МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«ДАГЕСТАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

Технологии и методы программирования

Кафедра **Информатики и информационных технологий**

Факультет **Информатики и информационных технологий**

Образовательная программа

10.03.01 «Информационная безопасность»

Профиль подготовки:
Безопасность компьютерных систем

Уровень высшего образования
Бакалавриат

Форма обучения
очная

Статус дисциплины: **базовая**

Махачкала, 2018

Рабочая программа дисциплины **«Технологии и методы программирования»** составлена в 2018 году в соответствии с требованиями ФГОС ВО по направлению подготовки **10.03.01 «Информационная безопасность»**, уровень Бакалавриат, утвержденного приказом Министерства образования и науки от 1 декабря 2016 г, № 1515, вступил в силу 20 декабря 2016 г.

Разработчик: Абдуллаев Габид Шаванович, к.э.н., доцент кафедры информатики и информационных технологий

Рабочая программа дисциплины одобрена:

На заседании кафедры ИиИТ от «2» 07 2018г., протокол № 12

Зав. Кафедрой _____ Ахмедов С.А.

(подпись)

На заседании Методической комиссии факультета ИиИТ

от «3» июля 2018г., протокол № 10.

Председатель _____ Камилов К.Б.

(подпись)

Рабочая программа дисциплины согласована с учебно – методическим управлением «04» 07 2018г. _____

(подпись)

Аннотация рабочей программы дисциплины

Дисциплина «Технологии и методы программирования» входит в базовую часть образовательной программы бакалавриата по направлению 10.03.01 «Информационная безопасность»

Дисциплина реализуется на факультете Информатики и ИТ кафедрой Информатики и ИТ.

Содержание дисциплины охватывает круг вопросов, связанных с изучением современных технологий и методов программирования, основных принципов объектно-ориентированного программирования, механизмов доступа к базам данных и работы с ними, приобретением практических навыков использования современных инструментальных средств для разработки, отладки и тестирования создаваемых прикладных программ.

Дисциплина нацелена на формирование следующих компетенций выпускника: общекультурных – ОК-8, общепрофессиональных – ОПК-4, профессиональных – ПК-1.

Преподавание дисциплины предусматривает проведение следующих видов учебных занятий: *лекции, практические занятия, лабораторные занятия, самостоятельная работа.*

Рабочая программа дисциплины предусматривает проведение следующих видов контроля успеваемости в форме *контрольной работы, сетевого компьютерного тестирования, коллоквиума* и промежуточный контроль в форме экзамена.

Объем дисциплины 6 зачетных единиц, в том числе в академических часах по видам учебных занятий

Семестр	Учебные занятия							Форма промежуточной аттестации (зачет, дифференцированный зачет, экзамен
	в том числе							
	Контактная работа обучающихся с преподавателем						СРС, в том числе экзамен	
	Всего	из них						
		Лекции	Лабораторные занятия	Практические занятия	КСР	консультации		
4	72	20	18		2	2	30	
5	144	18	16	34	2	2	72	экзамен

1. **Цели освоения дисциплины*:***

Подготовка к самостоятельной профессиональной работе, ознакомление с методами и технологиями программирования, умение ориентироваться во всем многообразии технологий программирования, умение применять практические навыки использования инструментальных и прикладных технологий в различных отраслях техники, экономики, управления и бизнеса.

2. **Место дисциплины в структуре образовательной программы:**

Дисциплина "Технологии и методы программирования" является базовой дисциплиной образовательной программы 10.03.01 «Информационная безопасность», профиль подготовки «Безопасность компьютерных систем», изучается в 4-5 семестрах. Объем дисциплины: 6 ЗЕ /216 часов, в том числе 106 часов - контактная работа с преподавателем, 110 часов - самостоятельная работа.

Программа дисциплины разработана в соответствии с федеральным государственным стандартом высшего образования по направлению подготовки бакалавриата 10.03.01 «Информационная безопасность», утвержденным приказом Минобрнауки России от 1 декабря 2016 г. № 1515, вступил в силу 20 декабря 2016 г.

Для изучения данной учебной дисциплины необходимы следующие знания, умения и навыки, формируемые предшествующими дисциплинами:

Из курса «Языки программирования»:

Знания: ядро языка программирования высокого уровня, его синтаксис и семантику; основы проектирования программ: типовые алгоритмы.

Умения: описывать разработанные программы посредством блок схем, тестировать и отлаживать разработанные программы; реализовывать на языке программирования высокого уровня типовые алгоритмы: табуляцию функций, формирование таблиц, работу с датчиком случайных чисел, ввод и вывод одномерных и двумерных массивов, поиск элементов в массиве, обработку массивов с выводом таблиц, сортировку, ввод и вывод текстов, сравнение фрагментов текста, изменение фрагмента текста по определенному правилу, запись информации в файл, чтение информации из файла, поиск и изменение информации в файле по заданному условию.

Владения: приемами работы в среде программирования (составление, отладка и тестирование программ; разработка и использование интерфейсных объектов) Перечень последующих учебных дисциплин, для которых необходимы знания, умения и владения, формируемые данной учебной дисциплиной: Интернет программирование;

Планируемые результаты обучения:

Дисциплина направлена на формирование компетенций ОПК-1, ПК-36, ПК-37 и планируемых результатов обучения, представленных в таблице.

Код компетенции из ФГОС ВО	**Наименование компетенции из ФГОС ВО**	**Планируемые результаты обучения**
ОК-8	способностью к самоорганизации и самообразованию	Знает: теоретические основы и источники информационных технологий Умеет: использовать информационные технологии для организации учебы, труда в повседневной жизни Владеет: техническими способами и методами получения новой информации
ОПК-4	способностью понимать значение информации в развитии современного общества, применять информационные технологии для поиска и обработки информации	Знает: различные способы, методы, принципы создания, преобразования, поиска, хранения и передачи информации Умеет: применять программные и аппаратные способы, методы, принципы для получения информации в новом качестве Владеет: навыками и применять имеющиеся информационные технологии на практике
ПК-1	способностью выполнять работы по установке, настройке и обслуживанию программных, программно-аппаратных (в том числе криптографических) и технических средств защиты информации	Знает: принципы организации информационных систем в соответствии с требованиями по защите информации. Умеет: анализировать и оценивать угрозы информационно безопасности объектов, использовать программные и аппаратные средства современного компьютера. Владеет: методами установки и настройки программно- аппаратных и технических средств защиты информации

4. Объем, структура и содержание дисциплины.

4.1. Объем дисциплины составляет 3 зачетных единиц, 108 академических часов.

4.2. Структура дисциплины.

№ п/п	Разделы и темы дисциплины	Семестр	Неделя семестра	Виды учебной работы, включая самостоятельную работу студентов и трудоемкость (в часах)				Самостоятельная работа	Формы текущего контроля успеваемости *(по неделям семестра)* Форма промежуточной аттестации *(по семестрам)*
				Лекции	Практические занятия	Лабораторные занятия	Контроль самост. раб		
	Модуль 1. *Понятия программной инженерии*								
1	Введение в технологию программирования	4	1	2				4	
2	Элементы программной инженерии	4	2	2		2		4	Выполнение и защита лаборат. работы 1
	Итого по модулю 1:			4		2		6	Тестирование по мод
	Модуль 2. *Визуальное моделирование на основе UML Методология (MSF)*								
3	Визуальное моделирование при анализе и проектирование. Основы UML	4	3-5	4		4		4	Выполнение и защита лаборат. работ 2-6
4	Методология MSF. Версии. Модель проектной группы	4	7	2		2		6	
5	Методология MSF. Управление рисками. Модель процессов	4	9	2		4	2	4	
	Итого по модулю 2:			8		10	2	14	Тестирование по мод
	Модуль 3. *Разработка и оценка качества программного обеспечения*								
6	Методология MSF. Выработка концепции. Планирование	4	11-13	4		2		4	Выполнение и защита лаборат. работ 7-9
7	Методология MSF. Фазы разработки и стабилизации	4	15-17	4		2		6	
	Итого по модулю 3:			6		4		10	Тестирование по мод
	Модуль 4. *Введение в объектно-ориентированное программирование на C#*								
8	ООП как подход к программированию	5	1	2	4			4	
9	Основные понятия ООП в C#: объекты, классы и методы	5	2-3	2	4	2		4	Выполнение и защита лаборат. работ 1-2
10	Понятие инкапсуляции, наследования, полиморфизма и его	5	4-5	2	2	2		4	

	применение в C#								
	Итого по модулю 4:			6	10	4		12	Тестирование по мод
	Модуль 5. *Платформа .NET Framework и ее применение для ООП*								
11	Основные понятия платформы .NET Framework	5	6-7	2	4			4	Выполнение и защита лаборат. работ 3-6
12	Платформа ASP.NET. Система типизации в .NET	5	8-9	2	2	2		4	
13	Событийно управляемое программирование в .NET	5	10-11	2	4	2		4	
14	Компонентное программирование в .NET	5	12-13	2	4	2		4	
	Итого по модулю 5:			8	16	6		16	Тестирование по мод
	Модуль 6. *Технологии Веб программирования и ADO.NET*								
15	Введение в Web приложения. Создание приложений Web Forms	5	14-15	2	4	2		6	Выполнение и защита лаборат. работ 7-9
16	Работа с базами данных в .NET. Технология ADO.NET. Автономный и подключенный уровни	5	16-17	2	4	2	2	4	
	Итого по модулю 6:			4	8	6	2	10	Тестирование по мод
	Экзамен							36	Письменная работа
	ИТОГО:			38	34	18	4	106	

4.3. Содержание дисциплины, структурированное по темам (разделам).

4.3.1. Содержание лекционных занятий по дисциплине

Модуль 1. Понятия программной инженерии

Тема 1. Введение в технологию программирования

Основные понятия. Программирование. IT-проекты. Программы и программное обеспечение (программные продукты). Бизнес и IT-проекты. Рынок ПО в России и в мире.

Причины неудачи IT-проектов.

Технологии программирования: структурное программирование, модульное программирование, объектно-ориентированное программирование, компонентное программирование

Тема 2. Элементы программной инженерии

Программная инженерия, основные понятия. Инженеры и программные инженеры. Программная инженерия как инженерная дисциплина. Область действия программной инженерии. Цели программных инженеров. Программные инженеры и научная среда

Процесс создания программного обеспечения. Понятие процесса. Модели процесса

Модуль 2. Визуальное моделирование на основе UML. Методология MSF

Тема 3. Визуальное моделирование при анализе и проектирование. Основы Unified modeling language (UML)

Анализ и проектирование. Некоторые частные вопросы. Обзор принципов объектного подхода. Повторное использование

Визуальное моделирование. История языка UML. Вместо введения. Идея визуального моделирования. История языка UML

Структура языка UML. Модели UML. Диаграммы UML. Понятия UML

Учебный пример. Постановка задачи. Система бронирования билетов для авиакомпании

Визуальное описание функциональной модели средствами UML. Актеры и варианты использования в UML

Структура системы и ее описание средствами UML. Классы. Шаблоны классов. Объекты. Интерфейсы. Пакеты. Подсистемы. Компоненты. Комментарии. Отношения между элементами модели

Тема 4. Методология Microsoft solutions framework. Версии. Модель проектной группы

Введение в методологию MSF. Основные концепции методологии MSF 4.0. Направления в MSF 4.0. Основные положения MSF for Agile Software Development. Инструментальная поддержка MSF 4.0

Формирование команды. Модель проектной группы MSF for Agile Software Development. Основные принципы построения команды. Ролевые группы и роли, зоны ответственности ролевых групп. Рекомендации по возможному объединению ролей. Учебный пример. Формирование команды

Тема 5. Методология MSF. Управление рисками. Модель процессов

Вспоминая предыдущую лекцию

Управление рисками в MSF for Agile Software Development. Основные сведения о рисках. Планирование управления рисками. Процесс управления рисками. Управление рисками как составная часть жизненного цикла проекта. Учебный пример. Выделение рисков

Модель процессов MSF for Agile Software Development. Принципы модели процессов. Управление компромиссами. Схема процесса разработки

Модуль 3. Разработка и оценка качества программного обеспечения

Тема 6. Методология MSF. Выработка концепции. Планирование

Вспоминая предыдущую лекцию

Старт проекта. Фаза выработки концепции

Планирование проекта. Фаза планирования

Тема 7. Методология MSF. Фазы разработки и стабилизации

Вспоминая предыдущую лекцию

Разработка решения. Фаза разработки

Стабилизация решения. Фаза стабилизации

Внедрение решения. Фаза внедрения

Модуль 4. Введение в объектно-ориентированное программирование на C#

Тема 8. ООП как подход к программированию

Введение в современные подходы к программированию. Современные подходы к программированию. Особенности декларативного подхода. Особенности процедурного подхода. Особенности функционального подхода. Основные понятия ООП. Абстракция, инкапсуляция, наследование и полиморфизм. Преимущества и недостатки ООП

Тема 9. Основные понятия ООП в C#: объекты, классы и методы

Интуитивные определения объектов, классов и методов. Соотношение понятий объекта и класса. Концептуальная модель для классов и объектов.

Классы, объекты, свойства и методы в языке C#. Классы и структуры в языке C#. Конструкторы и деструкторы классов в языке C#. Преимущества и недостатки объектных теорий.

Тема 10. Понятие инкапсуляции, наследования, полиморфизма и его применение в C#

Инкапсуляция в ООП. Примеры инкапсуляции в языке C#. Виды областей видимости объектов. Рекомендации по разграничению областей видимости. Преимущества инкапсуляции

Наследование в ООП. Отношение частичного порядка. Базовые и производные классы в C#. Иерархия классов в .NET. Отображение классов .NET в типы языка C#

Понятие полиморфизма в ООП. Примеры полиморфизма в C#. Виды полиморфизма. Абстрактные типы данных. Методы вызова процедур. Преимущества программирования с полиморфизмом. Абстрактные структуры данных, методы, свойства и индексаторы в C#. Интерфейсы в языке C# и их связь с абстрактными классами. Реализация интерфейсов на основе классов и структур

Модуль 5. Платформа .NET Framework и ее применение для ООП

Тема 11. Основные понятия платформы .NET Framework

.NET как концепция. .NET как вычислительная модель. .NET как технологическая платформа. .NET как инструментальное средство. Common Language Runtime и .NET Framework. Система типов Common Type System в .NET. Веб-сервисы в .NET.

Компонентное программирование в .NET. Сравнение компонентного программирования с ООП. Преимущества и недостатки .NET

Тема 12. Платформа ASP.NET. Система типизации в .NET

Неформальное и формальное определения типов. Преимущества теорий с типами. Классификация систем типизации. Система типов (Common Type System, CTS) в .NET. Базисные типы языков C# и их отображение в CTS. Пространства имен. Преобразования типов в .NET.

Структура Web-приложений. Структура .NET Framework. Компоненты Web-форм. Языки программирования в .NET Framework

Тема 13. Событийно управляемое программирование в .NET

Понятие события в программировании. Методы моделирования событий. Фреймы и функции как модели событий.

Делегаты в языке C#. Конструкторы для делегатов в языке C#. Делегаты с множественным вызовом в языке C#

События как особый вид делегатов. Исключения и их обработка в языке C#.

Тема 14. Компонентное программирование в .NET

Компонентный подход к программированию как расширение ООП. Обзор архитектурного решения .NET. Понятия сборки и манифеста в .NET

Пространства имен в .NET. Гетерогенное компонентное программирование в .NET.

Модуль 6. Технологии Веб программирования и ADO.NET

Тема 15. Введение в Web приложения. Создание приложений Web Forms

Типы Интернет-приложений. Принцип работы Web-приложений. Возможности и преимущества ASP. NET.

Создание проекта Web-приложения. Управление проектом при помощи IIS. Обработка событий eb-приложения. Обработка данных. Управление процессами.

Тема 16. Работа с базами данных в .NET. Технология ADO.NET. Автономный и подключенный уровни

Обзор ADO.NET. Архитектура ADO.NET. Создание базы данных. Генератор поставщиков данных. Подключение к базе данных. Команды

Вставка, удаление, обновление записей в базе данных. Хранимые процедуры. Транзакции баз данных.

Понятие автономного уровня ADO.NET. Основные свойства и методы класса DataSet. Работа с объектами DataColumn. Работа с объектами DataRown. Работа с объектами DataTable

4.3.2. Содержание лабораторно-практических занятий по дисциплине.

Темы практических занятий

Тема 1**. C++Builder 2010 и современные информационные технологии**

План занятия

1. Объектно-ориентированное программирование C++ Builder 2010
2. Основы визуального программирования интерфейса
3. Взаимодействие приложений в информационных системах
4. Распределенные многозвенные приложения
5. Переносимость данных и программ
6. Сетевые службы

Тема 2. О**бъектно-ориентированное проектирование в IDE C++ Builder 2010**

План занятия

1. Общие сведения о программах на C++ Builder 2010
2. Структура головного файла проекта
3. Структура файлов модулей форм
4. Области видимости и доступ к объектам модуля
5. Указатели на объекты
6. Общий вид окна IDE
7. Главное меню. Быстрые кнопки
8. Палитра компонентов
9. Окно формы
10. Окно Редактора Кода
11. Инспектор Объектов
12. Перетаскивание и встраивание окон в IDE C++Builder 2010

Тема 3. **Компоненты ввода и отображения информации**

План занятия

1. Компоненты Label, StaticText, Panel
2. Окна редактирования Edit, LabeledEdit и MaskEdit
3. Многострочные окна редактирования Memo и RichEdit
4. Компоненты выбора из списков - ListBox, CheckListBox, ValueListEditor, ComboBox,
5. Таблица строк - компонент StringGrid
6. Компоненты ввода и отображения целых чисел - Up Down и CSpinEdit
7. Компоненты ввода и отображения дат и времени - DateTimePicker, MonthCalendar, CCalendar
8. Компонент генерации страницы Excel - FIBook
9. Компонент отображения иерархических данных - ListView

Тема 4. **Кнопки, индикаторы, управляющие элементы**

План занятия

1. Общая характеристика управляющих элементов
1. Управляющие кнопки Button и BitBtn
2. Кнопка с фиксацией SpeedButton
3. Группы радиокнопок - RadioGroup, RadioButton и GroupBox
4. Индикаторы CheckBox и CheckListBox
5. Ползунки и полосы прокрутки - компоненты TrackBar и ScrollBar
6. Таймер - компонент Timer
7. Заголовки - компоненты HeaderControl и Header

Тема 5. **Системные диалоги**

План занятия

1. Общая характеристика компонентов – диалогов
2. Диалоги открытия и сохранения файлов - компоненты OpenDialog и SaveDialog
3. Диалог выбора шрифта - компонент FontDialog
4. Диалоги поиска и замены текста -компоненты FindDialog и ReplaceDialog
5. Диалоги выбора цвета -компоненты ColorDialog
6. Диалоги печати и установки принтера -компоненты PrintDialog и PrinterSetupDialog

Тема 6. **Проектирование графических, мультимедиа и анимационных приложений**

План занятия

1. 1 Построение графических изображений - компонент Image
2. Канва - холст для рисования
3. Режимы рисования. События OnPaint
4. 1 Процедуры воспроизведения звуков Windows
5. 2 Начала анимации - создание мультипликаций
6. 3 Универсальный проигрыватель MediaPlayer
7. 4 Воспроизведение немых видео клипов - компонент Animate

Тема 7. **Архитектура приложений для локальных баз данных в C++ Builder 2010**

План занятия

1. Модели баз данных
2. Организация связи с базами данных в C++Builder 2010
3. Обзор компонентов, используемых для связи с базами данных
4. Наборы данных Table.Основные свойства и события.
5. Компоненты визуализации и управления данными
6. Проектирование приложений с несколькими связанными таблицами
7. Состояние набора данных, пересылка записи в базу данных

8. Методы доступа к полям, навигации и поиска записей
9. Методы установки диапазона допустимых значений
10. Методы создания и модификации таблиц

Тема 8. **Основы языка SQL, создание приложений для работы с базами данных в сети**

План занятия

1. Оператор выбора Select
2. Операции с записями
3. Операции с таблицами
4. Операции с индексами
5. Основные свойства компонента Query
6. Основные методы и события компонента Query
7. Работа с базами данных в сети
8. InterBase - работа на платформе клиент/сервер
9. Доступ к базам данных через ADO
10. Обзор компонентов наборов данных
11. Доступ к InterBase через InterBase Express
12. Доступ к базам данных с помощью компонентов dbExpress
13. Технология MIDAS

Тема 9. **Проектирование приложений для Интернет**

План занятия

1. Создание собственного браузера
2. Динамические страницы Web - приложения CGI
3. Сервер Web C++ Builder 2010
4. Использование форм и таблиц в HTML
5. Использование шаблонов HTML
6. Использование активных форм
7. Обзор дополнительных возможностей работы с Интернет

Темы лабораторных занятий

Лабораторная работа № 1

Общие замечания

Процесс создания программы в C++Builder состоит из двух шагов: сначала нужно создать форму программы (диалоговое окно), а затем функции обработки *событий.* Форма *приложения* Windows создается путем добавления в нее *компонентов* и последующей их настройки.

В форме практически любого приложения есть компоненты, которые обеспечивают интерфейс между программой и пользователем. Такие компоненты называют базовыми. К базовым компонентам относятся:

- Label - поле вывода текста; Edit - поле редактирования текста;
- Button - командная кнопка; CheckBox - независимая кнопка выбора;
- RadioButton - зависимая кнопка выбора; ListBox - список выбора;
- ComboBox - комбинированный список выбора.

Вид компонента, его размер и поведение определяют значения *свойств* (характеристик) компонента.

Основную работу в программе выполняют функции обработки *событий*.

Исходную информацию программа может получить из полей редактирования (компонент Edit), списка выбора (компонент ListBox) или комбинированного списка (компонент ComboBox). Для ввода значений логического типа можно использовать Компоненты CheckBox и RadoiButton.

- Результат программа может вывести в поле вывода текста (компонент Label) или в окно сообщения (функции ShowMessage, MessageDlg).

- Для преобразования текста, например, находящегося в поле редактирования, в целое число нужно использовать функцию StrToint, а в дробное - функцию StrToFloat. Для преобразования целого, например, значения переменной, в строку нужно использовать функцию IntTostr, а для преобразования дробного - функцию FloatToStr или FloatToStrF.

Первые простые приложения в IDE C++ Builder 2010

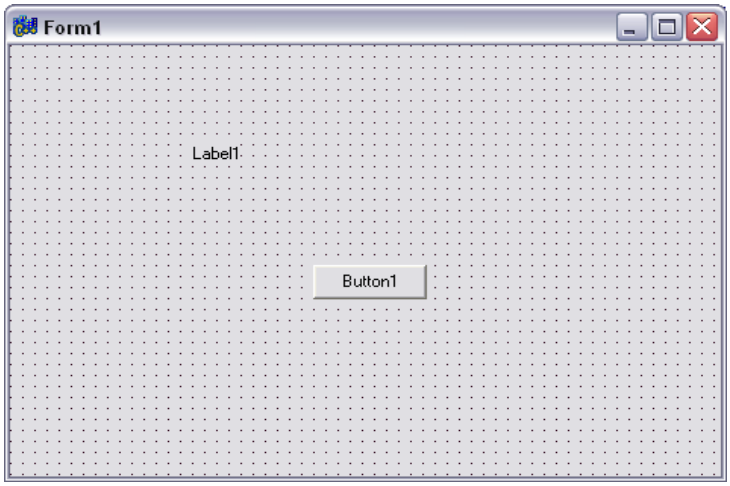
Создать приложение, в котором при щелчке пользователя на кнопке появится надпись на форме. Выполните для этого последовательно следующие шаги.

1. Запустите C++Builder.

2. Поместите на пустую форму кнопку **Button** со страницы **Standard** палитры компонентов, Для этого нужно выделить пиктограмму кнопки и затем щелкнуть курсором мыши в нужном месте формы. На форме появится кнопка, которой C++Builder присвоит имя по умолчанию - Button1.

3. Перенесите на форму со страницы **Standard** палитры компонентов метку **Label**. В этой метке в процессе выполнения приложения будет появляться текст при нажатии пользователем кнопки. C++Builder присвоит метке имя **Label1**.

4. Разместите компоненты на форме примерно так, как показано на рисунке.



Выделите на форме компонент **Button1**. В Инспекторе Объектов измените ее свойство **Caption**, которое по умолчанию равно **Button1**, на «Пуск».

5. Укажите метке **Label1**, что надписи на ней надо делать жирным шрифтом. Для этого выделите метку, в окне Инспектора Объектов раскройте свойство **Font** (шрифт), затем раскройте подсвойство **Style** (стиль) и установите в true свойство **fsBold** (жирный).

6. Сотрите текст в свойстве **Caption** метки **Label1**, чтобы он не высвечивался, пока пользователь не нажмет кнопку приложения.

Теперь осталось только написать оператор, который заносил бы в свойство **Caption** метки **Label1** нужный текст в нужный момент.

7. Выделите кнопку **Button1** на форме, перейдите в Инспектор Объектов, откройте в нем страницу событий **(Events),** найдите событие кнопки **OnClick** и сделайте двойной щелчок в окне справа от имени этого события,

или выполните двойной щелчок на кнопке Button1.

В обоих случаях откроется окно Редактора Кода и увидите там текст:

```
void _fastcall TForm1::Button1Click(TObject *Sender)
{

}
```

Заголовок этой функции складывается из имени класса формы (TForm1), имени компонента (Button1) и имени события без префикса On (Click).

8. Напишите в обработчике оператор задания надписи метки **Label1**.

Таким образом, обработчик события должен иметь вид:

```
void _fastcall TForm1: :Button1Click (TObject *Sender)
{
Label1->Caption = "Это мое первое приложение!";
}
```

Этот оператор означает следующее.

Символ "=" обозначает операцию присваивания.

Запись **Label1->Caption** означает, что присваиваете значение свойству **Caption** компонента **Label1**. Все указания свойств и методов производятся аналогичным образом: пишется имя компонента, затем ставятся символы операции стрелка **"->"**: символ минус "-" и символ больше ">", записанные без пробела. После этих символов пишется имя свойства или метода. В данном случае свойству **Caption** присваиваете строку текста «Это мое первое приложение!».

Закройте приложение, щелкнув на кнопке в его правом верхнем углу.

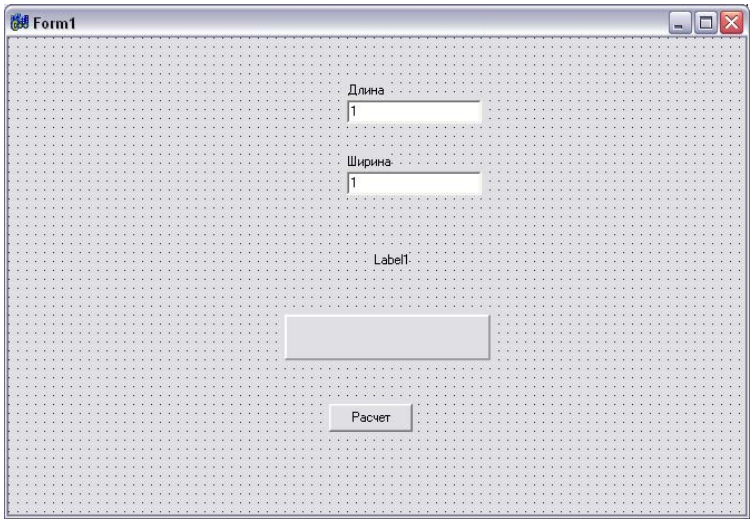
Немного более сложное приложение

Теперь создадим чуть-чуть сложное приложение, которое при нажатии кнопки перемножает два числа, введенных пользователем, и показывает результат умножения. Эти числа будем понимать, как длину и ширину сторон прямоугольника, и тогда результат - это площадь.

При построении этого приложения используйте компоненты - окна редактирования **LabeledEdit**. Результат нужно выводить не в метку **Label**, а в панель **Panel,** чтобы испытать новый компонент.

1. Откройте новое приложение.

2.Перенесите на форму со страницы библиотеки **Additional** два окна компонента **LabeledEdit**, а со страницы библиотеки **Standard** - одну панель **Panel**, одну кнопку **Button** и одну метку **Label** для надписи. Разместите все это примерно так, как показано на рисунке.



3. Измените надписи в метках компонентов **LabeledEdit** на «Ширина», «Высота». Для этого щелкните на символе "+" в свойстве **EditLabel** этих компонентов и измените надпись в свойстве **Caption** раскрывшихся списков свойств меток. Задайте для меток жирный шрифт.

4. Измените свойство **Caption** кнопки на **«Расчет».** Очистите свойство **Caption** у панели.

В свойстве **Text** компонентов **LabeledEdit** задайте 1" - начальное значение текста. Установите свойства **BevelInner = bvLowered и BevelOuter = bvRaised** панели, которые

определяют вид (утопленный - **bvLowered** или выпуклый **bvRaised**) основного поля и рамки панели.

Напишите обработчик щелчка кнопки. Операторы этого обработчика имеют вид:

Label1->Caption=”Площадь прямоугольника равна:”;

Panel1->Caption = LabeledEdit1->Text + " * " + LabeledEdit2->Text + " = " + FloatToStr(StrToFloat(LabeledEdit1->Text) * StrToFloat(LabeledEdit2->Text));

Во втором операторе свойству **Caption** компонента **Panel1** присваивается значение выражения, указанного в правой части оператора. Это выражение должно иметь тип строки текста. Начинается строка с текста, введенного пользователем в окно редактирования **Edit1** - этот текст хранится в свойстве **Text**. Затем прибавляете к этому тексту символы " * ". Знак "+" в выражениях для строк означает конкатенацию - сцепление двух строк символов. Затем аналогичным образом к строке добавляется текст второго окна редактирования и символы " = ". После вставляется результат перемножения двух целых чисел. Этот результат будет числом и, чтобы вставить его в текст, надо сначала преобразовать это число в строку. Эту операцию выполняет функция **FloatToStr(...),** которая преобразует в строку само произведение двух чисел. Но числа заданы пользователем в виде текстов - строк символов в окнах редактирования. Прежде, чем перемножать, эти строки надо перевести в числа. Эту операцию выполняют функции **StrToFloat(),** преобразующие символьное изображение числа в его значение типа действительного числа. Знак '*', указанный между двумя функциями **StrToFloat,** обозначает операцию умножения.

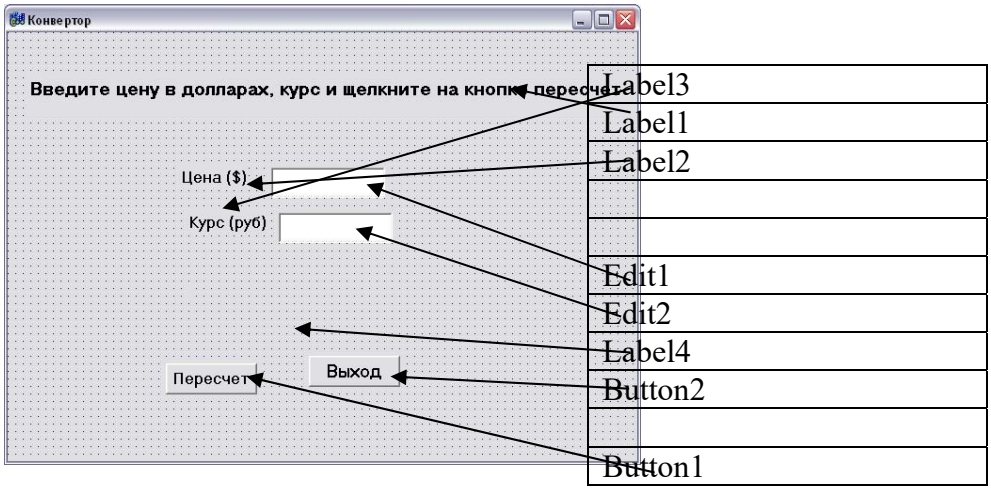
Лабораторная работа 2

Задание 1. Проект «Конвертор»

Составить проект **Конвертор,** который пересчитывает цену из долларов в рубли. Поместить на форму: 4 компонента Label и 2 компонента Edit для ввода и отображения числовых данных, 2 компонента Button.

Проект проектировать так, чтобы пользователь мог ввести в поля редактирования только правильные данные (число). Внешний вид формы приведен на рис.

Задайте свойства компонентов согласно рисунку. Затем напишите обработчики событий согласно приведенных ниже.



1.Обработчик, который проверяет, вводится в поле Text компонента Edit1 число или другой символ:

Создайте **обработчики, которые проверяют, вводится в поле Text компонент Edit число или другой символ.**

Например, для создания обработчика Edit1KeyPress для компонента Edit1нужно выделить компонент Edit1, перейти на страницу Events Инспектора Объектов (Object Inspector) и в

правой колонке таблицы строки OnKeyPress выполнить двойной щелчок мыши. Откроется редактор кода и в этом окне написать код обработчика события Edit1KeyPress. Пример кода приведен ниже.

```
void _fastcall TForm1::Edit1KeyPress(TObject *Sender, char &Key)
{
    // код запрещенного символа заменим нулем, в результате  символ в поле редактирования не появится
    // Key - код нажатой клавиш,  проверим, является ли символ допустимым
if ((Key >= '0') && (Key <= '9'))              //цифра
    return;
if (Key == DecimalSeparator)    // глобальная переменная DecimalSeparator содержит символ,
                                //используемый в качестве разделителя при записи дробных
                    чисел
{
if ((Edit1->Text).Pos(DecimalSeparator) != 0)
     Key = 0;                          // разделитель уже введен
     return;
}
if (Key == VK_BACK)                    // клавиша  <Backspace>
return;
if   (Key == VK_RETURN)                      // клавиша <Enter>
{
Edit2->SetFocus();
     return;
     }
                                       // остальные клавиший запрещены
 Key = 0;                              // не отображать символ
}
```

Обработчик для кнопки «Пересчет»

```
void _fastcall TForm1::Button1Click(TObject *Sender)
{
float usd;                                    // цена в долларах
float k;                                       // курс
float rub;                                    // цена в рублях
                                              // проверим, введены ли данные в поля Цена и Курс
if (((Edit1->Text).Length() ==0) || ((Edit2->Text).Length() == 0))
{
MessageDlg("Надо ввести цену и курс", mtInformation, TMsgDlgButtons() << mbOK, 0);
if ((Edit1->Text).Length() == 0)
    Edit1->SetFocus();                                // курсор в поле Цена
    Edit2->SetFocus();                                 // курсор в поле Курс
    return;
    };
usd = StrToFloat(Edit1->Text);                        // ввод исходных данных
k = StrToFloat(Edit2->Text);

rub = usd * k;                                        // вычисление
                                                      // вывод результата
Label4->Caption = FloatToStrF(usd,ffGeneral,7,2) +"$ = "+FloatToStrF(rub,ffGeneral,7,2) + "
руб." ;
}
```

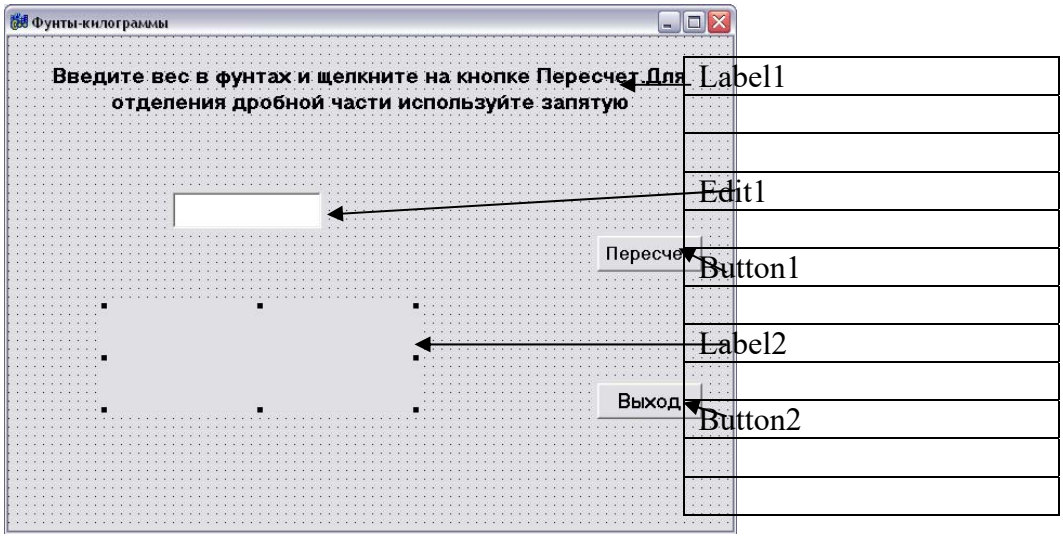
Обработчик для кнопки «Выход»

```
void _fastcall TForm1::Button2Click(TObject *Sender)
{
    Form1->Close();
    }
```

Задание 2. Проект «Фунты-килограммы»

Написать проект **Фунты-килограммы,** форма которой приведена на рис. 2, который позволяет пересчитать вес из фунтов в килограммы. Проект проектировать так, чтобы пользователь мог ввести в поля редактирования только правильные данные (числа) и кнопка «Пересчет» стала доступной только в том случае, если пользователь ввел исходные данные.

Компоненты: Label1 и Label2, Edit1, Button1 и Button2. Задайте свойства компонентов согласно рисунку. Затем Напишите обработчики событий согласно приведенных ниже.



Обработчик кода, который делает кнопку «Песчет» недоступным до ввода данных в поле редактирования Edit1

```
void _fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner)
{
    Button1->Enabled = False;
    }
```

Создайте обработчик, который проверяет, вводится в поле Text компонентов Edit1 и Edit1 число или другой символ (Обработчик EditKeyPress)

Обработчик, который проверяет, введены ли данные в поле Text компонента Edit1. Если да, то кнопка «Пересчет» становится доступной.

```
void _fastcall TForm1::Edit1Change(TObject *Sender)
{
if ( (Edit1->Text) .Length() == 0)
Button1->Enabled = False;
else
Button1->Enabled = True;
Label2->Caption = "";
}
```

Обработчик для кнопки «Пересчет»

```
void _fastcall TForm1::Button1Click(TObject *Sender)
{
double funt;
```

```
double kg;
funt =  StrToFloat(Edit1->Text);
kg = funt * 0.4995;
Label2->Caption = FloatToStrF(funt,ffGeneral,5,2) +" ф. - это " +FloatToStrF(kg,ffGeneral,5,2) +
" кг";
}
```

Задачи для самостоятельной работы

1. Скидка

Напишите программу вычисления стоимости покупки с учетом скидки. Скидка предоставляется, если сумма превышает 1000 руб., **а также** в выходные дни. Рекомендуемый вид формы приведен на рис. 1. В результате щелчка на кнопке **Скидка** в поле компонента Label должно появляться сообщение, информирующее о **предоставлении** скидки, и итоговая сумма с учетом скидки. Информацию о том, является ли день выходным, программа должна получать на основе анализа текущей даты

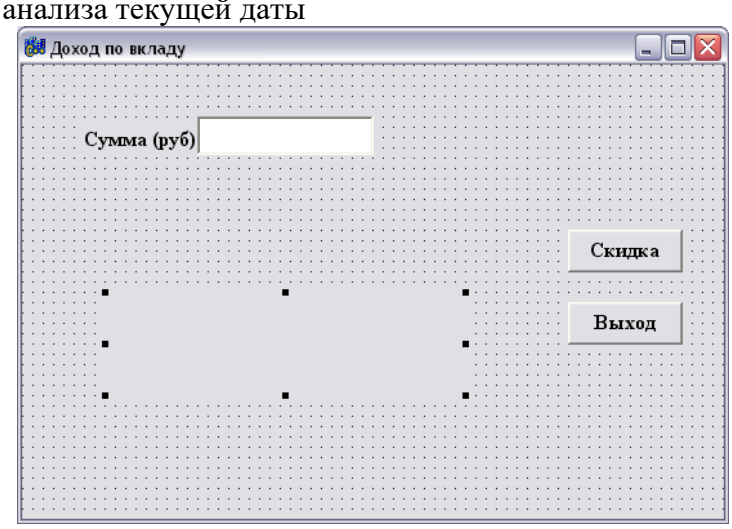


Рис. 1. Форма программы **Скидка**

2. Доход по вкладу

Напишите программу **вычисления** дохода по вкладу *в* банке. Доход вычисляется по формуле: Д = С * (СР / 360) * (СТ / 100), где: С - сумма вклада; СР - срок вклада (количество дней); СТ - процентная ставка (годовых). Рекомендуемый вид формы приведен на рис. 2.

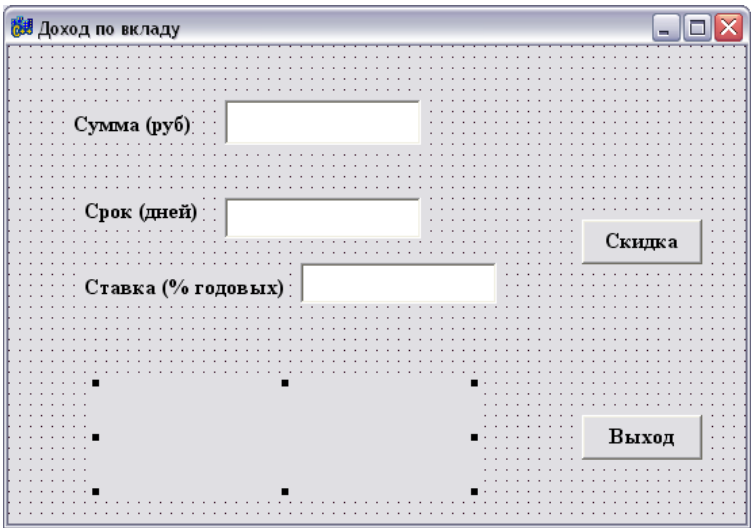
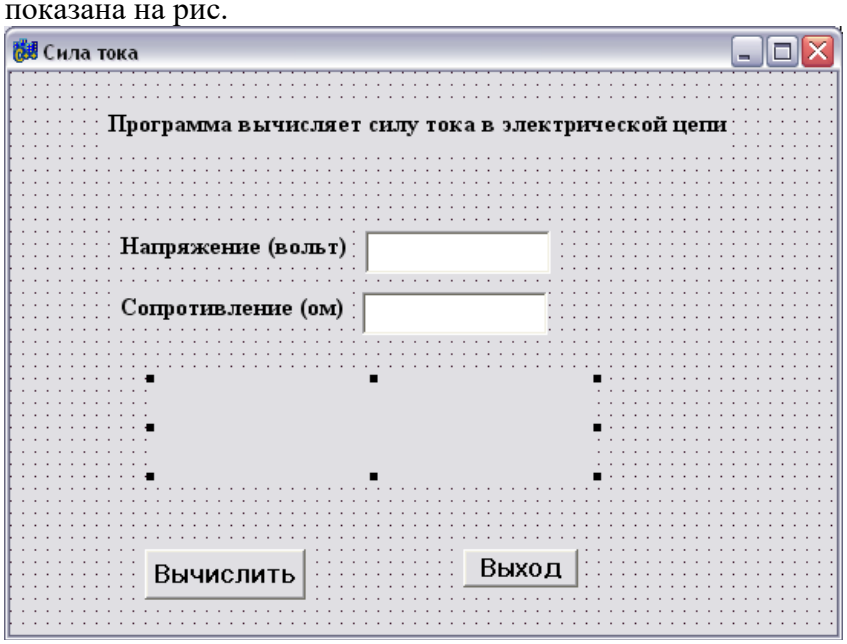


Рис. 2. Форма программы Доход по вкладу

Лабораторная работа 3

Задание 1. Проект «Сила тока»

Создайте проект **Сила тока,** в котором нужно реализовать использование компонентов TextBox и Label, а также обработку исключения "деление на ноль". Форма программы показана на рис.



Форма программы Сила тока.

```
// щелчок на кнопке Вычислить
void __fastcall TForm1::Button1Click(TObject *Sender)
{


}

void _fastcall TForm1::Edit1KeyPress(TObject *Sender,char &Key)
{


}

void _fastcall TForm1::Edit2KeyDown(TObject *Sender,WORD &Key,TShiftState Shift)
{
}
// щелчок на кнопке Завершить
void _fastcall TForm1::Button2Click (TObject *Sender)
{
Forml->Close(); // закрыть форму приложения
}
/* Процедура Edit1Change обрабатывает событие Change как поля Edit1, так и поля Edit2.
Сначала- надо создать процедуру обработки события Change для поля Edit1, затем - в
строке события Change компонента Edit2 щелкнуть на значке раскрывающегося списка и
выбрать Edit1Change. */
void _fastcall TForm1::Edit1Change(TObject *Sender)
{
Label4->Caption = "";
}
```

Задание 2. Проект Сопротивление

Создать проект **Сопротивление**, ее форма приведена на рисунке 4, который вычисляет сопротивление электрической цепи, состоящей из двух резисторов, которые могут быть соединены последовательно или параллельно. Демонстрирует использование компонента

RadioButton.

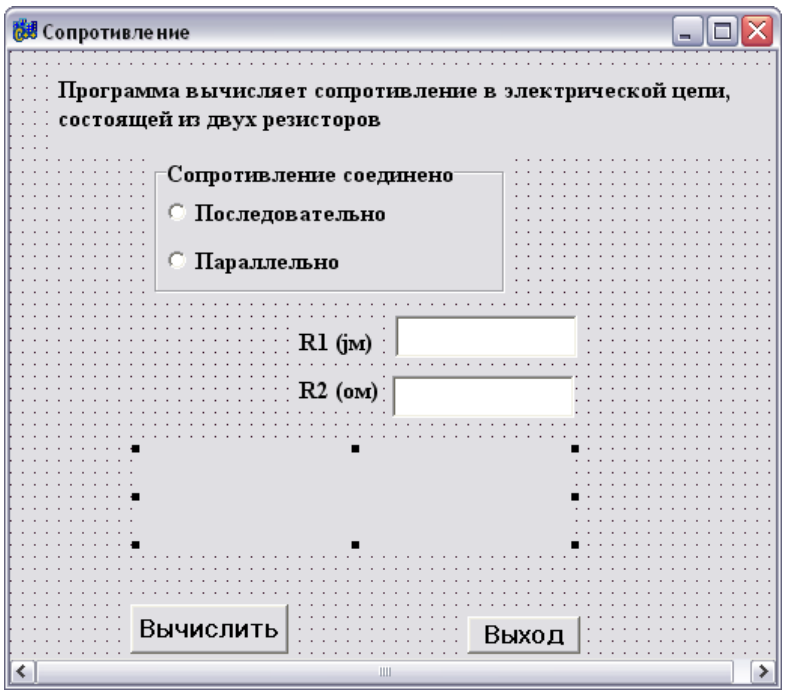


Рис. 4. Форма программы Сопротивление

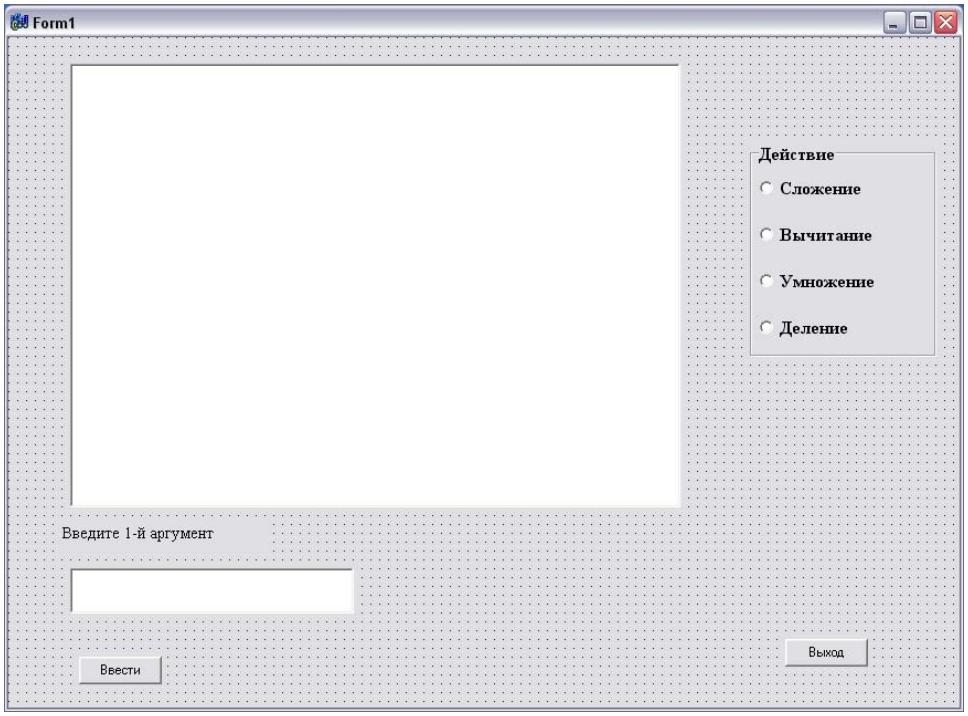
```
void _fastcall TForm1::Button1Click(TObject *Sender)
{
    }
// щелчок на переключателе "параллельно"
void __fastcall TForm1::RadioButton2Click(TObject *Sender)
{
    Label4->Caption = "";
    }
```

Лабораторная работа № 4. Проект “Арифмометр».

На форму нужно разместить следующие компоненты:

1. компонент Memo
2. компонент Edit
3. компонент RadioGroup
4. компонент Label
5. 3 компонента Button

Внешний вид приложения приведен на рисунке.



На этапе проектирования выполните следующие действия:

Для компонента Memo очистить содержимое свойства Lines, выполнив двойной щелчок на **…** В открывшемся окне редактора удалите слово Memo1.

Для компонента Label установите значение свойства Caption **Введите 1-й аргумент**

Для компонента Edit очистите значение свойства Text

Разместите два компонента Button один поверх другого. Измените значение свойства Caption первого компонента на **Ввест**и, а другого – **Вычислить.** Разместите третью компоненту Button как указано на рисунке и измените значение свойства Caption на **Выход**.

Для компонента Radiogroup выполните двойной щелчок на свойстве **Items** и в открывшемся окне редактора введите четыре строки:

Сложение Вычитание Умножение Деление, и нажмите **Ок**

Измените значение свойства Caption на **Действие.**

Приложение должно выполнять следующие действия:

1. На форме должна быть доступна кнопка **Ввести** и скрыта кнопка **Вычислить**
2. Нужно ввести в окно редактирования **Edit1** число и нажать кнопку **Ввести.**
3. В поле Memo1 должна появиться строка: **Первый аргумент равен** – введенное вами число. Метка **Label1** должна измениться на Введите 2-й аргумент. Строка редактирования **Edit1** должна очиститься. Кнопка **Ввести** должна скрыться и на его месте должна появиться кнопка **Вычислить.**
4. Нужно ввести значение второго аргумента. Выбрать одно из действий и нажать **на** кнопку **Вычислить.**
5. В поле Memo1 должны появиться строки: **Второй аргумент равен** – введенное вами число.

   Действие – выбранное вами действие.

   Результат равен

   Метка **Label1** должна измениться на **Введите 1-й аргумент**. Строка редактирования **Edit1** должна очиститься. Кнопка **Вычислить** должна скрыться и на его месте должна появиться кнопка **Ввести.**

Для отображения кнопки используется либо свойство Visible=true, либо метод Show. Для скрытия кнопки используется либо свойство Visible=false либо метод Hide, т.е.

Button1->Hide();

Для добавления строки в поле Memo1 используйте метод Add.
Memo1->Lines->Add(“Добавляемая строка”)

Для выбора действия используйте оператор switch. В качестве параметра оператора switch используйте свойство **ItemIndex** компонента **RadioGroup**

```
        switch(RadioGroup->ItemIndex)       // действия, зависящие от выбора
          {
case 0: Memo1->Lines->Add(“Действие- Сложение”); c=a+b; break;
case 1: Memo1->Lines->Add(“Действие- Вычитание”); c=a-b; break;
case 2: Memo1->Lines->Add(“Действие- Умножение”); c=a*b; break;
case 3: Memo1->Lines->Add(“Действие- Деление”); c=a/b; break;
default: Memo1->Lines->Add(“Действие не выбрано”);

        }
```

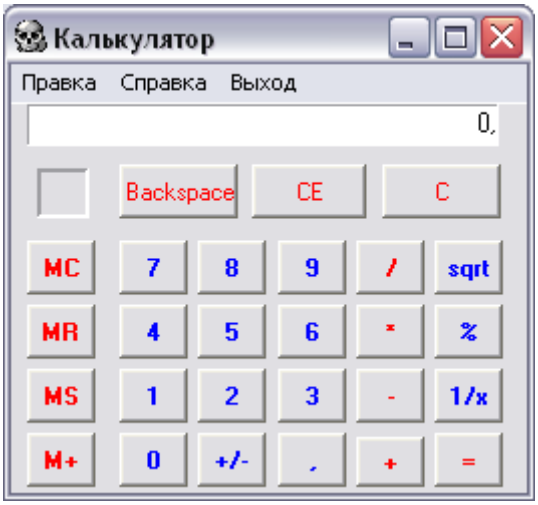
Для хранения значений1-го и 2-го аргументов и результата используйте переменные.
Объявление переменных выполняется следующим образом:
int a,b,c; - объявляются переменные **a,b,c** как целочисленные.
float a,b,c; - объявляются переменные **a,b,c** как вещественные.
double a,b,c; - объявляются переменные **a,b,c** как вещественные с плавающей точкой.

Для присваивания содержимого строки редактирования Edit1 переменной a можно воспользоваться командой:
a=StrToInt(Edit1->Text); - преобразовывает строку символов Edit1->Text в целое число и присваивает переменной **a**.
a=StrToFloatt(Edit1->Text); -преобразовывает строку символов Edit1->Text в вещественное число и присваивает переменной **a**.

Для очищения содержимого строки редактирования используйте метод Clear();, т.е.
Edit1->Clear();

Лабораторная работа № 5. Проект **Windows Калькулятор**

Составьте проект **Windows Калькулятор**. В качестве кнопок используйте компоненты **SpeedButton**. В качестве строки ввода значений используйте **Statictext**



Кнопки от 0 – до 9 формируют число в строку в StaticText .
Нажатие кнопки с соответствующей цифрой добавляет ее к содержимому строки в StaticText.

В файле реализации нужно объявить следующие глобальные переменные:

```
double a = 0, b = 0, zn = 0;
int op=0, DS = 0, z=0, dn = 0, d=0, kn=0, key=0;
AnsiString t1="", t2="", t3="";
```

Обработчик события на нажатие кнопки «5»:

```
if (dn!=1)
{
if ((StaticText1->Caption == "0,") ||
(StaticText1->Caption == FloatToStrF(a,ffGeneral,12,5)) ||
(StrToFloat(StaticText1->Caption) == a))
{
StaticText1->Caption = "";
StaticText1->Caption = StaticText1->Caption+"5,";
}
 else
 if (DS==0)
 {
 StaticText1->Caption = StaticText1->Caption-",";
 StaticText1->Caption = StaticText1->Caption+"5,";}
 else
 StaticText1->Caption = StaticText1->Caption+"5";
if (z==1)
kn=1;
d=1;
}
```

Обработчик события нажатия кнопки «=»

```
if (dn!=1)
{
DS=0;
key=0;
z=1;
if (kn==1)
{
b=StrToFloat(StaticText1->Caption);
switch (op)
{
 case 1: a=a+b; break;
 case 2: a=a-b; break;
 case 3: a=a*b; break;
 case 4: if (b==0)
{
StaticText1->Caption = "Íà íóëü äåëèòü íåëüçÿ";
dn = 1;
}
else
a=a/b; break;
}
if (dn!=1)
{
t1=FloatToStrF(a,ffGeneral,12,5);
```

```
if (t1.Pos(DecimalSeparator) != 0)
StaticText1->Caption = FloatToStrF(a,ffGeneral,12,5);
else
StaticText1->Caption = FloatToStrF(a,ffGeneral,12,5)+",";
}
kn=0;
}
op=4;
a=StrToFloat(StaticText1->Caption);
d=0;
}
```

Обработчик события нажатия на кнопку выбора действия «+»

```
if (dn!=1)
{
DS=0;
key=0;
z=1;
if (kn==1)
{
b=StrToFloat(StaticText1->Caption);
switch (op)
{
 case 1: a=a+b; break;
 case 2: a=a-b; break;
 case 3: a=a*b; break;
 case 4: if (b==0)
{
StaticText1->Caption = "На нуль делить нельзя";
dn = 1;
}
else
a=a/b; break;
}
if (dn!=1)
{
t1=FloatToStrF(a,ffGeneral,12,5);
if (t1.Pos(DecimalSeparator) != 0)
StaticText1->Caption = FloatToStrF(a,ffGeneral,12,5);
else
StaticText1->Caption = FloatToStrF(a,ffGeneral,12,5)+",";
}
kn=0;
}
op=1;
a=StrToFloat(StaticText1->Caption);
d=0;
}
```

Обработчик события нажатия на кнопку «BackSpace»

```
{
if (dn!=1)
{
 if (d!=0)
```

```
{
t1 = t2 = t3 = StaticText1->Caption;
 if (StaticText1->Caption.Length()==2)
 {
 StaticText1->Caption = "0,";
 }
 else
 {
 if (StaticText1->Caption.Length()==3)
  {
   while (t2.Length()!=1)
  {
  t2.Delete(t3.Length(),1);
  t3=t2;
  }
   if (t2=="-")
   {
   StaticText1->Caption = "0,";
   }
   else
   {
   t2 = t3 = StaticText1->Caption;
   while (t2.Length()!=1)
   {
   t2.Delete(t3.Length()-1,1);
   t3=t2;
   }
    if (t2==",")
    {
    t1.Delete(StaticText1->Caption.Length()-1,1);
    StaticText1->Caption = t1;
    }
    else
    {
    t1.Delete(StaticText1->Caption.Length(),1);
    StaticText1->Caption = t1;
    }
    }
    }
    else
    {
    while (t2.Length()!=1)
    {
    t2.Delete(t3.Length()-1,1);
    t3=t2;
    }
    if (t2==",")
    {
    t1.Delete(StaticText1->Caption.Length()-1,1);
    StaticText1->Caption = t1;
    }
    else
    {
```

```
    t1.Delete(StaticText1->Caption.Length(),1);
    StaticText1->Caption = t1;
    }
    }
   }
  }
 }
```

Обработчик события нажатия на кнопку «C»

```
StaticText1->Caption = "0,";
a=b=key=op=DS=kn=z=dn=d=0;
```

Аналогично реализуйте все необходимые обработчики нажатия на соответствующие кнопки.

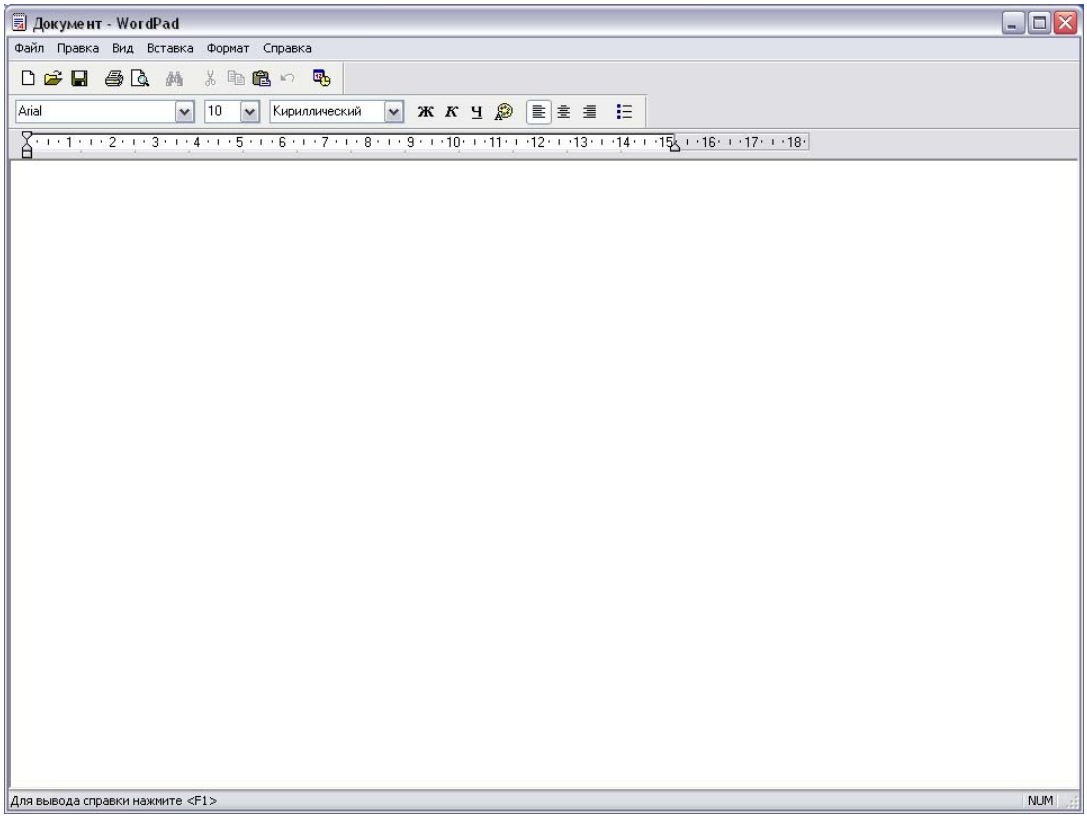
Лабораторная работа № 6

Проект: Текстовый редактор WordPad

Разработать проект, реализующий текстовый редактор Windows WordPad.
Внешний вид редактора приведен на рисунке.
Поместите на форму необходимые компоненты.

Компонент MainMenu - и с его помощью создайте главное меню приложения.
Компоненты SpeedButton – и с их помощью создайте панель быстрых кнопок.
Компоненты ComboBox – и с их помощью создайте выпадающие списки
Компонент RichEdit – рабочее окно приложения.

Установите необходимые свойства компонентов и создайте обработчики событий для пунктов мены и соответствующих кнопок.



Для обработки команды *Открыть* используйте фрагмент кода:

```
if (OpenDialog1->Execute() )
{
 MyFName = OpenDialog1->FileName;
RichEdit1->Lines->LoadFromFile(OpenDialog1->FileName);
```

Для обработки команды *Сохранить* используйте фрагмент кода:

```
if(MyFName!= "")
   RichEdit1->Lines->SaveToFile(MyFName) ;
   else
  if (SaveDialog1->Execute())
  {
   MyFName  =  SaveDialog1->FileName;
  RichEdit1->Lines->SaveToFile(SaveDialog1->FileName);
  }
```

Для обработки команды меню *Сохранить как* используйте фрагмент кода:

```
SaveDialog1->FileName = MyFName;
if (SaveDialog1->Execute())
{
 MyFName  = SaveDialog1->FileName;
RichEdit1->Lines->SaveToFile(SaveDialog1->FileName);
}
```

Для оустановки атрибутов шрифтов используйте фрагмент кода:

```
if (FontDialog1->Execute() )
RichEdit1->SelAttributes->Assign(FontDialog1->Font);
```

Для оустановки атрибутов шрифтов используйте фрагмент кода:

```
 if(ColorDialog1->Execute())
 RichEdit1->Color = ColorDialog1->Color;
```

Обработчик события OnFind компонента FindDialog1

```
void  _fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  int FoundAt, StartPos, ToEnd;
  TSearchTypes Option;
/* если было выделение, то поиск идет, начиная с его последнего символа, иначе - с
позиции курсора */
    StartPos = RichEdit1->SelStart;
    if (RichEdit1->SelLength)
     StartPos += RichEdit1->SelLength;
// ToEnd - длина текста, начиная с первой позиции поиска и до конца
    ToEnd = RichEdit1->Text.Length () - StartPos;
// поиск целого слова или нет в зависимости от установки пользователя
    if (FindDialog1->Options.Contains(frWholeWord) )
    Option << stWholeWord;
    else Option >> stWholeWord;
/* поиск с учетом или без учета регистра в зависимости от установки пользователя
    if (FindDialog1->Options.Contains(frMatchCase))
    Option << stMatchCase;
    else Option >> stMatchCase;
```

```
  FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd, Option);
  if (FoundAt != -1) // если найдено
  {
    RichEdit1->SetFocus() ; RichEdit1->SelStart = FoundAt;
    RichEdit1->SelLength = FindDialog1->FindText.Length();
    }
     else
    ShowMessage("Текст " + FindDialog1->FindText +" не найден");
    }
```

Обработчик события OnReplace компонента ReplaceDialog1

```
void _fastcall  TForm1::ReplaceDialog1Find(TObject *Sender)
{
…
// Если нажата кнопка "Заменить все", то уход на замену
if(ReplaceDialog1->Options.Contains(frReplaceAll))
ReplaceDialog1Replace(Sender);
}

void _fastcall TForm1::ReplaceDialog1Replace(TObject *Sender)
{
 if (RichEdit1->SelText != "")  // Если есть выделенный текст Замена выделенного текста
 RichEdit1->SelText = ReplaceDialog1->ReplaceText;
 else
 if (ReplaceDialog1->Options.Contains(frReplace))
 {
 ShowMessage("Текст " + ReplaceDialog1->FindText +' не найден");
 return;
 }
// Если нажата кнопка "Заменить все", то уход на поиск
 if (ReplaceDialog1->Options.Contains(frReplaceAll))
    ReplaceDialog1Find(Sender);
    }
```

Лабораторная работа
Графический редактор

Разработать проект графического редактора, воспроизводящего некоторые функции настоящих редакторов.

1. Поместите на форму два компонента типа TImage и расположите их в нижней левой части формы, придав квадратную форму, например, размером 20х 20. Это будут окна основного и вспомогательного цветов. Имена этих компонентов Image1 и Image2.
2. Перенесите на форму еще компонент типа TImage и расположите его в верхней части формы, несколько отступив от левого края и растянув так, чтобы он занимал основную часть формы. Это будет холст для рисования. Имя этого компонента будет Image3.
3. Перенесите на форму еще один компонент типа TImage и расположите его внизу правее первых двух на одном с ними уровне. Это будет палитра цветов. Ее высоту задайте той же, что у первых двух компонентов, а длину - в 10 раз большую. Имя этого компонента будет Image4.
5. Перенесите на форму кнопку типа **TSpeedButton** и расположите ее в верхнем левом углу формы. Эта кнопка будет соответствовать кисти - типичному инструменту графических редакторов. Назовите ее **SBBrush.** Установите у кнопки свойство **Grouplndex** равным 1 и

свойство **AllowAllUp** в **true.** Эти свойства обеспечат кнопке возможность фиксироваться в нажатом и не нажатом состоянии. Желательно загрузить в свойство **Glyph** пиктограмму кисти (файл ..\Images\Buttons\brush.bmp).

6. Перенесите на форму еще одну кнопку типа **TSpeedButton** и расположите ее ниже **SBBrush.** Эта кнопка будет соответствовать указателю цвета пиксела рисунка. Назовите ее **SBColor.** Установите свойство **GrouplndeX** равным 1 (это обеспечит, что только одна из двух кнопок может быть нажата) и свойство **AllowAllUp** в **true.** Желательно загрузить в свойство **Glyph** пиктограмму (например, файл „.\Images\ Buttons\one2one.bmp).
7. Перенесите на форму диалог **OpenPictureDialog.**
8. Перенесите на форму главное меню **MainMenu. В** меню задайте раздел Файл подразделом **Открыть**. Назовите этот подраздел **MOpen.** Задайте еще один раздел - **Правка** с подразделом **Отменить**. Назовите этот подраздел **Undo**.

Создайте обработчики событий.

9. **В заголовочный файл модуля включите оператор:**

```
Graphics::TBitmap *BitMap = new Graphics::TBitmap;
```

Этот оператор создает объект BitMap типа TBitmap. В этот объекте будет сохраняться изображение, чтобы его можно было восстановить командой **Отменить**.

10. **Для события OnCreate формы напишите обработчик вида:**

```
// задание свойств кисти основного и вспомогательного цветов
Image1->Canvas->Brush->Color = clBlack;
Image2->Canvas->Brush->Color = clWhite; // заполнение окон основного и
вспомогательного цветов
Image1->Canvas->FillRect(Rect(0,0,Image1->Width, Imagel->Height)) ;
Image2->Canvas->FillRect(Rect(0,0,Image2->Width, Image2->Height)) ; // задание ширины
элемента палитры цветов
int HW = Image4->Width /10; // закраска элементов палитры цветов
for(int i = 1; i <=10; i++)
{
   switch (i)
   {
   case 1:Image4->Canvas->Brush->Color = clBlack; break;
   case 2:Image4->Canvas->Brush->Color = clAqua; break;
   case 3:Image4->Canvas->Brush->Color = clBlue; break;
   case 4:Image4->Canvas->Brush->Color = clFuchsia; break;
   case 5:Image4->Canvas->Brush->Color = clGreen; break;
   case 6:Image4->Canvas->Brush->Color = clLime; break;
   case 7:Image4->Canvas->Brush->Color = clMaroon; break;
   case 8:Image4->Canvas->Brush->Color = clRed; break;
   case 9:Image4->Canvas->Brush->Color - clYellow; break;
   case 10:Image4->Canvas->Brush->Color = clWhite;
     }
   Image4->Canvas->Rectangle((i-1)*HW,0,i*HW, Image4->Height);
     }
      // рисование креста на холсте ~ только для тестирования
 Image3->Canvas->MoveTo(0,0) ;
Image3->Canvas->LineTo(Image3->Width,Image3->Height);
Image3->Canvas->MoveTo(0,Image3->Height) ;
Image3->Canvas->LineTo(Image3->Width,0) ;
BitMap->Assign(Image3->Picture) ;
```

11. В обработчик события формы **OnDestroy** запишите оператор BitMap->Free (); который освобождает память при закрытии приложения.
12. Для подраздела меню **Открыть** в обработчик включите операторы:

```
if (OpenPictureDialog1->Execute())
```

```
{
  Image3->Picture->LoadFromFile(OpenPictureDialog1->FileName);
  BitMap->Assign(Image3->Picture);
  }
```

13. Для подраздела меню **Отменить** в обработчик включите оператор:
    Image3->Picture->Assign(BitMap); восстанавливает на холсте изображение, сохраненное в **BitMap.**
14. В обработчик события **OnClick** кнопок **SBBrush** и **SBColor** запишите оператор
    if (((TSpeedButton *) Sender)->Down) BitMap->Assign(Image3->Picture) ;
    Этот оператор запоминает в **BitMap** текущий вид изображения перед началом работы с очередным инструментом.
15. В обработчик события **OnMouseDown** компонентов **Image3** и **Image4** вставить код:

```
if((Sender ==  Image4)  || SBColor->Down) // режим установки основного м
вспомогательного цветов
{
 if(Button == mbLeft)
{
// установка  основного цвета
   Image1->Canvas->Brush->Color =((Tlmage *)Sender)->Canvas->Pixels[X][Y];
   Image1->Canvas->FillRect(Rect(0,0,Image1->Width, Image1->Height));
 }
else
{ // установка вспомогательного цвета
Image2->Canvas->Brush->Color  = ((Tlmage *)Sender)->Canvas->Pixels[X][Y];
Image2->Canvas->FillRect(Rect(0,0,Image2->Width,Image2->Height));
      }
      }
      else  if (SBBrush->Down)
        // режим закраски указанной области холста
{
if (Button==mbLeft)
Image3->Canvas->Brush->Color = Imagel->Canvas->Brush->Color;
          else
          Image3->Canvas->Brush->Color =Image2->Canvas->Brush->Color;
          Image3->Canvas->FloodFill(X,Y, Image3->Canvas->Pixels[X] [Y] , fsSurface);
      }
```

Теперь усовершенствуйте графический редактор, т.е. реализуйте основные приемы, которые необходимы при создании различных инструментов.

Приложение должно выполнить следующие функции:

- Установка основного и дополнительного цветов. Щелчок на панели цветов левой кнопкой мыши устанавливает основной цвет, а щелчок правой кнопкой - вспомогательный.
- Кисть - кнопка SBBrush. Закрашивает замкнутую область, ограниченную цветом того пиксела, который указан щелчком мыши. При щелчке левой кнопкой закрашивание производится основным цветом, при щелчке правой кнопкой - вспомогательным.
- Индикация цвета - кнопка **SBColor. В** этом режиме вы можете указать курсором мыши любой пиксел на изображении и, щелкнув левой кнопкой, установить цвет этого пиксела как основной, а щелкнув правой кнопкой, установить его как вспомогательный цвет.
- Карандаш - кнопка **SBPen.** В этом режиме вы можете рисовать произвольную кривую основным цветом.
- Выделение фрагмента - кнопка **SBRect.** Фрагмент выделяется точечной рамкой. Выделенный фрагмент можно в дальнейшем перетащить мышью на другое место. Если в процессе перетаскивания нажата клавиша Ctrl, то производится копирование фрагмента, в

противном случае - вырезание, при котором область начального размещения фрагмента закрашивается вспомогательным цветом. Выделенный фрагмент может быть также скопирован или вырезан в буфер обмена Clipboard соответствующими командами меню.

- Стирание изображения (ластик) - кнопка **SBErase.** Перемещение ластика закрашивает область под ним во вспомогательный цвет.
- Рисование прямоугольника - кнопка **SBRectang.** Рисуется прямоугольная рамка основным цветом.
- Рисование заполненного прямоугольника - кнопка **SBFillRec.** Рисуется прямоугольная рамка основным цветом и прямоугольник внутри закрашивается вспомогательным цветом.
- Рисование прямой линии - кнопка **SBLine.** Рисуется прямая линия основным цветом.
- Открытие графического файла - команда **Файл | Открыть.**
- Сохранение изображения в графическом файле - команда **Файл | Сохранить как.**
- Отмена операций, выполненных последним использованным инструментом - команда **Правка | Отменить.**
- Копирование или вырезание выделенного фрагмента изображения в буфер обмена Clipboard - команды **Правка | Копировать или Правка | Вырезать.**
- Вставка графического изображения типа битовой матрицы из буфера обмена Clipboard - команда **Правка | Вставить.**

Функция выделения фрагмента осуществляется методом **DrawFocusRect.** В этом режиме при событии **OnMouseDown** холста - компонента **Image3, нужно** выполнить операторы:

```
// Запоминание начального положения курсора мыши
x0=x;
 y0=y;
// формирование начального положения области фрагмента
R.Top  = x;
R.Bottom = x;
R.Left  =  y;
R.Right  =  y;
//  Рисование рамки
Image3->Canvas->DrawFocusRect(R) ;
RBegin  =  true;
```

Эти операторы запоминают координаты мыши **x и y** в переменных **x0 и y0,** задают начальные координаты прямоугольной области - переменной **R** типа **TRect и** рисуют рамку (пока нулевого размера) методом **DrawFocusRect.** Устанавливается также флаг начала выделения фрагмента - переменная **RBegin.**

При событии **OnMouseMove** компонента **Image3,** если установлен флаг **RBegin,** выполняются операторы:

```
// Стирание прежней райки
Image3->Canvas->DrawFocusRect(R);
// формирование, области R
if (x0  < x)
{
R.Left  =x0; R.Right  = x;
}
  else
  {
  R.Left  =x; R.Right  = x0;
  }
  if (y0 <  y)
  {
   R.Top =  y0; R.Bottom =  y;
  }
```

```
  else
  {
   R.Top  =  y; R.Bottom =  y0;
  }
// Рисование новой рамки
Image3->Canvas->DrawFocusRect(R);
```

Первый из этих операторов стирает прежнее изображение рамки (напомним, что метод **DrawFocusRect** рисует рамку с помощью операции XOR). Два следующих оператора формируют новую область R из начальных координат (x0, y0) и текущих координат курсора (x, y). Дело в том, что область, передаваемая в функцию **DrawFocusRect,** должна быть сформирована «правильно» - значение **R.Left** должно быть меньше **R.Right, а R.Top** - меньше **R.Bottom.** Поскольку пользователь может перемещать курсор в любом направлении и соотношение координат (x0, y0) и (x, y) может быть любым, требуется упорядочивание координат.

Последний оператор рисует рамку в новом положении.

Итак, рамка, ограничивающая фрагмент нарисована. Теперь рассмотрим процедуру перетаскивания пользователем выделенного фрагмента. Если пользователь помещает курсор внутрь выделенной области и нажимает кнопку мыши, выполняются операторы:

```
// Стирание прежней рамки
 Image3->Canvas->DrawFocusRect(R) ;
// Установка флага перетаскивания
RDrag  =  true;
// Запоминание начального положения курсора ыьшт
x0  = x;
 y0  =  y; // Запоминание начального положения перетаскиваемого фрагмента
 R0 =  R;
// Запоминание изображения
BitMap->Assign(lmage3->Picture);
// Установка цвета кисти
Image3->Canvas->Brush->Color =  Image2->Canvas->Brush->Color;
```

Первый оператор стирает рамку. Второй - устанавливает флаг перетаскивания - переменную **RDrag.** Два следующих оператора запоминают начальное положение перетаскиваемого фрагмента в переменной R0 типа **TRect.** Следующий оператор запоминает методом **Assign** изображение в момент начала перетаскивания в переменной **Bitmap.** Это необходимо, чтобы в процессе перетаскивания можно было восстанавливать испорченные места изображения и чтобы при желании пользователя можно было в дальнейшем отменить результат перетаскивания. Последний оператор задает цвет кисти равным вспомогательному цвету, хранящемуся в компоненте **lmage2.**

При событии OnMouseMove компонента **Image3,** если установлен флаг **RDrag,** выполняются операторы:

```
// Восстановление изображения под перетаскиваемым фрагментом
Image3->Canvas->CopyRect(R,BitMap->Canvas,R);
// Если не нажата  клавиша Ctrl - стирание изображения в R0
 if (!Shift.Contains(ssCtrl))
      Image3->Canvas->FillRect(R0);
// Формирование нового положения фрагмента
R.Left  = R.Left  + x -  x0;
R.Right  =  R.Right  +  x  -  x0;
R.Top  = R.Top +  y  -  y0;
R.Bottom =  R.Bottom +  y  - y0;
// Запоминание положения курсора  мыши
```

```
x0 = x;
y0 = y;
// Рисование фрагмента в новом положении
   Image3->Canvas->CopyRect(R,BitMap->Canvas,R0);
// Рисование рамки
   Image3->Canvas->DrawFocugRect(R);
```

Первый оператор восстанавливает изображение под перетаскиваемым фрагментом в его прежней позицией (т.е. стирает фрагмент), копируя соответствующую область методом **CopyRect** из компонента **BitMap.** Далее, если не нажата клавиша Ctrl, то очищается область начального размещения фрагмента (осуществляется вырезание) методом **FillRect. В** противном случае начальное изображение фрагмента остается на месте. Затем запоминаются новые координаты курсора и новое положение фрагмента, после чего фрагмент и его рамка рисуются в новом положении.

Режимы рисования заполненного и не заполненного прямоугольников. Начало этих режимов по событию **OnMouseDown** и их продолжение по событиям **OnMouseMove** не отличаются от рассмотренного ранее режима выделения фрагмента. Отличие только в том, что при завершении формирования пользователем прямоугольной рамки, т.е. при событии **OnMouseUp,** надо в данном случае нарисовать прямоугольник. Рисование заполненного прямоугольника осуществляется операторами:

```
Image3->Canvas->Brush->Color = Image2->Canvas->Brush->Color;
Image3->Canvas->Pen->Color = Image1->Canvas->Brush->Color;
Image3->Canvas->Rectangle(R.Left,R.Top,R.Right,R,Bottom);
```

Они задают цвета кисти и пера и рисуют прямоугольник методом **Rectangle.** Рисование не закрашенного прямоугольника осуществляется операторами:

```
Image3->Canvas->Brush->Color = Image1->Canvas->Brush->Color;
Image3->Canvas->FrameRect(R) ;
```

Обратите внимание, что равным основному цвету задается цвет кисти, а не пера, поскольку метод **FrameRect** рисует цветом кисти.

Рисование прямой линии осуществляется следующим образом. Начало рисования по событию **OnMouseDown** сводится к операторам:

```
x0 = x; y0 = y;
 x1 = x;
y1 = y;
Image3->Canvas->Pen->Mode = pmNotXor;
Image3->Canvas->Pen->Color = Image1->Canvas->Brush->Color;
```

Они запоминают положение курсора в двух наборах переменных: **(x0,y0)** и **(x1, y1).** Зачем нужны два набора - будет сказано позднее. Затем устанавливается цвет пера и режим **pmNotXor,** который позволит при движении мыши стирать изображение линии.

При событиях OnMouseMove работают следующие операторы:

```
// Стирание прежней линии
Image3->Canvas->MoveTo(x0,y0) ;
Image3->Canvas->LineTo(x1,yl);
// Рисование новой линии
Image3->Canvas->MoveTo(x0,y0);
Image3->Canvas->LineTo(x,y);
// Запоминание новых координат конца линии
x1 =x;
y1 = y;
```

В этих операциях сначала парой методов MoveTo и LineTo стирается линия в прежнем положении, а затем такой же парой методов рисуется новая линия. После этого запоминаются новые координаты конца линии.

Обработчик события OnMouseUp дополните переводом пера в режим pmCopy, при

котором рисуется окончательная линия:

```
// Стирание прежней линии
Image3->Canvas->MoveTo(x0,y0);
Image3->Canvas->LineTo(xl,yl); // Рисование новой линии}
 Image3->Canvas->Pen->Mode = pmCopy;
 Image3->Canvas->MoveTo(x0, y0) ;
 image3->Canvas->LineTo(x,y);
```

Инструмент **Перо** позволяет рисовать произвольные линии. Казалось бы, естественной реализацией этого инструмента был бы следующий оператор, включаемый в обработчик события OnMouseMove:

```
Image3->Canvas->LineTo(x, y);
```

Реализуйте теперь следующие инструменты:

1. Ластик реализуется методом **FillRect,** очищающим изображение под его рамкой. Сохранение файла осуществляется с использованием компонента типа **SavePictureDialog** оператором

```
if (SavePictureDialogl->Execute() )
{
    BitMap->Assign(Image3->Picture);
    BitMap->SaveToFile(SavePictureDialogl->FileName);
    }
```

2. Изображение с холста сохраняется в компоненте **Bitmap,** из которого методом SaveToFile записывается в выбранный пользователем файл.

Команды меню **Копировать и Вырезать** осуществляются процедурой

```
void  _fastcall  TForm1: :MCopyClick (TObject  *Sender)
 {
// Стирание рамки
Image3->Canvas->DrawFocusRect(R);
// Создание временного объекта BMCopy
 Graphics::TBitmap *BMCopy = new Graphics::TBitmap;
 BMCopy->Width = R.Right - R.Left;
 BMCopy->Height = R.Bottom - R.Top;
 try
  { // Копирование фрагмента в BMCopy
 BMCopy->Canvas->CopyRect(Rect(0,0,BMCopy->Width,
                          BMCopy->Height),Image3->Canvas,R) ; // Восстановление рамки
 Image3->Canvas->DrawFocusRect(R);
// Копирование в Clipboard
 Clipboard!)->Assign(BMCopy);
if (Sender = =MCut) ( // Вырезание
 Image3->Canvas->Brush->Color = clWhite;
 Image3->Canvas->FillRect(R) ; } }
  _finally
{
// Освобождение памяти
BMCopy->Free();
}
}
```

3. Копированию или вырезанию подлежит ранее выделенный пользователем объект, местоположение и размеры которого определяются переменной **R**. Поэтому сначала создается временный объект типа **TBitmap,** в который переносится копируемый фрагмент. Затем этот объект копируется в буфер обмена **Clipboard**.

4. Для работы с буфером обмена используйте функцию **Clipboard,** создающую объект типа **TClipboard,** инкапсулирующий свойства буфера обмена Windows.

Аналогично реализуйте команду **Вставить,** копирующую изображение из буфера обмена Clipboard:

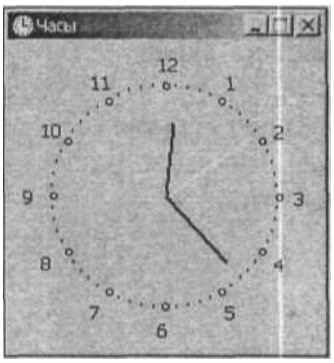
```
void _fastcall  TForm1::MPasteClick(TObject  *Sender)
 {
  Graphics::TBitmap  *BMCopy = new Graphics::TBitmap;

 try
 {
  try
  {
   BMCopy->LoadFromClipboardFormat(CF_BITMAP,Clipboard () ->GetAsHandle
   (CF_BITMAP) , 0) ,Image3->Canvas->CopyRect(Rect(0,0,BMCopy->Width,
  BMCopy->Height), BMCopy->Canvas, Rect(0,0,BMCopy->Width, BMCopy->Height));
  }
  _finally
  {
 BMCopy->Free();
 }
 } catch (EInvalidGraphic &)
 {
 ShowMessage("Ошибочный формат  графики");
  }
  }
```

. Попробуйте усовершенствовать редактор, добавив, в него выбор ширины линий, рисование эллипсов и т.д.

Лабораторная работа

Составить проект Идущие цифровые часы как показано на изображении.




```
#include "DateUtils.hpp"      // для доступа к SecondOf, MinuteOf и/ HourOf
#include "math.h"             // для доступа к sin и cos
#define R=75                  // радиус циферблата часов
int x0, y0;                   // центр циферблата
int ahr,amin,asec;            // положение стрелок (угол)

        fastcall TForml::TForml(TComponent* Owner): Tform1 Owner)
{
     TDateTime t;
```

```
        // зададим размер        формы в соответствии с размером циферблата
        ClientHeight = (R+ 30)*2;
        ClientWidth = (R+ 30)*2;
        x0  = R + 30;
        У0  = R + 30;
        t = Now();
        /*  Определить положение стрелок. Угол между метками  (цифрами)  часов,
    например,  цифрами 2 и 3,  - 30 градусов. Угол между метками минут -  6 градусов.
    Угол отсчитываем от 12-ти часов */

        ahr = 90 - HourOf(t)*30-(MinuteOf(Today() ) / 12) *6;
        amin = 90 - MinuteOf (t) *6;
        asec = 90 - SecondOf ( Today () )*6;
        Timerl->Interval = 1000;                          // период сигнала от таймера
    Timerl->Enabled = true;                          // пуск таймера
    }
    // рисует вектор из точки (x0,y0) под углом a относительно // оси X. Длинна вектора
1
    void _fastcall TForml::Vector(int x0, int y0, int a, int l)
    {
        // x0,y0 - начало вектора, a - угол между осью x и вектором, 1 - длина вектора
    #define TORAD 0.0174532          // коэффициент пересчета угла из //градусов в
                                     радианы
      int x, y;                                       // координаты конца вектора
      Canvas->MoveTo(x0,y0);
      x =  x0 + 1 * cos(a*TORAD);
      y =  y0 - 1 * sin(a*TORAD);
      Canvas->LineTo(x,y);
    }
    // прорисовка циферблата
    void  fastcall TForml::FormPaint(TObject *Sender)
    {
        int x, y;          // координаты маркера на циферблате
        int a;                    // угол между OX и прямой (x0,yo) (x,y)
        int h;                    // метка часовой риски
        TBrushStyle bs;  // стиль кисти
        TColor pc;                // цвет карандаша
    int pw;                  // ширина карандаша
    bs = Canvas->Brush->Style;
    pc = Canvas->Pen->Color;
    pw = Canvas->Pen->Width;
    Canvas->Brush->Style = bsClear;
    Canvas->Pen->Width =  1;
    Canvas->Pen->Color = clBlack;
    a = 0;          // метки ставим от 3-х часов, против часовой стрелки
    h = 3;          // угол 0 градусов - это 3 часа // циферблат
```

```
        while ( a < 360 )
        {
            x = x0 + R * cos (a * TORAD);
            y = x0 - R * sin(a * TORAD);
            Form1->Canvas->MoveTo(x,y);
             if ( (a % 30) == 0 )
            {
                Canvas->Ellipse(x-2,y-2,x+3,y+3); // цифры по большему радиусу
                x = x0 +  (R+15) * cos(a * TORAD);
                y = x0 -  (R+15) * sin(a * TORAD);.
                Canvas->TextOut(x-5,y-7,IntToStr(h));
                h--;
         if ( h == 0 ) h = 12;
         }
         }
     Canvas->Ellipse (x-l,y-1,x+1,y+1) ;
      a = a + 6;                          // 1 минута - 6 градусов 1 ,  восстановить карандаш и
кисть
            Canvas->Brush->Style = bs;
            Canvas->Pen->Width = pw;
            Canvas->Pen->Color = pc;
            DrawClock() ;
        }
        void _fastcall TForml: :DrawClock(void)
        {
            TDateTime t;
           Canvas->Pen->Color = clBtnFace;
           Canvas->Pen->Width = 3;
           Vector(x0,y0, ahr, R-20); // часовую
           Vector(x0,y0, amin, R-15); // минутную
           Vector(x0,y0, asec, R-7); // секундную
           t = Now() ;
           ahr =  90 - HourOf(t)*30-(MinuteOf(t)% 12)*6;
           amin = 90 - MinuteOf (t) *6;
           asec = 90 - SecondOf(t)*6;
           Canvas->Pen->Width = 3; // часовая стрелка
           Canvas->Pen->Color = clBlack;
           Vector(x0,y0, ahr, R-20);
           Canvas->Pen->Width = 2 ; // минутная стрелка
           Canvas->Pen->Color = clBlack;
           Vector(x0,y0, amin, R-15);
           Canvas->Pen->Width  = 1;
           Canvas->Pen->Color = clYellow;  // секундная стрелка
           Vector (x0,y0, asec, R-7) ;
        }
        //сигнал от таймера
```

```
void _fastcall TForml::TimerITimer(TObject *Sender)
{
    DrawClock();
}
```

Лабораторная работа
Разработка проекта "База данных" в C++ Builder

Цель работы

Написать проект для работы с базой данных MS Access, содержащей оценки студентов по изучаемым дисциплинам.

Спецификация проекта:

проект должна подключаться к базе данных, содержащей оценки студентов по изучаемым дисциплинам. База данных должна быть создана в MS Access (формат 2002-2003). проект должна позволять пользователю создавать, редактировать и удалять записи. Кроме этого проект должна подсчитывать средний балл для выбранного в таблице студента. Соединение с базой данной с помощью технологии ADO (от англ. ActiveX Data Objects — «объекты данных ActiveX») — интерфейс программирования приложений для доступа к данным, разработанный компанией Microsoft.

Для создания формы использовать компоненты:
Label – для подписей
Button – для инициирования действий
Edit – для вывода количества полей (колонок) и записей (строк) таблицы
ADOConnection –компонент для подключения к базе данных
ADOTable – компонент для работы со структурой и данными таблицы базы данных
DataSource – компонент для передачи данных компоненту DBGrid и DBNavigator.
DBGrid – компонент для визуализации таблицы из БД
DBNavigator – компонент для редактирования записей подключенной таблицы БД

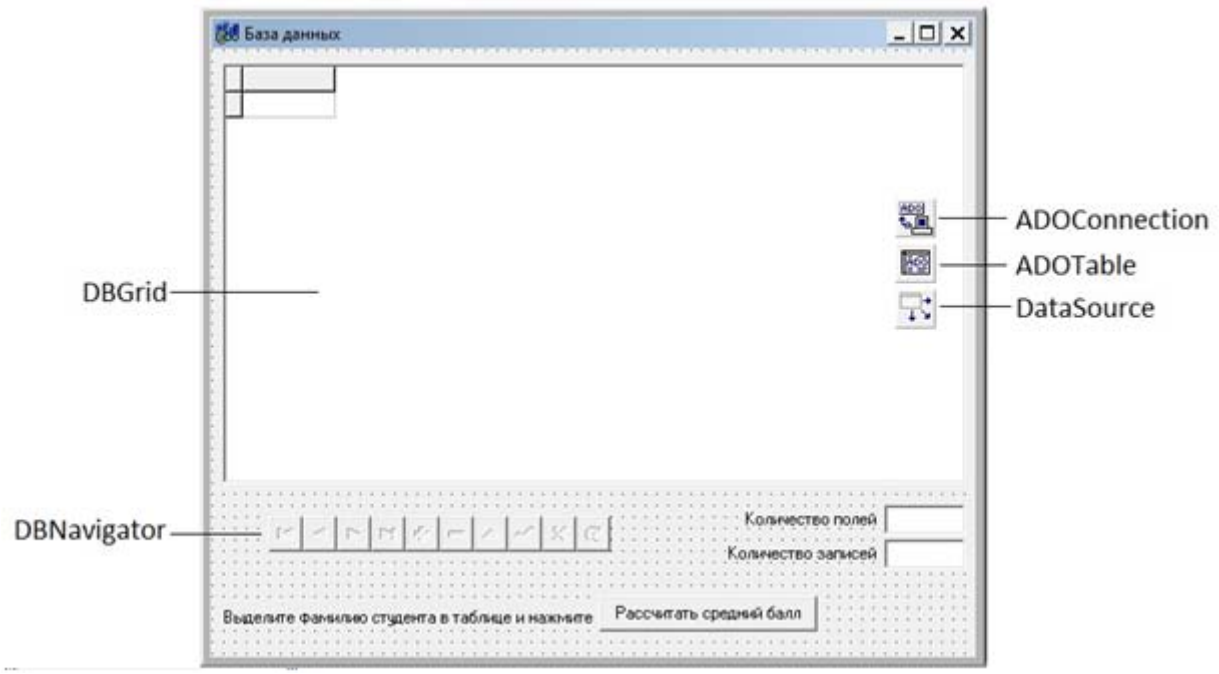


Рисунок 1. Рекомендуемая компоновка формы

Рекомендации для выполнения лабораторной работы:

1) Создать базу данных «Ведомость» в MS Access. Запустить Microsoft Office Access. В появившемся окне выбрать пункт «Новая база данных» и указать путь для сохранения базы данных,

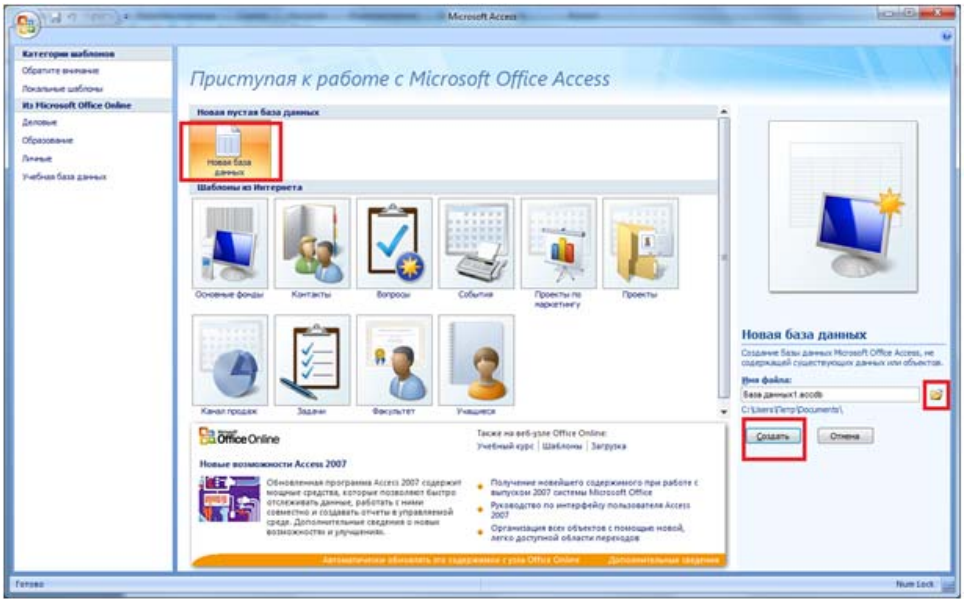


Рисунок 2. Создание БД

2) В появившемся окне выбрать расположение создаваемого файла БД и указать тип «Базы данных Microsoft Office Access 2002-2003 (*.mdb)» как показано на рисунке 3.

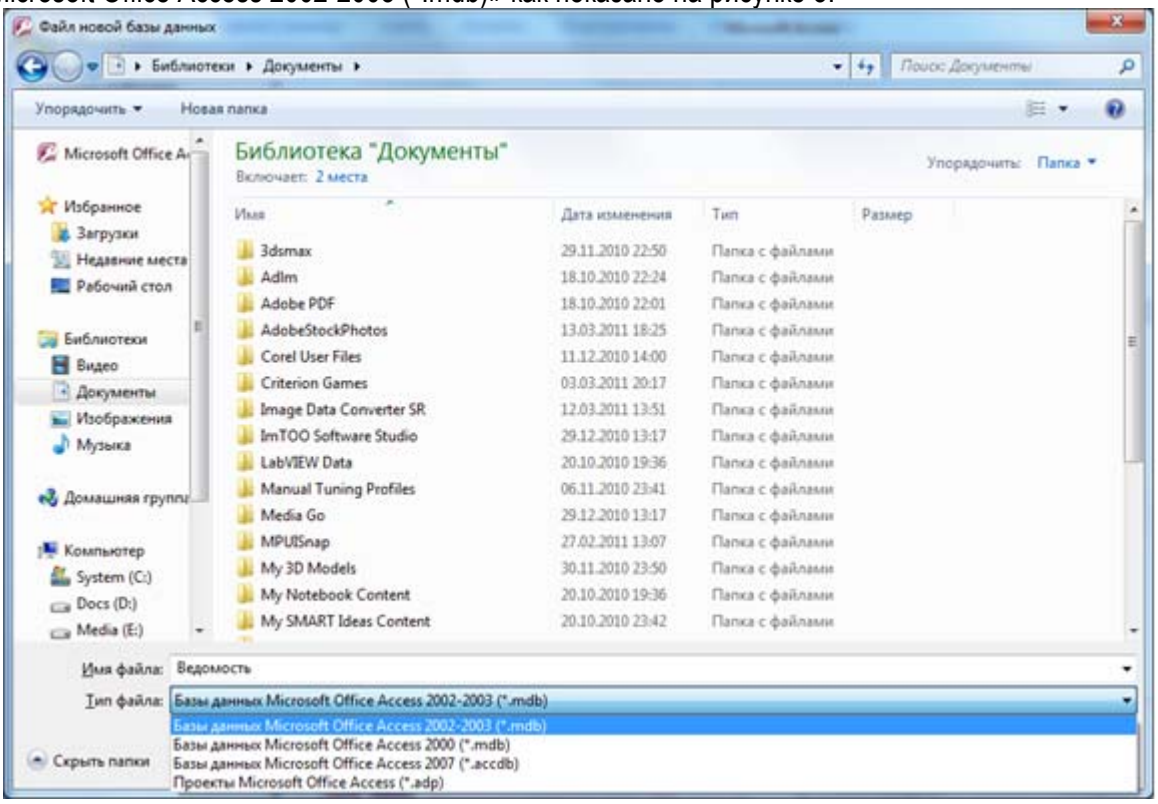


Рисунок 3. Выбор типа файла БД

3) Нажать кнопку «Создать» (рисунок 2).

4) В появившемся окне нажать кнопку режим (рис. 4) и задать имя таблицы «Студенты»

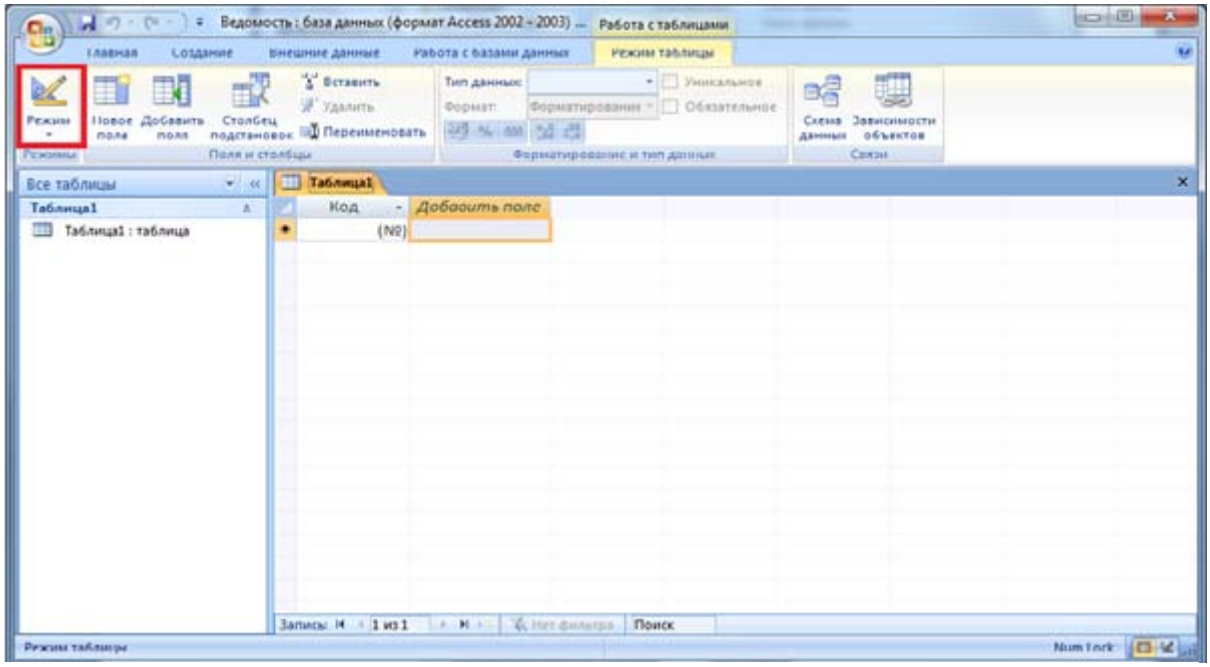


Рисунок 4. Переключение режима «Конструктор»

5) В БД создать таблицу «Студенты», структура которой показана на рисунке 5.

Студенты

Имя поля	Тип данных
Номер зачетки	Числовой
ФИО	Текстовый
Информатика	Числовой
Геодезия	Числовой
Математика	Числовой
Физика	Числовой
Моделирование систем	Числовой

Рисунок 5. Поля таблицы «Студенты»

Поле «Номер зачетки» сделать ключевым (уникальным), для этого выделить это поле и нажать кнопку на панели инструментов:

Для поля ФИО указать длину поля 50 символов:

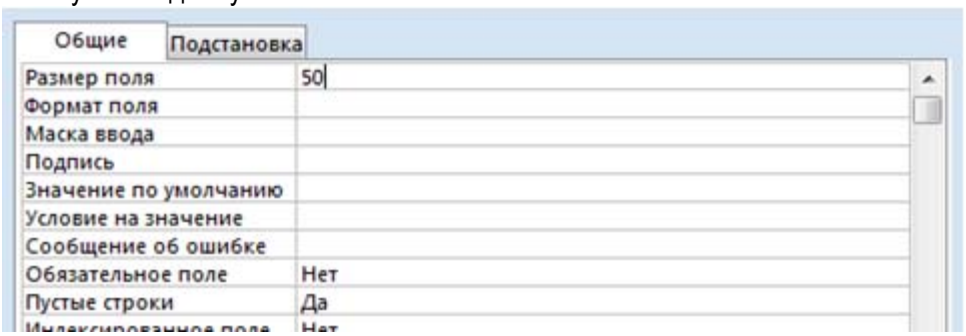


Рисунок 6. Настройка размера текстового поля (кол-ва символов)

Остальные поля должны быть числовыми. Они будут содержать оценку за соответствующую дисциплину.

6) Сменить режим на «Таблица», нажав кнопку «Режим» на панели инструментов

7) В таблицу ввести несколько записей.

Например:

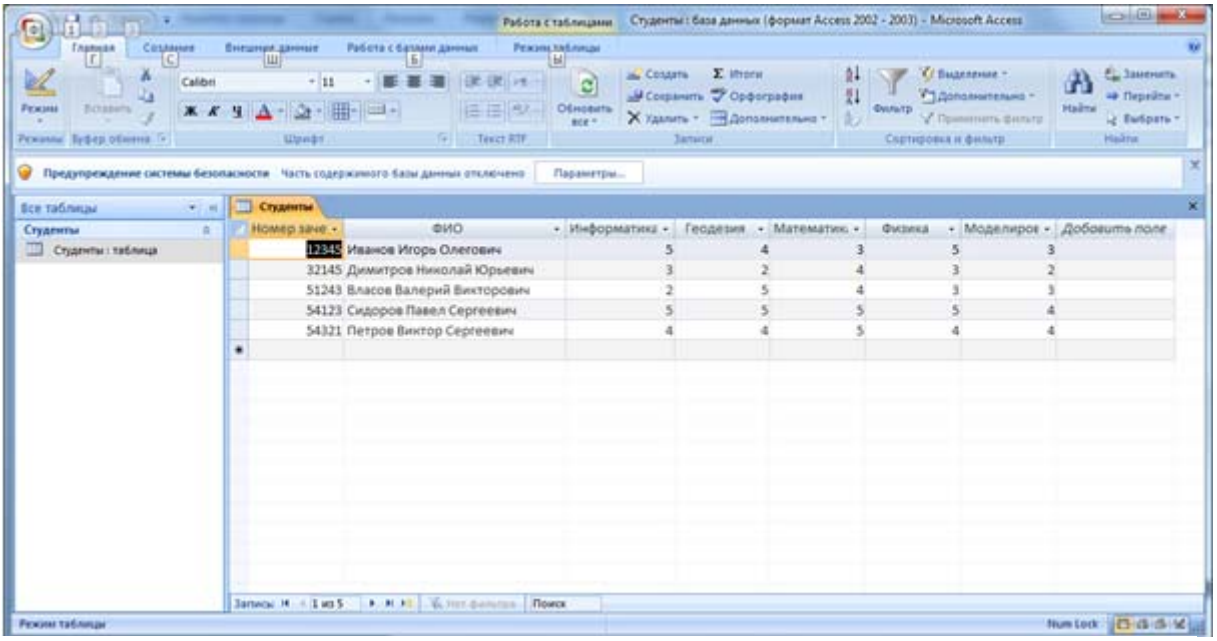


Рисунок 7. Пример заполнения таблицы

8) Сохранить и закрыть базу данных. Переместить файл базы данных в папку будущей проекта.

9) Запустить C++ Builder. При запуске автоматически создается новый проект. Окно C++ Builder 10) Сохранить проект в свою рабочую папку, выполнив команду меню File / Save Project As. Будет сохранено несколько файлов проект.

11) Расположить на форме требуемое количество объектов (см. рис.1).

Вкладка Standard: Label A, Button OK, Edit ab|.

Вкладка ADO: ADOConnection, ADOTable.

Вкладка DataAccess: DataSource.

Вкладка DataControls: DBGrid, DBNavigator.

12) Изменить подписи объектов Label и пользовательской формы Form1. Для этого необходимо у перечисленных объектов отредактировать свойство Caption в соответствии с рисунком 1.

13) У объектов Edit и ComboBox очистить поле свойства Text.

14) Поскольку объекты Edit используются только для вывода, то необходимо присвоить свойству ReadOnly для этих объектов значение true.

15) Настройка подключения к БД осуществляется за несколько шагов:

1. Выделить компонент ADOConnection1. Установить значение false для свойств Connected (соединение) и LoginPromt (вход с паролем) Сформировать строку подключения ConnectionString (строка параметров подключения к базе данных), нажав на кнопку с тремя точками.

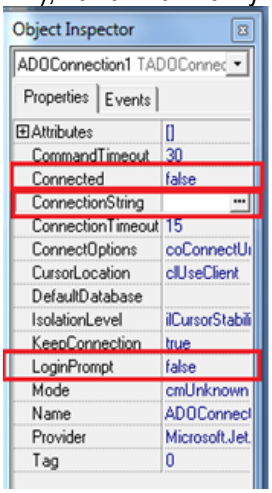


Рисунок 8. Настройка свойств объекта ADOConnection1

В появившемся окне выбрать пункт «Use Connection String» и нажать на кнопку «Build»:

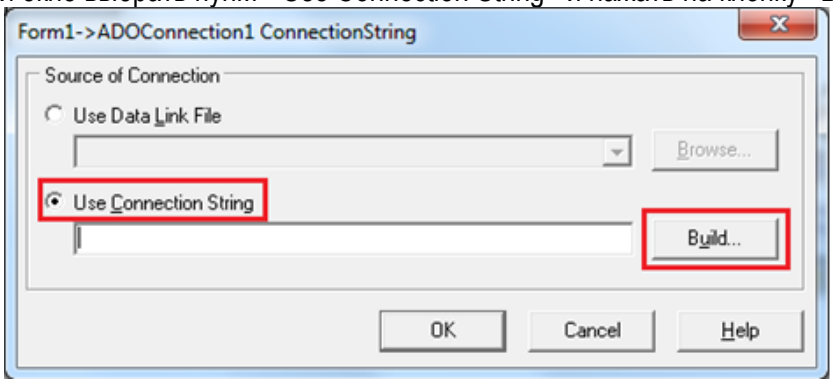


Рисунок 9. Окно настройки подключения

Далее необходимо выбрать поставщика данных и нажать на кнопку далее:

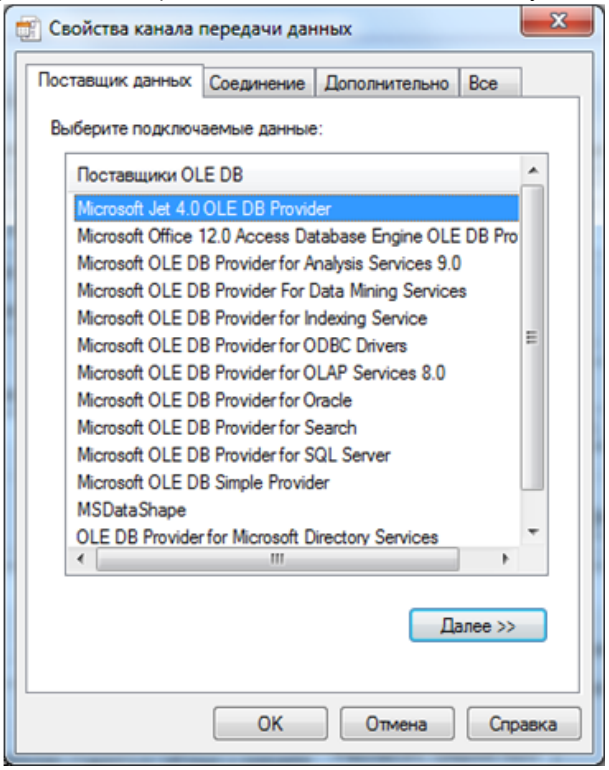


Рисунок 10. Выбор поставщика данных

Указать путь к базе данных и проверить соединение.

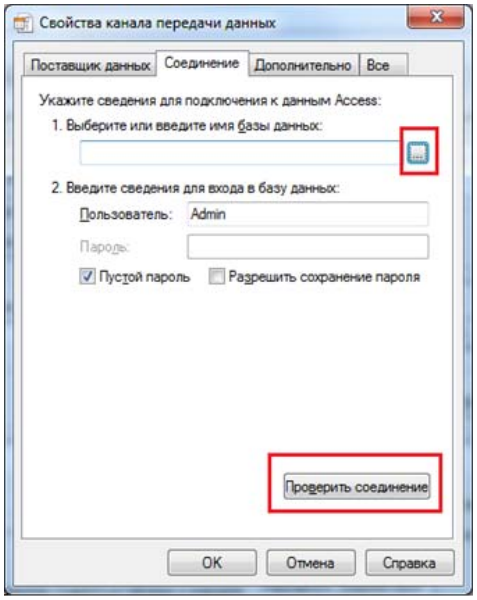


Рисунок 11. Выбор файла базы данных

Применить все изменения и поменять значение свойства Connected на true.

Важное примечание: Строка подключения представляет собой обычную строку, в которой перечислены параметры подключения проекта к базе данных.

Пример строки подключения:

Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\ALL_WORK\ПРИКЛАДНАЯ ИНФОРМАТИКА\Прикладная информатика\Лабораторные работы\5-БД\Студенты.mdb;Persist Security Info=False

Как видно из примера, строка подключения содержит путь к базе данных, который при необходимости можно заменять программно. Это необходимо для подключения различных баз данных одного типа к проекте.

На этом настройка компонента ADOConnection1 закончена.

2. Выделить объект ADOTable и в окне «Object Inspector» в поле Connection выбрать объект ADOConnection1, а в поле TableName выбрать таблицу из базы данных. Если при выборе таблицы возникает, то соединение с базой данных не было установлено. В этом случае следует проверить строку подключения в объекте ADOConnection1. В самую последнюю очередь установить переключатель Active в положение true.

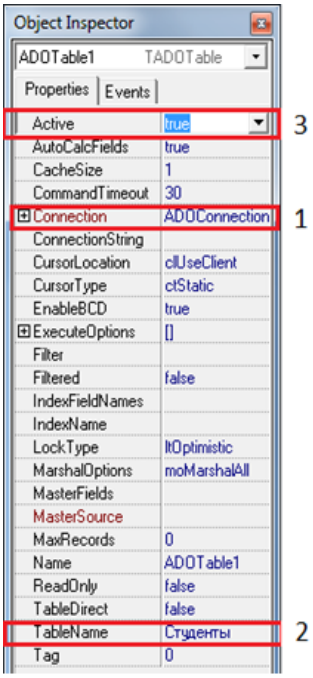


Рисунок 12. Настройка объекта ADOTable1

3. Выделить объект DataSource1 и в списке свойств в поле DataSet выбрать объект ADOTable1.

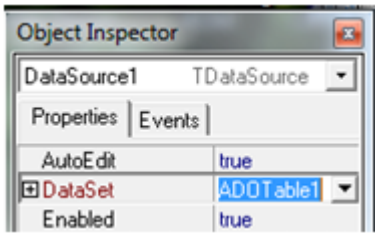


Рисунок 13. Настройка объекта DataSource1

4. Для объектов DBGrid и DBNavigator в поле свойства DataSource выбрать объект DataSource1:

.

5. При правильном выполнении всех вышеперечисленных операций в объект DBGrid должна быть загружена таблица из базы данных (рис.14).

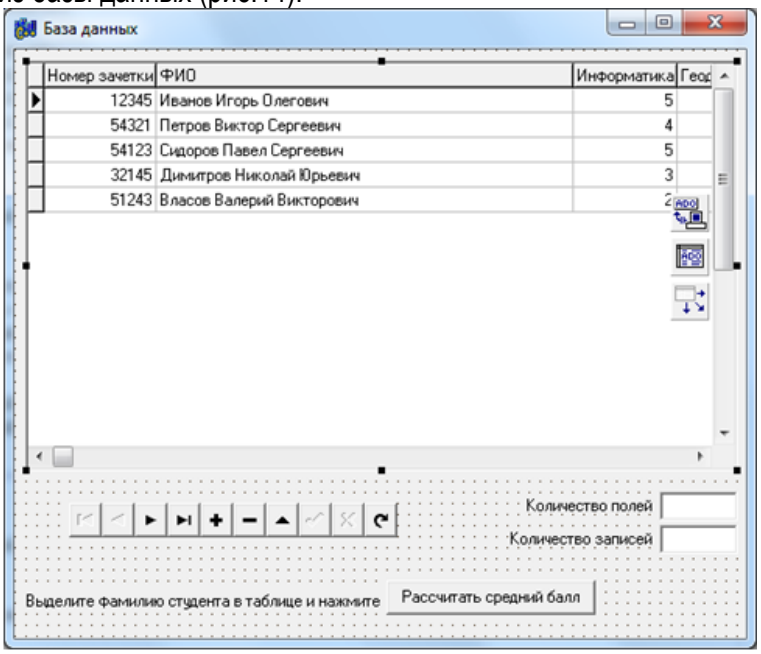


Рисунок 14. Загруженная таблица

Настройка подключения таблицы базы данных к проекте завершена. Теперь необходимо написать код для расчета количества полей и записей таблицы для последующего их вывода в объекты Edit1 и Edit2.

16) Перерасчет количества полей и записей таблицы необходимо производить каждый раз при запуске проекта, при добавлении или удалении записей, то есть при каждом изменении данных в таблице.

При изменении данных в таблице генерируется событие OnDataChange объекта DataSource. Для обработки этого события необходимо выделить объект DataSource1 и в списке событий Events дважды щелкнуть левой кнопкой мыши по полю OnDataChange. В созданной заготовке функции следует написать следующий программный код:

```
void __fastcall TForm1::DataSource1DataChange(TObject *Sender,
      TField *Field)
{
Edit1->Text=ADOTable1->FieldCount;    //определяем количество полей (колонок)
                                      //таблицы и записываем в Edit1

Edit2->Text=ADOTable1->RecordCount;   //определяем количество записей (строк)
                                      //таблицы и записываем в Edit2
}
```

17) Подсчет среднего балла студента должен происходить по нажатию кнопки «Рассчитать средний балл». Для обработки нажатия этой кнопки дважды щелкнуть по ней и в заготовке функции написать следующий код:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
int i;      //переменная для цикла
int kol;    // переменная для подсчета количества колонок
double sum,sredn;  //переменные для суммы баллов и среднего балла
sum=0;
kol=0;
for (i=2;i<ADOTable1->FieldCount;i++)    //цикл "Перемещаться по столбцам
                                         //начиная со второго и заканчивая
                                         //последним в таблице"
    {               //начало тела цикла
    sum=sum+ADOTable1->Fields->Fields[i]->AsInteger; //суммируем баллы в колонках
                                                     //текущей (выделенной) записи
    kol=kol+1;           //подсчитываем количество колонок
    }               //конец тела цикла
sredn=sum/kol;      //подсчет среднего балла
//вывод на экран сообщения с фамилией студента и его средним баллом
ShowMessage(ADOTable1->Fields->Fields[1]->AsString+": "+FloatToStr(sredn));
}
```

18) Сохранить проект нажатием кнопки на панели инструментов.

19) Провести отладку и тестирование проекта

20) Изучить назначение кнопок объекта DBNavigator1 самостоятельно.

21) Отредактировать таблицу средствами проекта. Отредактировать уже существующие записи в таблице, добавить 4 и удалить 2 записи.

Для защиты проекта необходимо:

1) Иметь рабочий вариант проекта

2) Знать используемые в проекте свойства компонентов Label, Button, ComboBox, Edit, ADOConnection, ADOTable, DataSource, DBGrid, DBNavigator, DBGrid, DBNavigator и уметь их использовать.

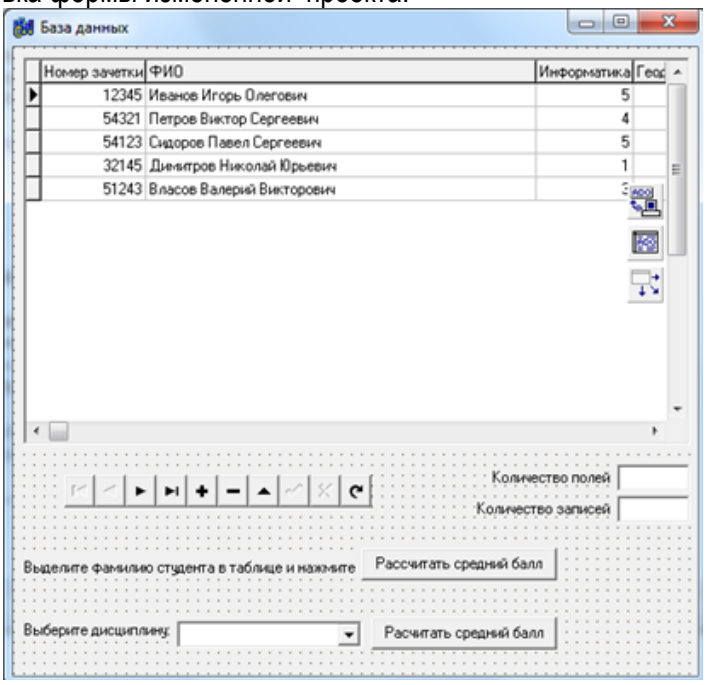
3) Ориентироваться в программном коде и знать все операторы, используемые в проекте

4) Выполнить самостоятельную часть лабораторной работы.

Задача для самостоятельной работы №6

Задание: Дополнить проект «База данных». По желанию пользователя проект должна подсчитывать средний балл по выбранной дисциплине.

Примерная компоновка формы измененной проекта:



Справка:

- Переход по записям (строкам) в базе данных в прямом направлении осуществляется командой

```
ADOTable1->Next();
```

- Для установки на первую запись предназначена команда

```
ADOTable1->First();
```

- Объявить цикл перехода по записям таблицы можно двумя способами:

```
for (i=0;i<ADOTable1->RecordCount;i++)
```

Или

```
while (!ADOTable1->Eof)
```

- Получение целого числа из поля по его имени:

```
ADOTable1->Fields->FieldByName(s)->AsInteger;
```

- Получение целого числа из поля по его индексу:

```
ADOTable1->Fields->Fields[i]->AsInteger;
```

- Заполнение объекта ComboBox1 списком имен полей из таблицы:

```
ComboBox1->Items=ADOTable1->FieldList;
```

Лабораторная работа

Составить проект **Базы данных**

В состав C++Builder включены компоненты, поддерживающие различные технологии доступа к данным.

Общие замечания

- Компоненты BDE для доступа к данным используют процессор баз данных Borland Database Engine.
- Компоненты ADO для доступа к данным используют ActiveX-компоненты (библиотеки) Microsoft.
- Для того чтобы программа, которая для доступа к данным использует BDE-компоненты, могла работать с базой данных, на компьютере должен быть установлен процессор баз данных - Borland Database Engine (BDE). BDE устанавливается на компьютер

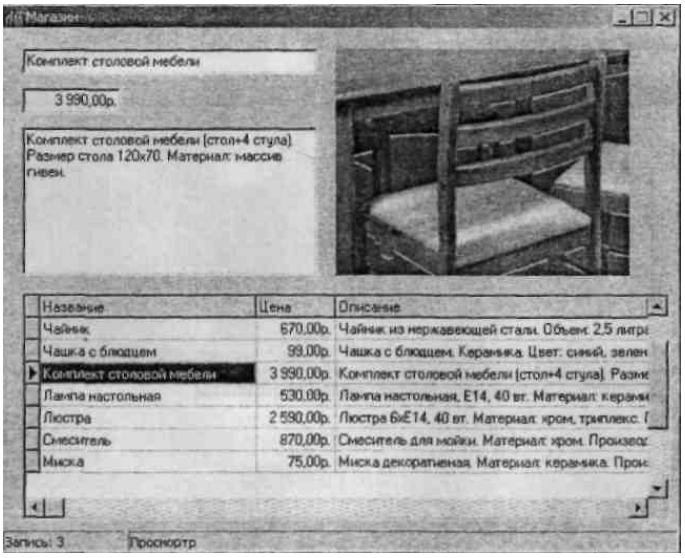
программиста в процессе инсталляции C++Builder.

- База данных, для доступа к которой используются BDE компоненты, должна быть зарегистрирована в системе. Зарегистрировать базу данных, создать псевдоним (Alias) можно при помощи утилиты BDE Administrator.
- Создать базу данных (таблицу) и наполнить ее информацией можно при помощи утилиты Database Desktop или SQL Explorer. Перед тем как приступить к созданию таблицы данных надо создать псевдоним (Alias) базы данных.
- Для того чтобы перенести программу работы с базой данных на другой компьютер, надо создать установочный CD. Для решения этой задачи Borland рекомендует использовать утилиту InstallShield Express, которая поставляется вместе с C++Builder.

Лабораторная работа по теме № 9.
Проектирование приложений для работы с базами данных.
Приложение « Магазин»

Разработать приложение **Магазин**

База данных "Магазин" должна состоять из таблицы stock.db и содержать информацию о товарах.

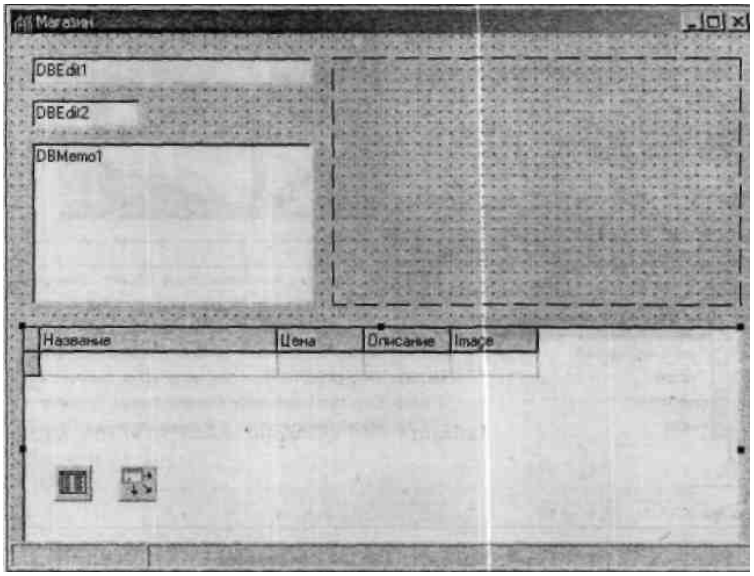


Поля таблицы stock (stock.db)

Поле	Тип	Размер	Комментарий
Title	A (Alpha)	50	Название товара
Price	$ (Money)	-	Цена
Memo	A (Alpha)	100	Описание товара
Image	A (Alpha)	30	Файл иллюстрации (в формате BMP)

Для доступа к базе данных используйте псевдоним slock. Создать псевдоним можно при помощи утилиты BDE Administrator.

Форма программы **Магазин** приведена на рисунке, значения свойств компонентов установите приведенные в следующих таблицах



Форма программы работы с базой данных **Магазин**

Значения свойств компонента Table1

Свойство	**Значение**	**Комментарий**
DatabaseName	stock	Псевдоним базы данных
TableName	stock.db	Файл, в котором находится таблица

Значения свойств компонента DataSource1

Свойство	Значение
DataSet	Table1

Значения свойства компонента DBGrid1

Свойство	**Значение**
DataSource	DataSource1
Columns[0].FieldName	Title
Columna[0].Title.Caption	Название
Columns[l].FieldName	Price
Columns[l].Title.Caption	Цена
Columns[2].FieldName	Memo
Columns[2].Title.Caption	Описание
Columns[3].FieldName	Image
Columns[3].Title.Caption	Image

*Значения свойств компонентовDBEdi*t и *DBMemo*

Свойство	**Значение**
DBEditl.DataSource	DataSource1
DBEditl.DataField	Title
DBEdit2.DataSource	DataSource1
DBEdit2.DataField	Price
DBMemol.DataSource	DataSource1
DBMemol.DataField	Memo

Обработчик FormShow

```
void _fastcall TForm1::FormShow(TObject *Sender)
 {
     try
      {
          Table1->Open();
```

```
    }
    catch ( EDBEngineError &e)
    {
ShowMessage("Для доступа к базе данных надо создать псевдоним stack");
}
}
```

Обработчик DataSource1StateChange - изменение состояния набора данных

```
void _fastcall TForm1::DataSource1StateChange(TObject *Sender)
{
    if ( DataSource1->State == dsBrowse)
        StatusBar1->Panels->Items[l]->Text = "Просмортр";
        else
        StatusBar1->Panels->Items[l]->Text = "Редактирование";
        }
```

Обработчик события AfterScroll - возникает после перехода к другой записи

```
void _fastcall TForm1::.Table1AfterScroll(TDataSet *DataSet)
 {
    AnsiString Picture;
    if ( Table1->RecNo!= -1)
    (
        StatusBar1->Panels->ltems[0]->Text ="Запись: " + IntToStr(Table1->RecNo );
        /* Доступ к значению поля текущей записи можно
        получить через свойство FieldValue.  Если поле Image пустое, то при попытке
        чтения из него данных возникает ошибка. */
        try
        {
            Picture =Table1->Database->Directory +DataSet->FieldValues["Image"];
            }
        catch (EVariantTypeCastError &e)
        {
        Image1->Visible = false;
        return;
        }
        ShowPhoto(Picture) ;
        }
    else
    {
StatusBar1->Panels->Items[0]->Text =  "";
StatusBar1->Panels->Items[1]->Text = "Новая запись";
Image1->Visible = false;
}
}
```

Обработчик ShowPhoto отображает картинку в поле компонента Image1

```
void _fastcall TForm1::ShowPhoto(AnsiString Picture)
 {
    try
    {
        Image1->Picture->LoadFromFile(Picture);
         }
        catch ( EFOpenError &e)
```

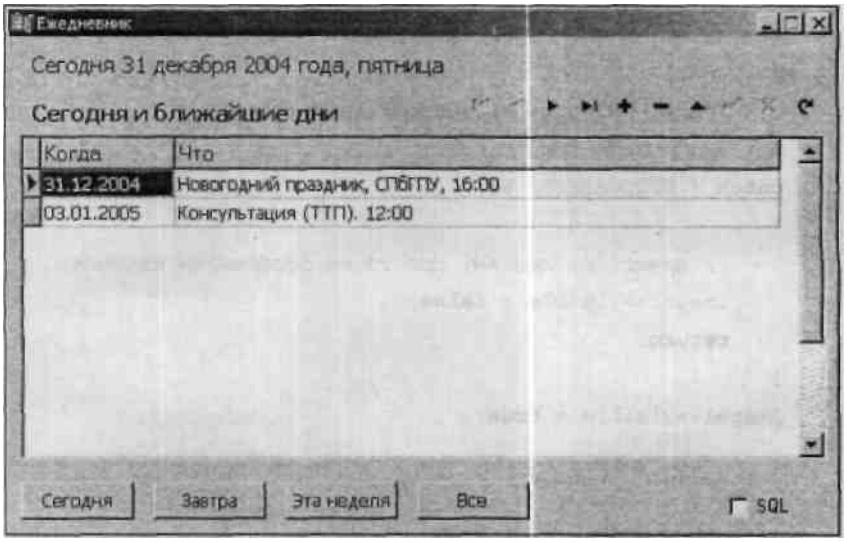
```
    {
        Image1->Visible = false;
        Return;
    }
    Image1->Visible = true;
}
```

Обработчик FormClose завершение работы программы

```
void _fastcall TForm1::FormClose(TObject *Sender,TCloseAction &Action)
{
    if (Table1->state == dsEdit  )          // таблица в режиме редактирования
    Table1~>Post0;                          // сохранить внесенные изменения
}
```

На самостоятельную работу
Разработать проект «Ежедневник»

В приложении «Ежедневник» нужно реализовать использование компонентов ADO для доступа к базе данных формата Microsoft Access.
База данных должна содержать информацию о запланированных мероприятиях (дата, задача). В приложении нужно предусмотреть возможность вносить в базу данных изменения (добавлять, удалять и редактировать записи), а также обеспечивать выбор информации по запросу - вывод список мероприятий, запланированных "на сегодня", "на завтра" и "на эту неделю". При запуске приложение должно автоматически выводить список мероприятий, запланированных "на сегодня" или, если запущена в пятницу, субботу или воскресенье, "на сегодня и ближайшие дни".



Лабораторная работа

Работа с базами данных в Borland C++ Builder

Цель работы

Усвоение основных алгоритмов доступа к базе данных MS Access средствами Borland C++ Builder.

1. Создание базы данных данными средствами Microsoft Access

Что такое база данных

База данных представляет собой компьютерный аналог организованной информации. Обычно элементы информации объединяет общая тема или назначение, как, например, прайс-лист магазина средств связи, приведенный ниже:

Товары				
Код товара	**Марка**	**На складе**	**Заказано**	**Цена**
1	SonyEricsson W880i	1	200	16 470,00р.
2	SonyEricsson W700i	1	30	7 590,00р.
3	SonyEricsson W300i	0	80	6 190,00р.
4	Nokia 6111	0	100	6 690,00р.
5	Nokia 6131	1	80	7 020,00р.
6	Samsung E480	0	20	6 580,00р.

Список организован в виде столбцов и строк, называемых полями и записями. Каждому товару соответствует отдельная запись, а каждое поле содержит определенную характеристику товара: название марки, наличие товара на складе, количество заказанных товаров данной марки, цену и тому подобное.

Внешне база данных, которая содержит только одну таблицу, похожа на обычный список, представленный на бумаге. Но поскольку информация хранится в электронном формате, ее можно сортировать и отображать различными способами, используя с максимальным эффектом.

Простые программы, которые хранят данные только в одной таблице (такие как Database, компонент Microsoft Work), часто называют плоскими базами данных. Более сложные программы (типа Microsoft Access) хранят информацию в нескольких связанных (related) между собой таблицах и поэтому называются реляционными базами данных. При правильной организации информации все таблицы можно трактовать как единую область памяти и извлекать из них данные в соответствии с возникающими потребностями.

Таблицы играют ключевую роль в базах данных, поскольку именно в них хранится информация. База данных может содержать тысячи таблиц, размеры которых ограничиваются только доступным пространством на жестком диске компьютера.

В табличном режиме содержимое таблицы отображается в виде столбцов (полей) и строк (записей), как показано ниже.

Таблицы представляют собой один из типов объектов, входящих в базу данных Access. На следующем рисунке представлено окно базы данных, где перечислены все типы объектов.

Из всех типов объектов только таблицы предназначены для хранения информации. Остальные используются для просмотра, редактирования, обработки и анализа данных - иначе говоря, для обеспечения эффективного доступа к информации.

Создание таблиц простейшим способом

1. Запустите приложение Microsoft Access: Программы-> Microsoft Office -> Microsoft Office Access.
2. Создайте новую базу данных Файл->Создать->Новая база данных. Сохраните ее под именем local.mdb.
3. В окне базы данных выберите Создание таблицы с помощью мастера.
4. В окне Создание таблиц в поле Образцы таблиц выберите Товары. С помощью кнопки > перенесите образцы полей КодТовара, Марка, НаСкладе, Заказано и Цена в Поля новой таблице. Нажмите Далее.

5. В следующем окне оставьте имя новой таблицы Товары. Убедитесь, что отмечен элемент Microsoft Access автоматически определяет ключ. Нажмите Далее.
6. В следующем окне убедитесь, что отмечен элемент Ввести данные непосредственно в таблицу. Нажмите Готово.
7. Заполните таблицу Товары произвольными данными. Поле Код товара имеет тип Счетчик и заполняется последовательно и автоматически. Сохраните таблицу после заполнения.

2. Работа с базами данных в Borland C++ Builder

Используя Borland C++ Builder, можно создать приложения, работающие как с однопользовательскими базами данных (БД), так и с серверными СУБД, такими как Oracle, Sybase, Informix, Interbase, MS SQL Server, DB2, а также с ODBC-источниками.

Набор данных в C++ Builder - это объект, состоящий из набора записей, каждая из которых, в свою очередь, состоит из полей, и указателя текущей записи. Набор данных может иметь полное соответствие с реально существующей таблицей или быть результатом запроса, он может быть частью таблицы или объединять между собой несколько таблиц.

В первых версиях C++Builderосновой работы с базами данных являлсяBorlandDatabaseEngine– процессор баз данных фирмыBorland.

Ключевой механизм BDE (Borland Database Engine), обеспечивающий работу визуальных компонент баз данных, действует как интерфейс между вашим приложением и самой базой данных. BDE реализован в виде набора системных DLL файлов. Взаимодействие компонентных объектов с BDE никак не специфицирует конкретную базу данных и не зависит от реализации обмена информацией на нижнем уровне иерархии. Именно BDE обращается в свою очередь к драйверам, специфическим для базы данных указанного типа, возвращая вашему приложению запрошенные фактические данные. BDE играет роль, аналогичную контроллеру драйверов ODBC (Open Database Connectivity) производства фирмы Microsoft, изолируя приложения от нижнего уровня взаимодействия с базой данных и увеличивая общую производительность связи за счет использования кэш-памяти. Используя BDE, вы получаете доступ ко всем локальным стандартным базам данных вашего компьютера, к источникам данных ODBC и к SQL серверам баз данных в архитектуре сетевой связи клиент/сервер.

Унифицированная технология BDE применяется во всех продуктах производства корпорации Borland: C++Builder, Borland C++, Delphi, IntraBuilder и JBuilder. Чтобы получить доступ к содержимому базы данных, приложению необходимо знать только идентификатор ее псевдонима (alias).

Использование BDEне теряет своей актуальности. Но начиная сC++ Builder 5, в библиотеке компонентов появились альтернативные механизмы связи с данными, что связано с ориентацией на работу с разными платформами. Такой дополнительной

возможностью является разработанная в Microsoft технология ActiveX Data Object (ADO) – пользовательский интерфейс к любым типам данных: различным базам данных, электронной почте, системным, текстовым и графическим файлам. Связь с данными осуществляется посредством технологии OLE DB.

Доступ к базам данных через ActiveXDataObject

Задание соединения компонентов ADOс базой данных

1. Запустите Borland C++ Builder. Перенесите на форму следующие компоненты:

DBGrid (Data Controls), ADOTable (ADO), DataSource (Data Access), Button (Standard), OpenDialog (Dialogs).

Компонент DBGrid обеспечивает табличный способ отображения на экране строк данных из компонентов Table или Query. Приложение может использовать DBGrid для отображения, вставки, уничтожения, редактирования данных БД.

Компонент DataSource действует как посредник между компонентами DataSet (Table, Query, StoredProc) и компонентами Data Controls - элементами управления, обеспечивающими представление данных на форме.

Компонент OpenDialog является методом реализации стандартного диалога открытия файлов.

2. Доступ к базе данных осуществляется с помощью строки соединения – свойства ConnectionStringкомпонентаADOTable. Нажмите на кнопку с многоточием возле этого свойства в инспекторе объектов. Откроется окно, показанное на рисунке.

Верхняя радиокнопка UseDataLinkFilепозволяет использовать файл связи .udl. Нижняя радиокнопкаUseConnectionStringпозволяет в режиме диалога сформировать строку соединения. Отметьте эту радиокнопку и нажмите кнопкуBuild…

На вкладке Поставщик данных окна Свойства связи с данными вы должны указать провайдер OLE DB, который собираетесь использовать для доступа к данным. Выберите Microsoft Jet 4.0 OLE DB Provider. Нажмите Далее.

На вкладке Подключение в окне Выберите или введите имя базы данных укажите путь к базе local.mdb. Нажмите кнопку Проверить подключение и убедитесь в успешном соединении с базой.

3. Задайте следующие значения свойств и событий компонентов формы в инспекторе объектов:

- DBGrid: Events-Data Sourse-DataSourse1;
- ADOTable: Table Name-Товары;
- DataSourse: DataSet-ADOTable1; Events-DataSet-ADOTable1.

4. Дважды нажмите на компонент Button. В обработчике кода впишите следующий код для события Button1Click:

```
const String ConnStr = "Provider=%s;Data Source=%s;Mode=%s";
```

```
if (Form1->OpenDialog1->Execute())

{

ADOTable1->ConnectionString = Format (ConnStr, ARRAYOFCONST(("Microsoft.Jet.OLEDB.4.0",(String)Form1->OpenDialog1->FileName, "Read")));

DBGrid1->DataSource=DataSource1;

DataSource1->DataSet=ADOTable1;

ADOTable1->Active=true;

}
```

5. Запустите приложение (F9). При нажатии на кнопку задается возможность указания пути к базе данных, после чего записи таблицы Товары загружаются вDBGrid.

Обработка записей базы данных

Создание новой записи в таблице

1. Добавьте на форму проекта компонент MainMenu(Standard). Нажмите на компонент правой кнопкой мыши и выберите в контекстном меню Дизайнер меню… Добавьте следующие пункты меню: File-> Open, New, Delete, Exit: Help-> Contents, About….
2. Создайте новую форму в проекте: Файл-> Новый-> Form. Разместите на ней четыре компонентаEdit, четыре соответствующих компонентаLabeli компонентButton.

3. На первой форме в обработчике пункта главного меню Newнапишите следующий код:

```
void __fastcall TForm1::New1Click(TObject *Sender)

{

Form2->Show();

}
```

На второй форме Form2 в обработчике кода для кнопки напишите следующий код:

```
void __fastcall TForm2::Button1Click(TObject *Sender)

{

const String ConnStr = "Provider=%s;Data Source=%s;Mode=%s";

Form1->ADOTable1->Active=true;

Form1->ADOTable1->Fields->Fields[0]->ReadOnly=false;

Form1->ADOTable1->Fields->Fields[1]->ReadOnly=false;
```

```
Form1->ADOTable1->Fields->Fields[2]->ReadOnly=false;

Form1->ADOTable1->Fields->Fields[3]->ReadOnly=false;

Form1->ADOTable1->Fields->Fields[4]->ReadOnly=false;

Form1->ADOTable1->Insert();

Form1->ADOTable1->Fields->Fields[0]->Value=Form1->ADOTable1->RecordCount+1;

Form1->ADOTable1->Fields->Fields[1]->Value=Form2->Edit1->Text;

Form1->ADOTable1->Fields->Fields[2]->Value=StrToInt(Form2->Edit2->Text);

Form1->ADOTable1->Fields->Fields[3]->Value=StrToInt(Form2->Edit3->Text);

Form1->ADOTable1->Fields->Fields[4]->Value=StrToCurr(Form2->Edit4->Text);

Form1->ADOTable1->UpdateBatch();

Form1->ADOTable1->Refresh();

Form1->ADOTable1->Active=false;

}
```

В результате выбор пункта главного меню формы 1 инициирует открытие второй формы, предназначенной для ввода новых данных в таблицу.

Задание на лабораторную работу

На основе приведенного примера написать приложение, реализующее доступ к базе данных MSAccessсредствамиBCB6.0сиспользованием понятий выбранной предметной области.

Примечание

В процессе выполнения курсовой работы предполагается реализация просмотра текущей записи в отдельной форме и удаления выбранной записи.

Лабораторная работа № 6

Проектирование справочной системы

Цель работы: изучение методики создания файлов справочной системыWindows(*.hlp) при разработке приложений.

Краткие теоретические сведения.

Разработка справочной системы состоит из двух основных этапов:

- Создание файлов или нескольких файлов, содержащих темы справок, например, с помощью Microsoft Word.
- Компиляция справки в файл и отладка всей справочной системы, с помощью специальных программ, например, HCRTF-MicrosoftHelpWorkshop(C:\Program Files\Borland\CBuilder6\Help\Tools\hcw.exe).

При создании справочной системы необходимо в первую очередь продумать систему в целом. А именно решить следующие вопросы:

- об информации, выносимой в основное, дополнительное, всплывающие окна;
- о включении разделов в предметный указатель и окно содержания;
- о представлении информации в основном окне (например, наличие начальной части, неподдающейся прокрутке) и общем стиле справки (наличие кнопок, пиктограмм, "горячих" областей);
- о разделении функций между основным и дополнительными окнами.

При создании файлов тем справок можно использовать текстовый редактор MicrosoftWord, созданный файл должен сохраняться в форматеRTF.

Каждая тема (кадр) должна начинаться с новой страницы (разрыв страницы выполняется нажатием клавиш Ctrl-Enter). Первой должна располагаться страницаСодержание, порядок остальных безразличен.

При написании темы можно использовать многие возможности оформления шрифта. Но если нет уверенности, что соответствующие шрифты будут иметься на компьютерах потенциальных пользователей, лучше использовать обычные системные шрифты (MSSansSerif).

По умолчанию при отображении текста в окне справки часть строки, которая не помещается в слишком узком для нее пространстве, переносится на новую строку. В случаях, когда это нежелательно, следует выделить соответствующие строки, и выполнить команду Формат│Абзаци в открывшемся окне на страницеПоложение на страницеустановить опциюНе разрывать абзац. Тогда, если ширины справки не хватает, чтобы отобразить всю длину отмеченных таким образом строк, в окне появится полоса горизонтальной прокрутки.

Для того чтобы при перемещении по тексту темы какая-то его часть (заголовок, начальная часть) не прокручивалась вертикальной прокруткой и всегда оставалась на экране, следует выделить соответствующий текст, выполнить команду Формат│Абзаци в

открывшемся окне на страницеПоложение на страницеустановить опциюНе отрывать от следующего.

В кадр могут специальным образом включаться рисунки, кнопки и т.д.

Добавлять изображения в тему можно, например, в виде файлов .bmp, пользуясь или буфером обмена, или одной из команд:

- {bmc<имя файла>}- размещение рисунка как обычного символа по ходу текста;
- {bml<имя файла>} – размещение рисунка с левой стороны;
- {bmr<имя файла>} – размещение рисунка с правой стороны.

Темы могут содержать горячие области: выделенные слова или кнопки, позволяющие пользователю выполнять переход от одной темы к другой. Кнопка вставляется командой {button<надпись>,<список макросов>}. В этой команде указывается текст надписи, которая будет присутствовать на изображаемой кнопке, а также перечень макросов, выполняемых при ее нажатии. Рассмотрение всего множества макросов, которые можно использовать при проектировании справочной системы выходит за рамки данной лабораторной работы. Описание этих макросов можно получить в файлehcw.hlp.

Каждая тема снабжается сносками (команда Вставка|Сноска; далее в диалоге свойств сноски надо выбрать тип нумерации «Другая» и в поле для ввода ввести один из символов), для которых используются определенные символы. Все сноски располагаются в первых позициях кадра. Символы, используемые для сносок различного типа приведены в таблице 1, там же раскрыто и назначение сносок каждого типа.

Таблица 1. Используемые символы при создании файла справки

Символ сноски	Назначение сноски
#	Обозначает уникальный идентификатор темы. По нему на данную тему могут ссылаться другие темы. Этому идентификатору ставится в соответствие номер, по которому на данную тему ссылается использующее справку приложение.
K	Позволяет отображать тематику кадра в предметном указателе. Название темы в предметном указателе представляет собой текст сноски K. Для одного кадра можно ввести несколько обозначений, разделяемых точкой с запятой. Пример KОбъект Поле; Объекты В предметном указателе две строки «Объект Поле» и «Объекты» будут вызывать один и тот же кадр. Можно сформировать двух уровневые ссылки: темы первого и второго уровня разделяются запятой. Пример KОбъект Поле, Создание; Объект Поле, Ввод данных; Объект Поле, Модификация свойств Кроме того, элементы, указанные в сносках K, используются при организации переходов между темами переход по ключевым словам (с помощью макросаKlink)
$	Определяет заголовок данной темы, используется в некоторых режимах работы, например, Поиск,Назади др. Например, если указанное пользователем

	ключевое слово соответствует нескольким темам, то от пользователя требуется уточнение. В этом случае появляется окноНайденные разделы.В нем отображается текст сносок $. Поэтому сноски $ включают только в те кадры, которые имеют в своих сноскахKэлементыKlink, используемые в сноскахKдругих кадров.
A	Используется для организации перехода по ключевым словам (с помощью макроса Alink)
+	Используется для указания последовательности просмотра тем. Включаются в кадр только тогда, когда соответствующие кнопки (кнопки Browse) предусмотрены в окне справки. Если сноски + включены в кадр, но их значения не указаны, последовательность просмотра тем будет установлена автоматически в соответствие с последовательностью тем в файле. Тексты сносок могут быть номерами или идентификаторами.
!	Используется для указания макросов, которые должны сработать перед появлением окна темы (для сложных справочных систем).
*	Используется для указания связанных друг с другом тем, которые должны быть встроены в справку (при создании справочных систем связанных программных продуктов).
>	Используется для указания идентификатора окна, в котором должна отображаться тема

В темы можно вводить переходы от одной темы к другой. Существуют непосредственные переходы и переходы по ключевым словам.

Их создание заключается в выделении слова-ссылки двойной линией, и сразу после него, без пробелов записать название ссылки раздела к которому будет осуществлен переход. Причем название ссылки должно быть оформлено как скрытый текст. В справочной системе ссылка будет подчеркнута одной линией. Далее при простом просмотре такой текст не будет виден и если вам в будущем понадобится его изменить, то в этом случае необходимо включить опцию отображения скрытых символов () на панели инструментов Word.

Непосредственный переход выполняется при щелчке пользователя на горячей области текста. Для этого сразу после нужных слов (без пробела) надо написать идентификатор темы, на которую нужно перейти. Соответствующие слова следует выделить двойным подчеркиванием (команда Формат|Шрифт, диалоговое окноШрифт, опцияПодчеркиваниев положенииДвойное), а идентификатор темы оформить как скрытый текст (командаФормат|Шрифт, диалоговое окноШрифт, разделЭффекты, опцияСкрытыйустановлена, опцияПодчеркиваниев положении "(нет)").

Можно отображать тему, на которую осуществляется переход во всплывающем окне. Такие окна обычно используются для дополнительной информации и различных пояснений. Всплывающее окно не удаляет окно темы его вызвавшей, но при любом действии пользователя исчезает. Переход к теме, отображаемой во всплывающем окне, осуществляется аналогично, но с выделением одинарным подчеркиванием.

Переходы по ключевым словам позволяют осуществлять два макроса, имеющие одинаковый синтаксис KlinkиAlink. Например,

Klink("<список ключевых слов>", <тип>, "<идентификатор темы>",

<имя окна>)

Обязательным элементом вызова макроса является только первый - "<список ключевых слов>". Он представляет собой одно или несколько ключевых слов или словосочетаний, перечисленных через точку с запятой. Если входящее в этот перечень словосочетание содержит запятую, то весь список заключается в двойные кавычки. Поиск ведется по первому слову, при нахождении нескольких тем пользователю предъявляется окно Найденные разделы, если не найдено ни одной темы, начинается поиск по второму ключевому слову и т.д.

<тип> определяет реакцию на найденные или не найденные ключевые слова и может принимать значения, представленные в таблице 2.

Таблица 2. Возможные значения событий для работы с ключевыми словами

Символическое значение	Численное значение	Описание значения
JUMP	1	Если найдена только одна тема, соответствующая ключевым словам, то на нее сразу осуществляется переход
TITLE	2	Если ключевое слово находится более чем в одном файле справки (при справке, состоящей из нескольких файлов), то в окне Найденные разделы после названия темы пишется имя файла, определенное в файле содержания (*.cnt)
TEST	4	Возвращается величина указывающая нашлось или нет хотя бы одно соответствие ключевым словам.

<идентификатор темы> определяет, что если не найдено соответствия ключевым словам, то появляется всплывающее окно с текстом, содержащимся в теме, на которую указывает данный идентификатор. Если идентификатор не задан, то при безуспешном поиске появляется окно с текстом «Дополнительные сведения отсутствуют».

<имя окна> задает окно для отображения. Если этот параметр не задан, то используется окно, заданное в кадре темы или окно по умолчанию.

В качестве примера использования макросов можно рассмотреть следующий текст

Объект Поле!Klink(Объект Поле; Объекты) позволяет осуществить ввод данных.

В готовом файле справки словосочетание «Объект Поле» будет выделено цветом. Щелкнув клавишей мыши на этом словосочетании, пользователь увидит список тем, содержащих в своих сносках Кключевые слова: «Объект Поле» и «Объекты». Если такая тема является единственной, то сразу осуществится переход на нее.

В тексте может быть использован оператор вида

{buttonОбъекты,Alink(Объекты)}

Это приведет к появлению в кадре кнопки с надписью «Объекты», при нажатии на которую будет осуществляться поиск тем, у которых ключевое слово присутствует в сносках A.

Конец каждого раздела должен быть оформлен в виде <Разрыва страницы> через пункт меню <Вставка->Разрыв…> или через нажатие Ctrl+Enter.

Когда сформированы темы проектируемой справочной системы, разработана ее структура, решен вопрос об общем стиле справки, переходят ко второму этапу проектирования – компиляции справки в файл *.hlp.

Компиляция и отладка справочной системы может проводиться с использованием программы HCRTF-MicrosoftHelpWorkshop. Эта программа позволяет создать файл Проекта справки, без которого ее нельзя компилировать, файл содержания справки, а также проверить справочную систему в работе.

Для создания файла проекта простой справочной системы нужно выполнить следующие действия:

1. Выполнить File|Newи в открывшемся окне выбрать опциюHelp Project.
2. В окне Project File Nameзадать имя и каталог файла Проекта справки (каталог выбрать тот, в котором лежит файл текстов тем .rtf).
3. В открывшемся окне Проекта нажать кнопку Files….
4. В появившемся окне Topic Files нажать кнопку Add… и выбрать среди файлов подготовленный файл текстов справки.Теперь, нажатием на кнопку Options в главном окне нашего проекта мы вызываем окно настроек нашей справки, где в первую очередь нам следует заполнить поле Default Topic записав туда название ссылки на раздел который будет отображаться по умолчанию при открытии справки. Тут же можно настроить и компрессию, указать язык, параметры полнотекстового поиска и т.д.

5. Следующим пунктом настройки проекта, является пункт Windows.Здесь можно настроить, размер, позицию, цвет и прочие атрибуты окна будущей справки.

6. Следующая кнопка Марвызывает окно настройки карты справки. Здесь необходимо проставить числовые идентификаторы разделов. С их помощью приложение будет вызывать тот или иной раздел справки в зависимости от настроек и состояния. В этом окне необходимо присвоить идентификаторы всем разделам справки заполняя названием ссылки и порядковым номером соответственно поляTopic IDиMapped numeric value.

7. Когда файл проекта создан, необходимо сохранить и откомпилировать его – кнопка Save and Compile, использовать ее следует каждый раз при изменении файла Проекта. После компиляции можно просмотреть сведения о результатах компиляции, и если она завершилась удачно (указан размер файла) просмотреть файл справочной системы. Просмотр и работа по отладке выполняется с помощью командыFile|Run WinHelp.
8. Остается только заполнить соответствующими данными раздел содержание нашей справки при помощи главного меню, пункт New-Help Contents>.В результате чего появится окно с загруженным в него файлом содержания. В верхнем левом поле ввода задается имя файла *.hlp, верхнем правом поле ввода задается заголовок, который должен появиться в окне при работе со справочной системой. Нижнее поле ввода заполняется с помощью кнопокAdd Above… иAdd Below…. Каждая его строка

соответствует либо заголовку (будет отображаться в виде закрытой книги), либо теме (отображается в виде листа с вопросом). Для заголовка задается только его текст, для темы задается название и идентификатор. Кнопки позволяют формировать многоуровневую структуру файла содержания. КнопкаEdit…позволяет редактировать выделенную строку, а кнопкаRemove…удалять. Для создания заголовков справки (отображаются в виде книжек) мы в диалоговом окнеAdd Below/Add Aboveследует выбрать пунктHeadingи заполнить полеTitlea при добавлении пунктов справки, выбрать пунктTopic, и внести название пункта и название ссылки в поляTitleиTopic IDсоответственно .

Файл содержания имеет расширение *.cnt.

9. Для подключения файла справки к приложению запустите C++Builder и начните новый проект. Выберите в меню команду Project - Optionsи в появившемся окне диалога выберите страницуApplication. Щелкните на кнопкеBrowse...справа от редактора строкиHelp fileи укажите HLP-файл, который вы только что создали. Закройте окно диалогаProject Options, щелкнув на кнопке OK.

10. Для того, чтобы открыть справочник программным способом, например по нажатию кнопки Helpили при выборе команды менюHelp | Help Topics, вызывается один из следующих методов объектаApplication:HelpCommand, HelpContext, HelpJump. В каждом приложении автоматически создается объектApplicationтипаTApplication— приложение. Этот компонент отсутствует в палитре библиотеки, вероятно, только потому, что он всегда один в приложении.Applicationимеет ряд свойств, методов, событий, характеризующих приложение в целом.

- Метод HelpContext: вызывает переход в файл справки на тему с идентификаторомContext.Это идентификатор, который при проектировании справки поставлен в соответствие некоторой теме.
- Метод HelpJump выполняет аналогичные действия, но его параметр JumpID — не идентификатор темы, а имя соответствующей темы в файле справки, задаваемое в нем сноской #.
- Метод HelpCommand позволяет выполнить указанную параметромCommandкомандуAPI WinHelpс параметромData. Метод генерирует событиеOnHelpактивной формы или приложения, а затем выполняет указанную командуWinHelp.
- Команда HELP_CONTENTSс параметром 0 отображает окноСодержание справки.
- Команда HELP_INDEXс параметром 0 отображает окно Указатель справки.
- Команда HELP_CONTEXTс параметром, равным идентификатору темы, отображает тему с заданным идентификатором (это тождественно рассмотренному ранее методуHelpContext).
- Команда HELP_CONTEXTPOPUPс параметром, равным идентификатору темы, делает то же самое, но отображает тему во всплывающем окне.

Полный список команд WinHelpвы можете найти в темеWinHelpсправочного файлаwin32.hlp, расположенного в каталоге ...\ProgramFiles\CommonFiles\BorlandShared\MSHelp.

Например, метод обработки команды меню Help | Help Topicsможет выглядеть так:

```
void _fastcall TForml::HelpTopicsItemClick(TObject *Sender)
```

```
{
Application->HelpCommand(HELP_CONTEXT,0);

	}
```

3 Задание на лабораторную работу

Спроектировать справочную систему для приложения, определенного вариантом задания. Тексты тем справки должны быть составлены грамотно и корректно. Темы справочной системы должны содержать графические элементы (пиктограммы, рисунки) и горячие области. В справочной системе должны быть предусмотрены непосредственные переходы и переходы от одной темы к другой по ключевым словам. Кроме того, необходимо обеспечить возможность поиска по ключевым словам.

4 Порядок выполнения работы

4.1 Создать глоссарий – перечень уникальных понятий, используемых в приложении и его интерфейсе. В качестве примера таковых могут выступать названия элементов меню, окон, режимов, текст командных кнопок и т.д. Работа над созданием глоссария требует контакта с целевой аудиторией. Это нужно, чтобы описания понятий не содержали многозначности при восприятии их потенциальными пользователями. Недопустимо наличие различных терминов для определения одного и того же понятия. Описание понятий должно отвечать промышленному руководству, соответствующему выбранной платформе (MSMicrosoft).

4.2 Добавить в глоссарий описания общей концепции приложения, ее функциональности в целом, а также отдельных функций, предоставляемых пользователю для выполнения (обзорная справка).

4.3 Включить в глоссарий изложение алгоритмов выполнения пользователем отдельных функций (процедурная справка).

4.4 Разработать структуру справочной системы, отражающую взаимосвязь отдельных элементов глоссария.

4.5 Определить разделы, которые должны быть включены в содержание и предметный указатель справочной системы.

4.6 Решить такие вопросы проектирования справочной системы как вид основного окна, введение изображений, наличие горячих областей и ссылок, организация переходов и т.д.

4.7 Сформировать файл тем справок в виде файла *.rtf.

4.8 Сформировать файл справочной системы *.hlp, создав файл Проекта справки и откомпилировав его средствами программыMSHelpWorkshop(HCRTF).

4.9 Используя те же средства создания справочной системы, сформировать файл содержания *.cnt.

4.10 Подключить проект справки к приложению, разработанному на лабораторной работе №5.

5 Требования к оформлению отчета

Отчет по лабораторной работе должен содержать:

- цель работы;
- описание приложения (вариант задания);
- структуру справочной системы;
- комментарии, поясняющие содержание обзорной, предметной и процедурной справок;
- текст файла тем справок в формате rtf(включая скрытый текст);
- обоснования применения использованных в файле сносок;
- выводы по работе.

При защите лабораторной работы необходимо представить электронные версии разработанных файлов *.hlpи *.cnt.

Лабораторная работа

Проектирование Интернет приложений в BC++ Builder

Лабораторная работа посвящена работе с базами данных в Интернет. В качестве базы данных используйте **dbP,** содержащую сведения о сотрудниках некоторой организации.

Просмотр в Web ***таблицы данных***

Для этого на странице Internet имеется компонент **DataSetTableProducer.**

1. Создайте новый сервер Web и в нем один объект действия «Default», и установите его свойство **Default** в **true.**

2. Перенесите в окно сервера **Web** компоненты **Table** и **DataSetTableProducer.** Набор данных **Table1** свяжите с таблицей **Pers** базы данных **dbP. C** помощью редактора полей установите свойства объектов полей, введите вычисляемое поле возраста **Age,** индексацию набора данных, фильтрацию и другие особенности отображения данных.

3. Свяжите компонент **DataSetTableProducer** через свойство **DataSet=** Table1 с набором данных **Table1**.

4. Свойство **Header** определяет команды HTML, которые должны располагаться в документе перед таблицей. Занесите в него команды:
<html>
<h1>Кадровый состав</h1>
<body>

5. Свойство **Footer** определяет команды HTML, которые должны следовать после таблицы. Занесите в него команды:

</body> </html>

6. Свойство **Caption** определяет заголовок таблицы. Свойство **MaxRows** устанавливает максимальное число строк таблицы. Если число записей окажется меньше, то число строк таблицы определится числом записей. Но если число записей в таблице больше, то на экран будет выдано только **MaxRows** первых записей.

7. Основное, что нужно сделать - указать поля набора данных, которые должны отображаться в таблице. Это можно сделать или быстрой кнопкой Add New - добавить поле**,** или кнопкой Add All Fields - добавить все поля**.** Эта кнопка доступна, только если при создании объектов полей в **Table1** использован Редактор полей. При этом все введенные в

Редакторе атрибуты отображения полей автоматически перенесутся в отображаемую таблицу. Если редактор полей не использован, доступна только кнопка Add New. Причем, при вводе очередного элемента в таблицу нужно выделить его в списке и в Инспекторе Объектов установить свойство **FieldName** - имя поля, а в подсвойстве **Caption** свойства **Title** задать надпись.

8. Установите с помощью индикаторов в левой панели общие свойства таблицы:

в окошке **Border** задайте сетку таблицы; в окошко **Width** определите ширину таблицы в процентах от ширины страницы. (Если установить 100%, то при просмотре в браузере ширина таблицы будет выбираться автоматически исходя из ширины окна браузера и будет изменяться при изменении ширины таблицы. Если задать **Width** < 100%, то ширина таблицы будет фиксированной).

Создайте обработчик события **OnAction:**

```
Table1.Active = true;
Response.Content = DataSetTableProducer1->Content;
Table1.Active = false;
```

Эти операторы включают и выключают соединение с базой данных и возвращают в качестве ответа серверного приложения результат, формируемый компонентом **DataSetTableProducer1.**

9. Сохраните проект под именем DB1, скомпилируйте его и перенесите модуль **.exe** в исполняемую папку сервера Web. При вызове модуля из браузера увидите страницу Web.

В Microsoft Internet Explorer посмотрите текст HTML, сгенерированный компонентом DataSetTableProducer:

```
<html>
<h1>Кадровый  состав</h1>
<body>
<Table Width="100%" Align="Left" Border=l>
<TR><TH>Отдел</TH> <TH>Фамилия</TH> <TH>Имя</TH> <TH>Отчество</TH>
<TH>г.р.</TH> <TH>Пол</TH> <TH>Возраст</TH> </TK>
<TK><TD>Бухгалтерия/TD> <TO>Антонова</TD> <TD>Антонина</TD>
<TD>Антоновна</TD> <TD>1955</TD> <TD>ж</TD> <TD>45</TD> </TR>
<TK><TD>Бухгалтерия/TD> <TO>Иванов</TD> <TD>Иван</TD>
<TD>Иванович</TD> <TD>1950</TD> <TD>м</TD> <TD>40</TD> </TR>
…
</Table> </body>
</html>
```

Как видно, сгенерированы обычные теги таблицы и в ее ячейки занесены значения соответствующих полей набора данных.

10. Для реализации и индексации по выбору пользователя внесите изменения в текст HTML, вводимый перед таблицей, т.е. задайте в свойстве Header компонента DataSetTableProducer1 следующий текст HTML:

```
<html>
<h1>Кадровый состав</h1>
<body>
<Table Border=1><Caption> <b>Упорядочить</Caption>
<TR><TD><A HREF="http: //mycomputer/cgi-bin/db2.exe/index?l">по алфавиту</A> </TD>
<TD><A HREF="http: //mycomputer/cgi-bin/db2 . exe/index?2 ">по отделам</A></TD></TR>
<TR><TD>A  HREF="http://mycomputer/cgi-bin/db2 .exe/index?3">по  г.р. </A></TD>
<TD><A  HREF="http: //mycoraputer/cgi-bin/db2 . exe/index?4">не
упорядочено</A></TD></TR>
 </Table>
<p>
```

Этот текст создает таблицу с заголовком «Упорядочить» и в ее ячейки заносит ссылки на действие с путем «/index» и с параметром, изменяющимся от 1 до 4, и тексты, сопровождающие ссылки: «по алфавиту», «по отделам», «по г.р.» и «не упорядочено».

11. Напишите обработчик действия, задающего свойство **IndexName** компонента Table1 в соответствии с выбором пользователя, т.е. надо создать действие, воспринимающее эти ссылки и упорядочивающее таблицу в модуле Web и задать свойству **PathInfo** значение «/index», а в обработчик события **OnAction** занесите код:

```
switch (StrToInt(Request->Query))
{
case 1: Table1->IndexName:= "fio";break;
case 2: Table1->IndexName = "depfio'"; break;
case 3: Table1->IndexName = "year'"; break;
case 4: Table1->IndexName = "'"; break;
}
Table1->Active = true;
Response->Content = DataSetTableProducer1->Content;
Table1->Active = false;
```

12. Сохраните проект под именемDB2 (это имя указано в приведенном выше тексте свойства **Header),** скомпилируйте и перенесите в исполняемую папку сервера Web.

13. По такому же принципу организуйте фильтрацию данных по различным критериям, значения которых задаются клиентом, используя свойство **Filter** набора данных и формируя на странице Web окна, в которые нужно ввести критерии фильтрации.

Работа с отдельными записями

Компонент **DataSetTableProducer** обеспечивает отображение всех записей таблицы, причем только тех полей, значения которых хранятся в самой таблице. Поля Memo или поля изображений отображаться в таблице не могут. Компонент **DataSetPageProducer** обеспечивает работу с шаблонами HTML. С набором данных он связывается свойством **DataSet** и автоматически заменяет шаблоны, имена которых совпадают с именами полей, на значения этих полей в текущей записи. **DataSetPageProducer** позволяет отображать поля отдельных записей, причем не только текстовые, но и, например, графические.

14. Предусмотрите в приложении просмотра таблицы пользователю возможность посмотреть подробную информацию о выбранном им сотруднике, включая его характеристику и фотографию. Для этого создайте новый модуль Web, перенесите на него компоненты **Table** и **DataSetTableProducer и н**астройте их как в пункте 3.

15. В окне редактора компонента **DataSetTableProducer** занесите в таблицу дополнительно еще одну колонку, в Инспекторе Объектов для этой колонки в подсвойстве **Caption** свойства **Title** задайте заголовок колонки **«Подробнее»**. В свойстве **FieldName** задайте для этой колонки какое-нибудь имя.

16. В ячейках колонки **Подробнее** сформируйте ссылки, обеспечивающие отображение страницы Web, содержащие подробные сведения о выбранном пользователем сотруднике. Для этого используйте обработчик события **OnFormatCell** компонента **DataSetTableProducer.** Это событие наступает при генерации компонентом содержимого каждой ячейки. Заголовок обработчика имеет вид:

void _fastcall TWebModule1::DataSetTableProducer1FormatCell(TObject *Sender, int CellRow, int CellColumn, THTMLBgColor &BgColor, THTMLAlign &Align, THTMLVAlign &VAlign, AnsiString &CustomAttrs, AnsiString &CellData)

Параметры **CellRow** и **CellColumn** определяют индексы соответственно строки и столбца, на пересечении которых находится формируемая ячейка. Строка с индексом 0 - это строка заголовков, индексы строк записей начинаются с 1. Параметр **BgColor** определяет цвет фона ячейки. Параметры **Align** и **VAlign** указывают выравнивание текста соответственно по

горизонтали и вертикали. Типы этих параметров определены следующими перечислениями:
enum THTMLAlign { haDefault, haLeft, haRight, haCenter };
enum THTMLVAlign { haVDefault, haTop, haMiddle, haBottom, haBaseline } ;

17. В параметре **CellData** занесите текст или команды HTML, которые должны размещаться в ячейке. В параметре **CustomAttrs** задайте дополнительные атрибуты отображения.

18. Реализуйте тело обработчика события **OnFormatCell**:

```
void _fastcall TWebModule1::DataSetTableProducer1FormatCell(TObject *Sender, int CellRow,
int CellColumn, THTMLBgColor &BgColor, THTMLAlign &Align, THTMLVAlign &SVAlign,
AnsiString &CustomAttrs, AnsiString &CellData)
{
if ((CellRow > 0)&& (CellColumn == 7))
CellData = "<a href=\"/cgi-bin/db3.exe/record?N=" + Table1Num->AsString +
"\">просмотр</a>";
}
```

Этот код должен занести в каждую ячейку восьмого столбца (индекс равен 7), кроме ячейки заголовка, ссылку на действие приложения. В действие передается параметр с именем N и со значением, равным значению ключевого поля **Num** таблицы данных. По этому параметру приложение сможет идентифицировать запись, к которой относится запрос.

19. В обработчик события **OnAction** действия **Default** занесите код:

```
Table1->Active = false;
Table1->IndexName = "depfio";
Table1->Active = true;
Response->Content = DataSetTableProducer1->Content() ;
```

Он обеспечивает индексацию таблицы по отделам и алфавиту и заполняет ячейки таблицы методом **Content** компонента **DataSetTableProducer.**

20. Сохраните приложение под именем «DB3» и перенесите выполняемый файл в папку сервера. Ссылки «просмотр» пока не работают

21. Создайте в модуле Web действие, воспринимающее запрос о детальном просмотре записи. Создайте в модуле новое действие, например, «record» и задайте **PathName**="/record". Это действие должно взаимодействовать с компонентом **DataSetPageProducer.**

22. Перенесите этот компонент на модуль Web и свяжите с набором данных, установив свойство **DataSet** равным **Table1.**

23. В свойство **HTMLDoc** занесите текст:

```
<body>
<h1>Просмотр записи о сотруднике</h1>
<table width="100%" Align="Left">
<tr><td Allign="Left"><b>Фамилия </td><td><b>Имя </td><td> <b>Отчество</td> <td><b>r.p.
</td><td><b>Iюл </td> <td><b>Возраст</td></tr>
<tr><td><#Fam></td><td><#Nam></td><td><#Par></td><td><#Year_b></td>
<td><#Sex></td><td><#Age></td></tr>
</table>
<br><br><br><hr>
<table width="l00%">
<tr><td Align="Left"><b>Характеристика </td></tr>
<tr><td width="67%"><#Charact></td>
<td width="33%"><#Photo></td> </tr>
</table>
</body>
</html>
```

Этот текст формирует страницу HTML . В верхнюю таблицу в тексте HTML занесены шаблоны с именами полей. Эти шаблоны будут автоматически заменяться компонентом **DataSetPageProducer** на значения полей из текущей записи. Во второй таблице, расположенной ниже, будут размещаться характеристика и фотография сотрудника. Пока вместо них в текст HTML внесены шаблоны с именами **Charact** и **Photo.** Расшифровку этих шаблонов нужно организовывать в обработчике события **OnHTMLTag** компонента **DataSetPageProducer.**

24. В обработчике события **OuAction** действия **record** введите код:

```
Table1->Active = false;
Table1->IndexName = "";
Table1->Active = true;
TLocateOptions SearchOptions;
Table1->Locate("Num"r Request->QueryFields->Values["N"],
SearchOptions<<loPartialKey<<loCaseInsensitive);
Response->Content  =  DataSetPageProducer1->Content ();
```

Здесь методом **Locate** устанавливается в качестве текущей запись, в которой значение поля **Num** совпадает с переданным в запросе параметром N. После этого в качестве ответа задается документ HTML, формируемый методом **Content** компонента **DataSetPageProducer1.**

25. Создайте обработчик события **OnHTMLTag** компонента **DataSetPageProducer** для задания операций расшифровки шаблонов с именами **Charact** и **Photo.** Этот обработчик может иметь вид:

```
if (TagString == "Charact")
ReplaceText = Table1->FieldByName("Charact")->AsString;
else  if (TagString == "Photo")
ReplaceText = "<img src=photo?N=" +Table1->FieldByName("Num") -> AsString +">";
```

В приведенном обработчике значение текущей записи поля **Charact** заносится как строка вместо шаблона с именем «Charact». Значение графического поля передается в документ тег **<img>** и из документа обращается к серверному приложению, в ответ на которое приложение заносит в поток объект изображения. Для этого в модуль сервера Web введите еще одно действие **photo и** задайте свойство **PathIufo =** "/photo"

26. Для реализации действий, определенных в пункте 26 создайте обработчик события **OnAction.** В модуль необходимо вставить директиву препроцессора, подключающую файл *Graphics.hpp.* Без этой директивы создадутся проблемы с переменными типа TBitmap.

```
#include <Graphics.hpp>
void _fastcall TWebModule1::WebModule1photoAction(TObject *Sender, TWebRequest
*Request, TWebResponse *Response, bool &Handled)
{
Table1->Active  = true;
TLocateOptions SearchOptions;
Table1->Locate("Num",Request->QueryFields->Values["N"],
SearchOptions<<loPartialKey<<loCaseInsensitive);
Graphics::TBitmap *B =  new  Graphics::TBitmap();
B->Assign(Table1->FieldByName("Photo"));
TMemoryStream *S1 =  new  TMemoryStream() ;
B->SaveToStream(S1l);
S1->Position  =  0;
Response->ContentType  = "image/x-xbitmap";
Response->ContentStream  =  S1;
delete  B;
```

```
Table1->Active  = false;
}
```

В обработчике создается временный объект **B** типа TBitmap, в который заносится значение поля **Photo** текущей записи. Создается также объект потока **S1** типа TMemoryStream, который размещается в динамически распределяемой памяти, в него заносится содержимое объекта **B** и позиция потока устанавливается на его начало. В качестве типа возвращаемого результата Response->ContentType указывается «image/x-xbitmap». Свойство **ContentType** объекта ответа Response заполняется при необходимости вернуть объект медиа. В этом случае значение свойства **ContentType** должно содержать указание на тип данных (image - изображение) и на подтип (битовая матрица). Свойство **Response->ContentStream** должно содержать указатель потока, из которого берется объект. Последние операторы приведенного обработчика уничтожают временный объект **B** и закрывают соединение с базой данных.

Редактирование наборов данных

27. До сих пор рассмотренные задачи связаны с чтением из баз данных. Попробуйте теперь реализовать задачу записи в базы данных, т.е разработать следующий проект: Пусть нужно объявить в Интернет о имеющихся на предприятии вакансиях и предложить желающим зарегистрироваться. Результаты регистрации претендента на должность нужно заносить в таблицу Pers базы данных dbP.

28. В этом тексте используйте форму, в которой будет размещена таблица, а в ней - надписи и компоненты ввода. Используйте для ввода фамилии, имени, отчества и года рождения окна редактирования Edit с именами Fam, Nam, Par, Year.

29. Для осуществления запроса методом формы **Post** к приложению **DB4.exe** с путем «/resp», используйте кнопку с именем B1. Используйте две радиокнопки RadioButton с именем **Sex** к для ввода пола претендента, при получении в серверном приложении запроса из этой формы чтобы можно было определить, по значению **value** какая из этих кнопок нажата («м» для первой кнопки и «ж» для второй).

30. Для реализации пунктов 28 и 29 подготовьте с помощью редактора HTML следующий текст страницы Web

```
<html>
<head>
<title>Прием на работу</title>
</head>
<body>
<h1><b>Регистрация  претендента на должность</h1>
<p>Запишите сведения о себе и нажмите кнопку 'Регистрация'</p>
<form method="POST"
action="http://mycomputer/cgi-bin/DB4.exe/resp"> <table>
<tr><td><strong>Фамилия</strong></td><td><input  type="text"  name="Fam" size = "20"></td>
</tr>
<tr><td><strong>Имя</strong></td><td><input  type="text"  name="Nam" size="20"></td> </tr>
 <tr><td><strong>Отчество</strong></td> <td><input  type="text"  name="Par" size="20"></td>
</tr>
<tr><td><strong>Iiол</strong></td> <td><input  type="radio"  value="м"
checked  name="Sex"><strong>м</strong> <input  type="radio" name="Sex"
value="ж"><strong>ж</strong></td> </tr>
<tr><td><strong>Год рожд. </strong>  </td> <td><input type="text" name="Year"
size="4"></td> </tr>
 <tr><td></td><td><div  align="center"><center><p>,input type="submit"
value="Регистрация" name="Bl"></td> </tr>
</table>
```

```
</form>
</body>
</html>
```

Разместите подготовленную страницу непосредственно на сервере вызовите ее. Проектируйте теперь серверное приложение, из которой нужно загружать эту страницу.

31. Создайте новый модуль сервера Web. Перенесите на него компоненты **Table** и **PageProducer.** Компонент **Table** настройте на таблицу Pers базы данных **dbP.** В свойство **HTMLDoc** компонента **PageProducer1** занесите приведенный выше текст страницы Web. Создайте в модуле действие и установите его свойство **Default** в **true.**

32. В свойстве **Producer** данного действия укажите ссылку на компонент **PageProducer1.**В тексте HTML содержится ссылка на действие с путем «/resp».

33. Теперь создайте в модуле объект этого действия под названием **Response.** В обработчике его события **OnAction** предусмотрите анализ введенного клиентом в форме регистрации. Если в данных содержатся какие-то ошибки, предусмотрите выдачу клиенту соответствующих сообщений. А если все в порядке, предусмотрите записи результатов регистрации в базу данных.

34. Для реализации пункта 32 создайте обработчик события события **OnAction** действия **Response**:

```
void _fastcall  TWebModule1::WebModule1ResponseAction(TObject *Sender, TwebRequest
*Request, TWebResponse *^Response, bool &Handled)
{
word year;
if ((Request->ContentFields->Values["Fam"] == "") ||
 (Request->ContentFields->Values["Nam"] == "") ||
 (Request->ContentFields->Values["Par"] == "") ||
(Request->ContentFields->Values["Year"j == ""))
{
Response->Content ="<p>Вы не полностью заполнили бланк регистрации" "<p>Повторите
ввод данных";
 Handled = true;
}
else
{
 try
{
year  = StrToInt(Request->ContentFields->Values["Year"])
}
catch (EConvertErrors) // реакция на  неверный формат года рождения
{
 Response->Content ="<p>Вы ошиблись в  формате года  рождения" "<p>Повторите  ввод
данных";
Handled  =  true;
return;
}
if ((year  < 1917)  ||  (year  > 1980))
{
// реакция на  противоречащее ограничениям значение года
Response->Content  ="<p> Вы не подходите на эту должность по  возрасту"; Handled =  true;
return;
}
Table1->Active  = true;
Table1->Insert();
```

```
Table1Fam->AsString = Request->ContentFields->Values["Fam" ] ; Table1Nam->AsString =
Request->ContentFields->Values["Nam"];
Table1Par->AsString = Request->ContentFields->Values["Par"] ;
Table1Year_b->Value = year;
if (Request->ContentFields->Values["Sex"] == "м")
Table1Sex->Value = true;
else
Table1Sex->Value = false;
// Пересылка новой записи в базу данных
Table1->Post () ;
Table1->Active = false;
Response->Content ="<p>Спасибо!<p>Сведения о Вас включены в базу данных" "<p>Мы
рассмотрим Вашу кандидатуру и сообщим результаты"; Handled = true;
}
}
```

Пояснения к тексту кода

Первый оператор if проверяет, нет ли хотя бы одного незаполненного поля. Если есть, то пользователю выдается страница, содержащая сообщение об этом и предложение повторить ввод. Пользователь может в браузере вернуться на предыдущую страницу и ввести на ней недостающие данные.

Локальная переменная **year** получает текст, введенный пользователем в поле Year. Если текст не соответствует формату целого числа, генерируется исключение, которое перехватывается в блоке **catch** и пользователю выдается соответствующее замечание.

Следующий оператор **if** проверяет, не выходит ли его значение за ограничения. Если выходит, то пользователю выдается сообщение, что он по возрасту не подходит. Эта проверка необходима, так как если в момент записи в базу данных выяснится, что ограничения нарушены, будет сгенерировано исключение.

Если все нормально, открывается соединение с базой данных, методом **Insert** в наборе данных создается новая пустая запись, ее поля заполняются данными клиента, после чего запись пересылается в базу данных методом **Post** и пользователю выдается сообщение о включении его в базу данных.

Как видно, включение в Интернет в базу данных новых записей, как и редактирование существующих записей, никаких сложностей не представляет.

35. Сохраните проект под именем DB4, скомпилируйте, перенесите в исполняемую папку сервера.

Задание на самостоятельную работу

Все предлагаемые проекты работы с базами данных в Интернет использовали компонент набора данных **Table.** Разработайте аналогичные проекты работы с базами данных в Интернет, используя запросы SQL с помощью компонента **Query**, а для просмотра записей таблицы используя компонент **QueryTableProducer.** Его свойство **Query** должно содержать ссылку на набор данных типа **TQuery.** В остальном свойства **QueryTableProducer** и **DataSetTableProducer** не различаются.

Модуль 1. *Введение в объектно-ориентированное программирование на C#*

Лабораторная работа № 1.

Задание 1. Разработать проект «Печать фотографий», который позволяет рассчитать стоимость печати фотографий.

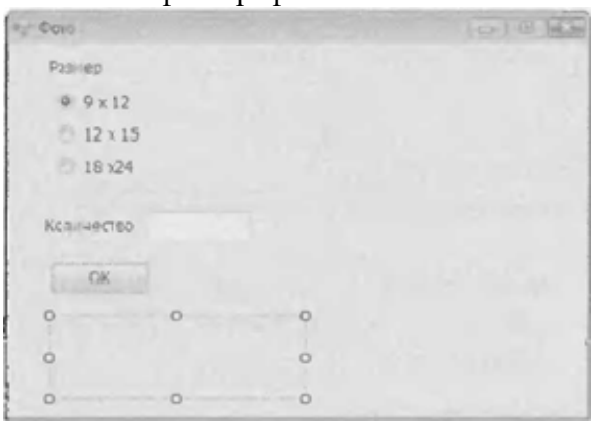
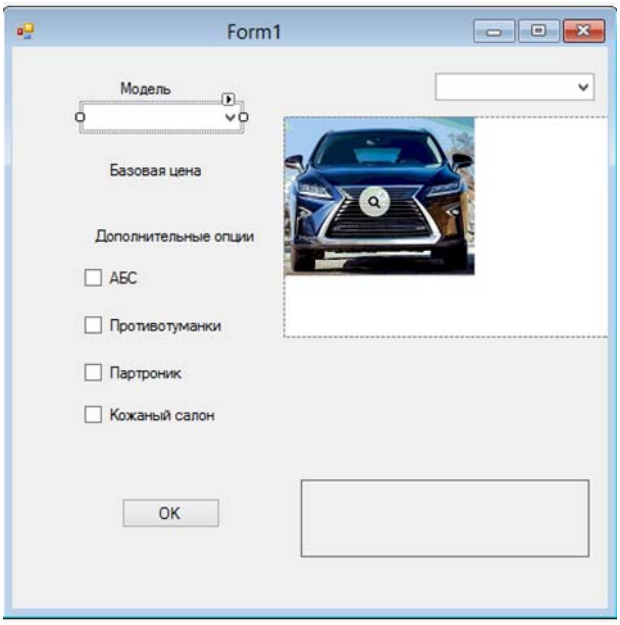


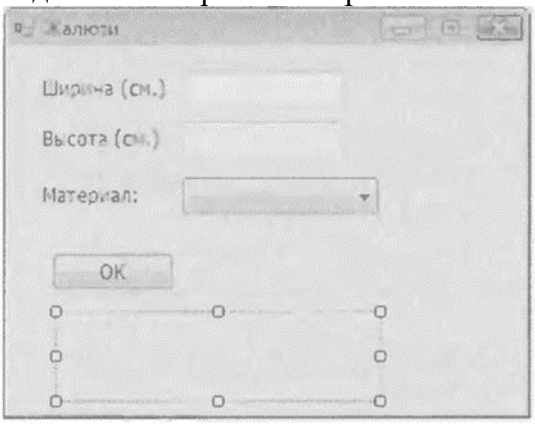
Рис. 1.4. Форма программы Фото

```
public Form1()
{
InitializeComponent();
// на стройка компонентов
radioButton1.Checked = true;
buttonl.Enabled = false;
}
// щелчок на кнопке OK
private void button1_Click(object sender, EventArgs e)
{
}
private void textBox1_TextChanged(object sender, EventArgs e)
{
if  (textBox1.Text.Length == 0)
button1.Enabled = false;
else
button1.Enabled = true;
label2.Text = "";
}
// щелчок на radioButton
private void radioButton1 Click(object sender, EventArgs e)
label2.Text=   "";
// установить курсор в поле Количество
textBox1.Focus();
}
}
```

Задание 2. Разработать проект **«Комплектация автомобиля»** позволяет рассчитать стоимость автомобиля в зависимости от выбранной комплектации. Отображение картинки обеспечивает компонент PictureBox.



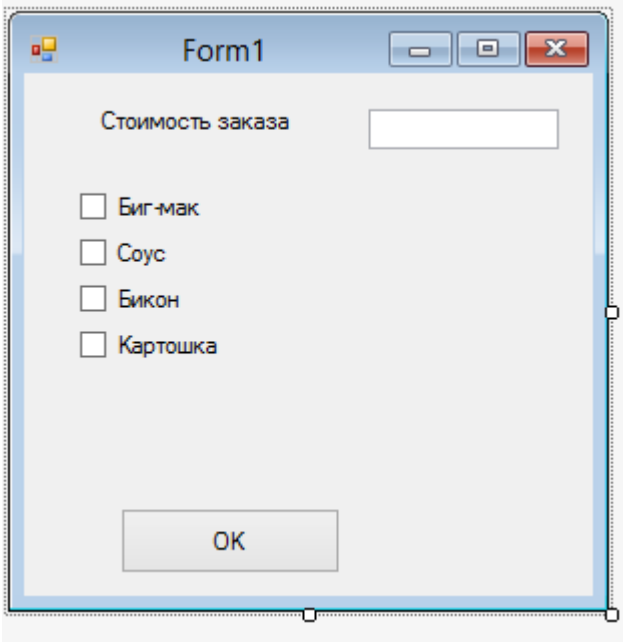
Задание 3. Разработать проект **«Жалюзи»**



Модуль 2. *Платформа .NET Framework и ее применение для ООП*

Лабораторная работа 2.

Задание 1. Разработать проект Windows Form **Кафе,** ее форма приведена на рис, которая демонстрирует использование компонента CheckBox.



Форма программы **Кафе**

Задание 2. Разработка проекта «Любимые напитки»

Цель работы

Написать программу демонстрирующую работу с объектами ListBox.

Спецификация программы:

На пользовательской форме должны быть расположены 3 списка. Один из списков содержит названия напитков. Пользователь может переносить элементы из этого списка в списки «Любимые» и «Нелюбимые» и обратно. При этом перемещаемый элемент должен удалятся из списка-источника. То есть, например, перенос элемента «Чай» из общего списка в список «Любимые» происходит в следующем порядке: он добавляется в список «Любимые» и удаляется из общего списка. Таким образом общее количество элементов всех трех списков остается постоянным. Перенос элементов между списками должен осуществляться по нажатию на соответствующие кнопки, либо мышью (система Drag and Drop).

Система Drag and Drop позволяет напрямую перетаскивать объекты между разными источниками, например, из одного списка в другой. «Перетаскивание» представляет собой нажатие и удерживание левой кнопки мыши на объекте и дальнейшее его перемещение за курсором в желаемую область.

Списки «Любимые» и «Нелюбимые» должны сохраняться в текстовые файлы.

Для создания формы использовать компоненты:
Label – для подписей
ListBox – для вывода списков
Button – для инициирования действий
Рекомендуемая компоновка формы программы представлена на рисунке.

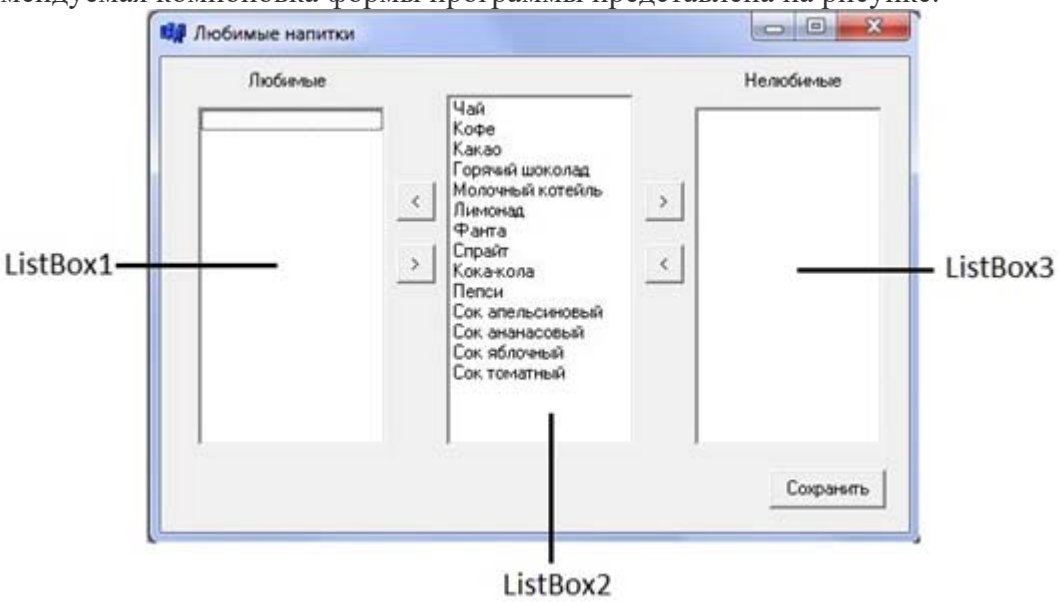


Рисунок. Рекомендуемая компоновка формы

Модуль 2. ***Платформа .NET Framework и ее применение для ООП***

Лабораторная работа № 3. Простые классы

Цель работы: Изучение правил разработки классов.

1. **Постановка задачи**

Создать новый проект и сохранить под под названием ***LabRab9.***

Имеются сведения об автомобилях и их владельцах:

- *Государственный регистрационный номер.*
- *Model.*
- *Color.*

- *FIO владельца.*

Требуется разработать приложение, позволяющее вводить указанные данные, хранить введенную информацию и удалять сведения об автомобиле и его владельце.

Интерфей спользователя

Список госномеров хранится в объекте ***comboBox1***. Поля ввода используются соответственно ***textBox1*** – ***textBox4***, кнопки – соответственно ***button1*** – ***button3***.

Постройте интерфейс пользователя как показано на рисунке.

Сценарий работы пользователя и программы

Пользователь видит на экране только список госномеров и кнопку **«УДАЛИТЬ»** – объекты группы данных не показываются. При выборе госномера из списка раскрываются данные. При вводе нового госномера также открываются поля данных, но они пустые.

Пользователь вводит новые значения (или изменяет имеющиеся) и сохраняет изменения (или отказывается от сохранения изменений). Данные о каждом Автомобиле –это отдельный объект. Сохраненные значения записываются в массив объектов, а в списке отображаются только госномера.

Госномер можно удалить из списка – в этом случае из массива удаляются и данные об объекте.

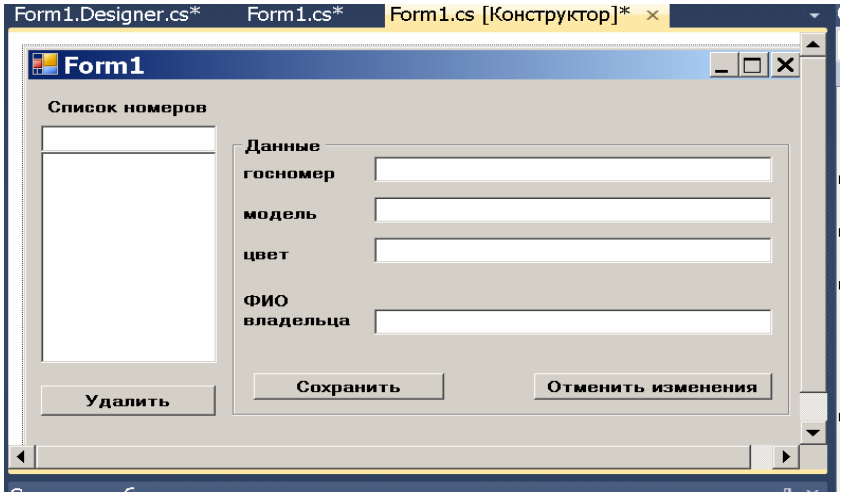


Рис. 20. Окно задачи № 5

2. Ввод и сохранение данных в массиве

Разработка некоторых элементов класса данных

Решение любой задачи начинается с описания используемых данных. Сведения о каждом транспортном средстве содержат четыре показателя, Автомобилей может быть любое количество. Фактически речь идёт о новом типе данных, содержащем в себе все перечисленные показатели.

Нужно оформить новый тип данных как класс ***Avto***.

Поместить заготовку описания этого класса в модуль ***Form1*** в пространство имён ***LabRab9*** после класса ***Form1***:

```
public class Avto
{ }
```

В составе класса **Gosnomer, Model, Color и FIO** определить как строки:

```
public class Avto
{
string Gosnomer; string Model; string Color; string FIO;
}
```

Далее необходимо описать конструктор объектов. Его назначение: при создании объекта дать исходные значения его полям. В классе описаны четыре поля, при создании объекта их инициализировать пустыми значениями. Тело конструктора будет содержать 4 оператора присваивания. Окончательно (фрагмент):

string FIO; public Avto()
{ Gosnomer = ""; Model = ""; Color = ""; FIO = ""; }

Обработка ввода *Gosnomer*

Логично представить себе, что пользователь при первом запуске приложения захочет ввести данные о первом Автомобиле. Он попробует набрать в верхней строке комбинированного списка госномер Это событие – набор с клавиатуры. Точнее, нажатие символов на клавиатуре, ***KeyPress***. Такое событие связано с объектом ***ComboBox***.

Нужно создать обработчик и подключить его к событию.

Надо описать алгоритм, который будет работать при наборе госномера. В соответствии с сущностью события ***KeyPress*** обработчик будет выполняться каждый раз при любом нажатии на любую клавишу. Но госномер будет набран целиком только после нажатия клавиши ***«ENTER»***. Значит, все остальные нажатия метод должен проигнорировать. Но как определить, что нажата именно клавиша ***«ENTER»***?

Ответ на этот вопрос станет очевиден, если знать смысл параметров метода. Параметр ***sender*** – это источник события, т. е. объект ***comboBox1,*** а параметр ***e*** – это контейнер, содержащий информацию о событии. Выражение ***e.KeyChar*** позволяет узнать символ нажатой клавиши. Проверку можно осуществить так:

if (e.KeyChar == (char) Keys.Enter) { }

Вставьте эту конструкцию в обработчик. Системное перечисление Keys включает в себя все символы клавиатуры, в том числе и Enter. Только при сравнении его надо преобразовать в символ. Между фигурными скобками можно записать операторы, которые будут работать только при завершении набора госномера – все остальные клавиши никак не будут «замечены» обработчиком.

Задание для самостоятельного выполнения

*Убедитесь, что обработчик реагирует на нажатие клавиши **«ENTER»**. Вставьте между фигурными скобками оператор выдачи на экран какого либо сообщения (через messageBox) и запустите программу. Наберите в поле ввода что-нибудь и нажмите **«ENTER»**. Убедившись – удалите оператор выдачи сообщения.*

Теперь определим, что делать, когда нажата клавиша ***«ENTER»***. Очевидно, нам придётся отобразить невидимые объекты окна – группу данных.

Задание для самостоятельного выполнения

Добавьте оператор, делающий группу объектов видимой.

Ввод данных

Теперь можно попробовать набирать данные в поля ввода. В общем-то всё нормально. Но! Ведь мы уже набрали Gosnomer – почему же приходится набирать его ещё раз?

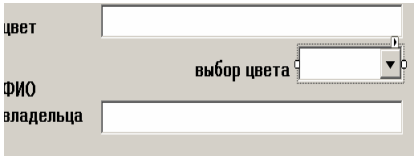
Задание для самостоятельного выполнения

Добавьте ещё один оператор. Как только группа объектов стала видимой, перенесите в первое поле ввода значение набранного госномера.

Есть и еще одно «но», значительно более существенное. Цвет лучше выбирать из какого-то списка, а не набирать. Пользователь вряд ли ошибётся с набором госномера и модели, а вот с цветом – запросто. Значит, придется запретить набор, т. е. установить у этого поля свойство ***Enable=false.***

Для правильного выбора цвета модифицируйте вид формы, как показано на рисунке.

добавьте между строками цвета и ФИО объект ***Label*** с текстом «Выбор цвета» и



объект ***comboBox2***, в котором нужно создать список цветов. У последнего объекта установите запрет набора цвета – это свойство ***DropDownStyle=DropDownList***.

Для заполнения списка цветов нужно в пространстве имен ***LabRab9*** между классами ***Form1*** и ***Avto*** тип данных – перечисление ***Color***. Включите в него шесть значений-констант:

public enum Color { неопределенный, белый, красный, фиолетовый, черный, серый, зелёный }

Затем нужно создать в comboBox2 список цветов на основе перечисления Color. Значения этого перечисления перепишем в коллекцию объекта comboBox2 при запуске программы – перед открытием окна.

В обработчик события **Form1.Load** вставить код:

int i;

for (i = 0; i < 6; i++) comboBox2.Items.Add((Color)i);

Нужно написать обработчик события ***SelectedIndexChanged*** двойным кликом по ***comboBox2***, которое выбранное значение цвета из свойства ***comboBox2.Text*** поместит в ***textBox3***. создаем обработчик этого события. В тело обработчика добавить строку:

textBox3.Text = comboBox2.Text;

Нужно также в приложении предусмотреть, чтобы при наборе данных пользователь мог перемещаться от поля к полю клавишей ***«TAB»***.

Для этого в модуле ***Form1.Designer.cs*** в секциях-описаниях каждого объекта есть для оператора, определяющий порядок обхода объектов окна с помощью клавиши ***«TAB»*** нужно установить индексы (TabIndex) следующим образом:

0 – comboBox1,
1 – textBox1,
2 – textBox2,
3 – comboBox2,
4 – textBox4,
5 – кнопка button2,
6 – кнопка button3,
7 – кнопка button1.

Остальные не играют роли. Теперь запустить программу и проверить работу клавишей ***«TAB»***. Курсор должен перемещаться в указанном порядке.

Обработчики нажатия на кнопки:

Кнопка отмены изменений

При отмене изменений (или отказе от сохранения набранных данных) надо сделать:

- очистить поля ***textBox1*** – ***textBox4***;
- убрать с экрана (скрыть) блок ввода данных;
- очистить поле ввода в объекте ***comboBox1***.

Задание для самостоятельного выполнения

- *Внесите необходимые изменения в программу.*

Кнопка сохранения результатов ввода данных

Введенные данные должны сохраниться в элементе массива данных Массив должен хранить все данные о каждом автомобиле. Соответственно, элементы массива должны быть типа ***Avto***. Определить хранилище на 1000 Автомобилей. Для объявления использовать оператор:

Avto [] md = new Avto[1000];

Нужно его записать в самом начале класса ***Form1***. Это гарантирует доступ к нему из любого метода этого класса, т. е. в пределах окна. Там же создать переменную-счетчик, которая будет содержать количество заполненных элементов массива:

int count=0;

Нужно учесть, что при объявлении массива в памяти выделено место только под его элементы, а они являются ссылками на объекты – при создании массива эти ссылки будут равны значению ***null***.

Поэтому нужно предусмотреть выделение памяти под каждый объект также при открытии окна с помощью оператора:

for (i = 0; i < 1000; i++) // создание объектов

{
md[i] = new Avto();
}

Можно было бы введенные значения записать в поля первого свободного элемента массива. Однако, просто так этого сделать не удастся: поля объекта являются закрытыми. Для этого нужно включить в класс ***Avto*** метод, позволяющий обновить значения полей объекта:

public void Obnov(string g, string m, string c, string fio)
{ Gosnomer = g; Model = m; Color = c; FIO = fio; }

Этот метод – динамический. Он может быть вызван для работы с объектом массива md. В обработчике нажатия на кнопку «Сохранить» нужно указать:

md[count]. Obnov(textBox1.Text, textBox2.Text, textBox3.Text, textBox4.Text);
count++;

После увеличения значения ***count*** следующий набор данных запишется уже в следующий по порядку объект.

Задание для самостоятельного выполнения

Добавьте в обработчик button2 очистку полей ввода данных и сокрытие группы объектов набора данных, как это сделано в обработчике button3.

Не пишите повторно операторы, которые уже есть в обработчике для button3. Воспользуйтесь уже имеющимся методом.

Отображение списка Госномеров

Но если с отменой набора все очевидно, то с сохранением не ясно: то ли сохраняются данные, то ли нет. В списке новый госномер не отображается.

Список госномеров – это список в коллекции ***comboBox1***, здесь нужно использовать массив объектов **md**

Первое поле каждого объекта – это госномер.

Задание для самостоятельного выполнения

В классе Avto создайте динамический метод, позволяющий получить Gosnomer объекта Avto.

*В обработчике кнопки **«СОХРАНИТЬ»** очистите коллекцию Items объекта comboBox1.*

*Просмотрите массив md в цикле, выберите из каждого объекта поле Gosnomer с помощью созданного ранее метода и добавьте Gosnomer в коллекцию Items объекта comboBox1. Алгоритм следует оформить в виде метода **CreateLisTGosn**. Метод вызвать в обработчике нажатием кнопки **«СОХРАНИТЬ»**.*

Обработка выбора Госномера из списка

Первая особенность – это другое событие: не набор, а выбор значения из списка. Имя события – ***SelectedIndexChanged***. Обработчик этого события можно получить двойным кликом по объекту. Внутрь обработчика надо записать операторы, извлекающие значения из выбранного объекта и записывающие их в поля ввода ***textBox1***. А затем – отобразить группу.

Для заполнения поля ввода госномера ***textBox1*** сам объект не нужен. Ведь после выбора госномера он автоматически отображается в поле ввода объекта ***comboBox1*** (свойство ***Text***), откуда его можно перекинуть в поле ***textBox1***. Вместе с оператором отображения группы это выглядит так:

textBox1.Text = comboBox1.Text;
groupBox1.Visible = true;

Две проблемы:

- получить номер выбранного объекта ***Avto***;
- извлечь из объекта значения модели, цвета и ФИО.

Свойство ***SelectedIndex*** объекта ***comboBox1*** содержит номер выбранной строки списка. Этот номер совпадает с индексом объекта в массиве. Выбрать номер можно оператором:

int nom=comboBox1.SelectedIndex;

Прямой доступ к полям ***Model***, ***Color***, ***FIO*** запрещен, поэтому извлечь значения сразу не получается.

Нужно расширить функциональность класса ***Avto***, т.е. добавить в него описания свойств, которые позволят получить значения закрытых полей. Итак:

```
public string mod { get { return Model; } }
public string col  { get { return Color; } }
public string  fiov { get { return  FIO; } }
```

Нужно воспользоваться этими свойствами для получения значений:

```
textBox2.Text = md[nom].mod;
textBox3.Text = md[nom].col;
textBox4.Text = md[nom]. fiov;
```

Задание для самостоятельного выполнения

Наберите и сохраните два разных набора значений. Затем последовательно отобразите их, выбрав госномера. Обратите внимание, что в одном из случаев значение в поле цвета противоречит значению в списке цветов – там осталось последнее набиравшееся значение. Сделайте так, чтобы значение цвета в поле comboBox2 и textBox3 были одинаковыми.

Теперь выберите какое-либо значение госномера и внесите изменение в данные. Например, замените FIO и сохраните. В списке появится двойной номер, и это правильно: не установлен контроль на повторную запись. Измененные данные должны записываться в тот же объект, если госномер остался прежним. Это значит, что алгоритм в обработчике ***button2*** должен различать, новый это госномер или существующий, т. е. в массив пишется как бы новый элемент.

Задания для самостоятельного решения

1. *Внесите изменения в алгоритм обработчика кнопки button2, не допускающие двойных записей.*
2. *Разрешите пользователю набирать новые цвета и обеспечьте их сохранение в списке. Двойных записей в список цветов не делать.*
3. *Создайте обработчик кнопки **«УДАЛИТЬ»** и составьте алгоритм работы. Не забудьте, что удалять надо не только госномер из списка, но и данные из массива.*

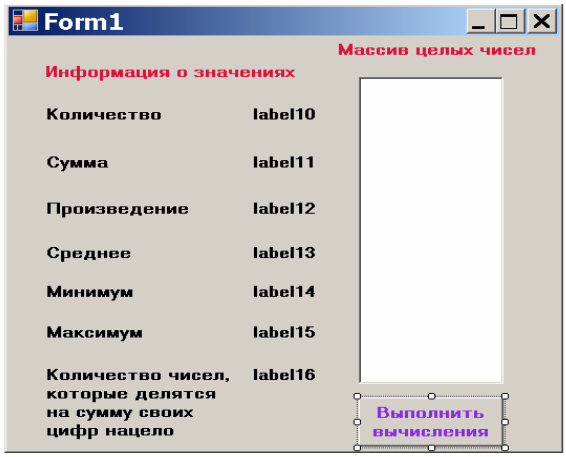
Лабораторная работа № 4. Типовые алгоритмы обработки массива

Цель работы: Изучение типовых алгоритмов обработки массивов.

Создать новый проект с именем ***Lab_rab6***. В объекте ***textBox1*** установить многострочный режим (свойство ***MultiLine*** = ***true***). В объектах label с 10 по 16 будут отображаться значения в соответствии с надписями в объектах.

В объект ***textBox1*** пользователь будет набирать значения элементов массива. Кнопка обеспечивает запуск алгоритма вычислений.

Порядок работы пользователя: сначала он вводит значения в массив, затем нажимает кнопку и получает результаты сразу всех вычислений.



Окно формы

Все характеристики массива (сумма, среднее и прочие) вычисляются по собственным алгоритмам. Все алгоритмы должны быть записаны внутри обработчика.

До нажатия кнопки данные хранятся в объекте ***textBox1***. Для хранения значений этот объект имеет коллекцию строк ***Lines*.** Это обычный массив типа ***string***. Сначала преобразовать элементы этого массива из строковых значений в численные и сохранить их значения в специально созданном целочисленном массиве. Будем считать, что пользователь задает в одной строке только одно число.

```
int[] m = new int[1000];
```

Вычислить количество чисел (строк) в коллекции:

```
int n = textBox1.Lines.Length;
```

В переменной k хранить количество значений в массиве:

```
int k = 0;
```

Цикл, формирующий массив ***m***

```
for (i = 0; i < n; i++)
{   m[k] = Convert.ToInt32(textBox1.Lines[i]); k++;  }
```

Задания для самостоятельной работы

1. *Обработайте исключение. Если в строке некорректное значение (не цифра или строка пуста), то по такой строке ничего делать не надо – никакое значение в массив не включается.*
2. *Запишите алгоритм выдачи на экран количества чисел в массиве.*
3. *Запишите алгоритм выдачи на экран суммы элементов массива.*
4. *Запишите алгоритм выдачи на экран произведения элементов массива.*

Запишите алгоритм выдачи на экран среднего арифметического элементов массива.

5. *Запишите алгоритм выдачи на экран минимального элемента массива.*
6. *Запишите алгоритм выдачи на экран максимального элемента массива.*
7. *Запишите алгоритм подсчета количества чисел, которые делятся нацело на сумму своих цифр.*
8. *Доработайте программу. Выдайте на экран номера строк, которые имеют ошибки ввода и не включены в вычисления. Вывод выполнить одним окном MessageBox.*

Указание. *Сохраните номера строк в отдельном массиве.*

Лабораторная работа № 5. Работа с массивами случайных чисел

Цель работы: Изучение методов классов Random и Math.

Задание для самостоятельноговыполнения *Создайте окно следующего вида:*

1. Два объекта ListBox:

- listBox1 – для аргументов функции;

- listBox2 – для значений функции.

2. Три объекта textbox:

- textBox1 – поле для источника повторения;
- textBox2 – поле для минимума;
- textBox3 – поле для максимума.

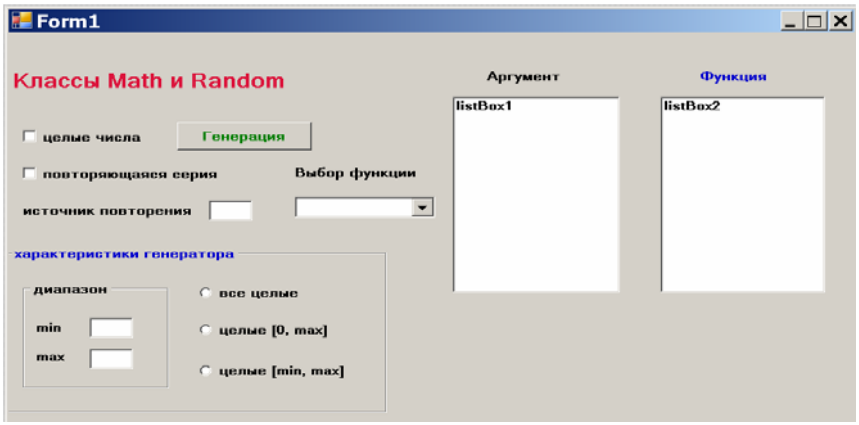
3. Объект comboBox1 – для выбора функции.
4. Два объекта checkbox:

- checkBox1 – для генерации повторяющейся серии чисел;
- checkBox2 – для генерации целых чисел.

5. Три объекта radioButton:

- radioButton1 – все целые;
- radioButton2 – целые в диапазоне [0, max];
- radioButton3 – целые в диапазоне [min, max].

6. Объект groupBox1 – группирует характеристики генератора случайных чисел.
7. Кнопка button1 – кнопка с надписью «Генерация».
8. Объект label3 – это надпись «Функции».
9. Объект label2 – это надпись *«источник повторения».*



Все остальные надписи – это объекты label под любыми номерами.

Окно формы

Предлагается реализовать следующий сценарий.

Пользователю нужна программа для изучения различных функций класса Math. Функцию он сможет выбрать из списка в объекте **comboBox1**. Убедитесь, что пока список пустой. При выборе функции надпись «Функция» должна замениться на имя выбранной функции.

Аргументы для вычисления функции будут размещаться в объекте **listBox1**, а результаты вычисления – в объекте **listBox2**. Вычисления должны производиться сразу же, как только пользователь выберет функцию.

Аргументы формируются в списке объекта **listBox1** при нажатии кнопки **«Генерация».** Генерируемые значения – это случайные числа. Всего их 10 штук.

Какие будут сгенерированы значения, зависит от настроек. Это могут быть вещественные числа в интервале от 0 до 1 (по умолчанию). Можно установить флажок – тогда будут генерироваться целые числа. Генерируемые серии чисел могут быть разными (по умолчанию), а при установке флажка повторения серии

– каждый раз одинаковыми для каждого заданного числа (источника повторений).

В случае целых чисел возможна генерация:

- из всех возможных положительных целых чисел;
- из отрезка от 0 до максимального значения;
- из отрезка от минимального до максимального – в зависимости от выбранного переключателя.

Значения минимума и максимума должны задаваться только в нужном случае.

Желательно, чтобы в каждый момент времени выполнения программы в окне отображалось бы только то, что необходимо. Например, если не установлен флажок целых чисел, то и все настройки для целых отображаться не должны. То же и с повторением серии: если флажок не установлен, то и источник отображаться не должен.

По условию задачи при запуске программы флажок «целые числа» не установлен. Значит, не должно отображаться ничего из настроек, то есть все объекты, заключенные в ***groupBox1*** должны быть невидимы. Найдём в свойствах объекта ***groupBox1*** свойство ***Visible*** и установим в нём значение ***false***.

Задание для самостоятельного выполнения

При запуске программы флажок повторения серии также не установлен. Сделайте, чтобы при запуске программы поле источника повторений и надпись не отображались на экране.

При установке флажка **«целые числа»** устанавливается режим генерации целых чисел. При этом должны показаться все режимы настройки, заключенные в объекте ***groupBox1***. И наоборот, когда флажок снят, режимы настройки вновь должны стать невидимыми. Данные функции реализуйте в обработчике события ***Click*** объекта ***checkBox2***:

Нужно добавить в обработчик код: если флажок установлен, то сделать группу объектов видимой, а если не установлен, сделать группу невидимой:

```
private void checkBox2_CheckedChanged(object sender, EventArgs e)
{

if (checkBox2.Checked == true)
        { groupBox2.Visible = true; }
else groupBox2.Visible = false;
}
```

Задание для самостоятельного выполнения

При установке флажка «повторяющаяся серия» поле источника и надпись должны отображаться. Наоборот, при снятии флажка оба объекта должны стать невидимыми. Запрограммируйте эту логику и проверьте работу.

При заполнении объекта ***comboBox1*** включите в список имена стандартных математических функций класса ***Math.***

Например, такие:

- **Геометрические:** Atan (арктангенс), Cos (косинус), Exp (экспонента), Log (натуральный логарифм), Sin (синус), Tan (тангенс).
- **Алгебраические:** Floor (округление), Ceiling (округление), Pow(x,3) (возведение числа в третью степень), Sqrt (квадратный корень), ABS() абсолютная величина.

Занесение в список ***comboBox1*** осуществляется через свойство ***Items***.

При выборе имени функции это выбранное имя должно заменить собой надпись **«Функция».**

Задание для самостоятельного выполнения

Замените надпись «Функция» при выборе имени из списка.

Указания.

*1. Выбранное имя функции – в свойстве **Text** объекта comboBox. Событие, которое наступает при выборе имени – **SelectedIndexChanged.***

2. Создайте соответствующий метод-обработчик – для создания заготовки достаточен двойной клик на объекте comboBox1.

Далее нужно создать обработчик события нажатия на кнопку. В обработчике нужно реализовать разные способы генерации последовательности чисел – их должно быть 10 штук.

Желательно использовать два генератора – два объекта класса ***Random***.

Один – для случая неповторяющейся последовательности.

Другой – для повторяющейся.

Для определения какой нужен генератор, используйте флажок ***checkBox1***.

В зависимости от того выбран он или нет, будут две ветви алгоритма. В ветвь **(if)** должна соответствовать неповторяющейся последовательности. Здесь должен быть создан простой объект класса ***Random***:

Random G = new Random();

Ветвь **(else)** должна соответствовать повторяющейся последовательности. Здесь должен быть создан сложный объект, учитывающий заданное значение в источнике повторения ***textBox3.Text***.

Однако, это значение типа ***string***, его надо преобразовать в целое число, а это возможно только тогда, когда это цифровое выражение. Следовательно, необходимо еще обработать исключение: если значение не задано или задано не цифрой, то взять в качестве источника любое число. Например, ноль.

```
Random G;
if (checkBox1.Checked == false)
        G = new Random();
else
{
int n;
try
{ n = Convert.ToInt32(textBox1.Text); }
catch {
        n = 0;
}
        G = new Random(n);
}
```

Теперь можно доставать из генератора случайные числа, используя метод ***Next*** или метод ***NextDouble.*** Каким методом – зависит от состояния флажка «целые числа». Если он не выбран, то используем ***NextDouble***, а если выбран, то ***Next***.

Если последовательность целая, то надо еще учесть, каков интервал выбора: все числа, от нуля до максимума или от минимума до максимума. А это зависит от того, какой из переключателей **(*radioButton1*, *radioButton2*, *radioButton3*)** выбран.

При выборе двух последних переключателей должны быть выбраны допустимые (цифровые) значения минимума и максимума. И весь этот выбор – в цикле. Каждое полученное случайное число надо разместить в ***listBox1*** как отдельную строку.

Цикл и ветвление в зависимости от состояния флажка:

```
for (n = 0; n < 10; n++)
{
if (checkBox2.Checked == false)
{ }
else
{ }
}
```

В секцию «if» добавьте операторы:

double d = G.NextDouble(); listBox1.Items.Add(d.ToString());

Первый оператор обеспечивает получение случайного вещественного числа. Второй с помощью метода Add добавляет это число в коллекцию объекта, предварительно преобразовав в строку.

Задания длясамостоятельного выполнения

1. Проверьте работу без флажка – генерация выполняется. А с флажком не выполняется – ещё не написан алгоритм.

2. Перезапустите программу. Нажмите на кнопку. Затем нажмите ещё раз. Должны быть новые значения, но старые тоже остались. Внесите соответствующие изменения в код.

3. Проверьте работу при установленном флажке повтора серии.

Действительно ли серия повторяется при повторном нажатии на кнопку? Убедитесь, что при разных значениях источника повтора получаются разные последовательности.

Запустите программу, установите флажок «целые числа». Отобразятся характеристики генератора для случая целых чисел. Обратите внимание: невыбран ниодин ив переключателей.

Задание для самостоятельного выполнения

Будем считать, что при выборе флажка надо установить значение «выбрано» в переключатель «все целые» (программно поставить точку). Это достигается изменением свойства Checked данного переключателя. Внесите нужный оператор в программу.

Для формирования блока **else** обработчика нажатия на кнопку нужно добавить следующие операторы:

```
int m;
if (radioButton1.Checked == true)
{ }
        if (radioButton2.Checked == true)
{ }
        if (radioButton3.Checked == true)
{ }
```

Переменная ***m*** предназначена для случайного целого числа. Каждый блок предназначен для своего переключателя: если он выбран, то будет обработка. Из генератора берется целое число из всего диапазона целых положительных чисел:

m = G.Next();

Во втором блоке надо сначала вычислить максимальное значение. Надо учесть, что оно может быть с ошибкой или не задано – в этом случае можно взять какое-то значение как максимально возможное, например, 100:

```
int max;
try {
        max = Convert.ToInt32(textBox3.Text);
}
catch
{
        max = 100;
}
        m = G.Next(max);
```

Задания для самостоятельного выполнения

1. Доработайте третий блок. Здесь метод Next используется с двумя параметрами – минимальное и максимальное значение. Не забудьте их также проверить на цифровое значение.

2. Запишите сразу после третьего блока оператор, добавляющий полученное

целое число в список listBox1.

Вычисление значений функции

Требуется алгоритм вычисления значений функции. При выборе функции в обработчик добавить блок вычислений. Фактически это разные варианты вычислений, но надо ещё распознать, какой именно вариант выбран – какая функция.

Для распознавания варианта используйте порядок их расположения в списке. Предпочтительнее в оспользоваться оператором ***switch***. Значения отправляются в ***listBox2***.

Добавьте в обработчик следующий код:

```
int i;
double d=0;
double r=0;
for (i = 0; i < 10; i++)
{
        d = Convert.ToDouble(listBox1.Items[i]);
switch (comboBox1.SelectedIndex)
{
    case 0: r= Math.Atan(d); break;
    …

}
    listBox2.Items.Add(r.ToString());
}
```

Задания для самостоятельного выполнения

1. *Проверьте работу программы для функции Atan.*
2. *При повторном выборе функции старые значения не исчезают. Внесите исправление.*
3. *После генерации новых аргументов в окне результатов остаются старые значения. Обеспечьте очистку окна результатов при генерации новых аргументов.*

Сейчас вещественные аргументы и результаты сохраняются с большим числом знаков после запятой. Отформатируйте вывод значений: аргументы – для вещественных два знака после запятой, результаты – четыре.

4. *Доработайте программу. Обеспечьте вычисление остальных функций.*

Лабораторная работа № 6. *Файлы произвольного доступа*

Цель работы: Изучение правил организации файлов произвольного доступа.

Продолжительность работы: 8 часов.

Постановка задачи

Требуется организовать простую базу данных. База хранит сведения о жителях города: FIO, Пол, Дата рождения, Улица, Дом,

Номер квартиры, Общая площадь жилья. Необходимо обеспечить: ввод новых данных, составление списка данных по известным значениям (полные сведения или частичные), отображение полных сведений по элементу из списка.

Обсуждение проблемы

На первый взгляд задача похожа на задачу № 5. Также можно

организовать сериализуемый класс. Введённые данные хранить в файле, который в начале работы программы считывать в массив, а после закрытия окна – записывать данные из массива в файл. Однако, здесь объем данных может быть таким, что файл целиком просто не уместится в массиве – представим себе, что кроме перечисленных сведений есть еще паспортные данные, сведения об образовании и т. д. Да и количество жителей само по себе велико. Следовательно, подобный способ организации программы не годится. Вместе с тем, разнотипность данных всё-таки заставит нас использовать класс. И сериализацию нам

придётся использовать – иначе объект на диск не записать. А в остальном – решение будет иным.

Сценарийработыпользователя

Мы не станем отображать на экране список всех FIO. Изначально на экране отобразим только две кнопки: ***«ВВОД ДАННЫХ»***

и ***«ПОИСК ДАННЫХ»****. Рассмотрим оба случая.*

1. При нажатии на кнопку ***«ВВОД ДАННЫХ»*** *появляются поля ввода данных для набора и сохранения новых данных. Вместе с полями появляются кнопки* ***«СОХРАНИТЬ»****,* ***«ОТМЕНИТЬ»*** *и*

«ВЫБРАТЬ»*. Кнопка* ***«ВЫБРАТЬ»*** *обеспечивает выбор пола:*

М или Ж. Должны быть заполнены все поля ввода. При нажатии кнопки ***«СОХРАНИТЬ»*** *проверяется правильность заполнения полей (всё ли заполнено) и данные сохраняются в файле «База» – выполняется запись в конец файла. Пользователь информируется о записи сообщением на экран: «Сведения добавлены». При нажатии кнопки* ***«ОТМЕНИТЬ»*** *ничего не происходит, поля ввода скрываются, и вновь на экране отображаются две исходные кнопки.*

2. При нажатии на кнопку ***«ПОИСК ДАННЫХ»*** *отображаются все поля для набора данных, кроме номера квартиры и площади жилья. Отображаются также кнопки* ***«ПОИСК»****,* ***«ВЫБРАТЬ»*** *и* ***«ОТМЕНИТЬ».*** *В поля пользователь может набрать значения в каждое поле или только в некоторые. Введенные значения являются ключами поиска информации в базе данных. При нажатии кнопки* ***«ОТМЕНИТЬ»*** *ничего не происходит, поля ввода скрываются, и вновь на экране две исходные кнопки.*

3. При нажатии на кнопку ***«ПОИСК»*** *выполняется поиск записей в файле «База». После завершения поиска на экране отображается список найденных данных в виде FIO. Список может быть и пустым, если при поиске не найдено ни одной записи. Вместе со списком отображается кнопка* ***«ЗАКРЫТЬ»****. Теперь пользователь может выбрать любой элемент списка, и на экране отображается полная информация.*

При нажатии на кнопку ***«ЗАКРЫТЬ»*** *поля ввода скрываются, и вновь на экране – две исходные кнопки.*

Интерфей спользователя

Интерфейс представлен на рис. 22. Все надписи – это объекты ***Label***.

Поля ввода – это объекты ***TextBox***. Для отображения списка найденных записей используется объект ***comboBox1***.

Надпись на кнопке ***«BUTTON3»*** изначально не сформирована. Надпись появится после выбора варианта работы: при вводе данных – надпись «Сохранить», при поиске – «Поиск». Порядок появления объектов описан в сценарии.

Заданиядлясамостоятельноговыполнения

1. В визуальном режиме установите невидимость всех объектов, кроме двух верхних кнопок.

2. Оформите метод Видим Ввод, в котором установите видимость всех надписей, полей ввода и кнопок ***«BUTTON3»,***

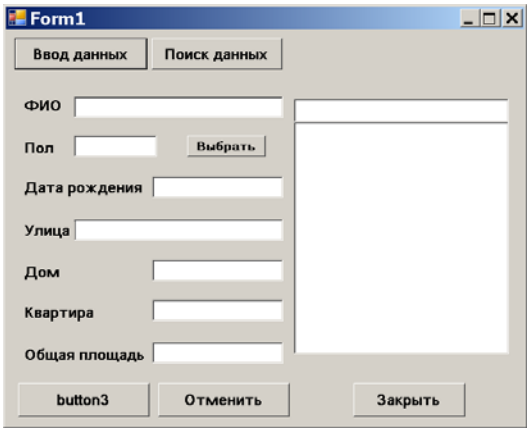


Рис. 22. Интерфейс пользователя

__«ВЫБРАТЬ»__ и __«ОТМЕНИТЬ»__. В этом же методе установите пустые значения всех полей ввода.

3. В обработчике кнопки __«ВВОД ДАННЫХ»__ запишите оператор вызова метода ВидимВвод. В button3 сформируйте надпись «Сохранить». Скройте кнопку __«ПОИСК ДАННЫХ»__, чтобы ее случайно не нажал пользователь.

4. Оформите метод СкрытьВвод, отменяющий видимость объектов (действия, противоположные методу ВидимВвод).

5. В обработчике кнопки __«ОТМЕНИТЬ»__ запишите оператор вызова метода СкрытьВвод.

6. В обработчике кнопки __«ВЫБРАТЬ»__ обеспечьте выбор пола. Значение М или Ж запишите в textBox2.

7. Оформите метод Набрано, в котором выполняется контроль полноты набора данных. Метод возвращает true, если все поля заполнены, и false в противном случае.

8. В обработчике кнопки __«СОХРАНИТЬ»__ запишите оператор вызова метода Набрано. Если метод возвращает false, то выдайте сообщение о неполном наборе на экран и завершите методобработчик (return).

Оформите метод Дата, в котором проверьте правильность заполнения поля Дата рождения. Предполагается, что оно должно быть заполнено в формате: ЧЧ.ММ.ГГГГ. И число, и месяц, и год – целые числа. Соответствие числа наименованию месяца не проверять (31 июня допускается). Метод возвращает true, если формат верен, и значения численные. В противном случае – false. Кроме того, при правильном формате метод формирует три выходных целых параметра: день, месяц и год. Значения надо сохранить в переменных.

9. В обработчике кнопки __«СОХРАНИТЬ»__ обеспечьте вызов метода Дата. Если он возвращает false, то сообщить об этом на экран и обработчик завершить.

10. В обработчике кнопки __«СОХРАНИТЬ»__ проверить правильность набора площади (численное значение). При неверном наборе сообщить об этом на экран, обработчик завершить. При правильном значении сохранить значение площади в переменной.

Теперь после всех контролей продолжим разработку алгоритма кнопки ***«СОХРАНИТЬ»***. Если все поля заполнены, то надо вывести новые данные в файл. Но файла у нас пока нет!

Определение формата файла

Определимся с форматом данных в файле. Проще всего описать новый класс объектов с полями, соответствующими вводимым данным.

Задания для самостоятельного выполнения

1. Создайте новый класс Житель с полями FIO (строка), Пол (символ), ДатаРождения (целочисленный массив), Улица (строка), Дом (строка), Квартира (строка), Площадь (вещественное число). Не забудьте, что мы будем выводить данные в файл – нужна сериализация. Да и пространства имён – тоже.

2. Включите в класс конструктор объектов, заполняющий поля объекта нулевыми (с точки зрения типа) значениями.

3. Включите в класс метод Заполнить, который заполняет поля объекта Житель.

Теперь можно в обработчике кнопки ***«СОХРАНИТЬ»*** объявить переменную-объект, создать этот объект с помощью конструктора

и заполнить его поля значениями (примерно так):

Житель T = new Житель();
T.Заполнить(textBox1.Text,textBox2.Text[0],c,m,g,textBox4.Text, textBox5.Text,textBox6.Text,p);

Здесь: c, m, g, p – это число, месяц, год рождения и площадь (получены вами ранее). Объект сформирован и готов к выводу в файл.

Создание файла и вывод данных

Теперь описываем поток, открываем файл для вывода данных в конец файла и выводим данные:

FileStream F = new FileStream("База", FileMode.Append, FileAccess.Write);
BinaryFormatter W = new BinaryFormatter(); W.Serialize(F, T);
F.Close(); СкрытьВвод();

Поиск данных Задания для самостоятельного выполнения

*Создайте метод ВидимПоиск, в котором устанавливается видимость всех надписей и полей ввода (кроме номера и площади квартиры). Отобразите кнопки **«BUTTON3»**, **«ВЫБРАТЬ»** и **«ОТМЕНИТЬ»**. В этом же методе установите пустые значения всех полей ввода.*

*1. В обработчике кнопки **«ПОИСК ДАННЫХ»** запишите оператор вызова метода ВидимПоиск. В button3 сформируйте надпись «Поиск».*

*Скройте кнопку **«ВВОД ДАННЫХ»**, чтобы случайно пользователь не смог ее нажать.*

Теперь нам надо определиться с алгоритмом поиска. Дело в том, что поиск мы заказали на ту же кнопку, которая использовалась

для сохранения данных – мы просто её переименовали. Если нажать на эту кнопку, то будет выполняться алгоритм сохранения данных – ведь обработчик события настроен именно на это. Нам нужен новый обработчик события, выполняющий поиск, но сработать он должен от той же кнопки.

Создадим вручную обработчик с похожим именем:

private void button3_Click_1(object sender, EventArgs e)
{ }

В обработчике кнопки ***«ПОИСК ДАННЫХ»*** сразу после переименования кнопки ***«BUTTON3»*** подменим обработчик события:

this.button3.Click -= new System.EventHandler(this.button3_Click);
this.button3.Click += new System.EventHandler(this.button3_Click_1);

Первый оператор операцией -= отключает прежний обработчик, второй подключает новый обработчик. Теперь в обработчике ***button3_Click_1*** можно реализовывать алгоритм поиска данных.

Задания для самостоятельного выполнения

Оформите метод НабранПоиск, в котором выполняется контроль набора данных для поиска. Метод возвращает true, если есть данные хотя бы в одном поле, и false в противном случае.

*1. В обработчике кнопки **«ПОИСК»** запишите оператор вызова метода НабранПоиск. Если метод возвращает false, то выдайте на экран сообщение об отсутствии ключей поиска и завершите метод-обработчик (return).*

Организуем поиск данных. По условию задачи, каждая запись файла, имеющая требуемые ключи, должна отображаться в списке

на экране – для этого у нас есть объект ***comboBox1***. Представим себе, что такой

список получен, и пользователь решил посмотреть данные, выбрав какую-то фамилию из списка. Данные надо будет вновь доставать из файла – ведь при поиске мы не сохраняли полные сведения нигде. Значит, в процессе поиска надо сохранять адрес данных в файле: номер байта, с которого начинаются нужные данные. Создадим в классе ***Form1*** массив:

long[] Адрес = new long[1000];

Массив ***Адрес*** является индексным массивом. У нас будет соответствие: FIO – в списке объекта ***comboBox1***, а номер байта начала данных в файле – в соответствующем элементе массива ***Адрес***.

Теперь надо организовать доступ к файлу. Переменную-поток надо разместить так, чтобы к ней был доступ не только при поиске, но и при выборе FIO – а это уже будет другой метод. Значит, поток следует создать как поле класса, сразу после объявления массива:

FileStream БД;

Откроем файл в обработчике ***button3_Click_1***:

FileStream БД = new FileStream("База", FileMode.OpenOrCreate,
FileAccess.ReadWrite);

А в конце обработчика (чтобы не забыть) сразу же запишем оператор закрытия потока: **БД.Close();**

Переходим к поиску данных в файле – все действия в том же обработчике.

Задания для самостоятельного выполнения

1. Очистите список в объекте comboBox1.

2. Создайте объект R класса BinaryFormatter для выполнения десериализации объектов и служебный объект Ж класса Житель.

*3. Сделайте видимым объект comboBox и кнопку **«ЗАКРЫТЬ»**.*

Запишите цикл считывания данных из файла (просмотр от начала до конца – по аналогии с предыдущей лабораторной работой):

```
long i=0;
while (i < БД.Length)
{ Ж = (Житель)R.Deserialize(БД);
// далее будет алгоритм сравнения ключей i = БД.Position;
}
```

После того, как в объект ***Ж*** считаны данные, можно сравнивать ключи. Но до полей объекта невозможнодобраться – они закрытые.

Задания для самостоятельного выполнения

Создайте в классе Житель свойства для доступа ко всем закрытым полям. Имена свойств составьте из имени поля с добавлением знака «подчёркивание». Например: FIO_. Поскольку пол и дата будут использоваться для сравнения, свойства должны возвращать их в виде строки.

1. Создайте метод Сравнение, в котором будет обеспечиваться сравнение заданных полей с полями введенного объекта. Метод имеет один входной параметр – объект класса Житель. Метод возвращает true, если ключи совпадают.

Примечание. Сравнивать надо только заданные ключи!

Далее запишем в обработчике операторы проверки результатов сравнения и сохранения сведений:

```
if (Сравнение(Ж)==true)
{ // далее сохранение сведений о записи в списке и в массиве
Адрес[comboBox1.Items.Count] = i; comboBox1.Items.Add(Ж. FIO_);
}
```

Задания для самостоятельного выполнения

1. Проверьте работу программы. Наберите несколько разных сведений, сохраните их и выполните поиск по разным ключам. Всё должно работать!

*2. Обработайте нажатие на кнопку **«ЗАКРЫТЬ»**. Объект comboBox1 и сама*

кнопка должны стать невидимыми.

Создайте обработчик события ***выбор из списка***. Вставьте в него такой алгоритм:

```
if (comboBox1.SelectedIndex >= 0)
{
FileStream БД = new FileStream("База", FileMode.
OpenOrCreate, FileAccess.ReadWrite); BinaryFormatter R = new BinaryFormatter();
Житель Ж;
БД.Position=Адрес[comboBox1.SelectedIndex]; // *** Ж=(Житель)
R.Deserialize(БД); БД.Close();
ВидимВвод();
textBox1.Text = Ж. FIO_; textBox2.Text = Ж.Пол_; textBox3.Text = Ж.Дата_;
textBox4.Text = Ж.Улица_;
textBox5.Text = Ж.Дом_; textBox6.Text = Ж.Квартира_; textBox7.Text =
Ж.Площадь_.ToString();
}
```

Обратите внимание на оператор, отмеченный тремя звёздочками.

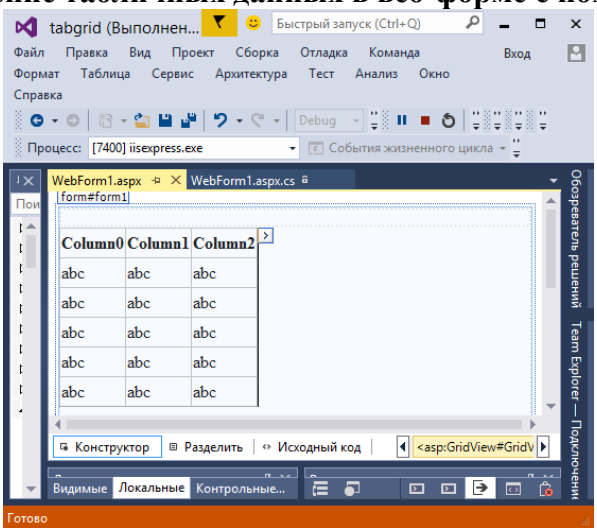
Он устанавливает файл в позицию начала нужного объекта, после чего объект считывается и с помощью свойств раскладывается по объектам ***TextBox***.

Задания для самостоятельного выполнения

*В алгоритме кнопки **«ОТМЕНИТЬ»** восстановите отображение кнопок **«ВВОД ДАННЫХ»** и **«ПОИСК ДАННЫХ»**.*

*Сейчас после выполнения поиска данных невозможно вернуться к вводу данных – ведь мы переключили кнопку **«BUTTON3»** на другой обработчик. Устраните дефект.*

Задание 7.
Отображение табличных данных в веб-форме с помощью *GridView*



Задание: Разработать проект отображения строковых массивов в виде таблицы на веб форме с помощью элемента управления **GridView** (Просмотр/обзор сетки данных в таблице) и объекта **DataTable.**

Таблица телефонов представлена в виде двух строковых массивов. Требуется написать приложение для вывода в веб-форму этих массивов в виде таблицы.

Для этого:

1.создать новый проект шаблона **Web приложение ASP.NET,** в поле **Имя** указать имя **TabGridweb**.

2. добавить к проекту веб-форму (в меню **Проект** выбрать команду **Добавить новый элемент** и в открывшемся окне выбрать шаблон **Веб форма**

3. перетащить на форму из **Панели элементов** (из подраздела **Данные)** элемент управления **GridView.**

Комментарии к программному коду:

Название страницы:**"Вывод таблицы в веб-форму";**

Содержимое первой строки:

String[] Имена = **{"Ахмед - раб", "Ахмед - дом", "Приемная ДГУ", "Кафедра ИИТ", "Далгатов-моб", "Шихнебиев Эмин"};**

Содержимое второй строки:

String[] Тлф = **{"8722-68-51-22", "8722-62-45-72", "8722-68-23-26", "8722-67-59-10", "906-482-66-55","989-665-14-41"};**

Создание обычной таблицы выполняется командой:

var Таблица = new System.Data.DataTable();

Заполнение шапки"шапки" таблицы вывода

Таблица.Columns.Add("ИМЕНА");

Таблица.Columns.Add("НОМЕРА ТЕЛЕФОНОВ");

Заполнение ячеек таблицы

for (var i = 0; i <= 5; i++)

Таблица.Rows.Add(Имена[i], Тлф[i]);

Отображение таблицы в компоненте GridView:

GridView1.Caption = "Таблица телефонов";

GridView1.BorderWidth = Unit.Pixel(2);

GridView1.BorderColor = System.Drawing.Color.Gray;

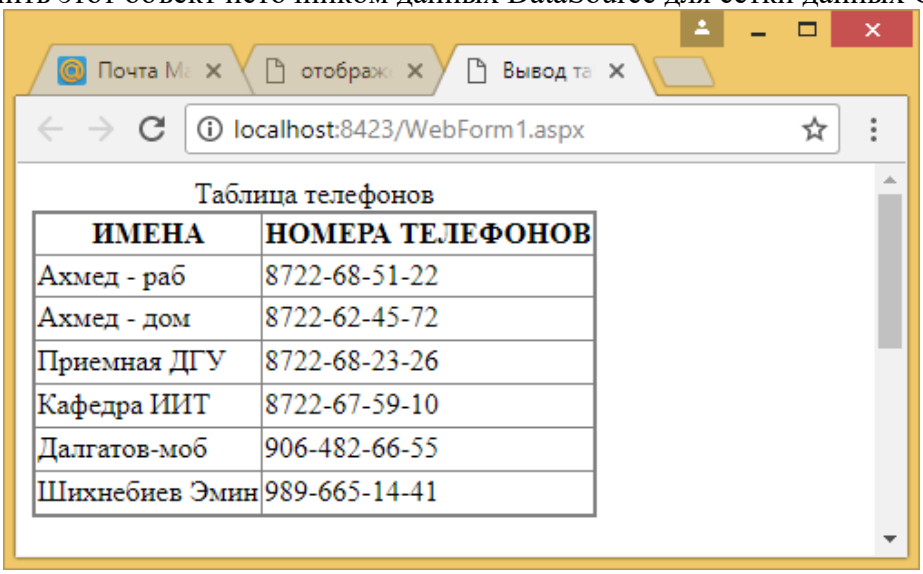
GridView1.DataSource = Таблица;

GridView1.DataBind();

Вначале нужно инициализировать два строковых массива: массив имен и массив телефонов

с помощью этих массивов заполнить ячейки объекта класса DataTable.

3. назначить этот объект источником данных DataSource для сетки данных GridView.



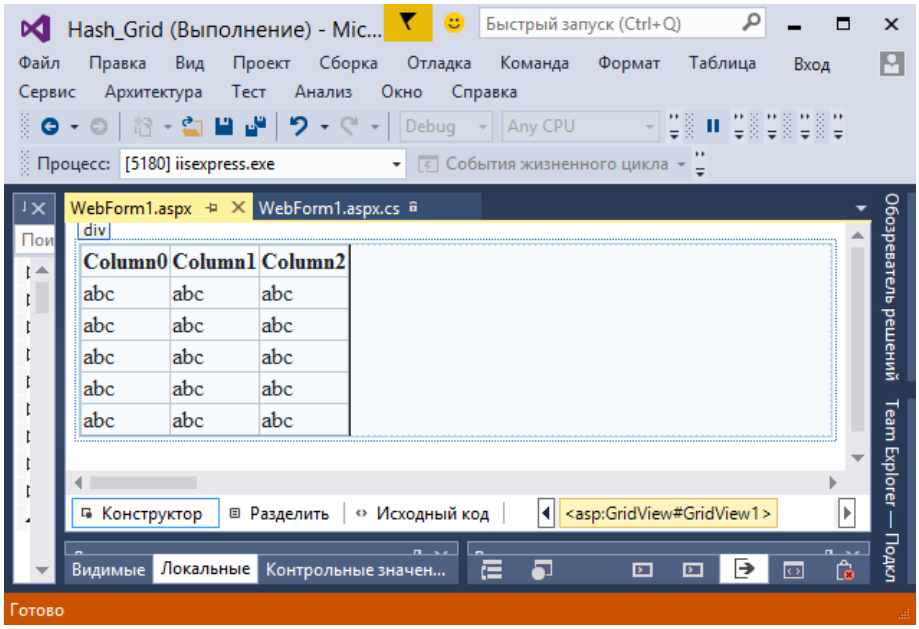
Лабораторная работа № 8. Разработка веб приложения отображения хэш-таблицы в веб-форме

Пояснение: *Хэш-таблица*- это таблица из двух столбцов, один из них содержит ключи, а второй - значения. То есть каждая строка в этой таблице образует пару "ключ - значение". Имея ключ в хэш-таблице, можно быстро найти значение. Хэш-таблицу можно назвать таблицей соответствий. Простейшим примером хэш-таблицы является таблица телефонов, которая участвовала в *задании 7*, однако там мы программировали ее просто как

два массива. Если эти два массива поместить в хэш-таблицу, то ключом в данном случае было бы ФИО, а значением - номер телефона.

При этом программирование поиска значения по ключу оказалось бы тривиальной задачей, операция добавления и удаления пары также упрощается, поскольку хэш-таблица-это объект, который содержит соответствующие эти и другие методы. В реальной жизни много разнообразных примеров представления данных в виде хэш-таблицы. Например, таблица, где расширения файлов (txt, jpg, mdb, xls) являются ключами, а соответствующими значениями - программы, которые открывают файлы с такими расширениями (Notepad.exe, Pbrush.exe, MSAccess.exe, Excel.exe). Примерами хэш-таблиц являются англо-русский и другие словари, а также база доменных имен, которая доменному имени сопоставляет IP-адрес.

Вид формы



Задание: Разработать проект решения задачи: сопоставить в хэш-таблице государства в качестве ключей, а их столицы - в качестве значений.

Используя элемент управления **GridView,** вывести эту хэш-таблицу на веб-страницу.

Для этого:

1.создать новый проект шаблона **Web приложение ASP.NET,** в поле **Имя** указать имя **Hash_Grid**.

2. добавить к проекту веб-форму (в меню **Проект** выбрать команду **Добавить новый элемент** и в открывшемся окне выбрать шаблон **Веб форма**

3. перетащить на форму из **Панели элементов** (из подраздела **Данные)** элемент управления **GridView.**

1. При обработке события загрузки веб-страницы нужно создать объект класса **Hash_table** и заполнить его парами: **"ключ - значение"**. Хэш-таблицу **"ключ - значение"** можно заполнить парами, допустимы разные формы записи:

1. через присваивание
2. посредством метода **Add.**

В качестве ключа указать Страну, в качестве значения – **Столицу**.

Создать вспомогательный объект **Таблица** класса **DataTable**, в который выгружаются данные из хэш-таблицы оператором цикла **foreach** Хэш-таблица имеет структуру типа **DictionaryEntry**, которая позволяет перемещаться по рядам в цикле и, таким образом, получить все пары из хэш-таблицы.

В цикле выполнить заполнение объекта класса **DataTable**.

Для **GridView1** указать в качестве источника данных заполненный объект **DataTable**.

Комментарии к программному коду:

Название страницы: **"отображение на веб форме хэш-таблицы";**
{
Код создания объекта Хеш:
var Хэш = new System.Collections.Hashtable();
// Заполнение хэш-таблицы:
// Можно добавлять записи

Добавлять записи в таблицу в формате "ключ-значение" можно разными способами:
1. Хэш["Белорусия"] = "Минск";
2. **Хэш.Add("Россия", "Москва");**
Нужно добавить до 15 записей

Создание обычной таблицы выполняется командой:
var Таблица = new System.Data.DataTable();

Заполнение шапки"шапки" таблицы вывода
Таблица.Columns.Add("ГОСУДАРСТВА");
Таблица.Columns.Add("СТОЛИЦЫ");
Цикл выгрузки пары "ключ-значение"из хеш-таблицы в обычную таблицу по рядам:
foreach (System.Collections.DictionaryEntry ОднаПара in Хэш) //структура DictionaryEntry определяет пару ключ-значение

Таблица.Rows.Add(ОднаПара.Key, ОднаПара.Value);

Отображение таблицы в компоненте GridView:
GridView1.Caption = "Таблица государств"; // Заголовок таблицы
GridView1.BorderWidth = Unit.Pixel(2);
GridView1.BorderColor = System.Drawing.Color.Gray;
GridView1.DataSource = Таблица;
GridView1.DataBind();

Лабораторная работа № 9. Базовые технологии доступа к БД

Цель работы: Изучить основные способы работы с наборами данных. Получить навыки проектирования несложных фактографических систем.

Указания к выполнению лабораторной работы

Ранее уже говорилось, что *наборы данных* представляют собой группы записей, переданных из базы данных в приложения для просмотра и редактирования. Каждый набор данных инкапсулирован в специальном компоненте доступа к данным. Основные свойства и методы базового класса наборов данных (**TDataSet)** уже были рассмотрены нами ранее. Кроме того, мы познакомились с основными компонентами наборов данных, взаимодействующих с базами данных посредством ADO (**ADODataTable, ADODataQuery, ADOConnection** и др.). Компоненты наборов данных предоставляют разработчику довольно широкие возможности для эффективной обработки данных. Рассмотрим некоторые из них:

- **Навигация по набору данных**. Осуществляется с помощью уже рассматривавшихся методов перемещения *курсора* (указателя текущей записи) по набору данных: **First()**,

Last(), **Next()**, **Prior()**, **MoveBy(int)**. Например, для последовательного перебора записей, начиная с текущей позиции до конца набора данных компонента **ADODataTable1**, можно использовать следующий цикл перебора:

```
ADODataTable1->Open()
...
while (!ADODataTable1->Eof)
{
      ...   // выполняются некоторые действия
      ADODataTable1->Next();
}
```

Интересным способом навигации является применение закладок. Подобно тому, как в книге можно заложить закладкой нужную страницу и впоследствии быстро к ней вернуться, также и в наборах данных можно осуществить аналогичные действия для отдельной записи. Для этого используются следующие методы набора данных:

- **void* GetBookmark()** – создает и возвращает закладку для текущей записи набора данных;
- **GotoBookmark(void *bookmark)** – перемещает курсор на запись, которая соответствует параметру bookmark (закладке).

- **Сортировка записей** является важной и довольно сложной проблемой. Можно выделить три способа определения нужного порядка следования записей:
  - *Индексирование полей таблиц БД*. Этот способ является возможным при использовании компонентов набора данных **ADODataTable**. При проектировании базы данных для каждой таблицы можно задать одно или несколько *индексированных полей*, позволяющих упорядочивать записи по значениям этого поля по возрастанию, либо по убыванию. После этого записи набора данных можно упорядочивать по любому индексированному полю, присвоив его наименование свойству IndexName. Рассмотрим, **например**, таблицу городов из базы данных, построенной для ИС «Телефонный справочник» (см. лабораторные работы № 3 и 4). Изначально список записей в наборе данных будет соответствовать действительному порядку размещения записей в таблице БД, поэтому при последовательном просмотре набора данных в программе получим:

```
Москва
Новосибирск
Санкт-Петербург
Бердск
```

Используя **конструктор таблицы** в Microsoft Access, сделаем поле city_name указанной таблицы индексированным. После этого, в качестве свойства IndexName для табличного набора данных можно указывать не только поле первичного ключа city_id, но и поле city_name. В последнем случае последовательный просмотр набора данных в программе даст упорядоченный список наименований городов:

```
Бердск
Москва
Новосибирск
Санкт-Петербург
```

  - *Определение порядка следования записей при построении SQL-запроса*. Этот способ используется при работе с компонентами **ADODataQuery** и их аналогами. Способы упорядочения записей в SQL-запросах на выборку данных будут подробно рассмотрены в курсе «Базы Данных».
  - Сортировка записей внутренними средствами компонента набора данных. Данный способ может быть использован не всегда, поскольку его применение ограничивается как свойствами компонента набора данных

(который может не поддерживать внутреннюю сортировку записей), так и свойствами базы данных, с которой взаимодействует приложение. Компонент **ADODataTable** имеют свойство **IndexFieldNames**, которому можно присваивать список наименований полей, по которым будет производиться упорядочение. Поля в списке должны быть только полями данных (вычисляимые и поля синхронного просмотра не допускаются); одно поле отделяется от другого через точку с запятой. Сортировка набора данных при непустом значении свойства **IndexFieldNames** выполняется следующим образом: сначала записи упорядочиваются по значению первого поля, затем, если эти значения совпадают, – то по значениям второго поля и так далее. Список полей может состоять из единственного поля. **Например**, можно сортировать записи рассмотренной выше таблицы городов по значению кода города. Для этого достаточно присвоить свойству **IndexFieldNames** его наименование: (city_code). При этом поле city_code не обязательно должно быть индексированным. Список записей при последовательном просмотре набора данных в этом случае примет вид:

```
Новосибирск
Бердск
Москва
Санкт-Петербург
```

- **Поиск записей** в наборе данных может выполняться двумя способами.
    - *Для позиционирования на нужной записи* используются методы **Locate** и **Lookup**:

Метод **Locate** ищет первую запись, удовлетворяющую критерию поиска, и, если такая запись найдена, делает ее текущей. Параметр **KeyFields** должен содержать список наименований одного или нескольких полей, по которым ведется поиск. В случае нескольких полей их названия разделяются точкой с запятой. Критерии поиска задаются в вариантном массиве **KeyValues** так, что *i*-тое значение в этом массиве автоматически ставится в соответствие *i*-тому полю в **KeyFields**. Параметр Options позволяет включить дополнительные необязательные режимы поиска. Метод Locate возвращает значение **true**, если запись найдена и установлена в качестве текущей, и **false** – в противном случае.

Метод **Lookup**, также как и метод **Locate**, находит запись, удовлетворяющую заданным условиям поиска, но не делает ее текущей, а возвращает значения некоторых полей этой записи. Кроме параметров **KeyFields** и **KeyValues**, смысл которых совпадает с одноименными параметрами метода **Locate**, функция метода **Lookup** содержит параметр **ResultFields**, который определяет список наименований полей, значения которых будут возвращены. Результатом метода является вариантный массив значений, причем *i*-тое значение в этом массиве соответствует *i*-тому полю списка **ResultFields**. Если запись, удовлетворяющая условиям поиска, не обнаружена, то возвращается пустое значение (*Null*()) Метод **Lookup** целесообразно использовать в тех случаях, когда изменять текущую запись нежелательно.

    - *Для фильтрации записей* по определенному критерию используются свойства **Filter**, **Filtered** и **FilterOptions**, а также событие **OnFilterRecord**.

Свойство **Filter** позволяет задать *критерий фильтрации*. В этом случае набор данных будет отфильтрован, как только его свойство **Filtered** станет равным **true**. Синтаксис описания критериев фильтрации предоставляет разработчику довольно широкие возможности, в частности:

- Использование наименований полей и константных значений;

- Использование операций сравнения (<,>,=,>=,<=,<>);
- Использование арифметических операций;
- Использование логических операций (NOT, AND, OR);
- Использование спецсимвола “*” для фильтрации по частичному соответствию.

Чтобы реализовать более сложные алгоритмы фильтрации набора данных можно использовать событие **OnFilterRecord**. Оно возникает всякий раз, когда значение свойства **Filtered** устанавливается в true. Обработчик этого события имеет два параметра: указатель на фильтруемый набор данных (приведенный к базовому классу **TDataset**) и переменную **Accept**, в которую в результате выполнения обработчика должно быть помещено либо значение **true**, ели запись удовлетворяет условиям фильтра, либо значение **false** в противном случае. Следует помнить, что при включении фильтра указанное событие вызывается *для каждой записи*, поэтому для больших наборов данных такая фильтрация может выполняться довольно долго.

Задания к лабораторной работе

1. В соответствии с вариантом задания спроектировать и реализовать простейшую фактографическую информационную систему.
2. При проектировании БД рекомендуется использовать результаты выполнения лабораторных работ 4 и 5 курса «Информационные системы».
3. При реализации программного приложения для работы с БД выполнить следующие требования:
    а. Разработку программного приложения начать с проектирования пользовательского интерфейса и подготовки прототипа приложения.
    б. Развитие проекта программного приложения осуществлять в соответствии с принципами итеративной разработки.
    в. Обеспечить возможность добавления, редактирования и удаления записей во все основные таблицы БД;
    г. Обязательно использовать компонент TDBLookupComboBox для редактирования и/или определения условий поиска информации.
    д. Обязательно использовать поля синхронного просмотра и вычисляемые поля;
    е. Обязательно использовать один из способов упорядочения набора данных;
    ж. Обязательно использовать один из способов поиска информации в наборе данных.
4. В отчет о выполнении лабораторной работы включить:
    а. Модель данных (ER-диаграмму) ИС;
    б. Перечень и содержание выполненных итераций по разработке программного приложения;
    в. Результаты тестирования программы;
    г. Выводы по проделанной работе.

Варианты заданий

1. *ИС пункта проката*. Таблица предметов содержит название предмета, количество предметов в пункте проката, стоимость одних суток проката. Таблица выданных предметов содержит наименование предмета, фамилию арендатора, дату выдачи, количество суток, на которое произведена выдача, ожидаемую дату возврата, реальную дату возврата, сумму оплаты.
   Обязательные требования к ИС :

а. Возможность получения информации о выданных предметах с дополнительным полем, в котором рассчитывается сумма оплаты за прокат. Учесть, что при выдаче предмета более, чем на десять дней, арендатору должна быть предоставлена скидка в 10%.
б. Возможность получения списка всех выданных предметов;
в. Возможность получения списка всех предметов, которые не были сданы в срок;

2. *ИС кадров предприятия*. Основная таблица содержит фамилию, отдел, оклад, дату рождения, дату приема, дату увольнения (если работает – пустое значение). Справочная таблица отделов содержит название отдела, фамилию начальника.
Обязательные требования к ИС :
а. Возможность получения информации о сотрудниках и их заработной плате. Учесть, что заработная плата сотрудников, работающих в организации более 5 лет, должна быть увеличена на 20% по сравнению с окладом.
б. Возможность получения информации об отделах с дополнительными вычисляемыми полями, в которых указывается численность отдела и общий фонд заработной платы (с надбавками).

3. *ИС склада*. Имеется справочник клиентов и справочник товаров. Справочник товаров содержит наименование, количество, цену. Справочник клиентов содержит имя и адрес клиента. Таблица фактур содержит наименование товара, приобретенное количество, наименование покупателя, дату приобретения, дату оплаты.
Обязательные требования к ИС :
а. Возможность получения информации по каждому товару с указанием стоимости каждой покупки;
б. Возможность получения информации по каждому клиенту с указанием списка неоплаченных им покупок и их количества;
в. Возможность получения списка всех неоплаченных фактур с указанием общей суммы к оплате.

4. *ИС телефонной станции*. Справочник абонентов содержит номер телефона, фамилию, адрес абонента. Справочник городов содержит коды городов, названия городов и зону. Тарифный справочник определяет стоимость одной минуты разговора в зависимости от зоны. Основная таблица содержит дату переговоров, время переговоров, дату оплаты (если отсутствует, то разговор не оплачен), номер телефона абонента, код города, продолжительность разговора.
Обязательные требования к ИС :
а. Возможность получения информации о междугородних переговорах с указанием стоимости звонка. Учесть, что если звонок был произведен в ночное время (от 22:00 до 06:00), абоненту предоставляется скидка в 50%;
б. Возможность получения списка городов из самой дорогой тарифной зоны.

5. *БД радиостанции*. Справочники звукооператоров и ведущих содержат фамилии, нормированный оклад (за 20 часов в эфире в месяц), дату принятия на работу. Основная таблица содержит название передачи, дату и время выхода в эфир, продолжительность в часах, фамилии звукооператора и ведущей.
Обязательные требования к ИС :
а. Возможность получения информации о выходах сотрудников радиостанции в эфир в течение текущего месяца (с указанием оплаты за каждый выход). Учесть, что расчет оплаты производится по вычисленному в соответствии с

окладом часовому тарифу с 50% надбавкой за работу в ночные часы (22:00-8:00) и с 10% надбавкой за стаж работы более 5 лет.
   б. Возможность получения информации о передачах, которые выходили в заданное время заданного дня недели.

6. *ИС библиотеки*. Основный справочник книг содержит шифр книги, фамилию автора, название книги, код УДК, стоимость, количество экземпляров Каталог читателей содержит номер билета, фамилию читателя, домашний адрес. Основная таблица выданных книг содержит шифр книги, номер билета, дату выдачи, дату фактического возврата (если книга на руках, то пустое значение), дату обязательного возврата (2 недели от даты выдачи).
   Обязательные требования к ИС:
   а. Возможность получения информации о выданных книгах с указанием количества просроченных дней и суммы штрафа. Учесть, что начисление штрафа начинается с 3-го дня просрочки и составляет 5% от стоимости книги.
   б. Возможность получения информации о количестве выданных и количестве просроченных книг для некоторого читателя.

7. *ИС деканата*. Справочник «Студенты» содержит номер студенческого билета, ФИО студента, номер группы. Справочник «Дисциплины» - шифр и название предмета. Таблица «Успеваемость» содержит номер студенческого билета, название дисциплины и полученные оценки.
   Обязательные требования к ИС :
   а. Возможность получения информации о дисциплинах, которые успешно сданы определенным студентом;
   б. Возможность получения информации о дисциплинах с указанием тех, по которым у студентов есть долги;
   в. Возможность получения информации о списке и количестве студентов определенной группы;

8. *ИС страховой компании*. Справочник видов страхования содержит наименование предметов страхования (недвижимость, транспорт), возможный срок страхования и процент страховой суммы от стоимости предмета. Основная таблица содержит фамилию страхователя, название предмета, оценочную стоимость, код вида страховки, дату страхования, дату наступления страхового случая (если он произошел), ежемесячный взнос, дата внесения последнего платежа.
   Обязательные требования к ИС :
   а. Возможность получения информации о количестве действующих договоров страховки определенного вида.
   б. Возможность получения информации о договорах страхования с указанием полной информации о виде страховки и размере страховой суммы;
   в. выдача информации о договорах страхования, по которым наступил страховой случай.

9. *ИС турагентства.* Справочник туров содержит информацию о курорте, программе тура, базовой стоимости тура. Таблица заездов содержит информацию о возможных датах заезда для каждого из тура, а также цену путевки для тура с указанной датой как процент от базовой стоимости тура. Журнал заказов содержит фамилию, имя, отчество клиента, код тура, код заезда.
   Обязательные требования к ИС :
   а. Возможность получения информации о турах, в которые можно отправиться в ближайшие 2 недели, – с указанием стоимости путевки;

б. Возможность получения информации о списке и количестве путешественников, отправляющихся в определенный заезд;
в. Возможность получения информации о путешественниках, которые отправлялись более чем в один тур.

10. *ИС банка.* Справочник вкладов содержит информацию о наименовании вклада, минимальном размере, сроке действия договора, процентной ставке по вкладу. Каталог клиентов содержит информацию о вкладчике (ФИО, паспортные данные, адрес), тип вклада, сумму вклада, дату заключения договора.
Обязательные требования к ИС :
    а. Возможность получения информации о клиентах, срок действия договора с которыми истек – с указанием суммы процентов и общей суммы возврата.
    б. Возможность получения информации о вкладчиках, заключивших договор в течение последнего месяца, упорядочив список по убыванию суммы.
    в. Возможность получения информации о денежных поступлениях и выплатах банка за последний месяц с указанием итогового сальдо.

11. *ИС супермаркета.* Справочник товаров содержит информацию о коде товара (штрих-код), его наименовании, цене, критическом количестве оставшихся товаров. Справочник операций содержит сведения о коде, наименовании и знаке операции: «+» – поступление товара, «–» – расход товара (например, совершение покупки), «0» – прочие операции. Журнал операций содержит сведения о типе операции, коде и количестве товара, участвующего в операции, номере обосновывающего документа (например, кассового чека).
Обязательные требования к ИС :
    а. Возможность получения информации о товарах, запасы которых необходимо пополнить (фактическое количество в магазине меньше критического)
    б. Возможность получения информации об операциях, связанных с расходом любого выбранного товара с указанием стоимости каждой операции;
    в. Возможность получения информации о самой крупной покупке любого выбранного товара за день;
    г. Возможность получения информации о самой крупной покупке, совершенной одним клиентом.

Контрольные вопросы

1. Понятие набора данных.
2. Класс **TDataSet** и производные компоненты наборов данных.
3. Навигация по набору данных.
4. Способы упорядочения записей в наборе данных.
5. Поиск данных методом **Locate**. Необязательные режимы поиска.
6. Зависит ли результат выполнения метода **Locate** от способа упорядочения записей. Ответ обосновать.
7. Поиск данных методом **Lookup**.
8. Фильтрация данных. Синтаксис строки критерия фильтрации.
9. Свойство **FilterOption**.
10. Фильтрация данных с использованием события **OnFilterRecord**.

5. **Образовательные технологии**

Основными образовательными технологиями проведения курса «Технологии программирования» являются:

- Лекции, сопровождаемые компьютерными презентациями;
- лабораторные работы, в рамках которых составляются и тестируются программы, иллюстрирующие теоретический материал лекций;
- самостоятельная работа студентов, включающая усвоение теоретического материала, поиск дополнительного материала и эффективных способов выполнения заданий, завершение выполнения лабораторных работ; оформление и подготовка к защите лабораторных работ, подготовка к текущему контролю знаний и к итоговому экзамену;
- разработанные индивидуальные задания для самостоятельной работы;
- рейтинговая технология контроля учебной деятельности студентов для обеспечения их ритмичной работы в течение семестра
- консультирование студентов по вопросам учебного материала и выполнения курсового заданий.

6. Учебно-методическое обеспечение самостоятельной работы студентов

Таблица – Технологическая карта самостоятельной работы студента

№	Темы дисциплины	Задания для самостоятельной работы	Трудоёмкость задания, часы	Перечень учебно-методического обеспечения
1.	Проблемы разработки сложных программных систем	Выполнить тест №1	2	Работа с источниками 2, 3.
2.	Жизненный цикл программного обеспечения	Выполнить тест №2	*4*	Работа с источниками 1, 3, 5.
3.	Унифицированный процесс разработки программного обеспечения	Выполнить тест №3	*6*	Использовать источники 1, 4, 5 и Интернет-ресурсы.
4.	Экстремальное программирование	Выполнить тест №4	8	Использовать источники 4, 6 и Интернет-ресурсы
5.	Анализ предметной области	Выполнить тест №5	8	Использовать источники 1,4, 5 и Интернет-ресурсы
6.	Качество программного обеспечения	Выполнить тест №6	8	Использовать источники 3,4, 5 и Интернет-ресурсы
7.	Подготовка к экзамену. Сдача экзамена.		36	Все темы курса

2.2 Контроль результатов освоения дисциплины

Текущий контроль успеваемости осуществляется путем оценки результатов выполнения заданий лабораторных, самостоятельной работ, посещения лекций.

Промежуточная аттестация осуществляется в форме экзамена, который выставляется по результатам проверки выполнения тестов и заданий.

Оценочные средства результатов освоения дисциплины, критерии оценки выполнения заданий представлены в разделе «Фонды оценочных средств для проведения промежуточной аттестации» и фонде оценочных средств образовательной программы.

7. Фонд оценочных средств для проведения текущего контроля успеваемости, промежуточной аттестации по итогам освоения дисциплины

7.1. Перечень компетенций с указанием этапов их формирования в процессе освоения образовательной программы.

Перечень компетенций с указанием этапов их формирования приведен в описании образовательной программы.

Код компетенции из ФГОС ВО	**Наименование компетенции из ФГОС ВО**	**Планируемые результаты обучения**	**Процедура освоения**
ОК-8	способностью к самоорганизации и самообразованию	Знает: теоретические основы и источники информационных технологий Умеет: использовать информационные технологии для организации учебы, труда в повседневной жизни Владеет: техническими способами и методами получения новой информации	Устный опрос, письменный опрос
ОПК-4	способностью понимать значение информации в развитии современного общества, применять информационные технологии для поиска и обработки информации	Знает: различные способы, методы, принципы создания, преобразования, поиска, хранения и передачи информации Умеет: применять программные и аппаратные способы, методы, принципы для получения информации в новом качестве Владеет: навыками и применять имеющиеся информационные технологии на практике	Устный опрос, письменный опрос
ПК-1	способностью выполнять работы по установке, настройке и обслуживанию программных, программно-аппаратных (в том числе криптографических) и технических средств защиты информации	Знает: принципы организации информационных систем в соответствии с требованиями по защите информации. Умеет: анализировать и оценивать угрозы информационно безопасности объектов, использовать программные и аппаратные средства современного компьютера. Владеет: методами	Устный опрос, письменный опрос

		установки и настройки программно- аппаратных и технических средств защиты информации	

7.3. Типовые контрольные задания

1. Задание {{ 1 }} ТЗ № 1

Признаки "небольшой" программы

□□решение четко поставленной, несущественной для практической деятельности задачи

□□решение одной или нескольких значимых для пользователей задач, не имеющих четкой постановки

□□периодически требуется доработка программы с появлением новых версий

□□для выполнения своих задач программа должна взаимодействовать с другими программами

□□есть существенная необходимость в документировании программы

2. Задание {{ 2 }} ТЗ № 2

Признаки "небольшой" программы

□□отсутствует необходимость в документировании программы

□□низкая производительность приносит существенный ущерб

□□для выполнения своих задач программа должна взаимодействовать с другими программами

□□в разработку вовлечено большое количество людей

3. Задание {{ 3 }} ТЗ № 3

Признаки сложной программной системы (программного комплекса)

□□для выполнения своих задач программа должна взаимодействовать с другими программами

□□в разработку вовлечено большое количество людей

□□неправильная работа программы наносит ощутимый ущерб

□□отсутствует необходимость в оптимизации производительности программы

4. Задание {{ 4 }} ТЗ № 4

Свойства сложных программных систем

□□низкая производительность приносит существенный ущерб

□□требуется документация для обучения пользователей

□□отсутствие проектной документации

□□ущерб от неправильной работы программы незначителен

5. Задание {{ 5 }} ТЗ № 5

Свойства сложных программных комплексов

□□удобство в использовании программы носит существенный характер

□□наличие проектной документации

□□в разработке участвует один человек

□□система решает одну четко поставленную задачу

6. Задание {{ 6 }} ТЗ № 6

Виды документации, требуемой для эксплуатации и развития программной системы

□□пользовательская

□□технический

□□инженерный

□□технологический

8. Задание {{ 8 }} ТЗ № 8

Системная инженерия изучает следующие аспекты создания программно-аппаратных систем …

□□разработка программно-аппаратных систем

□□эксплуатация программно-аппаратных систем

□□интеграция программной и аппаратной составляющих

□□разработка аппаратных устройств

9. Задание {{ 9 }} ТЗ № 9

Аспекты организации экономически эффективной работы

□□организация совместной работы коллектива разработчиков

□□учет требований к пользовательским свойствам программы

□□учет квалификации пользователя при проектировании пользовательских интерфейсов

□□создание безошибочно работающего программного продукта

10. Задание {{ 10 }} ТЗ № 10

Объективные причины отсутствия "безошибочно работающих" сложных программных систем

□□противоречие требований друг другу

□□изменение требований с течением времени

□□сложность поиска ошибок в коде программы

□□сложность исправления найденных ошибок

11. Задание {{ 11 }} ТЗ № 11

Сложные программные системы с точки зрения наличия в них ошибок условно делятся на …

□□достаточно качественные

□□недостаточно качественные

□□правильные

□□неправильные

12. Задание {{ 12 }} ТЗ № 12

Основные проблемы разработки сложных программных систем связаны с нахождением разумного компромисса между затратами на разработку и … ее результата

Правильные варианты ответа: качеством; качество;

13. Задание {{ 13 }} ТЗ № 13

К наиболее важным ресурсам при оценке затрат на программу относятся …

□□время выполнения проекта

□□бюджет проекта

□□персонал

□□стоимость оборудования

14. Задание {{ 14 }} ТЗ № 14

Функциональные возможности, надежность, гибкость, удобство внесения изменений являются аспектами … программной системы

Правильные варианты ответа: качества; качество;

15. Задание {{ 15 }} ТЗ № 15

К процессам создания программных систем относятся понятия …

□□жизненный цикл

□□качество

□□процесс разработки

□□разработка документации

7.3. Методические материалы, определяющие процедуру оценивания знаний, умений, навыков и (или) опыта деятельности, характеризующих этапы формирования компетенций.

Общий результат выводится как интегральная оценка, складывающая из текущего контроля - 30% и промежуточного контроля - 70%.

Текущий контроль по дисциплине включает:

- посещение занятий - _0__ баллов,
- участие на практических занятиях - 20 баллов,
- выполнение лабораторных заданий – 60 баллов,

- выполнение домашних (аудиторных) контрольных работ – 20 баллов.

Промежуточный контроль по дисциплине включает:

- устный опрос - 30 баллов,
- письменная контрольная работа - 30 баллов,
- тестирование - 40 баллов.

8. Перечень основной и дополнительной учебной литературы, необходимой для освоения дисциплины.

а) основная литература:

1. **Гагарина Л. Г.** Технология разработки программного обеспечения : [учеб. пособие] / Гагарина, Лариса Геннадьевна, Е. В. Кокорева ; под ред. Л.Г.Гагариной. - М. : ФОРУМ: ИНФРА-М, 2009, 2008. - 399 с. - (Высшее образование). - Допущено УМО. - ISBN 978-5-8199-0342-1 (ИД "ФОРУМ") : 246-84.
2. Савич, Уолтер. Программирование на C++ : [Перевод] / Савич, Уолтер. - СПб. и др. : Питер: Питер принт, 2004. - 779 с. ; 24 см. - ISBN 5-94723-582-X : 380-00..
3. **Бройдо В.Л.** Вычислительные системы, сети и телекоммуникации : учебник для вузов / В. Л. Бройдо, О. П. Ильина. - СПб.[и др.] : Питер, 2011, 2003. - 440-00.
4. **Макарова Н. В.** Информатика : учеб. для вузов: [для бакалавров] / Макарова, Наталья Владимировна, В. Б. Волков. - СПб. [и др.] : Питер, 2013, 2011. - 573 с. - (Учебник для вузов). - Рекомендовано УМО. - ISBN 978-5-496-00001-7 : 441-00.
5. **Информатика: Базовый курс** : учеб. для вузов: [для бакалавров и специалистов] / под ред. С.В.Симоновича. - 3-е изд. - СПб. [и др.] : Питер, 2011, 2009. - 637 с. - (Учебник для вузов). - Рекомендовано МО РФ. - ISBN 978-5-459-00439-7 : 419-00.

б) дополнительная литература

1. Терехов А., Ложечкин А. Microsoft Solutions Framework 4.0 – опыт Microsoft по организации командной разработки. Презентация с Microsoft Платформа 2006
2. Анашкина Н.В., Петухова Н.Н., Смольянинов В.Ю. Технологии и методы программирования
3. Г. Буч, Дж. Рамбо, А. Джекобсон. UML. Руководство пользователя. – ДМК-Пресс, Питер, 2004.
4. Г. Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. Второе издание. – Бином, 1998.
5. Эндрю Троелсен. Язык программирования C# 7 и платформы. NET и NET Core. Пер. с англ. - М.: "Вильямс", 2018. 1328с.
6. Эндрю Троелсен. Язык Программирования C# 5.0 и платформа .NET 4.5. Пер. с англ. - М.: "Вильямс", 2015. 1312с.
7. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 2.0 на языке C#. Пер. с англ. - М.: «Русская Редакция» ; СПб. : Питер , 2007. 656 стр.
8. Акчурин Э.А. Программирование на языке C# в MS Visual Studio .Net или SharpDevelop. Учебное пособие. Самара, ИУНЛ. ПГУТИ, 2011, 150 с.
9. Жоголев А.А. Технологии программирования. Компонентный подход. – М.: Научный мир, 2008
10. Иванова Г. С Технология программирования: Учебник для вузов Изд. 3-е, перераб., доп. 3-е, стереотип. / Иванова Г. С. - М.: Изд-во МГТУ им. Н.Э. Баумана, 2008
11. Кулямин В.В. Технологии программирования. Компонентный подход. – СПб.: Питер, 2014 г.
12. Модель проектной группы MSF. Белая книга, 2003, перевод eLine Software.
13. Модель процессов MSF. Белая книга, 2003, перевод eLine Software.

9. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины.

1. ЭБС IPRbooks: http://www.iprbookshop/ru/
2. Электронно-библиотечная система «Университетская библиотека онлайн»(архив):www.biblioclub.ru
3. Единое окно доступа к образовательным ресурсам. http://window.edu.ru/
4. http://www.microsoft.com/msf
5. http://www.uml.org
6. http://www.wikipedia.org
7. http://www.wikipedia.org
8. MSF for Agile Software Development Process Guidance: [http://go.microsoft.com/fwlink/?linkid=63524]
9. Алистер Кокбёрн. Каждому проекту своя методология: [http://software-testing.ru/lib/cockburn/methodology-per-project.htm] [http://alistair.cockburn.us/index.php/Methodology_per_project]).
10. С. Якимчук. MSF – философия создания IT-решений или голые амбиции лидера, 2004: [http://www.citforum.ru/SE/project/msf/].

10. Методические указания для обучающихся по освоению дисциплины.

Критерии и показатели сформированности компетенций

Степень (уровень) сформированности компетенций на этапе изучения дисциплины «Технологии программирования» оценивается по следующим критериям: мотивационно-ценностный, когнитивный, операционно-деятельностный. Показателями критериев являются результаты обучения по дисциплине (дескрипторы) таблицы 1. Инструментарий, этапы измерения показателей и критериев компетенции представлены в таблицах.

Критерии и показатели сформированности компетенции ОПК-1

Критерии сформированности компетенции	Способы оценки	
	Этапы контроля	Средства оценки
Мотивационно-ценностный критерий	2, 5, экзамен	1
Когнитивный критерий	1, 2, экзамен	1
Операционно-деятельностный критерий	2	1
	2, 5, экзамен	1
Интегральная оценка	Экзамен	

Критерии и показатели сформированности компетенции ПК-36

Критерии сформированности компетенции	Способы оценки	
	Этапы контроля	Средства оценки
Мотивационно-ценностный критерий	3, экзамен	1
Когнитивный критерий	2, 3	1
Операционно-деятельностный критерий	2, 3, экзамен	1
	3, экзамен	
Интегральная оценка	Экзамен	1

Критерии и показатели сформированности компетенции ПК-37

Критерии сформированности компетенции	Способы оценки	
	Этапы контроля	Средства оценки

Мотивационно-ценностный критерий	4, 6, 7, экзамен	1
Когнитивный критерий	4, 5, 6, 7	1
Операционно-деятельностный критерий	4, 5, 7	1
	6, экзамен	
Интегральная оценка	Экзамен	1

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ ДЛЯ ПРОВЕДЕНИЯ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ

Этапы контроля: раздел 2 (самостоятельная работа), раздел 3 (самостоятельная работа), раздел 4 (самостоятельная работа), раздел 5 (самостоятельная работа), раздел 6 (самостоятельная работа), раздел 7 (самостоятельная работа), экзамен.

Время на выполнение: 60 мин.

Метод оценивания: автоматизированный

Критерии оценки результатов выполнения: менее 50% правильных ответов - неудовлетворительно, менее 65% - удовлетворительно, менее 86% хорошо, 86% и более – отлично.

11. Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине, включая перечень программного обеспечения и информационных справочных систем.

Информационные технологии

Образовательный процесс осуществляется с применением локальных и распределенных информационных технологий (таблица 4, 5).

Таблица 4 – Локальные информационные технологии

Группа программных средств	Наименование программного продукта
Офисные программы	Microsoft Office
	Libre Office
Распознавание текста и речи	ABBYY FineReader 2010
Средства разработки	MicroSoft Visual Studio 2015 MicroSoft SQL Server 2012
Методические указания и материалы по видам занятий	Акчурин Э., Ильин А. Программирование на языке C#. ЛР в ИСР Visual C# 2010 Express или SharpDevelop. . Самара, ИУНЛ. ПГУТИ, 2011, 114 с.

Таблица 5 – Распределенные информационные технологии

Группа	Наименование
Система тестирования	Система сетевого компьютерного тестирования ДГУ www.ts.icc.dgu.ru
Библиотеки и образовательные ресурсы	Электронная библиотека ДГУ http://www.elib.dgu.ru
	Кафедральные сайты ДГУ http://cafedra.dgu.ru
	Сайте электронных образовательных ресурсов ДГУ http://eor.dgu.ru
Система электронного обучения	Сервер электронного обучения moodle http://moodle.dgu.ru

12. Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине.

Таблица 6 – Материально-техническая база

Помещения для	Перечень основного оборудования	

осуществления образовательного процесса	(с указанием кол-ва посадочных мест)	Адрес (местоположение)
Аудитории для проведения лекционных занятий		
Лекционные аудитории	Интерактивная доска, ноутбук; проектор. Количество посадочных мест – 30.	Ауд. 3-14, 4-16, 2-10, учебный корпус № 83, г.Махачкала, ул. Джержинского, 12.
Аудитории для проведения лабораторных занятий, контроля успеваемости		
Компьютерный класс	Компьютеры с выходом в Интернет и доступом в электронную информационно-образовательную среду вуза. Количество посадочных мест – 15.	Компьютерный зал № 1 учебный корпус № 3, г.Махачкала, ул. Джержинского, 12.
Помещения для самостоятельной работы		
Компьютерные классы	Компьютеры с выходом в Интернет и доступом в электронную информационно-образовательную среду вуза. Количество посадочных мест – 15+12=27.	Компьютерный зал № 2, № 3 учебный корпус № 3, г. Махачкала, ул. Джержинского, 12.
Читальный зал библиотеки ДГУ	Компьютеры с выходом в Интернет и доступом в электронную информационно-образовательную среду вуза. Количество посадочных мест – 30.	Электронный читальный зал научной библиотеки ДГУ, г. Махачкала, ул. Батырая, 4