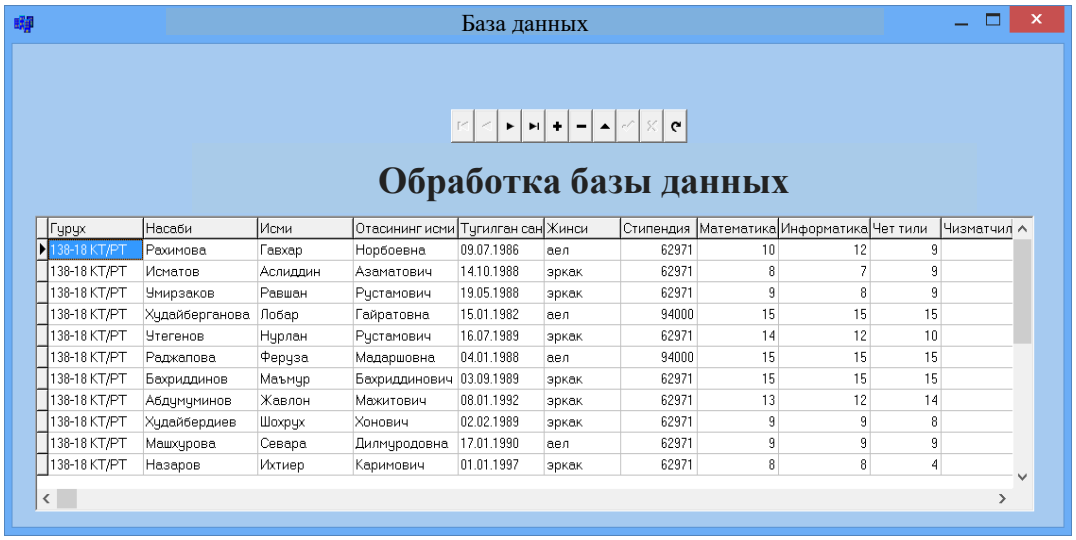


Рис. 13.8. Окно результатов.

После правильного ввода имени пользователя и пароля вы сможете войти в систему.



Гурух	Насаби	Исми	Отасининг исми	Тугилган сан	Жинси	Стипендия	Математика	Информатика	Чет тили	Чизматчил
138-18 КТ/РТ	Рахимова	Гавхар	Норбоевна	09.07.1986	ает	62971	10	12	9	
138-18 КТ/РТ	Исматов	Аслиддин	Азаматович	14.10.1988	эркак	62971	8	7	9	
138-18 КТ/РТ	Умирзаков	Равшан	Рустамович	19.05.1988	эркак	62971	9	8	9	
138-18 КТ/РТ	Худайберганова	Лобар	Гайратовна	15.01.1982	ает	94000	15	15	15	
138-18 КТ/РТ	Утегенов	Нурлан	Рустамович	16.07.1989	эркак	62971	14	12	10	
138-18 КТ/РТ	Раджапова	Феруза	Мадаршовна	04.01.1988	ает	94000	15	15	15	
138-18 КТ/РТ	Бахриддинов	Маъмур	Бахриддинович	03.09.1989	эркак	62971	15	15	15	
138-18 КТ/РТ	Абдумуминов	Жавлон	Мажитович	08.01.1992	эркак	62971	13	12	14	
138-18 КТ/РТ	Худайбердиев	Шохрух	Хонович	02.02.1989	эркак	62971	9	9	8	
138-18 КТ/РТ	Машхурова	Севара	Дилмуродовна	17.01.1990	ает	62971	9	9	9	
138-18 КТ/РТ	Назаров	Ихтиер	Каримович	01.01.1997	эркак	62971	8	8	4	

Рис. 13.9. Окно результатов.

Вопросы по главе 13

1. Что такое база данных?
2. Какие компоненты базы данных доступны в среде C ++ Builder 6?
3. Функция компонента AdoConnection?
4. Какова функция компонента DBNavigator?
5. Функция компонента DBgrid?

Тестовые вопросы:

1. Какое свойство используется для связи MS Acess с объектом AdoConnection?

а) ConnectionString;

б) LoginPrompt;

в) DataSourse;

г) Connections;

2. В каком меню компонентов находится объект TDataSource?

а) DataControls;

б) ADO;

в) DataAccess;

г) DataSourse;

3.  какой это компонент?

а) TADOTable;

б) TTable;

в) DataSourse;

г) TADO;

4.  Какой это компонент?

а) TADOTable;

б) DataSourse;

в) TTable;

г) DBNavigator;

5.  Какой это компонент?

а) TDBgrid;

б) DataSourse;

в) TTable;

г) TADOTable;

Ответы на раздел тестовых вопросов

Глава 1				
1	2	3	4	5
a	b	b	d	e
Глава2				
1	2	3	4	5
b	b	a	d	d
Глава3				
1	2	3	4	5
a	d	e	a	b
Глава4				
1	2	3	4	5
e	d	b	a	a
Глава5				
1	2	3	4	5
d	b	a	e	d
Глава6				
1	2	3	4	5
b	c	e	a	b
Глава7				
1	2	3	4	5
e	a	d	a	d
Глава8				
1	2	3	4	5
a	a	b	b	d
Глава9				
1	2	3	4	5
e	b	d	b	e

Глава10				
1	2	3	4	5
a	b	d	e	a
Глава11				
1	2	3	4	5
b	d	b	d	e
Глава12				
1	2	3	4	5
a	d	b	d	a
Глава13				
1	2	3	4	5
a	d	b	a	a

ГЛОССАРИЙ

A

Algoritm – amallarning cheklangan soni yordamida masala yechimini belgilovchi buyruqlarning to‘plami. Dasturlashdaalgoritmlarpsevdokod, blok-sxemava UML diagrammavositalariko‘rinishidatasvirlanadi.

Алгоритм– порядок действий, которые необходимо выполнить для решения определенной задачи. В программировании алгоритмы описывают средствами псевдокода, блок-схем и UML диаграмм.

Algorithm –the order of actions that must be performed to solve a particular task. In programming, algorithms are described by means of pseudocode, flowcharts and UML diagrams.

Amaliydasturlash – muayyanfaoliyatsohasidagiamaliymuammolarnihalqilishuchunmo‘ljallangandastur iyta’minotniishlabchiqishjarayoni. Ushbudasturlashniamaliydeyilishiningsababi, u apparat resurslaridanbevositafoydalanmaydivaunioperatsiontizimorqaliamalgaoshiradi.

Прикладное программирование – процесс разработки программного обеспечения, предназначенного для решения прикладных задач в определенной сфере деятельности. Такое программное обеспечение называют прикладным, и оно характеризуется тем, что не использует вычислительные ресурсы аппаратного обеспечения напрямую, а делает это посредством операционной системы.

Applied programming – the process of developing software designed to solve applied problems in a certain field of activity. Such software is called application software, and it is characterized by the fact that it does not use the hardware resources of the hardware directly, but does it through the operating system.

Amaliydasturlashinterfeysi – dasturlashilovalariinterfeysibo‘lib, dasturlashdarajasidadasturningtashqiko‘rinishixususiyatlarinio‘zichigaoladi.

Интерфейс прикладного программирования – интерфейс программирования приложения – функциональность приложения, доступная на программном уровне внешним программным компонентам.

Application programming interface (API) – application programming interface, available at the software level to external software components.

Assembler – ko‘rsatmalarimashinakodiko‘rsatmalarigamoskeladigan past darajadagidasturlashtili.Bundantashqari, assembler dasturi deb – past darajadagidasturlashtilidankompyuterkodigaaylantiruvchidasturgaaytiladi.

Ассемблер – язык программирования низкого уровня, инструкции которого соответствуют инструкциям машинного кода. Также, ассемблером называют программу – транслятор с языка программирования низкого уровня в машинный код.

Assembler – a low-level programming language, the instructions of which correspond to the instructions of the machine code. In addition, an assembler is called a program – a translator from a low-level programming language into machine code.

Axborottizimi – foydalanuvchilarniaxborotbilanta’minlashmaqsadidayaratilganma’lumotlarto‘plam ivaushbuma’lumotlargaxizmatko‘rsatuvchitexnik, dasturiyvatashkiliyresurslar.

Информационная система – совокупность данных и обслуживающих эти данные технических, программных и организационных ресурсов, создаваемая с целью информационной поддержки пользователей.

An information system is a collection of data and technical, program and organizational resources serving this data, created for the purpose of informational support of users.

B

Blok-sxema–algoritmlarnitavsiflashuchungrafikyozuv. Dasturiykomponentalarniishlabchiqishdavaishlashinimantiqiytahlilqilishjarayonida dasturchilartomonidanqo‘llaniladi.

Блок-схема – графическая нотация для описания алгоритмов. Используется программистами в процессе разработки и анализа логики работы программных компонентов.

Flowchart – graphical notation for the description of algorithms. Used by programmers in the process of developing and analyzing the logic of the operation of software components.

Boshqariladigankod–virtual mashinatomonidanbajariladigandasturkodi.

Управляемый код – программный код, исполняемый виртуальной машиной.

Managed code – the program code that is executed by the virtual machine.

Bulutlihisoblash – hisoblashnitashkiletishmodelibo‘lib, bundamijoztomonidanamalgaoshiriladiganhisoblashxizmatlarimijozresurslaridanko ‘rayuqoriimkoniyatgaegatarmoqtexnologiyalaridabajariladi. Server-mijozo‘rtasidagialoqatarmoqgaulanishyo‘liorqaliamalgaoshiriladi.

Облачные вычисления – модель организации вычислений, при которой вычислительные процессы, запрашиваемые клиентом, происходят на удаленных, намного более мощных по сравнению с клиентскими вычислительными ресурсами. Взаимодействиесерверасклиентомосуществляетсяпосредствомсетевогодосту па.

Cloud computing – a model of computing, in which the computing processes requested by the client, occur on remote, much more powerful compared to client computing resources. The interaction of the server with the client is carried out through network access.

D

Dastur – mualliftomonidanishlabchiqilganvayaratilgantizimdaishgatushurishmumkinbo‘lgan tugalanganmahsulot.

Программа – завершенный продукт, пригодный для запуска своим автором на системе, на которой он был разработан.

The program – a complete product, suitable for running by its author on the system on which it was developed.

Dasturiymahsulot– ixtiyoriyfoydalanuvchitomonidanishgatushuruluvchi, to‘g‘rilanuvchi, rivojlantiriluvchivatestlashimkoniyatimavjuddastur.

Программный продукт – программа, которую любой человек может запускать, тестировать, исправлять и развивать.

Software product – a program that any person can launch, test, correct and develop.

Dasturiymajmua– funksiyalarvaformatlarbo‘yichamuvofiqlashtirilganinteraktivdasturlarningto‘plami, muayyaninterfeysgaegavabirgalikdakattavazifalarnihaletishuchunto‘liqvositabo‘lib xizmatqiladi.

Программный комплекс – набор взаимодействующих программ, согласованных по функциям и форматам, точно определенным интерфейсам, и вкупе составляющих полное средство для решения больших задач.

Software package – a set of interacting programs that are coordinated by functions and formats, precisely defined interfaces, and together constitute a complete tool for solving large tasks.

Delphi – obyektgayo‘naltirilgandasturlashtili, Pascal dasturlashtiliasosidayaratilgan. Dasturiyta’minotniishlabchiqishmuhiti Borland kompaniyasigategishli.

Delphi – объектно-ориентированныйязыкпрограм-мирования, созданныйнаосновеязыкапрограммированияPascalисредаразработкипрограммныхпродуктовкомпанииBorland.

Delphi – an object-oriented programming language based on the Pascal programming language and the Borland software development environment.

E

ER diagrammasi– ER modeliningma'lumotlarinivizualsh-tirishninggrafikbelgilari.

ER диаграммы – графическая нотация визуализации данных ER модели.

ER diagrams – graphical notation of data visualization ER model.

ER model (Mohiyat-Munosabat) – ma'lumsoxadarelyatsionma'lumotlarbazasiniloyihalashuchunfoydalaniladigan model. U o'zichigama'lumotlarningmohiyativaularningo'zaromunosabatlariniqamraboladi.

ER модель (Сущность-Связь) – модель данных предметной области, используемая для проектирования реляционных баз данных в терминах сущностей и связей между ними.

ER model (Entity Relationship) – a domain data model used to design relational databases in terms of entities and the relationships between them.

F

Foydalanuvchininggrafikinterfeysi(FGI)– Windows (Microsoft), Mac OS (Apple) kabizamonaviyoperatsiontizimlartomonidantaqdimetilgangrafikfoydalanuvchiinterfeysi. FGI

sichqonchavaklaviaturayordamidamanipulyatsiyaqilinadigangrafikoynalar, tugmalar, ro‘yxatlarvaboshqaboshqaruvelementlaribilanifodalanadi.

Графический интерфейс пользователя (ГИП) – графический пользовательский интерфейс, предоставляемый современными операционными системами, такими как Windows (Microsoft), Mac OS (Apple) и т.п. ГИП представлен графическими окнами, кнопками, списками и прочими элементами управления, манипуляция которыми осуществляется посредством мыши и клавиатуры.

Graphical User Interface (GUI) is a graphical user interface provided by modern operating systems, such as Windows (Microsoft), Mac OS (Apple), etc. GUI is represented by graphic windows, buttons, lists and other controls, which are manipulated by mouse and keyboard.

Freymvork– turlidasturiyta’minotmahsulotlariningasosibo‘lgandasturiyta’minotturihisoblanadi.

Фреймворк – вид программного обеспечения, являющегося основой (каркасом) различных прикладных программных продуктов.

Framework – a kind of software, which is the basis (framework) of various application software products.

G

Geoaxborottizimi – ma’lumotlartizimi, uningvazifalariorasigajoylarninggeografikma’lumotlarnisaqlash, grafikko‘rinishita’minlashvama’lumotlardanfoydalanishninazoratqilishnio‘zichiga oladi.

Геоинформационная система (ГИС) – информационная система, в задачи которой также входит хранение, графическое отображение и управление доступом к пространственным (географическим) данным.

Geoinformation system (GIS) – is an information system, whose tasks also include storage, graphical display and control of access to spatial (geographic) data.

H

Hotiranidinamiktaqsimlash – dasturnibajarishvaqtidaxotiraniajratishnivabo‘shatishninazardatutadiganxotiraresurslariniboshqarish.

Динамическое распределение памяти – управление ресурсами памяти, предполагающее выделение, освобождение памяти во время выполнения программы.

Dynamic memory allocation – memory resource management, which involves allocation, freeing up memory during program execution.

HTML (HyperText Markup Language) – veb-sahifalaruchunbelgilashtilibo‘lib, ularnitashkiletuvchilarininiformatlashuchunmo‘ljallangan. Internet brauzerlarorqalinamoyishetiladi.

HTML (HyperText Markup Language) – языкразметкивеб-страниц, предназначенныйдляформатированияихсодержимого (контента), отображаемогоИнтернетбраузерами.

HTML (HyperText Markup Language) – a Web page markup language designed to format their content (content) displayed by Internet browsers.

I

Inkapsulyatsiya– axborotniyashirish, shuningdekobyektichidagima’lumotlarvafunksiyalarni (usullarni) birlashtirish.

Инкапсуляция – это сокрытие информации и комбинирование данных и функций (методов) внутри объекта.

Encapsulation – is the hiding of information and the combination of data and functions (methods) within an object.

J

Java –obyektgayo‘naltirilgandasturlashtili. Sun Microsystems kompaniyasitomonidanishlabchiqilgan.

Java – объектно-ориентированный язык программирования, разработанных компанией SunMicrosystems.

Java– an object-oriented programming language developed by Sun Microsystems.

JavaScript – Internet-brauzergao‘rnatilganprotsessualdasturlashtili. JavaScript tiliningmaqsadi - HTML formatlashelementlariyuklanganveb-sahifaningobyektmodeliningdasturinterfeyslariorqaliishlovberish.

JavaScript – встроенный в интернет браузер процедурный язык программирования. Назначение JavaScript - манипуляция элементами HTML разметки посредством программных интерфейсов объектной модели загруженной интернет страницы.

JavaScript is a procedural programming language built into the Internet browser. The purpose of JavaScript is to manipulate HTML markup elements through the program interfaces of the object model of the loaded web page.

jQuery – bibliotekahisoblanib, JavaScript dasturlashtilidayaratilgan. Maqsadidinamik Internet sahifalarniyaratishda HTML belgilashtilibilanishlashvauniboshqarishnisoddalashtirishuchunqo‘llaniladi.

jQuery – библиотека, написанная на JavaScript и созданная с целью упрощения взаимодействия с элементами HTML разметки при создании динамических интернет страниц.

jQuery is a library written in JavaScript and created to simplify interaction with HTML markup elements when creating dynamic web pages.

K

Kodnitekshirish– dasturiyta’minotkodinimuntazamvatizimlitahlilqilish, dasturiyta’minotniishlabchiqishningdastlabkibosqichlaridanxatoliklarnianiqlash, shuningdek, sifatsizyechimlarnivadasturdagitanqidiyjoylarnianiqlash.

Инспекция кода – систематический и периодический анализ программного кода, направленный на поиск необнаруженных на ранних стадиях разработки программного продукта ошибок, а также, на выявление некачественных архитектурных решений и критических мест в программе.

Code review – is a systematic and periodic analysis of program code aimed at searching for undetected errors in the early stages of software product development, and also for identifying poor-quality architectural solutions and critical locations in the program.

Kutubxona–dasturlashdadasturlardaishlatiladiganproseduralarni, ichkidasturlarni, funksiyalarni, makroslarnivaboshqama’lumotlarnisaqlaydiganfaylyokifayllarto‘plami.

Библиотека – в программировании – файл или совокупность файлов, в которых хранятся процедуры, подпрограммы, функции, макроопределения и другие данные, используемые программами.

Library – in programming – a file or a collection of files in which procedures, subroutines, functions, macros and other data used by programs are stored.

Kodniqaytaishlash – dasturkodigama’lumqoidalarasosidao‘zgartirishlarkiritishjarayoni. Qoidalargamuvofiqdasturningma’nosinio‘zgartirmasdandasturkodinitakomillashtirishimkoniniberadibuesainsonuchundasturkodiniosonvaqulayqiladi.

Рефакторинг кода – процесс внесения изменений в программный код в соответствии с некоторым набором правил – приемов рефакторинга, которые не меняют смысл программы, но делают ее код более стройным и легким для интерпретации человеком.

Refactoring – the process of making changes to the code in accordance with some set of rules – refactoring techniques that do not change the meaning of the program, but make its code more slim and easy to interpret by the person.

M

Manbakodi – algoritmiktildadasturningmatni. Kompyuterdadastlabkimatnkodito‘g‘ridan – to‘g‘riinterpretatordabajariladiyokioldinkompilyatortomonidanmuayyanhisob - kitobmuhitidatakroriybajarilishimumkinbo‘lganstandartyuklovchikodgakompilyatoryordamidatarjimaqilinadi.

Исходный код – текст программы на алгоритмическом языке. В компьютере исходный текст либо непосредственно выполняется интерпретатором, либо предварительно переводится компилятором в стандартный загрузочный код, способный многократно исполняться в определенной вычислительной среде.

Source code – the text of the program in an algorithmic language. In the computer, the source text is either directly executed by the interpreter, or it is previously translated by the compiler into a standard boot code that can be repeatedly executed in a certain computing environment.

Mashinatili – kompyuterbuyruqlariasosiyelementlarinitashkilqiluvchidasturlashtili.

Машинный язык – язык программирования, элементами которого являются команды компьютера.

Machine language is a programming language, the elements of which are computer commands

Moslashtirish– dasturiymahsulotlarniishlabchiqishnuqtainazaridandasturiymahsulotningfunktsiona

limkoniyatlarinifoydalanuvchiningtalablarigamoslashtirishjarayonisifatidanazardat utiladi.

Кастомизация – в контексте разработки программных продуктов может означать процесс настройки функциональ-ности программного продукта под требования конечного потребителя.

Customization in the context of software development can mean the process of setting the functionality of the software product to the requirements of the end user.

Modullidasturlash– budasturlashningshundayusullibo‘lib, bundadasturmodul deb ataladigantarkibiyqismlargabo‘linadivaularning har birinazoratqilinadigano‘lchamlari, aniqmaqsadivatashqimuhitbilanishlovchibatafsilinterfeysgaegabo‘ladi.

Модульное программирование – это такой способ программирования, при котором вся программа разбивается на группу компонентов, называемых модулями, причем каждый из них имеет свой контролируемый размер, четкое назначение и детально проработанный интерфейс с внешней средой.

Modular programming is a method of programming in which the whole program is divided into a group of components called modules, each of which has its own controlled size, a clear purpose and a detailed interface with the external environment.

Modul – 1) buyruqlarto‘plamibo‘lib, nomibo‘yichamurojaatetishimkoniniyaratadi;

2) dasturningoperatorlarito‘plamibo‘lib, chegaralanganelementlarvaindentifikatorgaega.

Модуль– 1) это совокупность команд, к которым можно обратиться по имени;

2) это совокупность операторов программы, имеющая граничные элементы и идентификатор.

Module – 1) is a set of commands, which can be accessed by name;

2) it is a collection of program statements that has boundary elements and an identifier.

Microsoft Visual Studio – dasturiymahsulotlarniishlabchiqishgamo‘ljallanganintegrallashganmuhitbo‘lib, Microsoft kompaniyasiningmahsulotihisoblanadi. Microsoft .NET Framework platformasigamo‘ljallangandasturlashtillariniqo‘llabquvvatlaydi.

Microsoft Visual Studio – интегрированнаясредаразработкипрограммныхпродуктовкомпании Microsoft, которая, втомчисле, поддерживаетязыкипрограм-мированиядляплатформы Microsoft .NET Framework.

Microsoft Visual Studio is an integrated development environment for Microsoft software products, which, among other things, supports programming languages for the Microsoft .NET Framework.

Microsoft .NET Framework – CLR virtual mashinasitomonidanfoydalaniladiganplatformalargabog‘liqbo‘lmaganmustaqildasturlarniishlabchiqishgamo‘ljallangan Microsoft kompaniyasiningengso‘nggidasturiyta’minottexnologiyasi.

Microsoft .NET Framework – одна из последних программных технологий компании Microsoft, созданная для разработки платформа- независимых приложений, исполняемых виртуальной машиной CLR.

Microsoft .NET Framework is one of the latest Microsoft software technologies designed to develop platform-independent applications run by the CLR virtual machine.

Ma’lumotlarbazasiniboshqarishtizimi (MBBT) – ma’lumotlarbazasiniboshqarishbo‘yichabarchajarayonlarnio‘zichigaolgandasturiyta’minot (axborottizimi). Bundayjarayonlargama’lumotlarbazasinisaqlash, SQL

yordamidaqaytaishlash, zaxiranusxalariniko‘paytirish, zaxiranusxalariniqaytatiklashvaboshqalarnio‘zichigaolganaxborottizimi.

Система Управления Базами Данных (СУБД) – программное обеспечение (информационная система), осуществляющее весь спектр операций по управлению базами данных, к которым относятся сама организация хранения данных, обработка инструкций SQL, организация резервного копирования, восстановление резервных копий и т.п.

Database Management System (DBMS) is a software (information system) that performs the whole spectrum of database management operations, which include the data storage organization itself, processing of SQL instructions, organizing backups, restoring backup copies, and so on.

O

Obeyktgayo‘nalatirilganma’lumotlaribazasi – obyektgayo‘naltirilganma’lumotlarmodeligaasoslanganma’lumotlarbazasi. Obyektgayo‘naltirilganmodelningasosiyelementlari: sinflar, obyektlar, interfeyslar, atributlar (xususiyatlar), usullarvaboshqalar.

Объектно-ориентированная база данных – база данных, основанная на объектно-ориентированной модели данных. Главными элементами объектно-ориентированной модели являются классы, объекты, интерфейсы, атрибуты (свойства), методы и т.п.

Object-oriented database is a database based on an object-oriented data model. The main elements of the object-oriented model are classes, objects, interfaces, attributes (properties), methods, and so on.

Obeyktgayo‘nalatirilgandasturlash– dasturniobyektlarto‘plamisifatidako‘rsatishgamo‘ljallangandasturlashmetodologiya sibo‘lib, ularning har birima’lumbirsinfningnamunasisifatidako‘riladi. Sinflarmerosningierarxiyasinitashkilqiladi.

Объектно-ориентированное программирование (ООП) – методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Object-oriented programming (OOP) - programming methodology based on the representation of the program in the form of a collection of objects, each of which is an instance of a particular class, and the classes form a hierarchy of inheritance.

P

Pascal – dasturlashnio‘rgatishdakengqo‘llaniladiganyuqoridarajalidasturlashtili.

Pascal – один из наиболее известных языков программирования высокого уровня, который широко используется в целях обучения программированию.

Pascal – one of the most famous high-level programming languages that is widely used for teaching programming.

R

Relayatsionma’lumotlarbazasi – ma’lumotlarningrelatsionmodeligaasoslanganma’lumotlarbazasi.

Реляционная база данных – база данных, основанная на реляционной модели данных.

Relational database– a database based on a relational data model.

S

SQL – ma’lumotlarbazasigatuzilgantizimliso‘rovlarnitavsiflashtilibo‘lib, natijasifatidatuzilganma’lumotlarto‘plamiyokima’lumotlartarkibidagio‘zgarishlarni amalgaoshirishuchunmo‘ljallangan. Undantashqari SQL so‘rovlariyordamidama’lumotlarbazasituzilishinio‘zgartirish, kirishparametrlariniturlibo‘limlargasozlashvatranzaktsiyalarniboshqarishimkoninib eradi.

SQL – язык описания структурированных запросов к базам данных, результатом выполнения которых может быть или структурированный набор информации или изменения в составе данных. Также, инструкции SQL позволяют изменять саму структуру базы данных, настраивать параметры доступа к различным ее разделам и управлять транзакциями.

SQL – a language for describing structured queries to databases, the result of which can be either a structured set of information or changes in the composition of the data. Also, SQL statements allow you to change the structure of a database, configure access parameters to its various sections and manage transactions.

C++ – Bjorn Stroustruptomonidanishlabchiqilganobyektgayo‘naltirilgandasturlashtili.

C++ – объектно-ориентированный язык программиро-вания, разработанный Бьерном Страуструпом.

C ++ – is an object-oriented programming language developed by Bjorn Stroustrup.

T

Tizimdasturchisi – tizimlidasturlarniishlabchiqish, ishlatishvaulardanfoydalanishgaxizmatko‘rsatishbilanshug‘ullanadiganmutaxassis.

Системный программист – специалист, занимается разработкой, эксплуатацией и сопровождением системного программного обеспечения.

The system programmer is a specialist, engaged in the development, operation and maintenance of system software.

V

Vizualdasturlash– ko‘rgazmalivositalaryordamidadasturiyta’minotlarniishlabchiqishgamo‘ljallangandasturlash.

Визуальное программирование – программирование, предусматривающее создание приложений с помощью наглядных средств.

Visual programming – programming, providing for the creation of applications using visual aids.

Vebdasturlash – Internet tarmog‘iuchunvebilovalarniyaratishgamo‘ljallangandasturlashyo‘nalishi. Foydalanuvchivebilovalargainternetbrauzerorqalio‘zaroboglanadi.

Веб–программирование – направление в программировании, ориентированное на разработку приложений для сети интернет (веб–приложений). Пользовательвзаимодействуетсвеб-приложениемчерезинтернетбраузер.

Web programming – a direction in programming, focused on the development of applications for the Internet (web applications). The user interacts with the web application via an Internet browser.

Y

Yuqori darajali dasturlashtili– insonqabulqilishiuchunqulaytushunchavatuzulishgaegabo‘lgandasturlashtili.

Язык высокого уровня – язык программирования, понятия и структура которого удобны для восприятия человеком.

High–level language is a programming language, the concepts and structure of which are convenient for human perception.

СПИСОК ЛИТЕРАТУРЫ

1. Kadirov M.M. «Axborot texnologiyalari» o‘quv qo‘llanma 1-qism, «Fan va texnologiya».-Toshkent 2018, -316 b.

2. Mark Lewin. Go Web Development Succinctly. Syncfusion Inc. USA 2017. p 240.

3. Kenneth C. Laudon, Jane. P. Laudon. Management Information Systems: Managing the Digital Firm, 13th Edition, Pearson Education, USA 2014. p 621.

4. Axel Rauschmayer. Speaking JavaScript. O‘Reilly Media. USA 2014. p 419.

5. Alex Allain. Jumping into C++. USA, 2014. p 340.

6. Стенли Липпман. Язык программирование C++. Базовой курс. Вильямс – М.: 2014.

7. СидхармаРао. Освой самостоятельно C++ за 21 день. Вильямс – М.: 2013.

8. Nazirov Sh.A., Qobulov R.V., Bobojonov M.R., Raxmanov Q.S. C va C++ tili. Voris-nashriyot. Toshkent 2013. 488 b.

9. Kjell Backman. Structured Programming with C++. BookBoon. USA. 2012. P 246.

10. Matt Doyle. Beginning PHP 5.3. Wiley Publishing, Inc. USA 2010. P 775.

11. G‘ulomov S.S., Begalov B.A..Informatika va axborottexnologiyalari. Darslik.Toshkent: Fan, 2010. 686b.

13. Nazirov Sh.A., Qobulov R.V. Obyektga mo‘ljallangan dasturlash.O‘quv qo‘llanma. –Toshkent: Aloqachi, 2007. –337b.

14. Aripov M.M., Yakubov A.X., Sagatov M.V., Irmuhamedova R.M. va boshqalar. Informatika. Axborot texnologiyalari. O‘quv qo‘llanma. 1,2-qism. – Toshkent: TDTU, 2005.

16. Xaldjigitov A.A., Madraximov Sh.F., Adamboev U.E.. Informatika va programmalash. O‘quv qo‘llanma. T.: O‘zMU, 2005.- 145 b.

СОДЕРЖАНИЕ

Глава4. ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ ПРОЦЕССОВ В СРЕДЕ BORLAND C++ BUILDER 6

Глава 5. ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЯЮЩИХСЯ ПРОЦЕССОВ В СИСТЕМЕ ПРОГРАММИРОВАНИЯ С++ BUILDER 6

РАЗДЕЛ 6. ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ ПРОЦЕССОВ В СИСТЕМЕ ПРОГРАММИРОВАНИЯ С++ BUILDER 6

Глава 7. РАБОТА С МАССИВАМИ В СИСТЕМЕ ПРОГРАММИРОВАНИЯ C++ BUILDER 6

Глава 8. СТРУКТУРНЫЙ ТИП НА ЯЗЫКЕ C++

Глава 9. ФУНКЦИИ И ПРОЦЕДУРЫ C++. ПРИМЕНЕНИЕ МЕТОДОВ СТРУКТУРНОГО ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C++

CONTENT

Chapter 4. BORLAND C ++ BUILDER 6 PROGRAMMING LINE PROGRAMMING SYSTEM

Chapter 5. C ++ BUILDER 6 PROGRAMMING PROCEDURES IN THE PROGRAMMING SYSTEM

SECTION 6. C ++ BUILDER 6 PROGRAMMING PROCEDURES IN THE PROGRAMMING SYSTEM

Chapter 7. BORLAND C ++ BUILDER 6 PROGRAMMING WITH MASSIVE

Chapter 8. C ++ PROGRAMMING LANGUAGE STRUCTURE

Chapter 9. FUNCTIONS AND PROCEDURES IN C ++. Using C ++ FOR STRUCTURAL PROGRAMMING METHODS

Chapter 10. Useful and Reference Data in C ++ Programming

Chapter 11. CLASSES AND OBJECTS IN BORLAND C ++ BUILDING PROGRAMS

Chapter 12. GRAPHIC OPTIONS IN BORLAND C ++ BUILDER 6

Chapter 13. BORLAND C ++ BUILDER 6 INTEGRATED INFORMATION DATABASE

MUNDARIJA

4. BOB. BORLAND C++ BUILDER 6DASTURLASH TIZIMIDA CHIZIQLI JARAYONLARNI DASTURLASH

5 BOB. C++ BUILDER 6 DASTURLASH TIZIMIDA TARMOQLANUVCHI JARAYONLARNI DASTURLASH

6 BOB. C++ BUILDER 6 DASTURLASH TIZIMIDA TAKRORLANUVCHI JARAYONLARNI DASTURLASH

7 BOB. BORLAND C++ BUILDER 6 DASTURLASH TIZIMIDA MASSIVLAR BILAN ISHLASH

8 BOB. C++ DASTURLASH TILINING STRUKTURA TOIFASI

9 BOB. C++ DA FUNKSIYA VA PROTSEDURALAR. STRUKTURALI DASTURLASH USULLARINI C++ DA QO‘LLASH

10 BOB. C++ DASTURLASH TIZIMIDA MA’LUMOTLARNING FAYLLI VA MUROJAAT TOIFASI

11 BOB. BORLAND C++BUILDER DASTURLASH TIZIMIDA SINFLAR VA OB’EKTLAR

12 BOB. BORLAND C++ BUILDER 6 NING GRAFIK IMKONIYATLARI

13 BOB. BORLAND C++ BUILDER 6 INTEGRALLASHGAN SOHASIDA MA’LUMOTLAR BAZASINI QAYTA ISHLASH

КАРИМОВА НОЗИМАХОН ОЙБЕКОВНА

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ТЕХНИЧЕСКИХ СИСТЕМАХ

Toshkent– «Fanvatexnologiya» – 2018

Muharrir: Sh.Aliyeva
Tex.muharrir: F.Tishaboyev
Musavvir: A.Moydinov
Musahhih: Sh.Mirqosimova
Kompyuterdasahifalovchi: N.Raxmatullayeva

E-mail: tipografiyacnt@mail.ru Tel:245-57-63, 245-61-61.

Nashr.lits. AI№149, 14.08.09. Bosishga ruxsat etildi:6.09.2018.

Bichimi 60x84 ⅛. «Times Uz» garniturasi.

Ofsetbosmausilidabosildi. Shartlibosmatabog‘i 18,25.

Nashriyotbosmatabog‘i 18,75. Buyurtma№405.

«Fan vatexnologiyalarMarkazi...maxonasi» da chop etildi.

100066, Toshkent sh., Olmazorko‘chasi, 171 uy