МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра информационных систем и технологий

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О. Г. Локтионова
«15» 12 2017 г.

"Визульное программирование "

Методические указания по выполнению
лабораторных работ по дисциплине
«Визуальное программирование»
для студентов направления подготовки бакалавров
09.03.02 Информационные системы
09.03.03 Прикладная информатика

Курск 2017

УДК 004.82 (075.8)
Составитель: Т.И.Лапина

Рецензент
Доктор технических наук, профессор *Р.А.Томакова*

Визуальное программирование: методические указания по выполнению лабораторных работ / Юго-Зап. гос. ун-т; сост.: Т. И. Лапина, Курск, 2017. 46 е.: ил. 31, табл. 4, Библиогр.: с. 13.

Содержат краткие теоретические сведения о методах разработки приложений при проектированию средств информатизации объектов профессиональной деятельности в различных областях.

Методические указания соответствуют требованиям программ по направлениям подготовки бакалавров: 09.03.02 Информационные системы, 09.03.03 Прикладная информатика.

Предназначены для студентов направления подготовки бакалавров 09.03.02 Информационные системы, 09.03.03, Прикладная информатика дневной и заочной форм обучения.

Текст печатается в авторской редакции

Подписано в печать Формат 60x84 1/16.
Усл. печ. л. . Уч. – изд. л. .Тираж 100 экз. Заказ. Бесплатно.
Юго - Западный государственный университет.
305040, г. Курск. ул. 50 лет Октября, 94.

1. Лабораторная работа № 1.
Знакомство со средой Microsoft Visual Studio

Цель работы:

Получение навыков работы со средой Microsoft Visual Studio при разработке приложений с визуальным интерфейсом.

2.1. Порядок выполнения лабораторной работы

Изучить теоретический материал и среду проектирования.

Автоматизация информационных процессов в настоящее время представляется, в первую очередь, разработкой программного приложения с графическим интерфейсом пользователя (GUI), управляющего потоками данных.

Графический интерфейс пользователя (Graphical User Interface, GUI) это система средств для взаимодействия пользователя с устройством, основанная на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана (окон, кнопок, полос прокрутки и т. п.).

Чаще всего элементы интерфейса в GUI реализованы на основе метафор и отображают их назначение и свойства, что облегчает понимание и освоение программ неподготовленными пользователями. Таким образом, работа пользователя осуществляется с экранными формами, содержащими объекты управления и панели инструментов с кнопками действий для обработки.

1.1. Теоретические сведения

Стандартный графический интерфейс пользователя должен отвечать ряду требований:

- поддерживать информационную технологию работы пользователя с программным продуктом;
- ориентироваться на конечного пользователя, который общается с программой на внешнем уровне взаимодействия;

- удовлетворять принципу «шести», когда в одну линейку меню включают не более 6 понятий, каждое из которых содержит не более 6 опций;

– сохранять стандартизованное назначение графических объектов и, по возможности, их местоположение на экране.

В объектно-ориентированном программировании мы имеем дело с классами и объектами. Объекты – это составные типы данных: они объединяют несколько значений в единый модуль и позволяют нам записывать и сохранять эти значения по имени. Другими словами, объект – это неупорядоченная коллекция свойств, каждое из которых имеет имя и значение. При разработке не консольных приложений, основным понятием является Форма.

Форма – это контейнер для размещения элементов управления среды разработки.

Свойства – возможность получения доступа к информации, которая хранится в этом элементе.

Методами называют набор действий, которые может совершать объект.

Событие – действие, распознаваемое объектом (например, щелчок мышью, нажатие клавиши), для которого можно запрограммировать отклик, т.е. реакцию объекта на произошедшее событие.

После запуска Visual Studio выбираем *Файл → Создать→ Проект*, далее выбираем пункт *CLR* отмечаем *Приложение Windows Forms*, даем имя проекта, к примеру Factorial и нажимаем OK.

Рисунок 1 – Начальная страница Microsoft Visual Studio

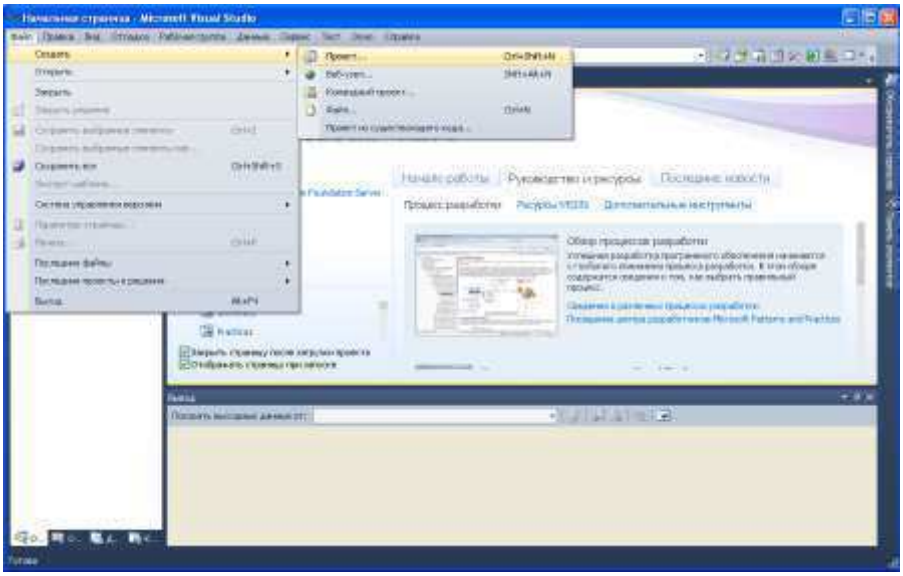


Рисунок 2 – Создание проекта

Не забудьте, указать имя проекта. Назовем проект Factorial (функционал в проект будет добавлен на практической работе).

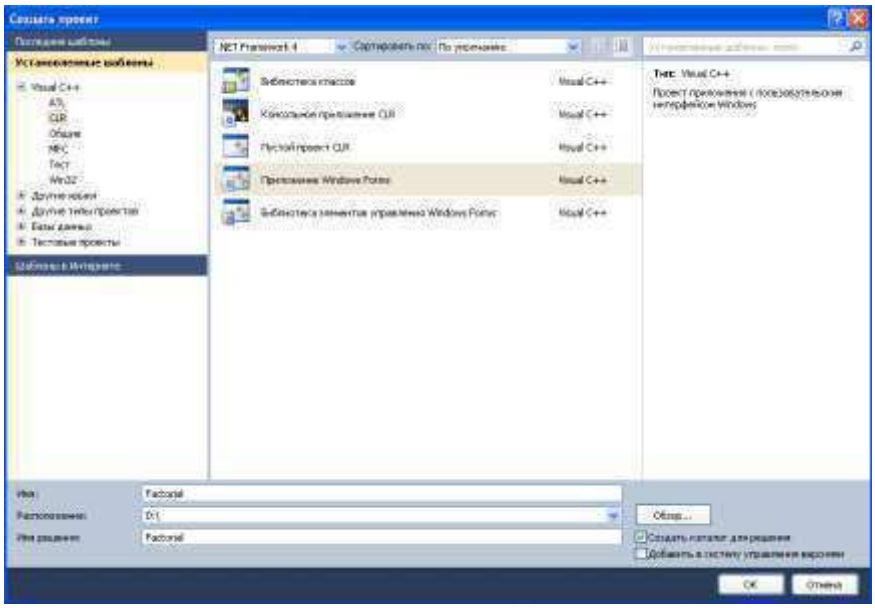


Рисунок 3 – Задание имени проекта

После нажатия на кнопку ОК, появиться пустая Форма:

Рисунок 4– Пуста Форма

Для добавления различных элементов на Форму, необходимо открыть Панель элементов. Для этого нужно выбрать раздел меню *Вид → Панель элементов*, либо нажать сочетание клавиш *Ctrl+ Alt+ X.*

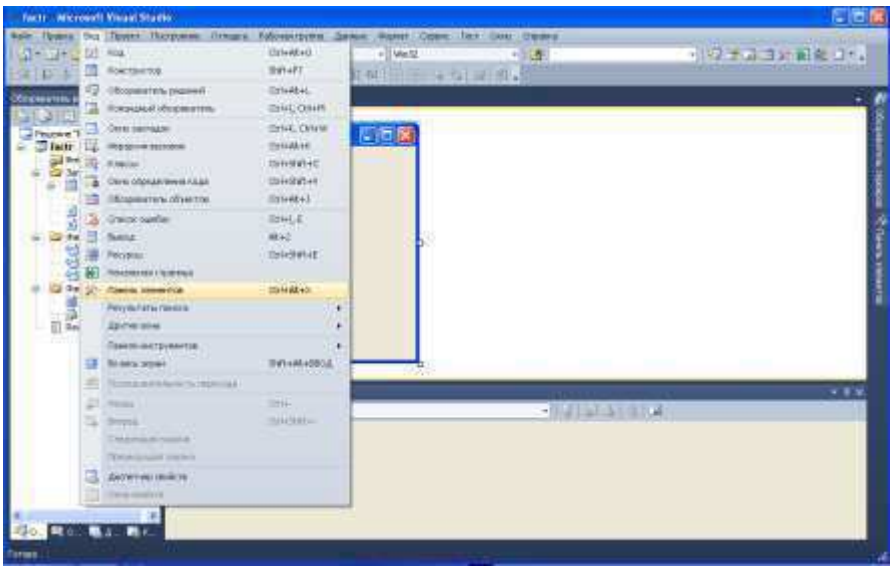


Рисунок 5 – Открытие Панели элементов

Обычно, *Панель элементов* расположена справа.

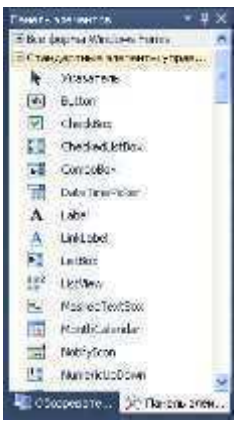


Рисунок 6– Панель элементов

Для удобства создания приложения, закрепите Панель элемент и Панель свойств справа.

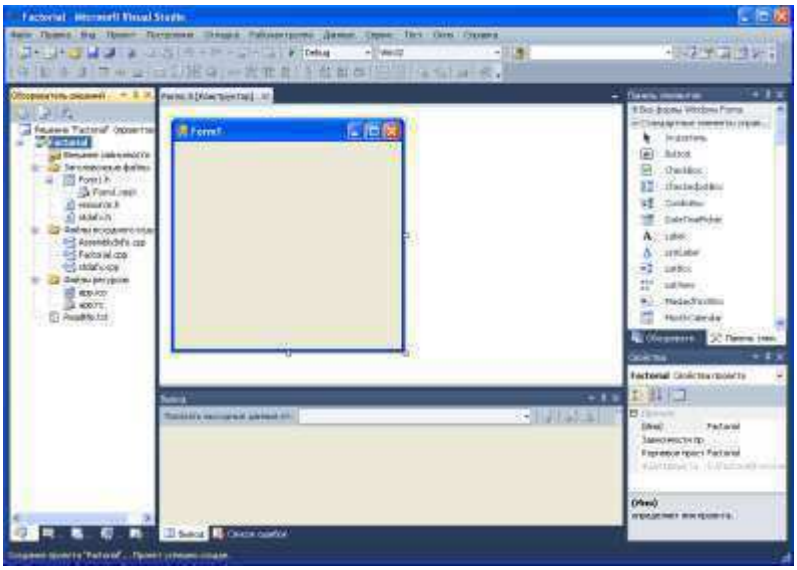


Рисунок 7– Рабочий вид среды разработки приложения

1.3. Элементы управления

Форма задает интерфейс будущего приложения. Он должен быть понятным для пользователя. Рассмотрим некоторые свойства *Form*.

Таблица 1 – Свойства Form

Свойство	Описание свойства	Подсвойства	Описание подсвойств
BackColor	Цвет фона окна	Другой	Произвольный цвет
		Интернет	Веб-цвета
		Система	Системные
Cursor	Вид курсора	При нажатии на выпадающий список появляются различные виды курсора мыши, которые отображаются при выполнение приложения	
Font	Шрифт текста	Name	Название шрифта
Text	Название Формы	-	-

Size	Размер окна, его ширина и высота	-	-
Icon	Иконка формы	-	-
Padding	Отступы от краев окна	All	Отступ со всех сторон
		Left	Отступ слева
		Top	Отступ сверху
		Right	Отступ справа
		Botoom	Отступ снизу

Таблица 2– Методы Form

Метод	Описание метода
Close()	Закрытие Формы
Hide()	Установление режима «невидимый» для Формы
Show()	Вывод Формы на экран
ShowDialog()	Вывод Формы в модальном режиме
Dispose()	Удаление Формы, освобождение занятой ею части памяти
Focus()	Активация Формы

Таблица 3 – Основные события Form

Событие	Описание событие
Activated	Возникает при активизации Формы
Click	Возникает при щелчке мышью на Форме
Load	Возникает перед первым выводом Формы

Button (кнопка) служит для выполнения действий с помощью мыши. Рассмторим некоторые свойства Button.

Таблица 4 – Свойсвта Button

Событие	Описание событие
AutoEllipsis	Получает или задает значение, указывающее, отображается ли знак многоточия (...) в правом углу элемента управления, обозначающий, что текст элемента управления выходит за пределы указанной длины этого элемента
AutoSize	Получает или задает значение, указывающее, основано ли изменение размеров элемента управления на его содержимом
Capture	Возвращает или задает значение, определяющее, была ли мышь захвачена элементом управления
DialogResult	Возвращает или задает значение, возвращаемое в родительскую форму при нажатии кнопки
Dock	Возвращает или задает границы элемента управления, прикрепленные к его родительскому элементу управления, и определяет способ изменения размеров элемента управления с его родительским элементом управления
FlatAppearance	Возвращает внешний вид границ и цвета, используемые для определения состояния флажка и состояние мыши
FlatStyle	Получает или задает плоский внешний вид для кнопки
Image	Получает или задает изображение, отображаемое в кнопке

Таблица 5 – Методы Button

Метод	Описание метода
Hide()	Установление режима «невидимый» для кнопки
Show()	Вывод кнопки на экран
Select()	Активирует элемент управления
Focus()	Задание фокуса ввода элемента управления

Таблица 6 –События Button

Событие	Описание события
Click	При щелчке элемента управления
Enter	При входе в элемент управления
MouseHover	При задержание мыши на элементе управления
MouseLeave	При убирание мыши с элемента управления

Рассмотрим простейший пример активации кнопок. Создадим Форму «Пример БПО-13-01». На ней создадим две кнопки: «Приветсвие» и «Закрыть».

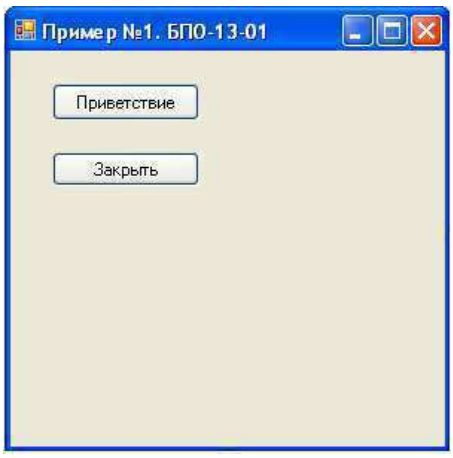


Рисунок 8 – Форма «Пример БПО-13-01»

Сам объект *Button* никаких действий не выполняет, его откликом является сигнал *isClick*. Для формирования реакции на нажатие необходим обработчик событий.

Обработчик событий – это метод Button_Click( ), содержащий список реакций на события.

Создадим обработчик событий для кнопок:

код для кнопки «Приветствие»: MessageBox::Show("Здравствуйте!");

код для кнопки «Закрыть»: Application::Exit();

Получим следующий результат.

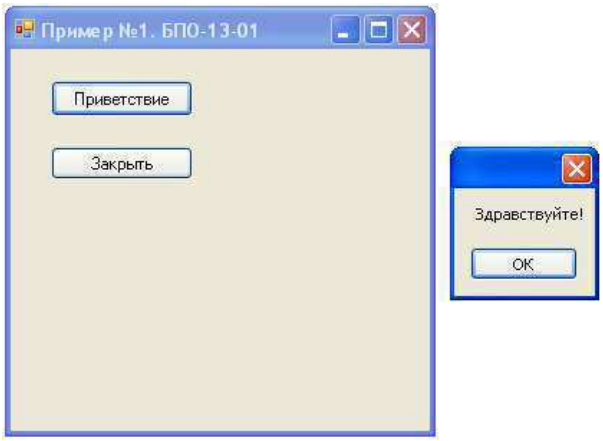


Рисунок 9 – Результат работы

Функция *MessageBox* служит для создания и отображения на экране окна сообщения, которое содержит определяемое программой сообщение и заголовок.

Данная функция используется во многих программах, в основном, для вывода предупреждения пользователю о некорректности введенных им данных.

Пример работы с данной функцией рассмотрен выше и будет использоваться в следующих работах.

Элемент управления TextBox

Элемент управления *TextBox* представляет собой текстовое поле для ввода или вывода данных типа *string*.

Каждый набор символов, который вводится с клавиатуры в *TextBox* имеет тип *string*. Для выполнения каких-либо математических расчетов, необходимо перевести введенные в текстовое поле пользователем данные к целочисленному, или вещественному типу. Обработчик событий будет иметь следующий вид:

```
int x= System::Convert::ToDouble(A->Text);
B->Text = System::Convert::ToString(A);
```

Таблица 8 – Свойства TextBox

Название свойства	Описание свойства
AcceptsReturn	Получает или задает значение, казывающее, что происходит в многострочном элементе управления TextBox при нажатии клавиши ENTER: создается

	новая строка текста или активируется кнопка стандартного действия формы
AcceptsTab	Получает или задает значение, указывающее, что происходит при нажатии клавиши TAB в многострочном элементе управления: вводится знак табуляции в текстовом поле или фокус ввода в форме перемещается к следующему элементу управления в последовательности переходов
Lines	Получает или задает строки текста в элементе управле-ния «Текстовое поле»
Multiline	Получает или задает значение, показывающее, является ли данный элемент управления многострочным TextBox
PasswordChar	Получает или задает знак, используемый для маскировки знаков пароля, вводимых в однострочный элемент управления TextBox
ReadOnly	Получает или задает значение, указывающее, является ли текст в текстовом поле доступным только для чтения
Text	Получает или задает текущий текст в текстовом поле TextBox
TextAlign	Получает или задает способ выравнивания текста в элементе управления TextBox

Таблица 9 – Методы TextBox

Метод	Описание метода
AppendText	Добавляет строку к содержимому текстового элемента управления
Clear	Удаляет из текстового поля все его содержимое
Copy	Копирует текущее выделение текста в элементе управления, поддерживающем редактирование текста
CreateGraphics	Задает объект Graphics для элемента управления
Cut	Перемещает текущий выбор из текстового поля в бу-фер обмена
DeselectAll	Указывает, что значение свойства SelectionLength равно нулю для отмены выделения символов в эле-менте управления
Dispose()	Освобождает все ресурсы, используемые объектом
Focus	Задает фокус ввода элемента управления
Hide	Скрывает элемент управления от пользователя

Paste()	Заменяет текущий выбор в текстовом поле содержи-мым буфера обмена
Select()	Активирует элемент управления
SelectAll	Выбирает весь текст в текстовом поле
Show	Отображает элемент управления для пользователя
Undo	Отменяет последнюю операцию редактирования в текстовом поле
Метод	Описание метода

Таблица 10 – События TextBox

Событие	Описание события
GotFocus	Событие, возникающее в момент активизации окна
LostFocus	Событие, возникающее в момент потери фокуса
KeyDown	Событие, возникающее в момент движения нажи-маемой клавиши вниз
KeyPress	Событие, возникающее при удержании нажатой клавиши
KeyUp	Событие, возникающее при отпускании нажатой клавиши
Change	Событие, возникающее при изменении, добавлении или удалении очередного символа в поле ввода

1.5. Пример выполнения задания

Создаем проект с именем Factorial. На форму добавляем шесть элементов: 3 - *Label*, 2 – *TextBox,* 1 – *Button,* как показано на рисунке.

Рисунок 9 – Вид Формы

В среде разработки, *Панель элементов* автоматически скрывается. Чтобы исключить свертывание Панели элементов, необходимо нажать на кнопку «кнопка» .

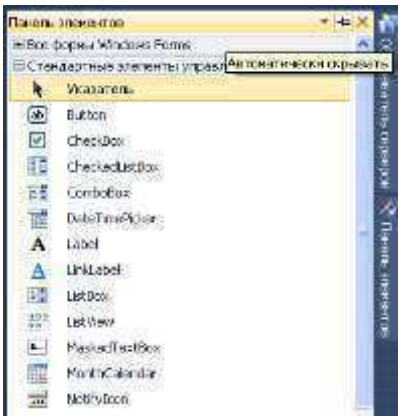


Рисунок 10 –Панель элементов

Для задания свойств элементам, необходимо раскрыть *Диспетчер свойств*. Выбираем *Вид → Диспетчер свойств.*

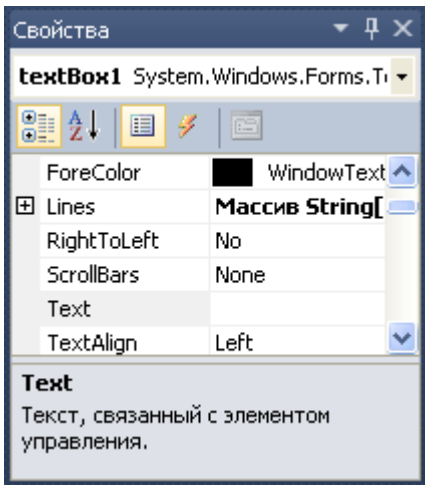


Рисунок 12 – Диспетчер свойств

Изменяет свойства элементов. В *Панели свойств*, изменяем свойство *Text*.

Размер элемента *Label* строго не изменяется по вертикали. В случае, большого текста, не все слова могут быть видны. Для того, чтобы размер окна изменялся по тексту, необходимо изменить свойство *AutoSize* и его значение True поменять на False. Изменим цвет введенного текста, при помощи свойства *ForeColor*. Текст выравнем по центру, используя свойств *TextAlign*. В первом *Label1*

изменим вид текста (свойство *Font*): размер шрифта (подсвойство *Size*) и начертание (полужирный, подсвойство – *Bold*).

Можно поэкспериментировать с цветом заливки Формы, работая со свойством *BackColor*.

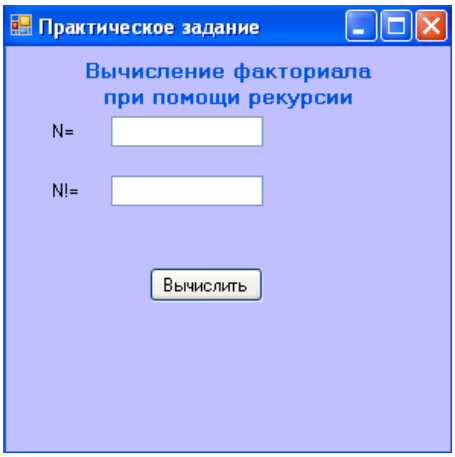


Рисунок 13 – Форма после выполненных изменений

Теперь перейдем к кодированию. Открываем в *Обозреватель решений* файл Factorial.cpp. После подключенных заголовочных файлов, создаем прототип функции вычисления факториала:

```
long double f(int N);
```

В этот же файл добавляем текст программы:

```
long double f(int N)
{if(N < 0) return 0;
 if (N == 0) return 1;
 else return N * fact(N - 1);
}
```

Для прикрепления к элементам *TextBox* переменных, изменяем их свойство *Name*, соответственно, на N и FactN.

Открываем файл Form1.h, в начало файла добавляем прототип функции f (сразу после #pragma once) .

Возвращаемся в конструктор нашей формы. Щелкаем два раза на кнопку *Выполнить* и добавляем код:

```
int number = System::Convert::ToDouble(N->Text);
double factor = f(number);
FactN->Text = System::Convert::ToString(factor);
```

Код добавляется между скобками, именно там, где стоит курсор.

```
#pragma endregion
    private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
    int number = System::Convert::ToDouble(N->Text);
    double factor = f(number);
    FactN->Text = System::Convert::ToString(factor);

            }
```

Рисунок 13 – Обработчик события «Вычислить»

Выполните отладку программы. Результат работы приложения представлен ниже.

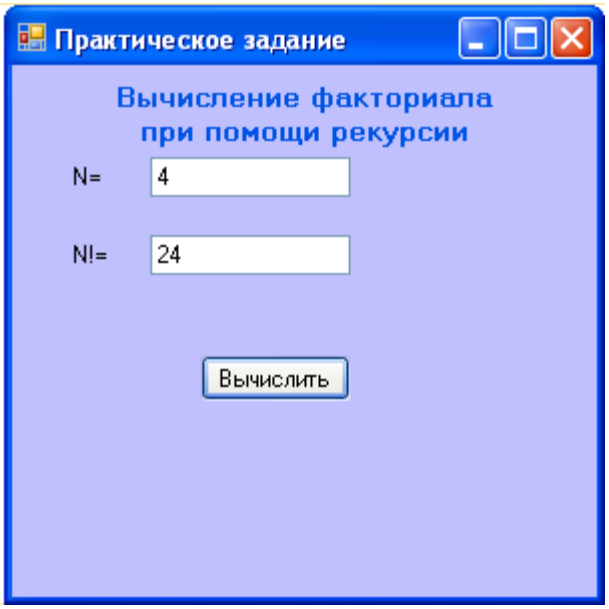


Рисунок 14 – Результат выполнения приложения

2. Лабораторная работа № 2.

Состав визуальной среда программирования. Палитра компонентов. Свойства и методы компонентов.

Цель работы:

Получение навыков работы с элементами управления, их свойствами, методами и событиями при разработке приложений с визуальным интерфейсом.

2.1. Порядок выполнения лабораторной работы

При выполнении заданий лабораторных работ, студенты

Количество элементов управлений на Форме будет несколько десятков. Если, нужно вывести весь список уже установленных элементов, необходимо выбрать *Структура документа* на панели меню, или нажать сочетание клавиш Ctrl+Alt+D. Слева появиться окно со списком всех элементов Вами созданной Формы.

Рисунок 15 – Структура Формы

Лабораторная работа состоит из трех заданий: 2 практических и одно теоретическое. Ниже представлен пример выполнения лабораторных работ, а также индивидуальные задания для каждого студента.

Пример 1. Табулирование функции и вычисление её значений в указанном интервале с заданным шагом

Откройте форму и установите на ней следующие элементы управления:

- *Label* – 8 элементов;
- *PictureBox* – 1 элемент (слева);
- *DataGridView* – 1 элемент (справа);
- *TextBox* – 6 элементов;
- Button – 2 элемента.

На форму будет установлен текст задания (вид кусочно-заданной функции). Чтобы поместить его на Форме, занесем его в буфер (при открытом задание, нажимаем на клавишу *PrintScreen*) и в графическом редакторе (например, *Paint*), «вырезаем» рисунок нужного размера. Затем, сохраняем его и помещаем в элемент управления *PictureBox* в свойсвто *Image*.

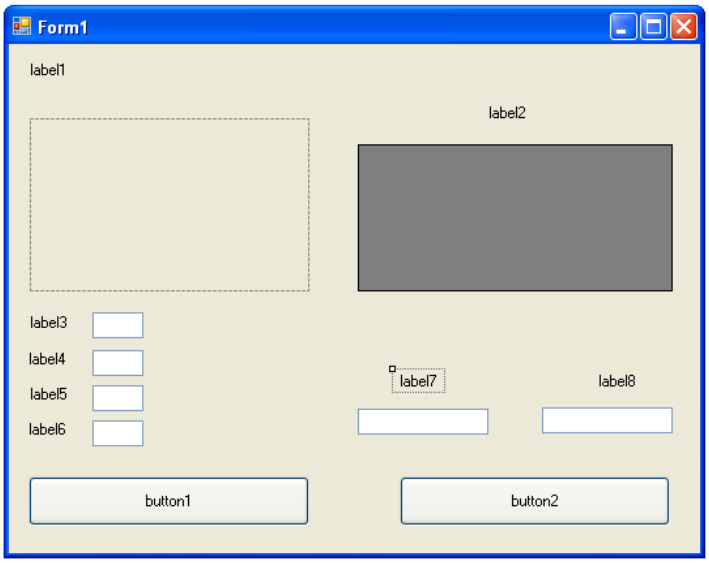


Рисунок 16–Расстановка элементов управления

Чтобы задать одинаковое свойство более чем одному элементу управления, выделите необходимые элементы с помощью мыши (или нажатием клавиши Shift), и выберите свойство.

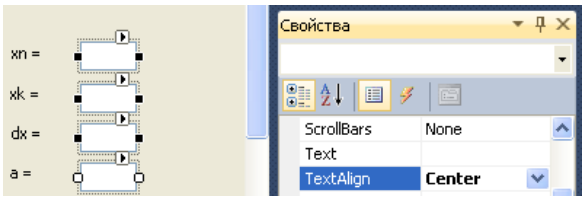


Рисунок 17– Установление одинакового свойства нескольким элементам

Задайте свойства элементам, согласно таблице значений.

Таблица 11 – Свойства Form1

Элемент управления	Свойство	Значение
Form1	Text	“Лабораторная работа №4. Задание 1”
	FormBorderStyle	FixedToolWindow *(при работе приложения, размер формы не может быть изменен)*
Label1	AutoSize	False
	Text	“Протабулировать функцию y=f(x) на отрезке [xn; xk] с шагом dx. Найти экстремумы функции на указанном отрезке.”
	Font	Начертание: жирный. Размер: 10.
PictureBox1	Image	“D:\picture”
DataGridView	BorderStyle	Fixed3D *(придать таблице легкий эффект объема)*
Label2	Text	“Таблица значений”
Label3	Text	“xn =”
Label4	Text	“xk =”
Label5	Text	“dx =”
Label6	Text	“a =”
TextBox1	TextAlign	Center
TextBox2	TextAlign	Center
TextBox3	TextAlign	Center
TextBox4	TextAlign	Center
Label7	Text	“Максимальное значение функции”
	AutoSize	False
	TextAlign	TopCenter
Label8	Text	“Минимальное значение функции”
	AutoSize	False
	TextAlign	TopCenter
TextBox5	ReadOnly	True (для невозможности ввода данных пользователем)
	TextAlign	Center
TextBox6	ReadOnly	True (для невозможности ввода данных

		пользователем)
	TextAlign	Center
Button1	Text	“Выполнить задание”
	Font	Начертание: жирный. Размер: 10.
	Size	215;38
Button2	Text	“Закрыть приложение”
	Font	Начертание: жирный. Размер: 10.
	Size	215;38

После установления всех свойств, Форма примет следующий вид:

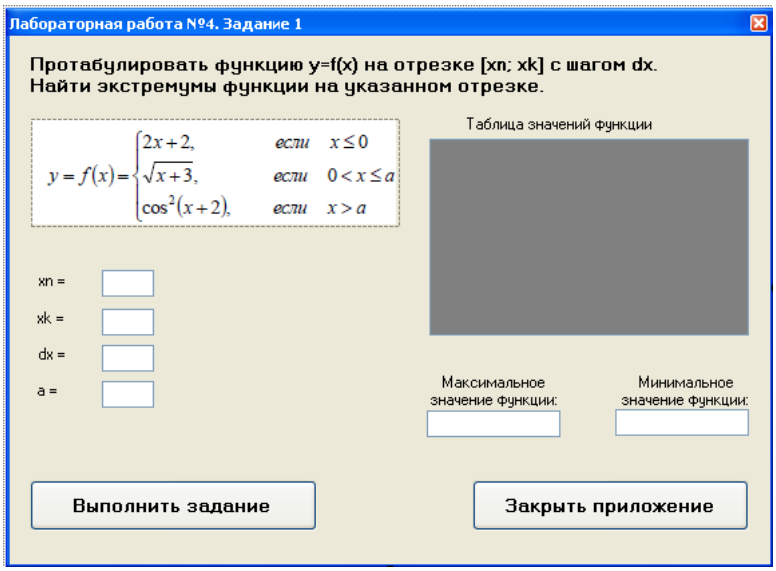


Рисунок 18 – Вид Формы с заданными свойствами

Рассмотрим код обработчика события *Click* кнопки «Выполнить задание»:

```
private: System::Void button1_Click(System::Object^ sender,
System::EventArgs^ e)
{ double xn,xk,xh,x,y,a,ymax,ymin,yt;
int n,i;
//Проверка ввода данных в компоненты textBox
if ((textBox1->Text!="")&&(textBox2->Text!="")&&
(textBox3->Text!="")&&(textBox4->Text!=""))
{ //Преобразование введенных данных в тип double
xn = Convert::ToDouble(textBox1->Text); xk =
Convert::ToDouble(textBox2->Text);
xh = Convert::ToDouble(textBox3->Text); a =
Convert::ToDouble(textBox4->Text);
```

```
//Очистка столбцов таблицы
dataGridView1->Columns->Clear();
//Создание двух столбцов в таблице
dataGridView1->ColumnCount = 2;
//Создание в таблице строк
dataGridView1->Rows->Add(ceil((xk-xn)/xh)+1);
//Занесение в верхнюю строку таблицы в первую ячейку текст «X», во вторую текст «Y»
dataGridView1->Columns[0]->Name=" X";
dataGridView1->Columns[1]->Name=" Y";
i=0; x=xn; ymax=-1.8e307;ymin=1.8e307;
while (x<=xk)
{ if (x<=0){ y=2*x+2;}
else if (x<=a) {y=sqrt(x+3);}
else {y=pow(cos(x+2),2);}
//Занесение в первый столбец значений аргумента X
dataGridView1->Rows[i]->Cells[0]->Value
=Convert::ToString(x);
//Переменной yt присваивает округленное до двух знаков после запятой значение y
yt=ceil(y*100)/100;
//Вывод во втором столбце таблицы значение функции Y
dataGridView1->Rows[i]->Cells[1]->Value
=Convert::ToString(yt);
//находит максимальное и минимальное значение и округляет до двух знаков после запятой
if (y>ymax) ymax=ceil(y*100)/100;
if (y<ymin) ymin=ceil(y*100)/100;
x=x+xh;
i++;}
//выводит в компоненты textbox максимальное и минимальное значение функции
textBox5->Text = Convert::ToString (ymax);
textBox6->Text = Convert::ToString (ymin); }
else {MessageBox::Show( "Заполните, пожалуйста, данные",
"Ошибка ввода данных",
MessageBoxButtons::OK, MessageBoxIcon::Exclamation );}}
```

Запустите приложение. Результат представлен ниже.

Вещественные числа вводятся в текстовое окно через запятую.

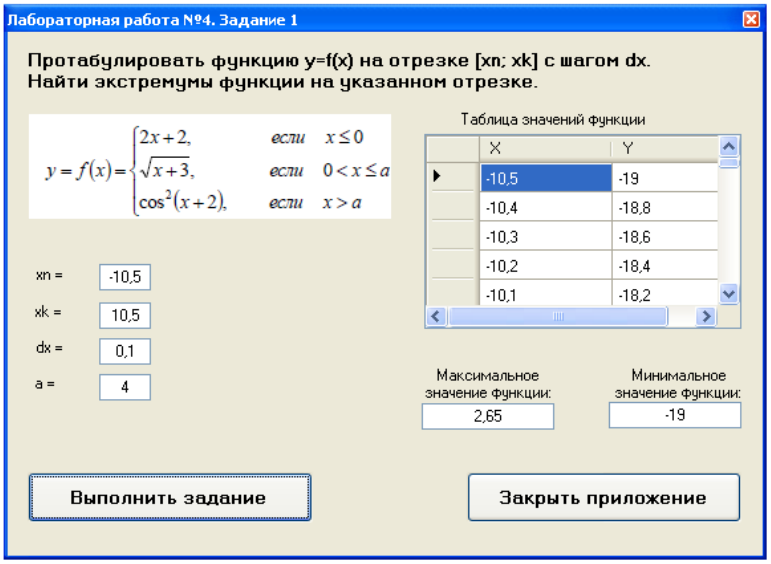


Рисунок 19 – Результат работы приложения

2.2. Задания для выполнения лабораторной работы

Протабулировать функцию

$y = \begin{cases} f1(x), если\ x < K; \\ f2(x), если\ K \le x < N; \\ f3(x), если\ x \ge N; \end{cases}$ в диапазоне изменения x от xn до xk, с шагом dx.

Значения xn, xk, dx, N, K вводятся пользователем при выполнении приложения.

Таблица 12 – Варианты заданий

№	Задание функции	Параметры	№	Задание функции	Параметры
1	$y = \begin{cases} \frac{1}{1+x^2} \cdot tg(x) \\ x^2 + 2\ , \\ 2 \cdot x + \ln(x), \end{cases}$	-	16	$y = \begin{cases} tg(a/x); \\ e^{x/a}\ ; \\ \lvert x - a \rvert\ ; \end{cases}$	$a = \pi/5.$
2	$y = \begin{cases} 1{,}53 \cdot x^2\ , \\ x + \ln(x \cdot a), \\ \frac{1}{\sqrt{x + 2{,}3 \cdot a}}, \end{cases}$	$\alpha = 4{,}345.$	17	$y = \begin{cases} x^2 + c\ ; \\ \sqrt{x + \sqrt{c}}\ ; \\ e^x + \ln(c); \end{cases}$	$c = \ln\lvert\omega \cdot t\rvert$ $\omega = 4{,}24;\ \ t = 2{,}91$

3	$y=\begin{cases}\ln(x+\alpha), \\ \dfrac{1}{\ln(x+\alpha)}, \\ 2-\ln(x+\alpha),\end{cases}$	$\alpha = 4{,}345.$	18	$y=\begin{cases}\dfrac{x+1}{x-1}; \\ \sin(\pi\cdot x); \\ \dfrac{x-1}{x+1};\end{cases}$	-
4	$y=\begin{cases}1,5, \\ e^{x\cdot a}+2\cdot x+a, \\ x\cdot a+\dfrac{x\cdot a+\ln(x+a)}{(1-x\cdot a)\cdot(x+a)},\end{cases}$	$a = 5{,}4.$.	19	$y=\begin{cases}a\cdot\sqrt{1-x^2/b^2}; \\ -\sqrt{\lvert x-b\rvert}; \\ \frac{a}{b}\cdot(x+b);\end{cases}$	$b=\sin(\omega\cdot t^2);$ $a=\cos^2(\omega\cdot t);\ \omega=1;$ $t=0{,}1$
5	$y=\begin{cases}\dfrac{x-y}{x+\dfrac{y}{x+y}}, \\ \sqrt{1-\frac{x^2}{y^2}}, \\ \dfrac{\sin(x)}{2+\cos(y)},\end{cases}$	$y=\lvert\sin(\omega^2\cdot t)\rvert;$ $\omega = 20{,}7;\ t = 1{,}3.$	20	$y=\begin{cases}1-\sin(x); \\ 0{,}5(1+\cos(\\ 0;\end{cases}$	-
6	$y=\begin{cases}\ln\lvert p+x\rvert, \\ \ln\lvert p-x\rvert, \\ e^{p+x^2},\end{cases}$	$p=\sqrt{0{,}17}\cdot\lg(5);$.	21	$y=\begin{cases}x+\dfrac{\ln(a)}{\ln(a-x)}; \\ \dfrac{a\cdot\ln(a-x)}{\sin(a-x)}; \\ 2{,}35\cdot\lg\lvert a-x\rvert;\end{cases}$	$a = 2{,}57$
7	$y=\begin{cases}\sin^2 x, \\ \lg(x/a), \\ \dfrac{x^2}{a}+\ln(a),\end{cases}$	$a = 5{,}34;$	22	$y=\begin{cases}(1+e^x)\cdot\ln(x+a); \\ x-a; \\ \dfrac{\sin(x+a)}{\ln(x\cdot\sqrt{a+x})};\end{cases}$	$a = 3{,}53.$
8	$y=\begin{cases}\sin(x+b), \\ x+b\cdot\sqrt{x^2+b^2}, \\ x-b,\end{cases}$	$b=\dfrac{\sin(a)}{\cos(2+a)};$ $a = 2;52$	23	$y=\begin{cases}0, \\ 1{,}53\cdot x^2, \\ x+\ln(x\cdot a), \\ \dfrac{x}{\sqrt{x+2{,}3a}},\end{cases}$	$a = 1{,}25;$
9	$y=\begin{cases}\pi+\dfrac{(x+y)}{x}, \\ \pi-(x-y), \\ \dfrac{x+y}{(x-y)},\end{cases}$	$y=\sin(x);$	24	$y=\begin{cases}a, \\ 2\cdot a+x^{0,5a}, \\ (x+a)^{0,7}+\dfrac{a^2}{x+a},\end{cases}$	$a = 2{,}23;$
10	$y=\begin{cases}10^{x-a}, \\ e^{x+a}, \\ tg(x+a),\end{cases}$	$a = 0{,}5$	25	$y=\begin{cases}\dfrac{\cos(x)\sin(x)}{1+x^2}, \\ (1+x^2)\cdot\cos(x) \\ \ln(1+x),\end{cases}$	$a = 3{,}25.$
11	$y=\begin{cases}0 \\ \lg(\ln(x)) \\ \sqrt[3]{\sin^2(ax)}\end{cases}$	$a = 3{,}75.$	26	$y=\begin{cases}1{,}178; \\ x\cdot\sqrt{x+\ln(x+a)}; \\ e^{x-a}+1;\end{cases}$	$a = 4{,}664.$
12	$y=\begin{cases}(a+x)^3 \\ 0 \\ (a+x)/(ax^3)\end{cases}$	$a = 0{,}3.$	27	$y=\begin{cases}\dfrac{n\cdot(n-2)(n-1)}{b}, \\ 2^{2\cdot n}, \\ (n-9),\end{cases}$	$a = 3{,}53.$

13	$y=\begin{cases} e^{x/a} \\ \ln(a/x) \\ \ln(a+e^{a\cdot x}) \end{cases}$	a = 2,1.	28	$y=\begin{cases} \frac{(-x-1)}{4}, \\ 3, \\ \cos(x), \end{cases}$	-
14	$y=\begin{cases} \sqrt{y+b} \\ \sqrt[3]{y-b} \\ \sqrt[5]{b-y} \end{cases}$	b = 3,75.	29	$y=\begin{cases} \sin(x+b); \\ (x+b)\cdot\sqrt{x^2+b^2}; \\ 3{,}437; \end{cases}$	b = sin(a · cos(2 + a)); a = 0,52.
15	$y=\begin{cases} 0; \\ e^{x-a}; \\ \lg(a-x); \end{cases}$	a = 1,5.	30	$y=\begin{cases} 3{,}14+\operatorname{ctg}\frac{x+1}{1-y\cdot x}; \\ -3{,}14+\operatorname{ctg}\frac{x+y}{1+x\cdot y}; \\ 1{,}57; \end{cases}$	y = cos(ω · t); ω = 2; t = 1,37

Контрольные вопросы

1. Опишите структуру программы.
2. Перечислите и опишите основные типы данных?
3. Назовите основные элементы визуальной среды.
4. Как определить переменную, константу?
5. Опишите известные вам компоненты ввода-вывода данных.
6. Перечислите основные свойства компоненты ввода-вывода данных при разработке приложений с визуальным интерфейсом.
7. Укажите основные методы и событиями компонентов.
8. Какая связь между понятием объект, класс и компонент?
9. Что такое директива препроцессору?
10. Как подключить библиотеку с математическими функциями?

2. Лабораторная работа №3.

Использование в приложении структурированных данных. Работа с массивами данных.

Цель работы:

Получение навыков работы со структурированными данными. Получение навыков разработки приложений по обработки массивов данных.

2.1. Порядок выполнения лабораторной работы

Откройте Форму. На Форму добавляем элемент управления *MenuStrip*, с помощью которого создадим меню для сохранения данных и представления информации о разработчике. Нажимаем на строку меню правой кнопкой мыши и выбираем *Вставить стандартные элементы.* Из появившегося списка, оставляем только то, что представлено на рисунке. Добавляем меню «*О разработчике».*

Также на Форму добавляем элемент *TabControl*, который будет содержать в себе две вкладки *tabPage1* и *tabPage2*. Зададим вкладкам свойство *Text*, «Условие» и «Выполнение».

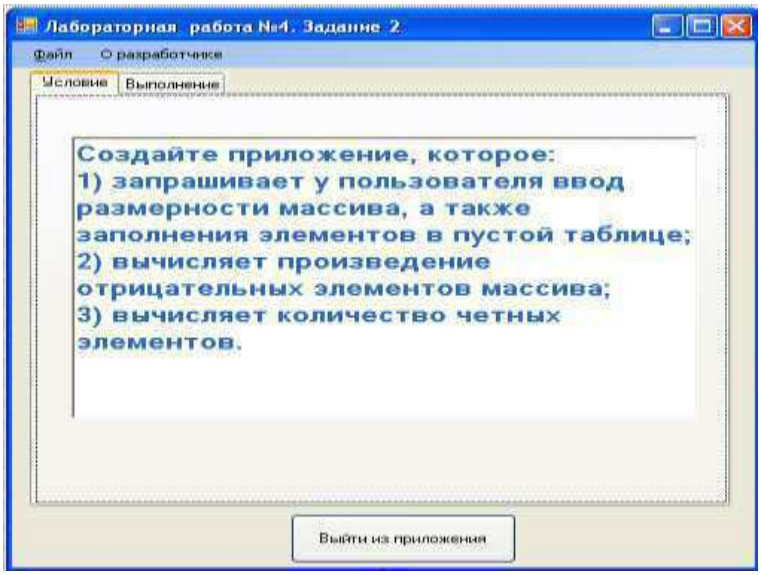


Рисунок 20 – Вкладка «Условие»

Во вкладку *tabPage1* добавляем элемент *RichTextBox1*. Данный элемент позволяет добавлять на Форму текст большого объема. В нашем задание, поместим в *RichTextBox1* условие. На Форму добавим кнопку «Выйти из приложения» и зададим события выхода из Формы, которое использовалось выше.

Откроем вкладку «Выполнение». Установим на ней следующие элементы:

- *GroupBox* – 2 элемента;
- *Label* - 2 элемента;
- *CheckBox* – 2 элемента;
- *Button* – 2 элемента;
- *DataGridView1* – 1 элемент;
- *TextBox* – 4 элемента.

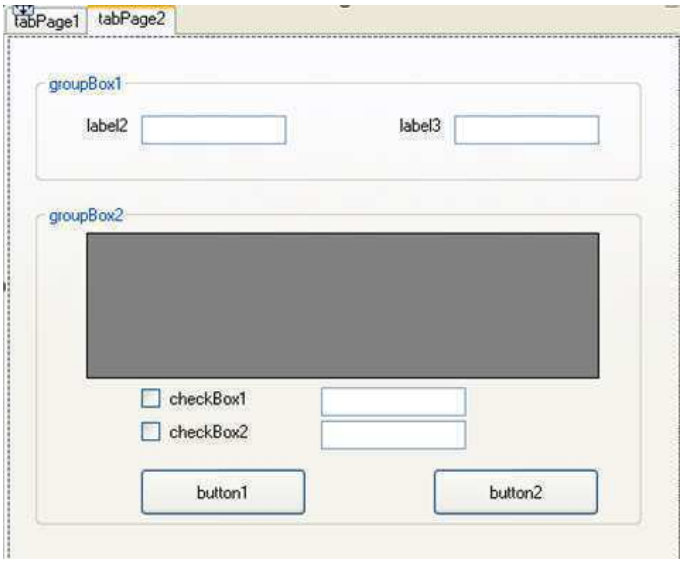


Рисунок 21 – Вкладка «Выполнение»

Установим элементам Формы свойства, представленные в таблице.

Таблица 13– Свойства Form1

Элемент управления	Свойство	Значение
Form1	Text	“Лабораторная работа №4. Задание 2”
	Cursor	Hand (*Установить курсор в виде руки* )

MenuStrip		«Файл», «О разработчике»
TabControl	Text	*Содержит две вкладки tabPage1 и tabPage2*
tabPage1	Text	“Условие”
RichTextBox1	Text	Создайте приложение, которое: 1) запрашивает у пользователя ввод размерности массива, а также заполнения элементов в пустой таблице; 2) вычисляет произведение отрицательных элементов массива; 3) вычисляет количество четных элементов.
	Font	Начертание: жирный. Размер: 14.
	ForeColor	MenuHighlight
tabPage2	Text	“Выполнение”
GroupBox1	Text	“Задайте размерность матрицы” *Это поле будет содержать следующие элементы управления: Количество строк, Количество столбцов, TextBox1, TextBox2*
	Font	Начертание: жирный. Размер: 8.
Button4	Text	“Справка”
GroupBox2	Text	“Исходная матрица” *Это поле будет содержать следующие элементы управления: DataGridView1, Button2, Button3*
Label1	Text	“Количество строк:”
	AutoSize	False
	TextAlign	TopCenter
Label2	Text	“Количество столбцов”

	AutoSize	False
	TextAlign	TopCenter
DataGridView1	RowHeadersVisible	False *(для того, чтобы не отображать заголовок строк)*
	ColumHeadersVisible	False *(для того, чтобы не не отображать заголовок столбцов)*
CheckBox1	Text	“Произведение отрицательных элементов”
	Font	Начертание: обычный. Размер: 8.
CheckBox2	Text	“Произведение отрицательных элементов”
TextBox1		
TextBox2		
Button1	Text	“Открыть таблицу для заполнения элементов”
Button2	Text	“Считать данные и выполнить задание”
Button3	Text	“Выйти из приложения”
Button4	Text	“Справка”
PictureBox1	SizeMode	StrechImage *(растягивание рисунка внутри области)*

Создадим в приложение еще две Формы, на которых установим необходимую при работе информацию. Для создания в приложение новой Формы, на панели меню выбираем *Добавить новый элемент (или Ctrl+Shift+A) ->Форма Windows Form.* Не забудьте задать имя формы (например, Form2, Form3 и т.д.).

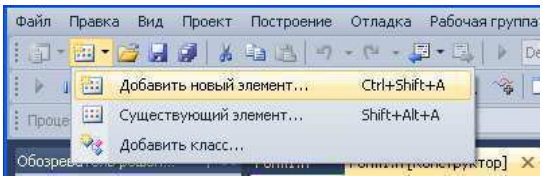


Рисунок 22 – Добавление новой Формы

На Form2 поместим информацию о разработчике. Прикрепим ее к вкладке в меню «О разработчике». Внешний вид Form2 представлен ниже.

Установим свойства элементам формы Form2.

Таблица 14 – Свойства Form2

Form2	Text	“О разработчике”
PictureBox1	Image	“D:\ ”
RichTextBox	Text	“ФИО: ”
Button1	Text	“На главную”
	Font	Начертание: обычный. Размер: 10.
Внешний вид	BackColor	“0;192;192”

Для того чтобы связать вкладку с формой, необходимо установить соответствующее событие. Щелкаем два раза мышью на вкладке «О разработчике», добавляем следующий код:

```
Form2-> p = gcnew Form2();
this->Hide();
p->ShowDialog();
this->Show();
```

Для того чтобы выйти обратно в главную Форму, на Form2 установим кнопку «На главную». При её нажатие текущая Форма станет невидимой. Зададим на неё следующее событие:

```
this->Hide();
```

Создадим Form3. На ней будет храниться информация о работе с приложением. Эта форма будет вызываться при нажатии на кнопку «Справка», которая будет установлена позже.

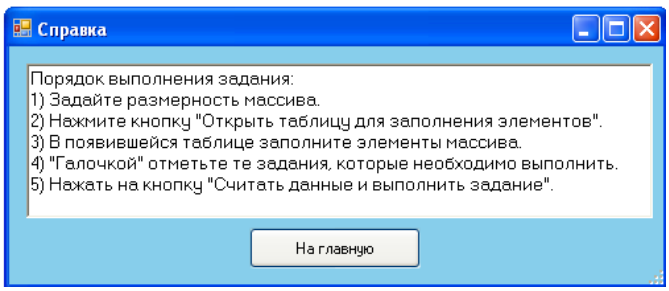


Рисунок 23 – Форма при нажатии на кнопку «Справка»

Установим свойства элементам формы Form3.

Таблица 15 – Свойства Form2

Form3	Text	“Справка”
RichTextBox	Text	“Порядок выполнения задания: 1) Задайте размерность массива. 2) Нажмите кнопку "Открыть таблицу для заполнения элементов". 3) В появившейся таблице заполните элементы массива. 4) "Галочкой" отметьте те задания, которые необходимо выполнить. 5) Нажать на кнопку "Считать данные и выполнить задание".”
	Font	Размер: 10.
Внейшний вид	BackColor	SkyBlue
Button1	Text	“На главную”

Перейдем на вкладку «Выполнение». Создадим событие для кнопки «Открыть таблицу для заполнения элементов». Щелкаем два раза мышью и записываем следующий код:

```
//Проверка, что не пустые компоненты textBox1 и textBox2
if ((textBox1->Text!="")&&(textBox2->Text!=""))
{m = Convert::ToInt32(textBox1->Text);
n = Convert::ToInt32(textBox2->Text);
//Чистка столбцов компонента DataGridView, если они не пусты
dataGridView1->Columns->Clear();
//Заполнение компонента DataGridView столбцами
dataGridView1->ColumnCount = n;
//Заполнение компонента DataGridView строками
dataGridView1->RowCount = m;}
else
{MessageBox::Show( "Заполните, пожалуйста, данные", "Ошибка ввода данных",
MessageBoxButtons::OK, MessageBoxIcon::Exclamation );}
```

Создадим обработчик для кнопки «Считать данные и выполнить данные». Запишем код:

```
//переменную kol и kol2 обнуляем, а переменную p присваиваем единице
```

```
kol=0;kol2=0;p=1;
//Производим считывание из ячеек таблицы и вносим данные в массив
for (int i = 0; i < m; i++)
for (int j = 0; j < n; j++)
{
A[i][j] = Convert::ToSingle(this->dataGridView1->Rows[i]->Cells[j]->Value);
if (A[i][j]<0) {p=p*A[i][j];kol2++;}
if (A[i][j]%2==0) {kol++;}
}
//Вывод данных нахождения произведения отрицательных элементов матрицы
if ((checkBox1->Checked==true)&&(kol2!=0)) {this->textBox3->Text=Convert::ToString (p);}
else
if (checkBox1->Checked==true) {this->textBox3->Text=Convert::ToString ("нет элементов");}
//Вывод данных нахождения количество четных элементов матрицы
if ((checkBox2->Checked==true)&&(kol!=0)) {this->textBox4->Text=Convert::ToString (kol);}
else
if (checkBox2->Checked==true) {this->textBox4->Text=Convert::ToString ("нет элементов");}
return;
```

На кнопку «Справка» зададим события вызова Form3 (рассмотрено выше). Запустите приложение.

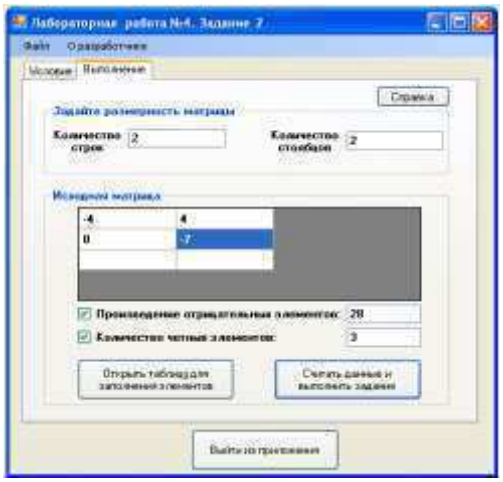


Рисунок 24 – Результат работы приложения

3.2. Задания для выполнения лабораторной работы

1. Для заданной квадратной матрицы сформировать одномерный массив из ее диагональных элементов. Найти след матрицы, суммируя элементы одномерного массива.Преобразовать исходную матрицу по правилу: четные строки разделить на полученное значение, нечетные оставить без изменения. Исходную и преобразованную матрицы напечатать по строкам.

2. Задан массив A размером N*Ì элементов и вектор B размером M. Элементы первого столбца массива A упорядочены по убыванию. Включить массив B в качестве новой строки в массив A с сохранением упорядоченности по элементам первого столбца. Вычислить сумму элементов полученного массива. Полученный массив и его сумму вывести на печать.

3. Заданы две матрицы A и B размером N*N элементов. Сформировать из них прямоугольную матрицу X размером N*2N, включая первые N столбцов матрицу A, в следующие — матрицу B. В матрице X найти положение максимального по абсолютной величине элемента. Матрицу X и полученные значения строки и столбца максимального элемента вывести на печать.

4. Задан массив X размером N. Сформировать из него матрицу A, содержащую по L элементов в строке. Недостающие элементы в последней строке (если такие будут) заполнить нулями. В полученной матрице найти элемент, имеющий минимальное значение. Напечатать матрицу и минимальный элемент.

5. Задана квадратная матрица A размером N*N. Сформировать два одномерных массива. В один переслать по строкам верхний треугольник матрицы, включая элементы главной диагонали, в другой — нижний треугольник. Подсчитать на сколько отличаются суммы элементов верхнего и нижнего треугольников. Исходный, полученные массивы и расчетные значения сумм вывести на печать.

6. Задана квадратная матрица. Переставить строку с максимальным элементом на главной диагонали со строкой с заданным номером. Исходную и полученную матрицы вывести на печать.

7. Задана квадратная матрица. Исключить из нее строку и столбец, на пересечении которых расположен максимальный элемент главной диагонали. Исходную и полученную матрицы вывести на печать.

8. Заданы матрица размером N*N и число К (1<=К<=N). Строку с максимальным по модулю элементом в К-м столбце переставить с К-й строкой. Исходную и полученную матрицы вывести на печать

9. Задана матрица размером N*N. Найти максимальный по модулю элемент матрицы. Переставить строки и столбцы матрицы таким образом, чтобы максимальный по модулю элемент был расположен на пересечении К-й строки и К~го столбца. Исходную и полученную матрицы вывести на печать.

10. Задана квадратная матрица размером N*N. Преобразовать ее таким образом, чтобы элементы строки матрицы были результатом деления исходного значения элемента на значение диагонального элемента, расположенного в этой строке. Вычислить сумму элементов полученной матрицы. Преобразованную матрицу и результаты расчета вывести на печать.

11. Задана квадратная матрица А размером М*N. Заменить строку ,содержащую минимальный элемент матицы на одномерный массив В размером N. Исходную и полученную матрицы вывести на печать.

Определить, является ли заданная целочисленная квадратная матрица А N-го порядка магическим квадратом, т.е. такой, в которой суммы элементов во всех строках и столбцах одинаковы.

12. Для заданной квадратной матрицы сформировать одномерный массив из элементов побочной диагонали. Найти сумму элементов одномерного массива. Преобразовать исходную матрицу по правилу: четные столбцы разделить на полученное значение, нечетные оставить без изменения. Исходную и преобразованную матрицы напечатать по строкам.

13. Задан массив A размером M*N и число K (1<=K<=M).. Сформировать одномерный массив B размером M, в который выбрать к-й столбец массива A. Поменять местами максимальные элементы массивов A и B.Вывести на печать исходный , полученный массивы.

14. Задан массив A размером M*N и число K (1<=K<=N).. Сформировать одномерный массив B размером N, в который выбрать к-ю строку массива A. Поменять местами минимальные элементы массивов A и B.Вывести на печать исходный , полученный массивы.

15. Задана квадратная матрица. Исключить из нее строку и столбец, на пересечении которых расположен минимальный элемент побочной диагонали. Исходную и полученную матрицы вывести на печать.

16. Задана матрица A размером N*N. Сформировать одномерный массив B размером N, в котором i-й элемент представляет собой сумму элементов принадлежащих главной и побочной диагоналям и расположенных в i-й строке матрицы A. Найти номер минимального элемента массива B. Вывести на печать исходный , полученный массивы и номер минимального элемента.

17. Задана матрица A размером M*N, одномерный массив B размером N и число K(1<=K<=N). Поместить массив B в матрицу A K-й строкой. Вычислить сумму элементов полученного массива.Вывести на печать исходный , полученный массивы и значение суммы элементов массива.

18. Задана матрица A размером M*N, одномерный массив B размером M и число K(1<=K<=N). Поместить массив B в матрицу A K-м столбцом. Вычислить среднее значение сумму элементов полученного массива.

3. Лабораторная работа №4.

Работа с файловой системой ПК. Виды компонентов диалогов и способы обращения к диалогам.

Цель работы:

Знакомство с компонентами диалогов, их свойствами и методами. Получение навыков работы с компонентами диалогов.

3.1. Теоретические сведения

Окна открытия и сохранения файла представлены классами **OpenFileDialog** и **SaveFileDialog**. Они имеют во многом схожую функциональность, поэтому рассмотрим их вместе.

OpenFileDialog и SaveFileDialog имеют ряд общих свойств, среди которых можно выделить следующие:

- DefaultExt: устанавливает расширение файла, которое добавляется по умолчанию, если пользователь ввел имя файла без расширения
- AddExtension: при значении true добавляет к имени файла расширение при его отсутствии. Расширение берется из свойства DefaultExtили Filter
- CheckFileExists: если имеет значение true, то проверяет существование файла с указанным именем
- CheckPathExists: если имеет значение true, то проверяет существование пути к файлу с указанным именем
- FileName: возвращает полное имя файла, выбранного в диалоговом окне
- Filter: задает фильтр файлов, благодаря чему в диалоговом окне можно отфильтровать файлы по расширению. Фильтр задается в следующем формате Название_файлов|*.расширение.

Например, Текстовые файлы(*.txt)|*.txt. Можно задать сразу несколько фильтров, для этого они разделяются вертикальной линией |. Например, Bitmap files (*.bmp)|*.bmp|Image files (*.jpg)|*.jpg

InitialDirectory: устанавливает каталог, который отображается при первом вызове окна

Title: заголовок диалогового окна

Отдельно у класса SaveFileDialog можно еще выделить пару свойств:

CreatePrompt: при значении true в случае, если указан не существующий файл, то будет отображаться сообщение о его создании

OverwritePrompt: при значении true в случае, если указан существующий файл, то будет отображаться сообщение о том, что файл будет перезаписан

Чтобы отобразить диалоговое окно, надо вызвать метод ShowDialog().

3.2. Порядок выполнения работы

Рассмотрим оба диалоговых окна на примере. Добавим на форму текстовое поле textBox1 и две кнопки button1 и button2. Также перетащим с панели инструментов компоненты OpenFileDialog и SaveFileDialog. После добавления они отобразятся внизу дизайнера формы. В итоге форма будет выглядеть примерно так:

Теперь изменим код формы:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

        button1.Click += button1_Click;
        button2.Click += button2_Click;
        openFileDialog1.Filter = "Text files(*.txt)|*.txt|All files(*.*)|*.*";
        saveFileDialog1.Filter = "Text files(*.txt)|*.txt|All files(*.*)|*.*";
    }
    // сохранение файла
    void button2_Click(object sender, EventArgs e)
    {
        if (saveFileDialog1.ShowDialog() == DialogResult.Cancel)
            return;
        // получаем выбранный файл
```

```
        string filename = saveFileDialog1.FileName;
        // сохраняем текст в файл
        System.IO.File.WriteAllText(filename, textBox1.Text);
        MessageBox.Show("Файл сохранен");
    }
    // открытие файла
    void button1_Click(object sender, EventArgs e)
    {
        if (openFileDialog1.ShowDialog() == DialogResult.Cancel)
            return;
        // получаем выбранный файл
        string filename = openFileDialog1.FileName;
        // читаем файл в строку
        string fileText = System.IO.File.ReadAllText(filename);
        textBox1.Text = fileText;
        MessageBox.Show("Файл открыт");
    }
}
```

По нажатию на первую кнопку будет открываться окно открытия файла. После выбора файла он будет считываться, а его текст будет отображаться в текстовом поле. Клик на вторую кнопку отобразит окно для сохранения файла, в котором надо установить его название. И после этого произойдет сохранение текста из текстового поля в файл.

FontDialog.

Для выбора шифта и его параметров используется FontDialog. Для его использования перенесем компонент с Панели инструментов на форму. И пусть на форме имеется кнопка button1. Тогда в коде формы пропишем следующее:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

        button1.Click += button1_Click;
        // добавляем возможность выбора цвета шрифта
        fontDialog1.ShowColor = true;
    }
```

```
    void button1_Click(object sender, EventArgs e)
    {
        if (fontDialog1.ShowDialog() == DialogResult.Cancel)
            return;
        // установка шрифта
        button1.Font = fontDialog1.Font;
        // установка цвета шрифта
        button1.ForeColor = fontDialog1.Color;
    }
}
```

FontDialog имеет ряд свойств, среди которых стоит отметить следующие:

- ShowColor: при значении true позволяет выбирать цвет шрифта
- Font: выбранный в диалоговом окне шрифт
- Color: выбранный в диалоговом окне цвет шрифта
- Для отображения диалогового окна используется метод ShowDialog().

И если мы запустим приложение и нажмем на кнопку, то нам отобразится диалоговое окно, где мы можем задать все параметры шрифта. И после выбоа установленные настройки будут применены к шрифту кнопки:

ColorDialog

ColorDialog позволяет выбрать настройки цвета. Также перенесем его с Панели инструментов на форму. И изменим код формы:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

        button1.Click += button1_Click;
        // расширенное окно для выбора цвета
        colorDialog1.FullOpen = true;
        // установка начального цвета для colorDialog
        colorDialog1.Color = this.BackColor;
    }

    void button1_Click(object sender, EventArgs e)
    {
```

```
            if (colorDialog1.ShowDialog() == DialogResult.Cancel)
                return;
            // установка цвета формы
            this.BackColor = colorDialog1.Color;
        }
    }
```

Среди свойств ColorDialog следует отметить следующие:

- FullOpen: при значении true отображается диалоговое окно с расширенными настройками для выбора цвета
- SolidColorOnly: при значении true позволяет выбирать только между отнотонные оттенки цветов
- Color: выбранный в диалоговом окне цвет

И при нажатии кнопку нам отобразится диалоговое окно, в котором можно установить цвет формы:

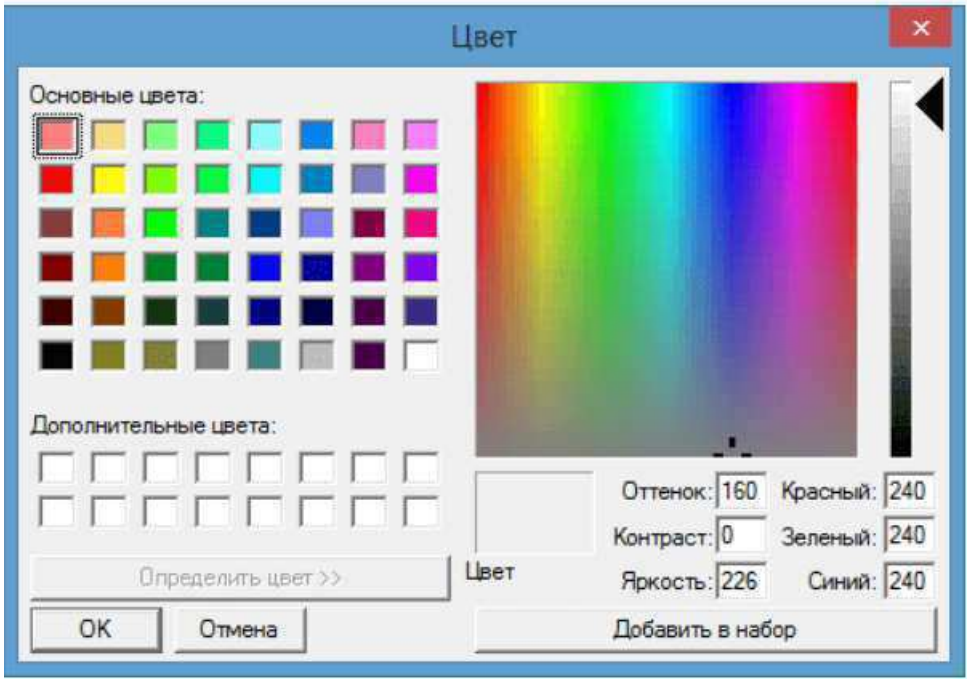


Рисунок 25 – Пример диалога выбора шрифта

3.3. Задания для выполнения лабораторной работы

1. Создайте простой текстовой редактор, позволяющий открывать и сохранять текстовые файлы TXT на основе компонентов Memo, OpenDialog, SaveDialog.. Функции открытия и сохранения файлов реализуйте с помощью обычных кнопок (Button).

2. Создайте программу, позволяющую просматривать рисунки в формате BMP (компоненты Image и OpenDialog). Функцию открытия файлов реализуйте с помощью кнопки с рисунком (BitBtn).

3. С помощью компонентов Animation и OpenDialog напишите программу, позволяющую просматривать файлы анимаций AVI. Функцию открытия файлов реализуйте с помощью обычной кнопки (Button).

4. Создайте простой текстовой редактор, позволяющий открывать и сохранять текстовые файлы TXT на основе компонентов Memo, OpenDialog, SaveDialog.. Функции открытия и сохранения файлов реализуйте с помощью кнопок с рисунками (BitBtn).

5. Создайте простой текстовой редактор, позволяющий менять гарнитуру и размер шрифта всего текста (RichEdit, FontDialog). Для реализации функции замены шрифта используйте обычную кнопку (Button).

6. Создайте программу, позволяющую просматривать рисунки в формате BMP (компоненты Image и OpenDialog). Функцию открытия файлов реализуйте с помощью простой кнопки (Button).

7. Создайте форму с 4 кнопками SpeedButton. На кнопках выведите надписи: «Запретить изменение размера», «Разрешить изменение размера», «Разрешить максимизацию окна», «Запретить максимизацию окна». Написать соответствующие обработчики для каждой кнопки.

8. Создайте программу, позволяющую просматривать рисунки в формате BMP (компоненты Image и OpenDialog). Функцию открытия файлов реализуйте с помощью главного меню (MainMenu).

9. Разместите на форме компоненты ComboBox, ListBox и Button. Реализуйте механизм добавления текста, введённого или выбранного в редакторе ComboBox, в список при нажатии кнопки. Добавьте в форму

кнопку, позволяющую при нажатии на неё удалять первое значение из списка.

10. С помощью компонентов Animation и OpenDialog напишите программу, позволяющую просматривать файлы анимаций AVI. Функцию открытия файлов реализуйте с помощью кнопки (SpeedButton). На кнопке выведите рисунок и текст «Открыть» слева от изображения.

3.4. Список контрольных вопросов

1. Какими компонентами диалогов располагает среда программирования?
2. Укажите методы обращения к диалогу?
3. Правила вызова на экран и проверки поведения пользователя при использовании компонентов диалогов.
4. Основные свойства компонентов диалогов.
5. Как установить тип файлов при использовании компонентов диалогов?
6. Какое свойства компонентов диалогов содержит имя выбранного в диалоге файла?

4. Лабораторная работа №5.
Компоненты оформления приложения: меню и панели инструментов.

Цель работы:

Знакомство с компонентами оформления приложения: меню и панели инструментов. Получение навыков работы с компонентами оформления приложения.

4.1. Теоретические сведения

Элемент ToolStrip представляет панель инструментов. Каждый отдельный элемент на этой панели является объектом ToolStripItem.

Ключевые свойства компонента ToolStrip связаны с его позиционированием на форме:

Dock: прикрепляет панель инструментов к одной из сторон формы

LayoutStyle: задает ориентацию панели на форме (горизонтальная, вертикальная, табличная)

ShowItemToolTips: указывает, будут ли отображаться всплывающие подсказки для отдельных элементов панели инструментов

Stretch: позволяет растянуть панель по всей длине контейнера

В зависимости от значения свойства LayoutStyle панель инструментов может располагаться по горизонтали, или в табличном виде:

HorizontalStackWithOverflow: расположение по горизонтали с переполнением - если длина панели превышает длину контейнера, то новые элементы, выходящие за границы контейнера, не отображаются, то есть панель переполняется элементами

StackWithOverflow: элементы располагаются автоматически с переполнением

VerticalStackWithOverflow: элементы располагаются вертикально с переполнением

Flow: элементы располагаются автоматически, но без переполнения - если длина панели меньше длины контейнера, то выходящие за границы элементы переносятся, а панель инструментов растягивается, чтобы вместить все элементы

Table: элементы позиционируются в виде таблицы

Если LayoutStyle имеет значения HorizontalStackWithOverflow / VerticalStackWithOverflow, то с помощью свойства CanOverflow мы можем задать поведение при переполнении. Так, если это свойство равно true (значение по умолчанию), то для элементов, не попадающих в границы ToolStrip, создается выпадающий список:

При значении false подобный выпадающий список не создается.

Типы элементов панели и их добавление

Панель ToolStrip может содержать объекты следующих классов

ToolStripLabel: текстовая метка на панели инструментов, представляет функциональность элементов Label и LinkLabel

ToolStripButton: аналогичен элементу Button.Также имеет событие Click, с помощью которого можно обработать нажатие пользователя на кнопку

ToolStripSeparator: визуальный разделитель между другими элементами на панели инструментов

ToolStripToolStripComboBox: подобен стандартному элементу ComboBox

ToolStripTextBox: аналогичен текстовому полю TextBox

ToolStripProgressBar: индикатор прогресса, как и элемент ProgressBar

ToolStripDropDownButton: представляет кнопку, по нажатию на которую открывается выпадающее меню

К каждому элементу выпадающего меню дополнительно можно прикрепить обработчик нажатия и обработать клик по этим пунктам меню

ToolStripSplitButton: объединяет функциональность

ToolStripDropDownButton и ToolStripButton

4.2. Порядок выполнения работы

Добавить новые элементы можно в режиме дизайнера:

Также можно добавлять новые элементы программно в коде. Их расположение на панели инструментов будет соответствовать порядку добавления. Все элементы хранятся в ToolStrip в свойстве Items. Мы можем добавить в него любой объект класса ToolStripItem (то есть любой из выше перечисленных классов, так как они наследуются от ToolStripItem):

```
public partial class Form1 : Form
{
   public Form1()
   {
      InitializeComponent();

      ToolStripButton clearBtn = new ToolStripButton();
      clearBtn.Text = "Clear";
      // устанавливаем обработчик нажатия
      clearBtn.Click += btn_Click;
      toolStrip1.Items.Add(clearBtn);
   }

   void btn_Click(object sender, EventArgs e)
   {
      MessageBox.Show("Производится удаление");
   }
}
```

Кроме того, здесь задается обработчик, позволяющий обрабатывать нажатия по кнопки на панели инструментов.

Элементы ToolStripButton, ToolStripDropDownButton и ToolStripSplitButton могут отображать как текст, так и изобажения, оибо сразу и то, и другое. Для управления размещением изображений в этих элементах имеются следующие свойства:

DisplayStyle: определяет, будет ли отображаться на элементе текст, или изображение, или и то и другое.

Image: указывает на само изображение

ImageAlign: устанавливает выравнивание изображения относительно элемента

ImageScaling: указывает, будет ли изображение растягиваться, чтобы заполнить все пространство элемента

ImageTransparentColor: указывает, будет ли цвет изображения прозрачным

Чтобы указать разместить изображение на кнопке, у свойства DisplayStyle надо установить значение Image. Если мы хотим, чтобы кнопка отображала только текст, то надо указать значение Text, либо можно комбинировать два значения с помощью другого значения ImageAndText:

Все эти значения хранятся в перечислении

ToolStripItemDisplayStyle.

Также можно установить свойства в коде c#:

```
ToolStripButton clearBtn = new ToolStripButton();
clearBtn.Text = "Поиск";
clearBtn.DisplayStyle = ToolStripItemDisplayStyle.ImageAndText;
clearBtn.Image = Image.FromFile(@"D:\Icons\0023\search32.png");
// добавляем на панель инструментов
toolStrip1.Items.Add(clearBtn);
```

Элемент ToolStrip представляет панель инструментов. Каждый отдельный элемент на этой панели является объектом ToolStripItem.

Ключевые свойства компонента ToolStrip связаны с его позиционированием на форме:

Dock: прикрепляет панель инструментов к одной из сторон формы

LayoutStyle: задает ориентацию панели на форме (горизонтальная, вертикальная, табличная)

ShowItemToolTips: указывает, будут ли отображаться всплывающие подсказки для отдельных элементов панели инструментов

Stretch: позволяет растянуть панель по всей длине контейнера

В зависимости от значения свойства LayoutStyle панель инструментов может располагаться по горизонтали, или в табличном виде:

HorizontalStackWithOverflow: расположение по горизонтали с переполнением - если длина панели превышает длину контейнера, то новые элементы, выходящие за границы контейнера, не отображаются, то есть панель переполняется элементами

StackWithOverflow: элементы располагаются автоматически с переполнением

VerticalStackWithOverflow: элементы располагаются вертикально с переполнением

Flow: элементы располагаются автоматически, но без переполнения - если длина панели меньше длины контейнера, то выходящие за границы элементы переносятся, а панель инструментов растягивается, чтобы вместить все элементы

Table: элементы позиционируются в виде таблицы

Если LayoutStyle имеет значения HorizontalStackWithOverflow / VerticalStackWithOverflow, то с помощью свойства CanOverflow мы можем задать поведение при переполнении. Так, если это свойство равно true (значение по умолчанию), то для элементов, не попадающих в границы ToolStrip, создается выпадающий список:

При значении false подобный выпадающий список не создается.

Типы элементов панели и их добавление

Панель ToolStrip может содержать объекты следующих классов

ToolStripLabel: текстовая метка на панели инструментов, представляет функциональность элементов Label и LinkLabel

ToolStripButton: аналогичен элементу Button.Также имеет событие Click, с помощью которого можно обработать нажатие пользователя на кнопку

ToolStripSeparator: визуальный разделитель между другими элементами на панели инструментов

ToolStripToolStripComboBox: подобен стандартному элементу ComboBox

ToolStripTextBox: аналогичен текстовому полю TextBox

ToolStripProgressBar: индикатор прогресса, как и элемент ProgressBar

ToolStripDropDownButton: представляет кнопку, по нажатию на которую открывается выпадающее меню

К каждому элементу выпадающего меню дополнительно можно прикрепить обработчик нажатия и обработать клик по этим пунктам меню

ToolStripSplitButton: объединяет функциональность ToolStripDropDownButton и ToolStripButton

Добавить новые элементы можно в режиме дизайнера:

Также можно добавлять новые элементы программно в коде. Их расположение на панели инструментов будет соответствовать порядку добавления. Все элементы хранятся в ToolStrip в свойстве Items. Мы можем добавить в него любой объект класса ToolStripItem (то есть любой из выше перечисленных классов, так как они наследуются от ToolStripItem):

```
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent();

ToolStripButton clearBtn = new ToolStripButton();
clearBtn.Text = "Clear";
// устанавливаем обработчик нажатия
clearBtn.Click += btn_Click;
toolStrip1.Items.Add(clearBtn);
```

```
}
 void btn_Click(object sender, EventArgs e)
MessageBox.Show("Производится удаление");
}
}
```

Кроме того, здесь задается обработчик, позволяющий обрабатывать нажатия по кнопки на панели инструментов.

Элементы ToolStripButton, ToolStripDropDownButton и ToolStripSplitButton могут отображать как текст, так и изобажения, оибо сразу и то, и другое. Для управления размещением изображений в этих элементах имеются следующие свойства:

DisplayStyle: определяет, будет ли отображаться на элементе текст, или изображение, или и то и другое.

Image: указывает на само изображение

ImageAlign: устанавливает выравнивание изображения относительно элемента

ImageScaling: указывает, будет ли изображение растягиваться, чтобы заполнить все пространство элемента

ImageTransparentColor: указывает, будет ли цвет изображения прозрачным

Чтобы указать разместить изображение на кнопке, у свойства DisplayStyle надо установить значение Image. Если мы хотим, чтобы кнопка отображала только текст, то надо указать значение ext, либо можно комбинировать два значения с помощью другого значения ImageAndText:

Все эти значения хранятся в перечислении ToolStripItemDisplayStyle.

Также можно установить свойства в коде c#:

```
ToolStripButton clearBtn = new ToolStripButton();

clearBtn.Text = "Поиск";

clearBtn.DisplayStyle = ToolStripItemDisplayStyle.ImageAndText;

clearBtn.Image = Image.FromFile(@"D:\Icons\0023\search32.png");

// добавляем на панель инструментов

toolStrip1.Items.Add(clearBtn);
```

Для создания меню в Windows Forms применяется элемент MenuStrip. Данный класс унаследован от ToolStrip и поэтому наследует его функциональность.

Наиболее важные свойства компонента MenuStrip:

Dock: прикрепляет меню к одной из сторон формы

LayoutStyle: задает ориентацию панели меню на форме. Может также, как и с ToolStrip, принимать следующие значения

HorizontalStackWithOverflow: расположение по горизонтали с переполнением - если длина меню превышает длину контейнера, то новые элементы, выходящие за границы контейнера, не отображаются, то есть панель переполняется элементами

StackWithOverflow: элементы располагаются автоматически с переполнением

VerticalStackWithOverflow: элементы располагаются вертикально с переполнением

Flow: элементы размещаются автоматически, но без переполнения - если длина панели меню меньше длины контейнера, то выходящие за границы элементы переносятся

Table: элементы позиционируются в виде таблицы

ShowItemToolTips: указывает, будут ли отображаться всплывающие подсказки для отдельных элементов меню

Stretch: позволяет растянуть панель по всей длине контейнера

TextDirection: задает направление текста в пунктах меню

MenuStrip выступает своего рода контейнером для отдельных пунктов меню, которые представлены объектом ToolStripMenuItem.

Добавить новые элементы в меню можно в режиме дизайнера:

Для добавления доступно три вида элементов: MenuItem (объект ToolStripMenuItem), ComboBox и TextBox.

Таким образом, в меню мы можем использовать выпадающие списки и текстовые поля, однако, как правило, эти элементы применяются в основном на панели инструментов. Меню же обычно содержит набор объектов ToolStripMenuItem.

Также мы можем добавить пункты меню в коде C#:

```
public partial class Form1 : Form
{
public Form1()
{
    InitializeComponent();

    ToolStripMenuItem fileItem = new ToolStripMenuItem("Файл");
```

```
        fileItem.DropDownItems.Add("Создать");
        fileItem.DropDownItems.Add(new ToolStripMenuItem("Сохранить"));

        menuStrip1.Items.Add(fileItem);

        ToolStripMenuItem aboutItem = new ToolStripMenuItem("О программе");
        aboutItem.Click += aboutItem_Click;
        menuStrip1.Items.Add(aboutItem);
    }

    void aboutItem_Click(object sender, EventArgs e)
    {
        MessageBox.Show("О программе");
    } }
```

ToolStripMenuItem в конструкторе принимает текстовую метку, которая будет использоваться в качестве текста меню. Каждый подобный объект имеет коллекцию DropDownItems, которая хранит дочерние объекты ToolStripMenuItem. То есть один элемент ToolStripMenuItem может содержать набор других объектов ToolStripMenuItem. И таким образом, образуется иерархическое меню или структура в виде дерева.

Если передать при добавление строку текста, то для нее неявным образом будет создан объект ToolStripMenuItem: fileItem.DropDownItems.Add("Создать")

Назначив обработчики для события Click, мы можем обработать нажатия на пункты меню: aboutItem.Click += aboutItem_Click

Отметки пунктов меню

Свойство CheckOnClick при значении true позволяет на клику отметить пункт меню. А с помощью свойства Checked можно установить, будет ли пункт меню отмечен при запуске программы.

Еще одно свойство CheckState возвращает состояние пункта меню - отмечен он или нет. Оно может принимать три значения: Checked(отмечен), Unchecked (неотмечен)
и Indeterminate (в неопределенном состоянии)

Например, создадим ряд отмеченных пунктов меню и обработаем событие установки / снятия отметки:

```
public partial class Form1 : Form
{
   public Form1()
   {
      InitializeComponent();
       ToolStripMenuItem fileItem = new ToolStripMenuItem("Файл");
       ToolStripMenuItem newItem = new ToolStripMenuItem("Создать")
 { Checked = true, CheckOnClick = true };
      fileItem.DropDownItems.Add(newItem);
       ToolStripMenuItem saveItem = new ToolStripMenuItem("Сохранить")
 { Checked = true, CheckOnClick = true };
      saveItem.CheckedChanged += menuItem_CheckedChanged;
       fileItem.DropDownItems.Add(saveItem);
       menuStrip1.Items.Add(fileItem);
   }
    void menuItem_CheckedChanged(object sender, EventArgs e)
   {
      ToolStripMenuItem menuItem = sender as ToolStripMenuItem;
      if (menuItem.CheckState == CheckState.Checked)
         MessageBox.Show("Отмечен");
      else if (menuItem.CheckState == CheckState.Unchecked)
         MessageBox.Show("Отметка снята");
  }
}
```

Клавиши быстрого доступа

Если нам надо быстро обратиться к какому-то пункту меню, то мы можем использовать клавиши быстрого доступа. Для задания клавиш быстрого доступа используется свойство ShortcutKeys:

```
public partial class Form1 : Form
{
   public Form1()
   {
      InitializeComponent();
       ToolStripMenuItem fileItem = new ToolStripMenuItem("Файл");
```

```
        ToolStripMenuItem saveItem = new ToolStripMenuItem("Сохранить")
    { Checked = true, CheckOnClick = true };
        saveItem.Click+=saveItem_Click;
        saveItem.ShortcutKeys = Keys.Control | Keys.P;
                fileItem.DropDownItems.Add(saveItem);
        menuStrip1.Items.Add(fileItem);
      }
       void saveItem_Click(object sender, EventArgs e)
      {
        MessageBox.Show("Сохранение");
      }}
```

Клавиши задаются с помощью перечисления Keys. В данном случае по нажатию на комбинацию клавиш Ctrl + P, будет срабатывать нажатие на пункт меню "Сохранить".

С помощью изображений мы можем разнообразить внешний вид пунктов меню. Для этого мы можем использовать следующие свойства:

DisplayStyle: определяет, будет ли отображаться на элементе текст, или изображение, или и то и другое.

Image: указывает на само изображение

ImageAlign: устанавливает выравнивание изображения относительно элемента

ImageScaling: указывает, будет ли изображение растягиваться, чтобы заполнить все пространство элемента

ImageTransparentColor: указывает, будет ли цвет изображения прозрачным

Если изображение для пункта меню устанавливает в режиме дизайнера, то нам надо выбрать в окне свойство пункт Image, после чего откроется окно для импорта ресурса изображения в проект

Чтобы указать, как разместить изображение, у свойства DisplayStyle надо установить значение Image. Если мы хотим, чтобы кнопка отображала только текст, то надо указать значение Text, либо можно комбинировать два значения с помощью другого значения ImageAndText. По умолчанию изображение размещается слева от текста:

Также можно установить изображение динамически в коде:

fileToolStripMenuItem.Image = Image.FromFile(@"D:\Icons\0023\block32.pn

StatusStrip представляет строку состояния, во многом аналогичную панели инструментов ToolStrip. Строка состояния предназначена для отображения текущей информации о состоянии работы приложения.

При добавлении на форму StatusStrip автоматически размещается в нижней части окна приложения (как и в большинстве приложений). Однако при необходимости мы сможем его иначе позиционировать, управляя свойством Dock, которое может принимать следующие значения:

Bottom: размещение внизу (значение по умолчанию)

Top: прикрепляет статусную строку к верхней части формы

Fill: растягивает на всю форму

Left: размещение в левой части формы

Right: размещение в правой части формы

None: произвольное положение

StatusStrip может содержать различные элементы. В режиме дизайнера мы можем добавить следующие типы элементов:

StatusLabel: метка для вывода текстовой информации. Представляет объект ToolStripLabel

ProgressBar: индикатор прогресса. Представляет объект ToolStripProgressBar

DropDownButton: кнопка с выпадающим списком по клику. Представляет объект ToolStripDropDownButton

SplitButton: еще одна кнопка, во многом аналогичная DropDownButton. Представляет объект ToolStripSplitButton

Либо можно обратиться на панели свойств к свойству Items компонента StatusStrip и открывшемся окне добавить и настроить все элементы:

Также мы можем добавить элементы программно. Создадим небольшую программу. Определим следующий код формы:

```
public partial class Form1 : Form
{
    ToolStripLabel dateLabel;
```

```
    ToolStripLabel timeLabel;
    ToolStripLabel infoLabel;
    Timer timer;
    public Form1()
    {
        InitializeComponent();

        infoLabel = new ToolStripLabel();
        infoLabel.Text = "Текущие дата и время:";
        dateLabel = new ToolStripLabel();
        timeLabel = new ToolStripLabel();
         statusStrip1.Items.Add(infoLabel);
        statusStrip1.Items.Add(dateLabel);
        statusStrip1.Items.Add(timeLabel);
         timer = new Timer() { Interval = 1000 };
        timer.Tick += timer_Tick;
        timer.Start();
    }
     void timer_Tick(object sender, EventArgs e)
    {
        dateLabel.Text = DateTime.Now.ToLongDateString();
        timeLabel.Text = DateTime.Now.ToLongTimeString();
    }
}
```

Здесь создаются три метки на строке состояния и таймер. После создания формы таймер запускается, и срабатывает его событие Tick, в обработчике которого устанавливаем текст меток.

ContextMenuStrip представляет контекстное меню. Данный компонент во многом аналогичен элементы MenuStrip за тем исключением, что контекстное меню не может использоваться само по себе, оно обязательно применяется к какому-нибудь другому элементу, например, текстовому полю.

Новые элементы в контекстное меню можно добавить в режиме дизайнера:

При этом мы можем добавить все те же элементы, что и в MenuStrip. Но, как правило, использует ToolStripMenuItem, либо

элемент ToolStripSeparator, представляющий горизонтальную полоску разделитель между другими пунктами меню.

Либо на панели свойств можно обратиться к свойству Items компонента ContextMenuStrip и в открывшемся окне добавить и настроить все элементы меню:

Теперь создадим небольшую программу. Добавим на форму элементы ContextMenuStrip и TextBox, которые будут иметь названия contextMenuStrip1 и textBox1 соответственно. Затем изменим код формы следующим образом:

```
public partial class Form1 : Form
{
   string buffer;
   public Form1()
   {
      InitializeComponent();

      textBox1.Multiline = true;
      textBox1.Dock = DockStyle.Fill;

       // создаем элементы меню
  ToolStripMenuItem copyMenuItem = new ToolStripMenuItem("Копировать");
  ToolStripMenuItem pasteMenuItem = new ToolStripMenuItem("Вставить");
  // добавляем элементы в меню
  contextMenuStrip1.Items.AddRange(new[] { copyMenuItem, pasteMenuItem });
  // ассоциируем контекстное меню с текстовым полем
  textBox1.ContextMenuStrip = contextMenuStrip1;

 // устанавливаем обработчики событий для меню
  copyMenuItem.Click += copyMenuItem_Click;
   pasteMenuItem.Click += pasteMenuItem_Click;
    }   // вставка текста

    void pasteMenuItem_Click(object sender, EventArgs e)
    {    textBox1.Paste(buffer);

    }    // копирование текста
```

```
void copyMenuItem_Click(object sender, EventArgs e)
{   // если выделен текст в текстовом поле, то копируем его в буфер
   buffer = textBox1.SelectedText;
}}
```

У многих компонентов есть свойство ContextMenuStrip, которое позволяет ассоциировать контекстное меню с данным элементом.

В случае с TextBox ассоциация происходит следующим образом: textBox1.ContextMenuStrip = contextMenuStrip1. И по нажатию на текстовое поле правой кнопкой мыши мы сможем вызвать ассоциированное контекстное меню.

С помощью обработчиков нажатия пунктов меню устанавливаются действия по копированию и вставке строк.

4.3 Задания для выполнения лабораторной работы

1.Создайте форму с панелью инструментов (ToolBar) и главным меню (MainMenu). Разместите на панели инструментов кнопки, между которыми вставьте разделитель (New Separator).

Назначьте всем кнопкам один обработчик на событие OnClick, в который вставьте следующую строку:

Application->MessageBox(“Нажата кнопка на панели инструментов”, ”Кнопка”, MB_OK);

При нажатии правой кнопки мыши на форме должно открываться контекстное меню с пунктами. В контекстном меню реализуйте функцию выхода, справки об авторе.

4.4. Список контрольных вопросов

1. Какие компоненты помогают создать меню в приложении?
2. Как создать контекстное меню?
3. Каким образовм определить пункты меня и связанные сними действия?

5. Лабораторная работа №6.

Компоненты для представления графики

Цель работы:

Знакомство с возможностями визуальной среды и компонентов для представлении графической информации.

5.1. Теоретические сведения

Обычно результаты расчетов представляются в виде графиков и диаграмм. Библиотека .NET Framework имеет мощный элемент управления Chart для отображения на экране графической информации (рис. 1).

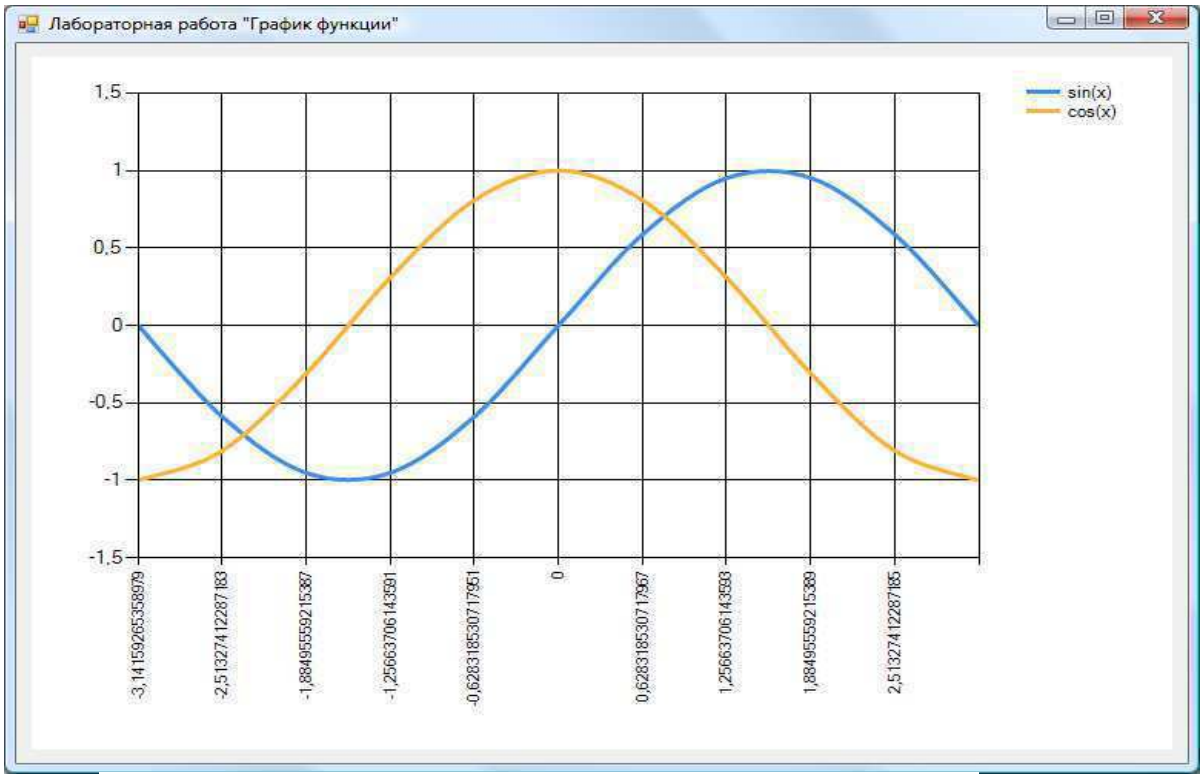


Рисунок 1 – Окно программы с элементом управления.

Построение графика (диаграммы) производится после вычисления таблицы значений функции y=f(x) на интервале [Xmin, Xmax] с заданным шагом. Полученная таблица передается в

специальный массив Points объекта Series компонента Chart с помощью метода DataBindXY. Компонент Chart осуществляет всю работу по отображению графиков: строит и размечает оси, рисует координатную сетку, подписывает название осей и самого графика, отображает переданную таблицу в виде всевозможных графиков или диаграмм. При необходимости компоненту Chart передаются данные о толщине, стиле и цвете линий, параметрах шрифта подписей, шагах разметки координатной сетки и другие настройки. В процессе работы программы изменение параметров возможно через обращение к соответствующим свойствам компонента Chart. Так, например, свойство AxisX содержит значение максимального предела нижней оси графика и при его изменении во время работы программы автоматически изменяется изображение графика.

5.2. Порядок выполнения работы

Пример: составить программу, отображающую графики функций sin(x) и cos(x) на интервале [Xmin, Xmax]. Предусмотреть возможность изменения разметки координатных осей, а также шага построения таблицы.

Прежде всего, следует определить в коде класса все необходимые переменные и константы. Конечно, можно обойтись и без этого, вставляя значения в виде чисел прямо в формулы, но это, во-первых, снизит читабельность кода программы, а во вторых, значительно усложнит изменение каких-либо параметров программы, например, интервала построения графика.

```
/// Левая граница графика
private double XMin = -Math.PI;
/// Правая граница графика
private double XMax = Math.PI;
/// Шаг графика
private double Step = (Math.PI * 2) / 10;
// Массив значений X - общий для обоих графиков
private double[] x;
// Два массива Y - по одному для каждого графика
private double[] y1;
private double[] y2;
```

Также в коде класса следует описать глобальную переменную типа Chart, к которой мы будем обращаться из разных методов:

Chart chart;

Поскольку данный класс не входит в пространства имен, подключаемые по умолчанию, следует выполнить дополнительные действия. Во-первых, в Обозревателе решений нужно щёлкнуть правой кнопкой по секции Ссылки и добавить ссылку на библиотеку визуализации (рис. 2):

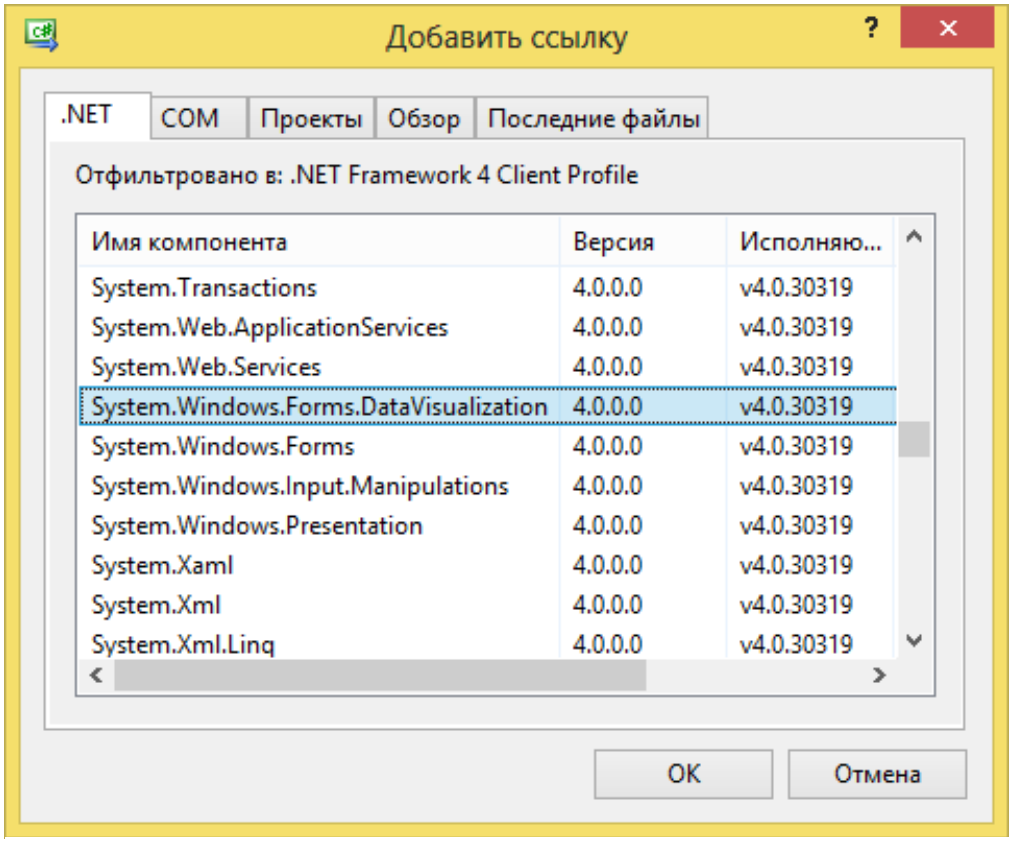


Рисунок 26 – Добавление ссылки на библиотеку визуализации

Кроме того, следует подключить соответствующее пространство имен:

using System.Windows.Forms.DataVisualization.Charting;

Далее следует определить метод, который будет расчитывать количество шагов и вычислять значения функций в каждой точке, внося вычисленные значения в массивы *x*, *y1* и *y2*:

```
///
/// Расчёт значений графика
///private void CalcFunction()
{
// Количество точек графика
int count = (int)Math.Ceiling((XMax - XMin) / Step)
+ 1;
// Создаём массивы нужных размеров
x = new double[count];
y1 = new double[count];
```

```
y2 = new double[count];
// Расчитываем точки для графиков функции
for (int i = 0; i < count; i++)
{
// Вычисляем значение X
x[i] = XMin + Step * i;
// Вычисляем значение функций в точке X
y1[i] = Math.Sin(x[i]);
y2[i] = Math.Cos(x[i]);
}
}
```

После расчёта значений нужно отобразить графики на форме с помощью элемента Chart. Элемент управления Chart нельзя выбрать с помощью панели элементов – его нужно создавать прямо в коде программы. Вторым шагом следует созать область отображения графика и настроить внешний вид осей:

```
/// Создаём элемент управления Chart и настраиваем его
///
private void CreateChart()
{
// Создаём новый элемент управления Chart
chart = new Chart();
// Помещаем его на форму
chart.Parent = this;
// Задаём размеры элемента
chart.SetBounds(10, 10, ClientSize.Width - 20,
ClientSize.Height - 20);
// Создаём новую область для построения графика
ChartArea area = new ChartArea();
// Даём ей имя (чтобы потом добавлять графики)
area.Name = "myGraph";
// Задаём левую и правую границы оси X
area.AxisX.Minimum = XMin;
area.AxisX.Maximum = XMax;
// Определяем шаг сетки
area.AxisX.MajorGrid.Interval = Step;
// Добавляем область в диаграмму
chart.ChartAreas.Add(area);
// Создаём объект для первого графика
Series series1 = new Series();
// Ссылаемся на область для построения графика
series1.ChartArea = "myGraph";
// Задаём тип графика - сплайны
```

```
series1.ChartType = SeriesChartType.Spline;
// Указываем ширину линии графика
series1.BorderWidth = 3;
// Название графика для отображения в легенде
series1.LegendText = "sin(x)";
// Добавляем в список графиков диаграммы
chart.Series.Add(series1);
// Аналогичные действия для второго графика
Series series2 = new Series();
series2.ChartArea = "myGraph";
series2.ChartType = SeriesChartType.Spline;
series2.BorderWidth = 3;
series2.LegendText = "cos(x)";
chart.Series.Add(series2);
// Создаём легенду, которая будет показывать названия
Legend legend = new Legend();
chart.Legends.Add(legend);
}
```

Наконец, все эти методы следует откуда-то вызвать. Чтобы графики появлялись сразу после запуска программы, надо вызывать их в обработчике события Load формы:

```
private void Form1_Load(object sender, EventArgs e)
{
// Создаём элемент управления
CreateChart();
// Расчитываем значения точек графиков функций
CalcFunction();
// Добавляем вычисленные значения в графики
chart.Series[0].Points.DataBindXY(x, y1);
chart.Series[1].Points.DataBindXY(x, y2);
}
```

5.3. Задания для выполнения лабораторной работы

Постройте графики функций для соответствующих вариантов из лабораторной работы №2. Таблицу данных получить путём изменения параметра X с шагом h. Самостоятельно выбрать удобные параметры настройки.

5.4. Список контрольных вопросов

1. Что такое device contex?

2. Какие возможности объектов используются для построения изображений?

3. Назовите компоненты, имеющие device contex?

4. Для чего можно использовать возможности компонента графики Chart?

5. Опишите порядок использования компонента графики Chart.

6. Лабораторная работа №7.

Разработка программных интерфейсов с элементами графики.

Цель работы:

Знакомство с возможностями визуальной среды для создания приложений с представлении графической информации. Изучение возможности Visual Studio по создание простейших графических изображений в приложении.

6.1. Теоретические сведения

Сообщение WM_PAINT. Прежде чем приступить к описанию способов рисования в окнах, применяемых приложениями .NET Frameworks, расскажем о том, как это делают «классические» приложения Microsoft Windows.

ОС Microsoft Windows следит за перемещением и изменением размера окон и при необходимости извещает приложения, о том, что им следует перерисовать содержимое окна. Для извещения в очередь приложения записывается сообщение с идентификатором WM_PAINT. Получив такое сообщение, функция окна должна выполнить перерисовку всего окна или его части, в зависимости от дополнительных данных, полученных вместе с сообщением WM_PAINT.

Для облегчения работы по отображению содержимого окна весь вывод в окно обычно выполняют в одном месте приложения — при обработке сообщения WM_PAINT в функции окна. Приложение должно быть сделано таким образом, чтобы в любой момент времени при поступлении сообщения WM_PAINT функция окна могла перерисовать все окно или любую его часть, заданную своими координатами.

Последнее нетрудно сделать, если приложение будет хранить где-нибудь в памяти свое текущее состояние, пользуясь которым функция окна сможет перерисовать окно в любой момент времени.

Здесь не имеется в виду, что приложение должно хранить образ окна в виде графического изображения и восстанавливать его при необходимости, хотя это и можно сделать. Приложение должно хранить информацию, на основании которой оно может в любой момент времени перерисовать окно.

Сообщение WM_PAINT передается функции окна, если стала видна область окна, скрытая раньше другими окнами, если пользователь изменил размер окна или выполнил операцию прокрутки изображения в окне. Приложение может передать функции окна сообщение WM_PAINT явным образом, вызывая функции программного интерфейса Win32 API, такие как UpdateWindow, InvalidateRect или InvalidateRgn.

Событие Paint.

Для форм класса System.Windows.Forms предусмотрен удобный объектно-ориентированный способ, позволяющий приложению при необходимости перерисовывать окно формы в любой момент времени. Когда вся клиентская область окна формы или часть этой области требует перерисовки, форме передается событие Paint. Все, что требуется от программиста, это создать обработчик данного события (рис.1.), наполнив его необходимой функциональностью.

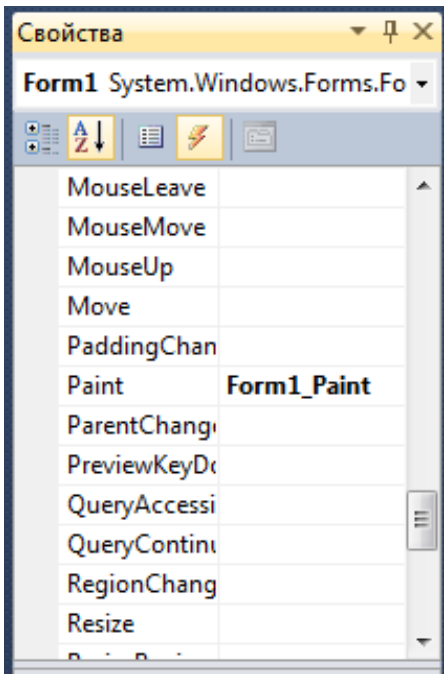


Рисунок 27 – Создание обработчика события Paint

Объект Graphics. Перед тем как рисовать линии и фигуры, отображать текст, выводить изображения и управлять ими в GDI необходимо создать объект Graphics. Объект Graphics представляет поверхность рисования GDI и используется для создания графических изображений. Ниже представлены два этапа работы с графикой.

1. Создание объекта Graphics.
2. Использование объекта Graphics для рисования линий и фигур, отображения текста или изображения и управления ими.

Существует несколько способов создания объектов Graphics. Одним из самых используемых является получение ссылки на объект Graphics через объект PaintEventArgs при обработке события Paint формы или элемента управления:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
Graphics g = e.Graphics; // Объявляется объект Graphics
// Далее вставляется код рисования
}
```

Методы и свойства класса Graphics.

Имена большого количества методов, определенных в классе Graphics, начинается с префикса Draw* и Fill*. Первые из них предназначены для рисования текста, линий и не закрашенных фигур (таких, например, как прямоугольные рамки), а вторые — для рисования закрашенных геометрических фигур. Мы рассмотрим применение только самых важных из этих методов, а полную информацию Вы найдете в документации.

Метод DrawLine рисует линию, соединяющую две точки с заданными координатами. Ниже мы привели прототипы различных перегруженных версий этого метода:

```
public void DrawLine(Pen, Point, Point);
public void DrawLine(Pen, PointF PointF;
public void DrawLine(Pen, int, int, int, int);
public void DrawLine(Pen, float, float, float, float);
```

Первый параметр задает инструмент для рисования линии — перо. Перья создаются как объекты класса Pen, например:

```
Pen p = new Pen(Brushes.Black,2);
```

Здесь мы создали черное перо толщиной 2 пиксела. Создавая перо, Вы можете выбрать его цвет, толщину и тип линии, а также другие атрибуты.

Остальные параметры перегруженных методов DrawLine задают координаты соединяемых точек. Эти координаты могут быть заданы как объекты класса Point и PointF, а также в виде целых чисел и чисел с плавающей десятичной точкой.

В классах Point и PointF определены свойства X и Y, задающие, соответственно, координаты точки по горизонтальной и вертикальной оси. При этом в классе Point эти свойства имеют целочисленные значения, а в классе PointF — значения с плавающей десятичной точкой.

Третий и четвертый вариант метода DrawLine позволяет задавать координаты соединяемых точек в виде двух пар чисел. Первая пара определяет координаты первой точки по горизонтальной и вертикальной оси, а вторая — координаты второй точки по этим же осям. Разница между третьим и четвертым методом заключается в использовании координат различных типов (целочисленных int и с плавающей десятичной точкой float).

6.2. Порядок выполнения лабораторной работы

Чтобы испытать метод DrawLine в работе, создайте приложение DrawLineApp (аналогично тому, как Вы создавали предыдущее приложение). В этом приложении создайте следующий обработчик события Paint:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
Graphics g = e.Graphics;
g.Clear(Color.White);
for (int i = 0; i < 50; i++)
{ g.DrawLine(new Pen(Brushes.Black, 2), 10, 4 * i + 20, 200, 4 * i
+ 20);
}}
```

Здесь мы вызываем метод DrawLine в цикле, рисуя 50 горизонтальных линий (рис. 2.).

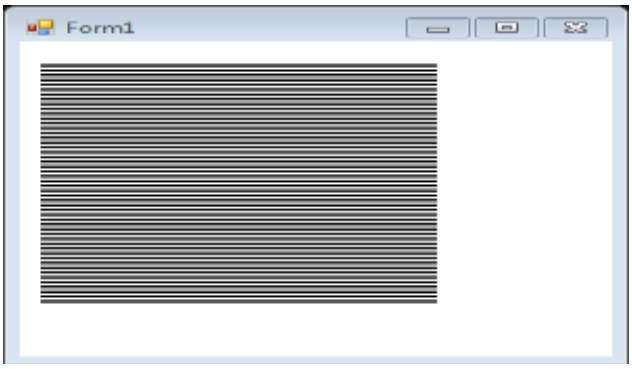


Рисунок 28 – Пример использования DrawLine

Для прорисовки прямоугольников можно использовать метод DrawRectangle(Pen, int, int, int, int); В качестве первого параметра передается перо класса Pen. Остальные параметры задают расположение и размеры прямоугольника.

Для прорисовки многоугольников можно использовать метод DrawPolygon(Pen, Point[]);

Метод DrawEllipse рисует эллипс, вписанный в прямоугольную область, расположение и размеры которой передаются ему в качестве параметров. При помощи метода DrawArc программа может нарисовать сегмент эллипса. Сегмент задается при помощи координат прямоугольной области, в которую вписан эллипс, а также двух углов, отсчитываемых в направлении против часовой стрелки. Первый угол Angle1 задает расположение одного конца сегмента, а второй Angle2 — расположение другого конца сегмента (рис.3.).

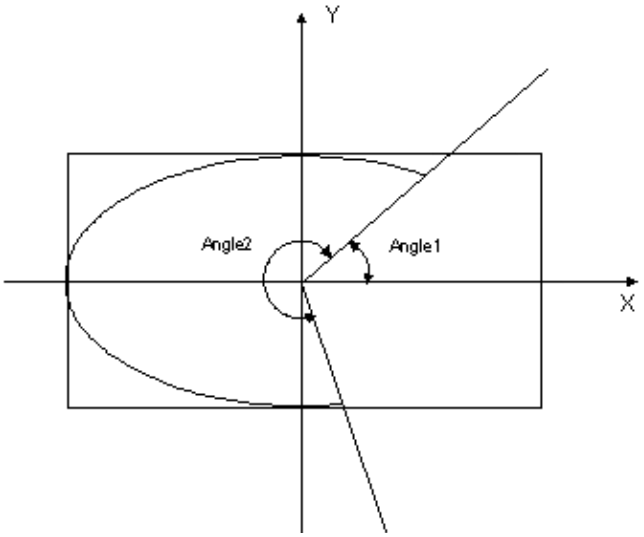


Рисунок 29 – Задание эллипса

В классе Graphics определен ряд методов, предназначенных для рисования закрашенных фигур. Имена некоторых из этих методов, имеющих префикс Fill:

FillRectangle (рисование закрашенного прямоугольника), FillRectangles (рисование множества закрашенных многоугольников), FillPolygon (рисование закрашенного многоугольника),FillEllipse (рисование закрашенного эллипса) FillPie (рисование закрашенного сегмента эллипса) FillClosedCurve (рисование закрашенного сплайна) FillRegion (рисование закрашенной области типа Region).

Есть два отличия методов с префиксом Fill от одноименных методов с префиксом Draw. Прежде всего, методы с префиксом Fill рисуют закрашенные фигуры, а методы с префиксом Draw — не закрашенные. Кроме этого, в качестве первого параметра методам с префиксом Fill передается не перо класса Pen, а кисть класса Brush.

6.3. Задание для выполнения лабораторной работы

Изучите методы и свойства классов Graphics, Color, Pen и SolidBrush.

Создайте собственное приложение выводящий на форму рисунок, состоящий из различных объектов (линий, многоугольников, эллипсов, прямоугольников и пр.), не закрашенных и закрашенных полностью. Используйте разные цвета и стили линий (сплошные, штриховые, штрих-пунктирные).

6.4. Список контрольных вопросов

1. Укажите способы рисования изображений на канве компонентов?
2. Какие методы рисования знаете?
3. Как нарисовать произвольные геометрические фигуры?
4. Как произвести закраску изображения?
5. Как установить цвета линий и закраски?

7. Лабораторная работа №8.

Компоненты для работы с базами данных

Цель работы:

Знакомство с компонентами для работы с базами данных. Получение навыков разработки приложений баз данных

7.1. Теоретические сведения

Для создания клиентского приложения используем пример базы данных с названием DB_Book.

1. Создаём приложение WindowsForms:

Файл → Создать → Проект → Приложение WindowsForms (Рис.30)

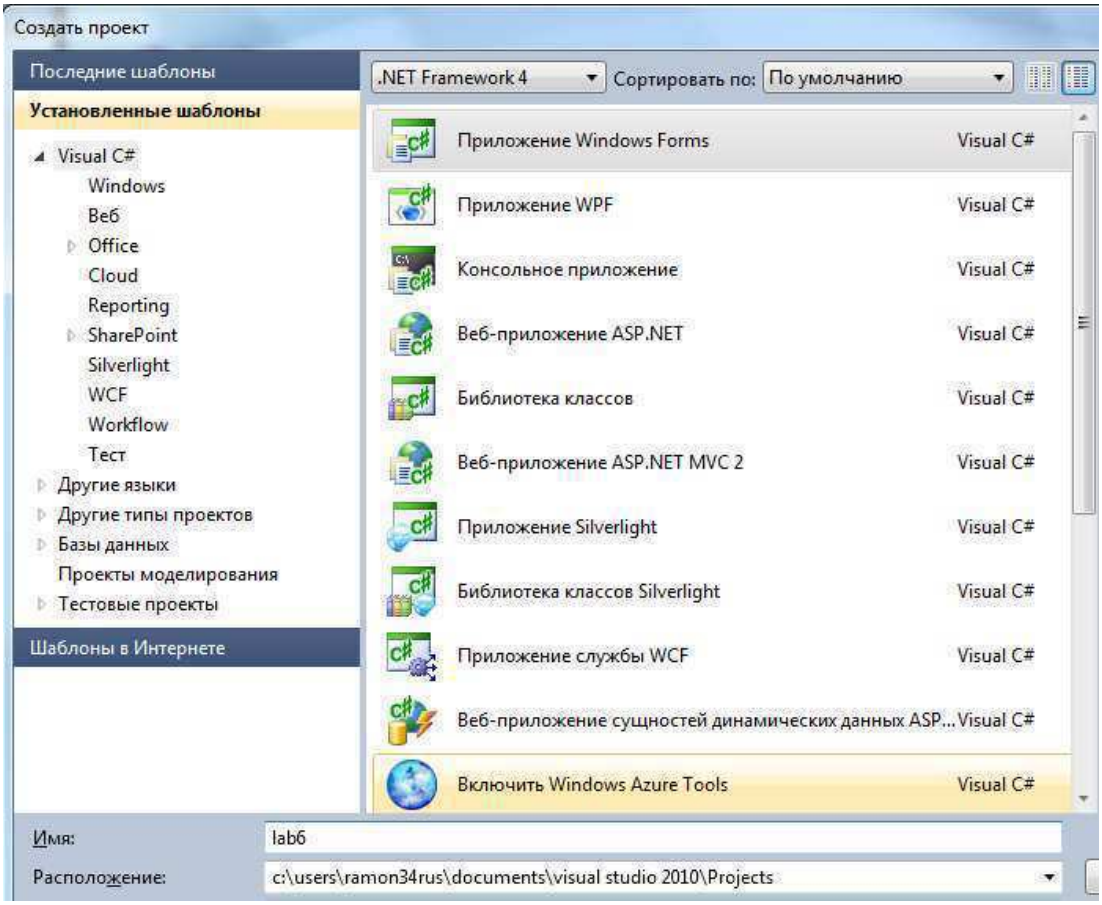


Рисунок 30 – Создание приложения WindowsForms

2. В проекте выбираем меню 'Сервис ⇒ Подключиться к базе данных (рис. 31).

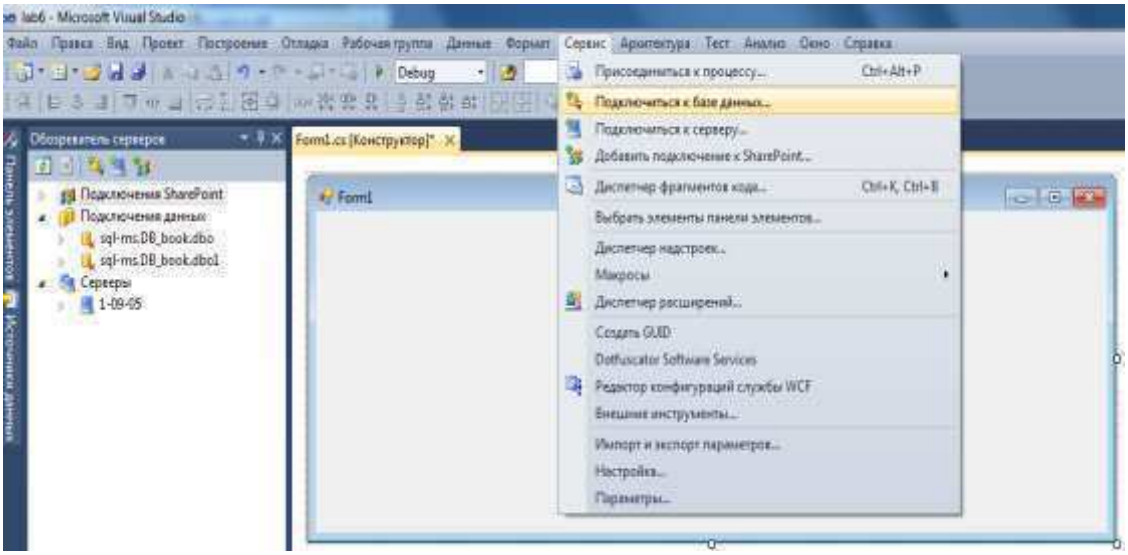


Рисунок 31 – Подключение к базе данных

3. В открывшемся окне введите имя сервера и базы данных, нажмите OK (рис. 32).

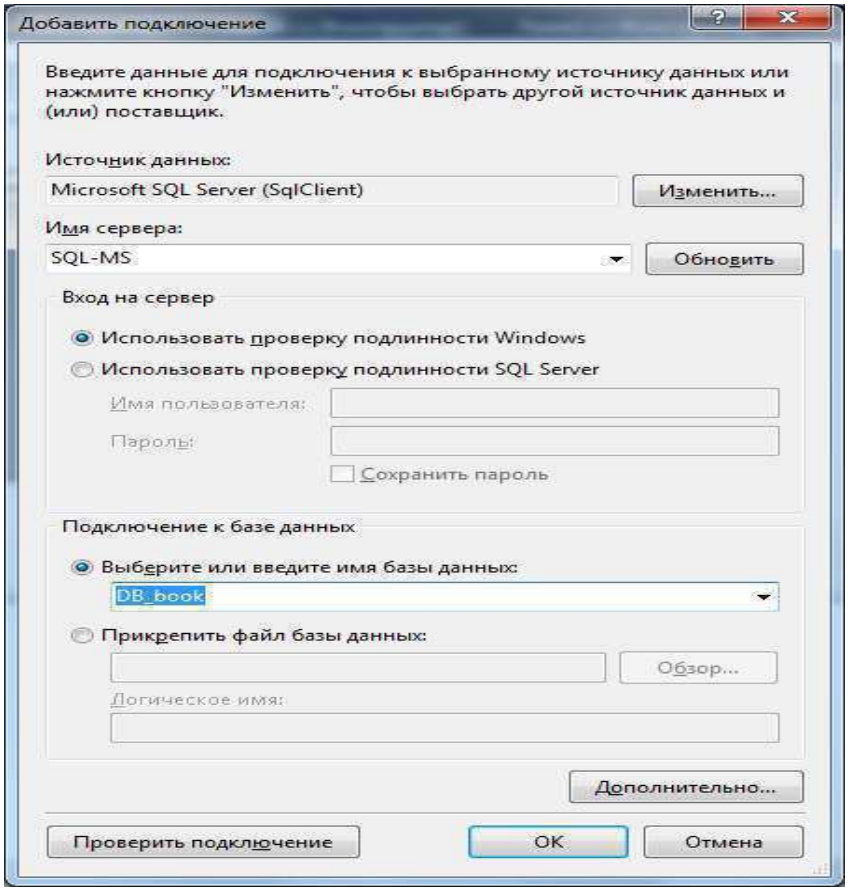


Рисунок 32 – Выбор сервера и базы данных

4. Слева в окне 'Обозреватель серверов' можно увидеть подключенную базу данных(рис. 33).

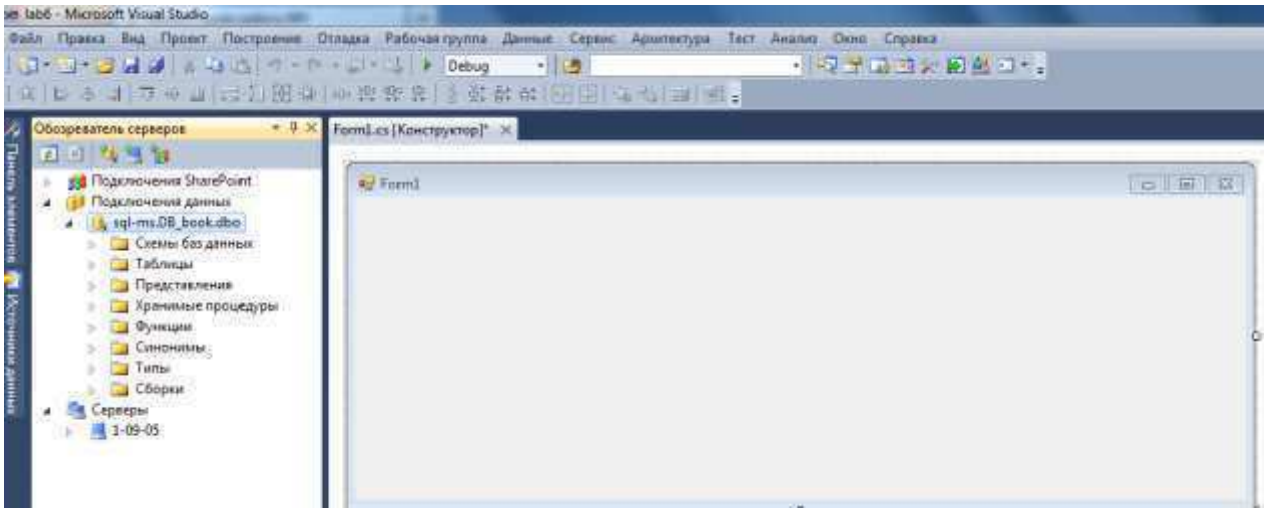


Рисунок 33 – Подключение базы данных

5. Создать 5 форм, переименовать их на FormPurchases, FormBooks, FormAuthors, FormDeliveries, FormPublish соответственно (Рис. 34-36).

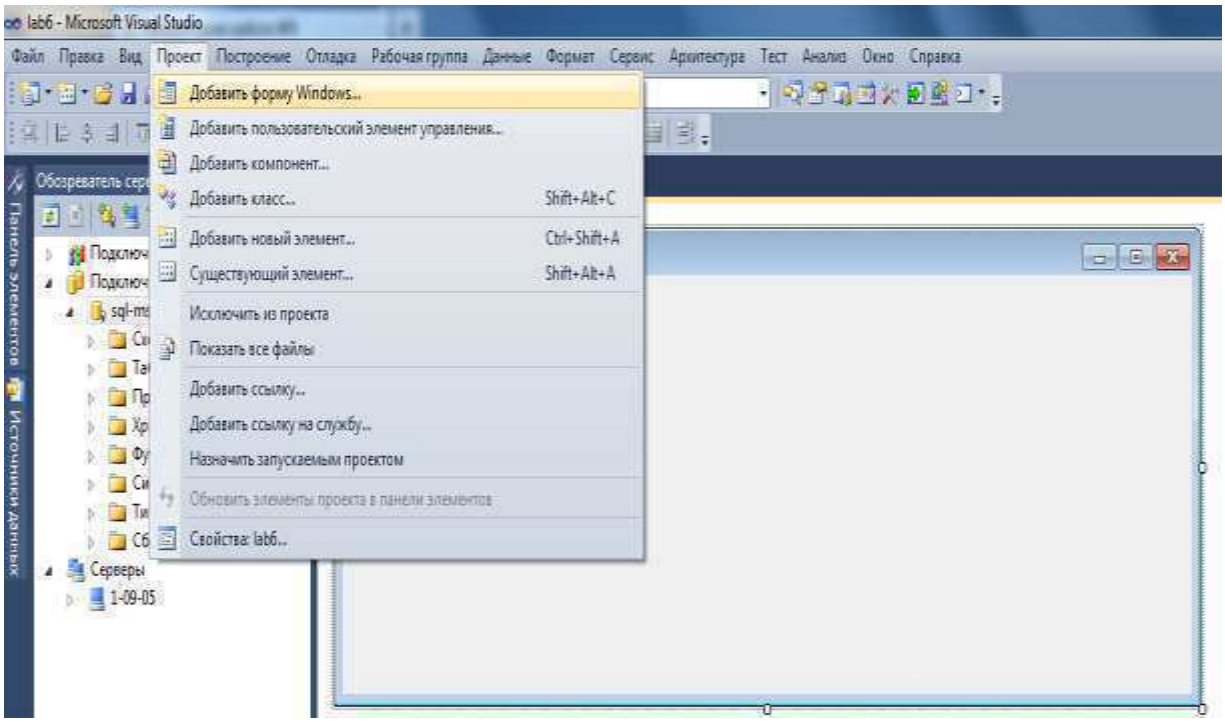


Рисунок 34 – FormPurchases

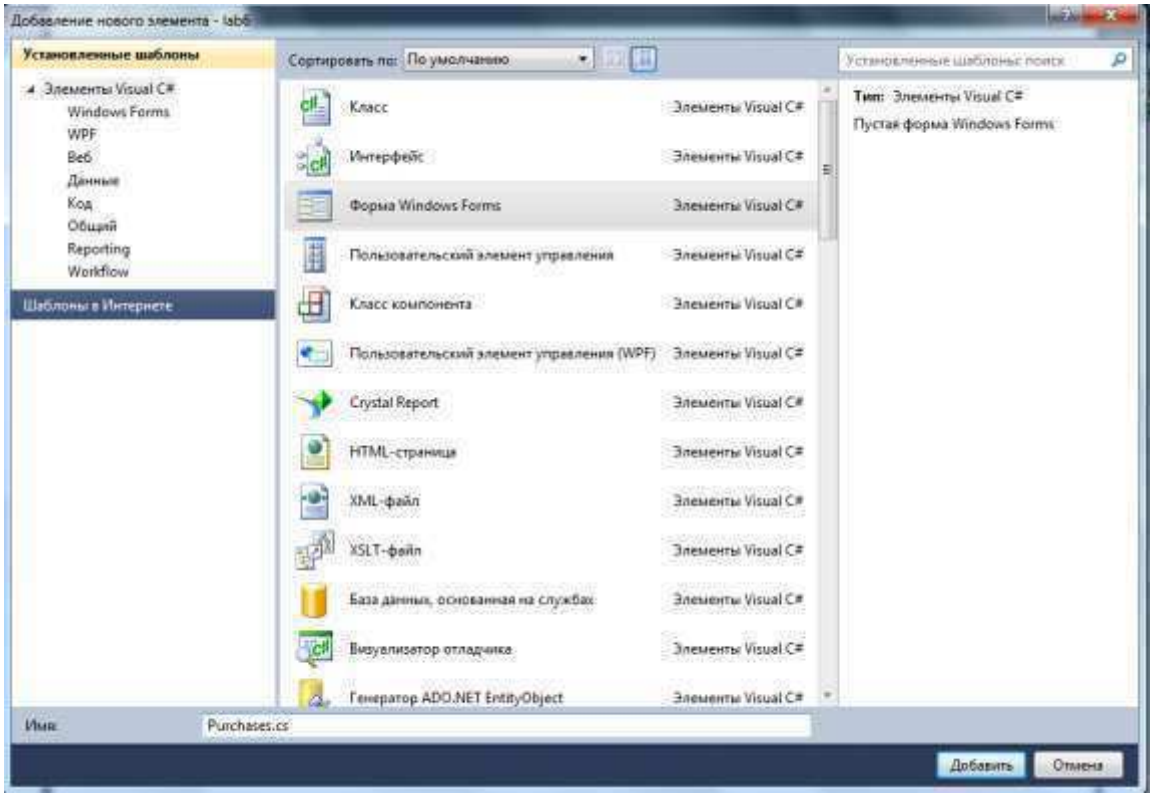


Рисунок 35 – FormBooks

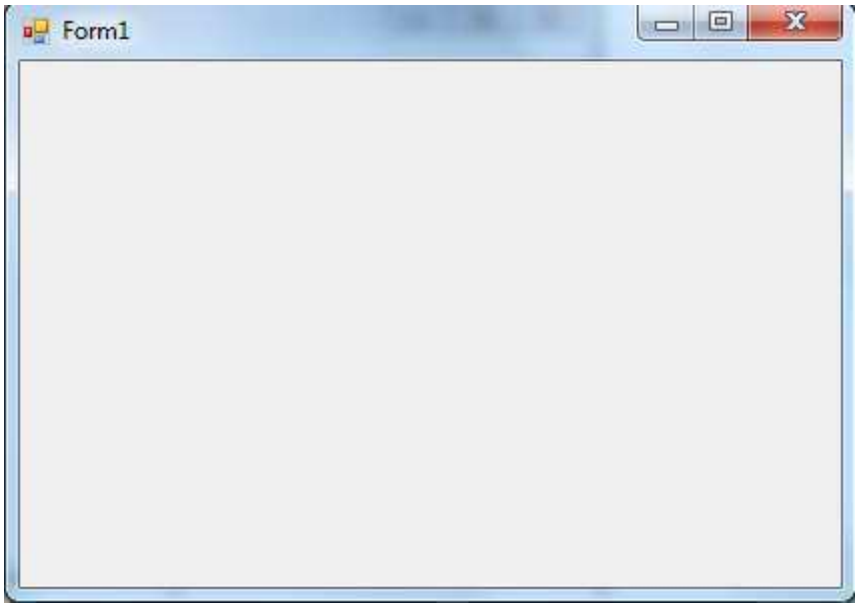


Рисунок 36 – FormAuthors

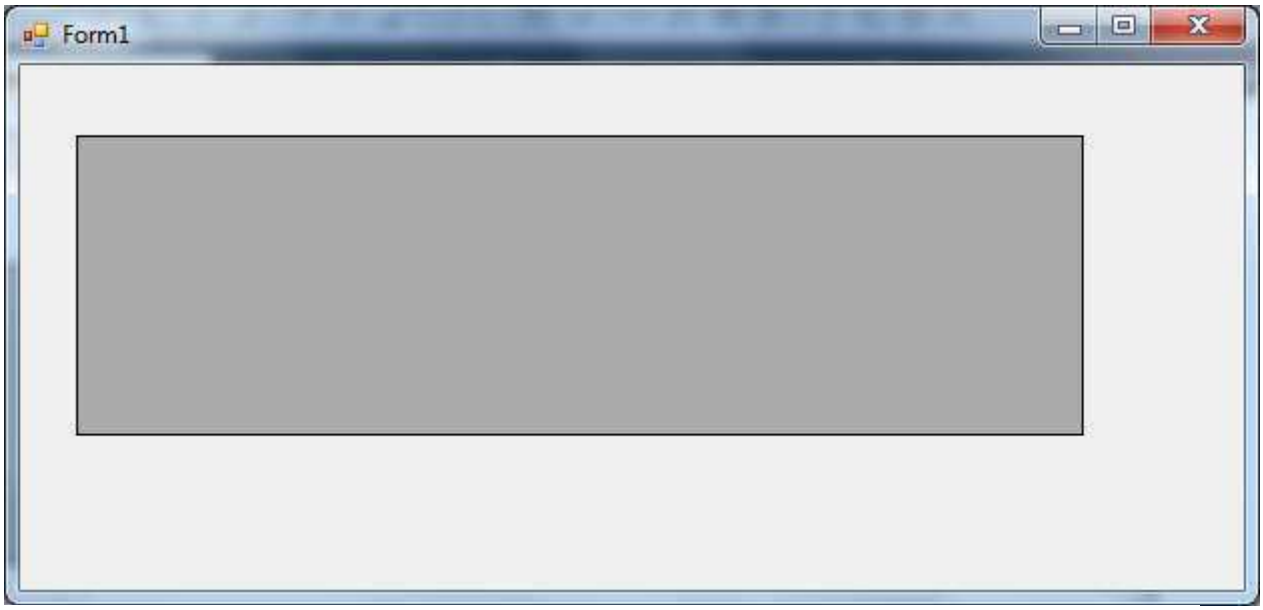


Рисунок 37 – FormPublish

6. На каждую форму добавить по компоненту типа DataGridView(рис. 38).

7. Выберем источник данных, для это во вкладке нажать 'Выберите источник данных ⇒ Добавить источник данных проекта ⇒ Базы данных ⇒ Набор данных ⇒ Далее' и выбрать необходимые таблицы (рис.38-41).

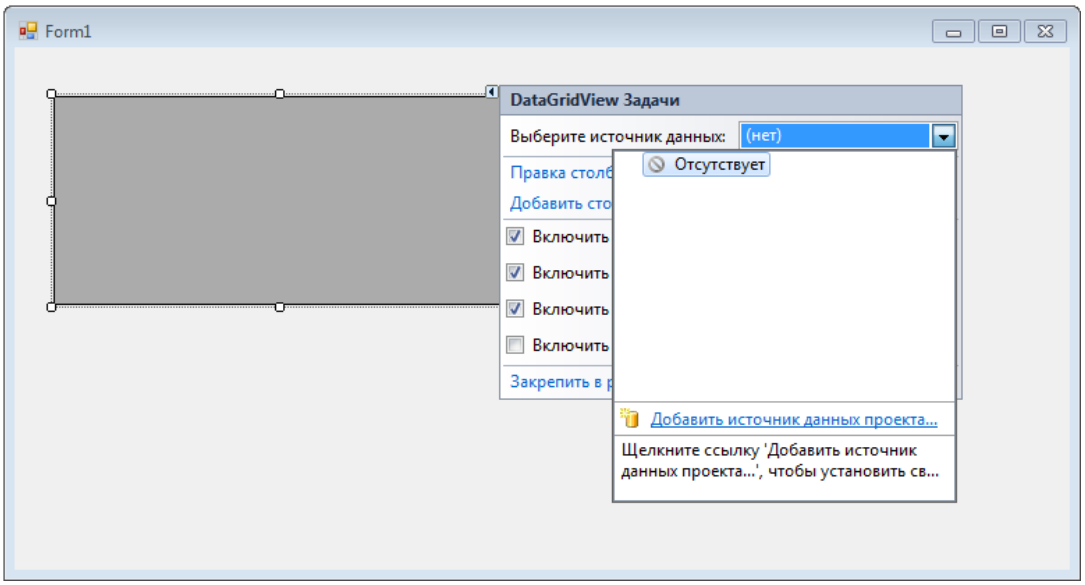


Рисунок 38 – Выбор таблицы

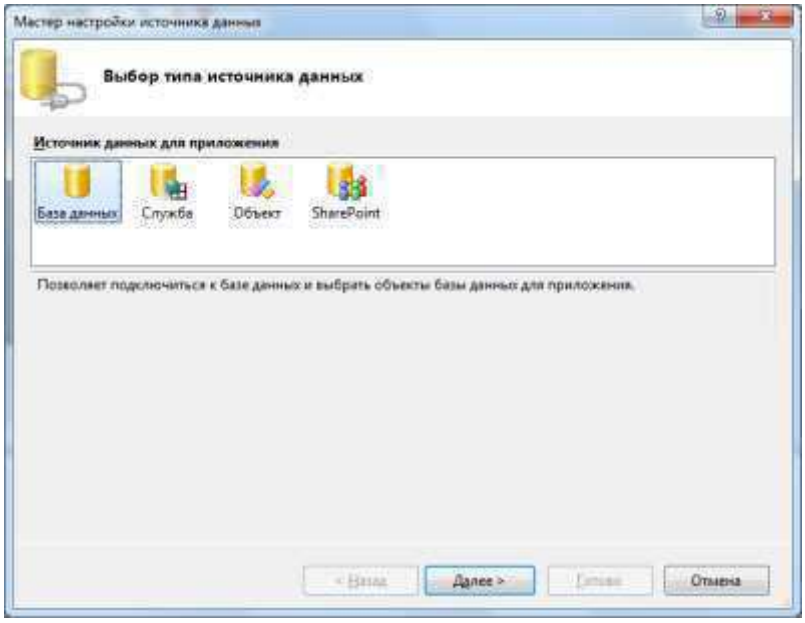


Рисунок 39 – Выбор источника данных

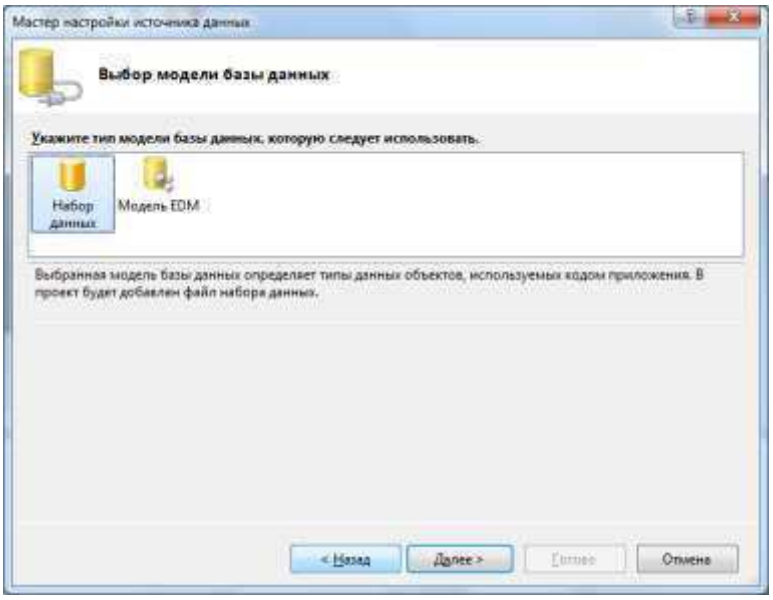


Рисунок 40 – Выбор модели данных

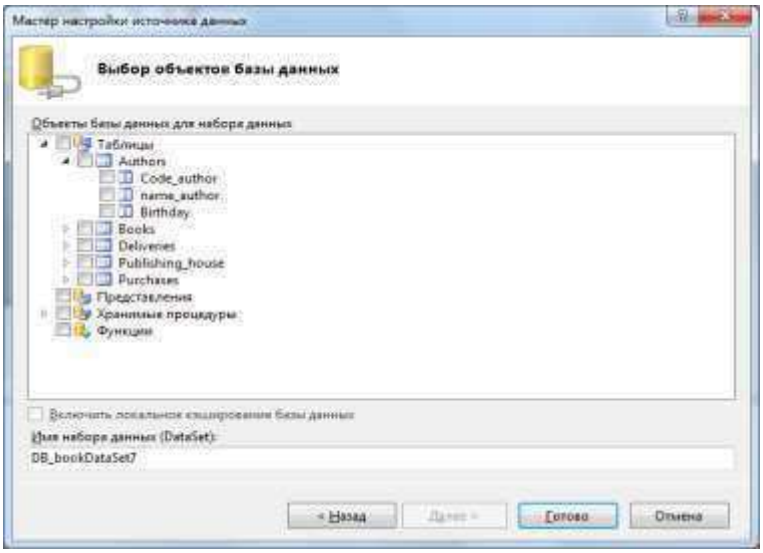


Рисунок 41 – Выбор объекта данных

8. На основной форме создать 4 компонента типа Button и в соответствующих методах Click вызвать соответствующие формы с помощью кода:

для FormAuthors:

```
FormAuthors myForm3 = new FormAuthors();
myForm3.Show();
```

для FormPurchases:

```
FormPurchases myForm3 = new FormPurchases();
myForm3.Show();
```

для FormBooks:

```
FormBooks myForm4 = new FormBooks();
myForm4.Show();
```

для FormDeliveries:

```
FormDeliveries myForm5 = new FormDeliveries();
myForm5.Show();
```

для FormPublish:

```
FormPublish myForm6 = new FormPublish();
myForm6.Show();
```

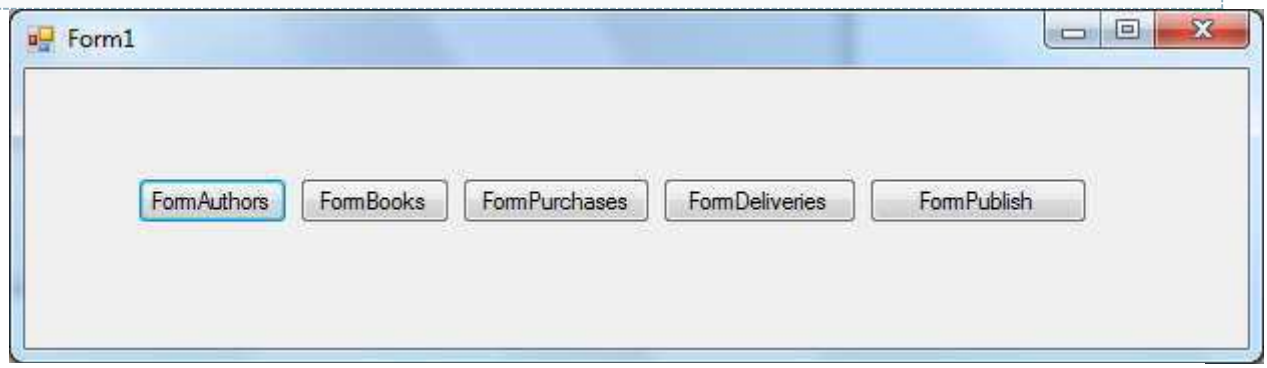


Рисунок 42 –Добавление компонентов на форму

9. На форму добавить компонент типа BindingNavigator.

Настроить у BindingNavigator свойство BindingSource для связи с созданной таблицей(значение должно совпадать со значением

свойства элемента DataGridView). Добавить компонент типа BindingNavigator на остальные формы (рис. 43).

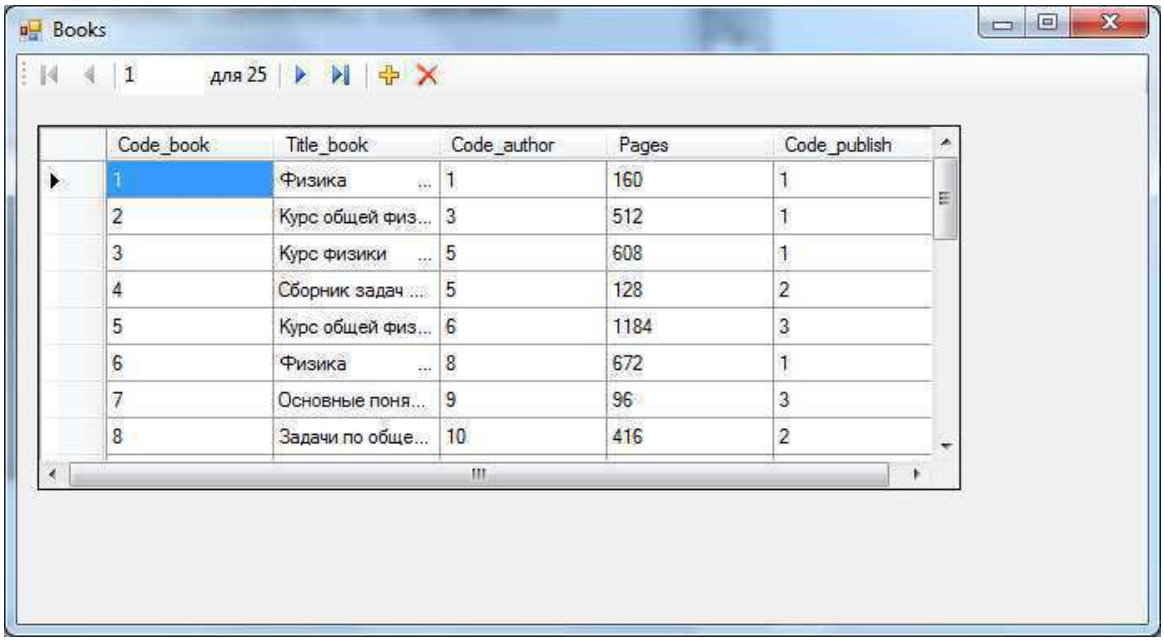


Рисунок 43 – Пример формы приложения баз данных

10. Проверить работу приложения.

11. На форму FormBooks добавить 3 компонента типа TextBox и 2 компонента типа ComboBox.

У 1-го компонента TextBox изменить свойства: (DataBinding) Text booksBindingSource - ID Книги

У 2-го компонента TextBox изменить свойства: (DataBinding) Text booksBindingSource - Название

У 1-го компонента ComboBox изменить свойства: (DataBinding) SelectedValue booksBindingSource – Code_author DataSource authorsBindingSource DisplayMember АвторValueMember Code_author

У 3-го компонента TextBox изменить свойства: (DataBinding) Text booksBindingSource - Количество страниц

У 2-го компонента ComboBox изменить свойства: (DataBinding) SelectedValue booksBindingSource – Code_publish DataSource publishinghouseBindingSource DisplayMember Публикация ValueMember Code_publish (рис. 44)

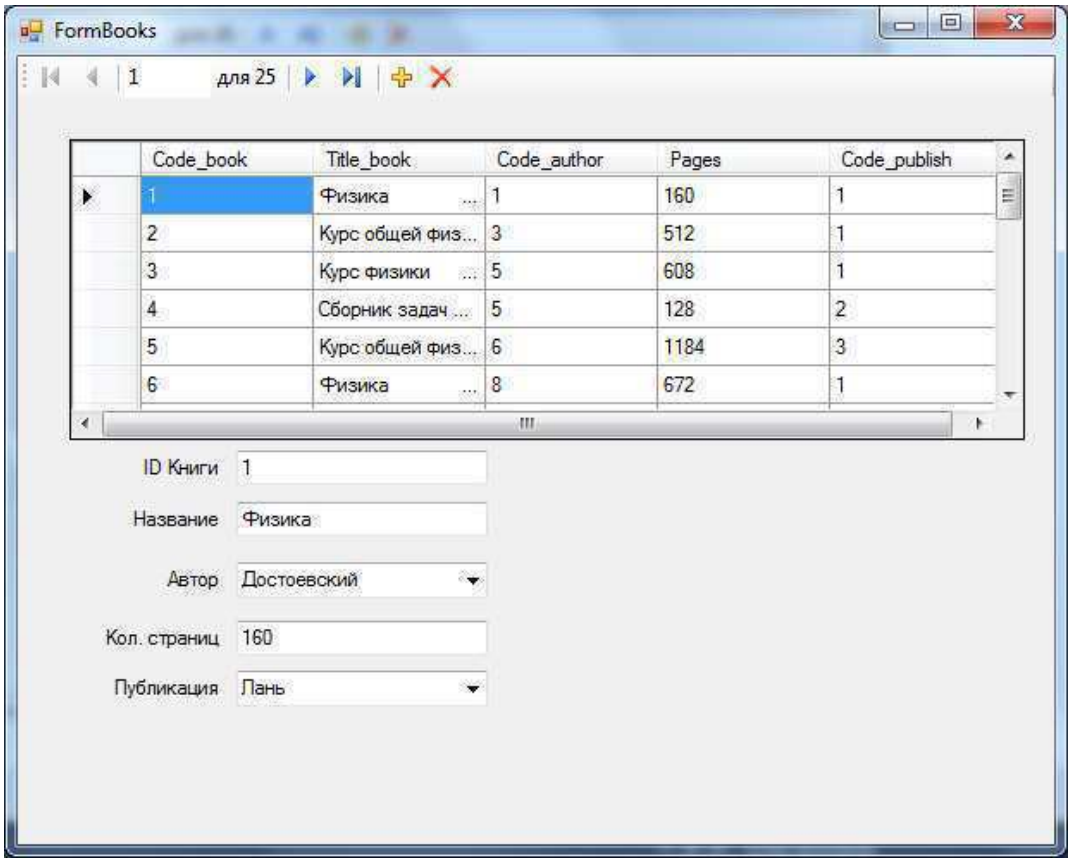


Рисунок 44 – Пример работы приложения

12. У компонента DataGridView убрать все галочки со свойств редактирования и добавления (рис. 45).

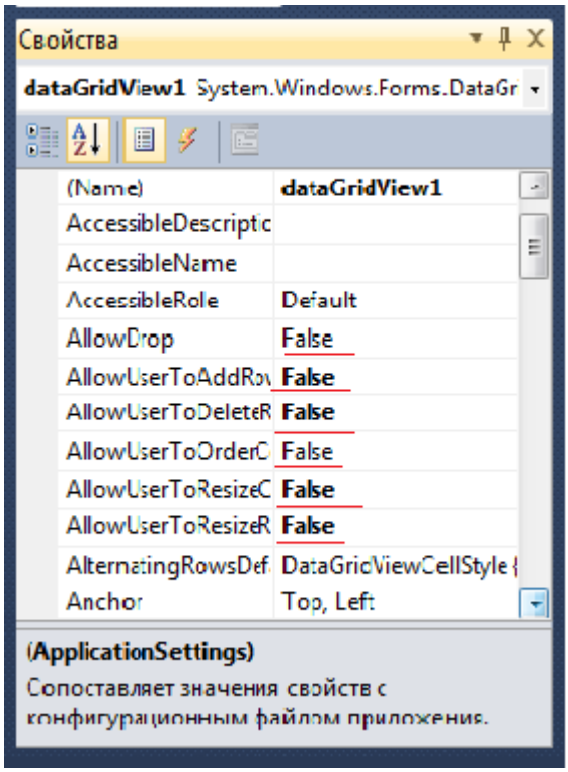


Рисунок 45 – Свойства редактирования и добавления

13. На форму FormBooks добавить компонент типа Button (кнопка обновления данных), свойство Text изменить на «Обновить» и прописать событие Click:

```
this.Validate();
this.booksBindingSource.EndEdit();
this.booksTableAdapter.Update(this.dB_BOOKSDataSet.Books);
```

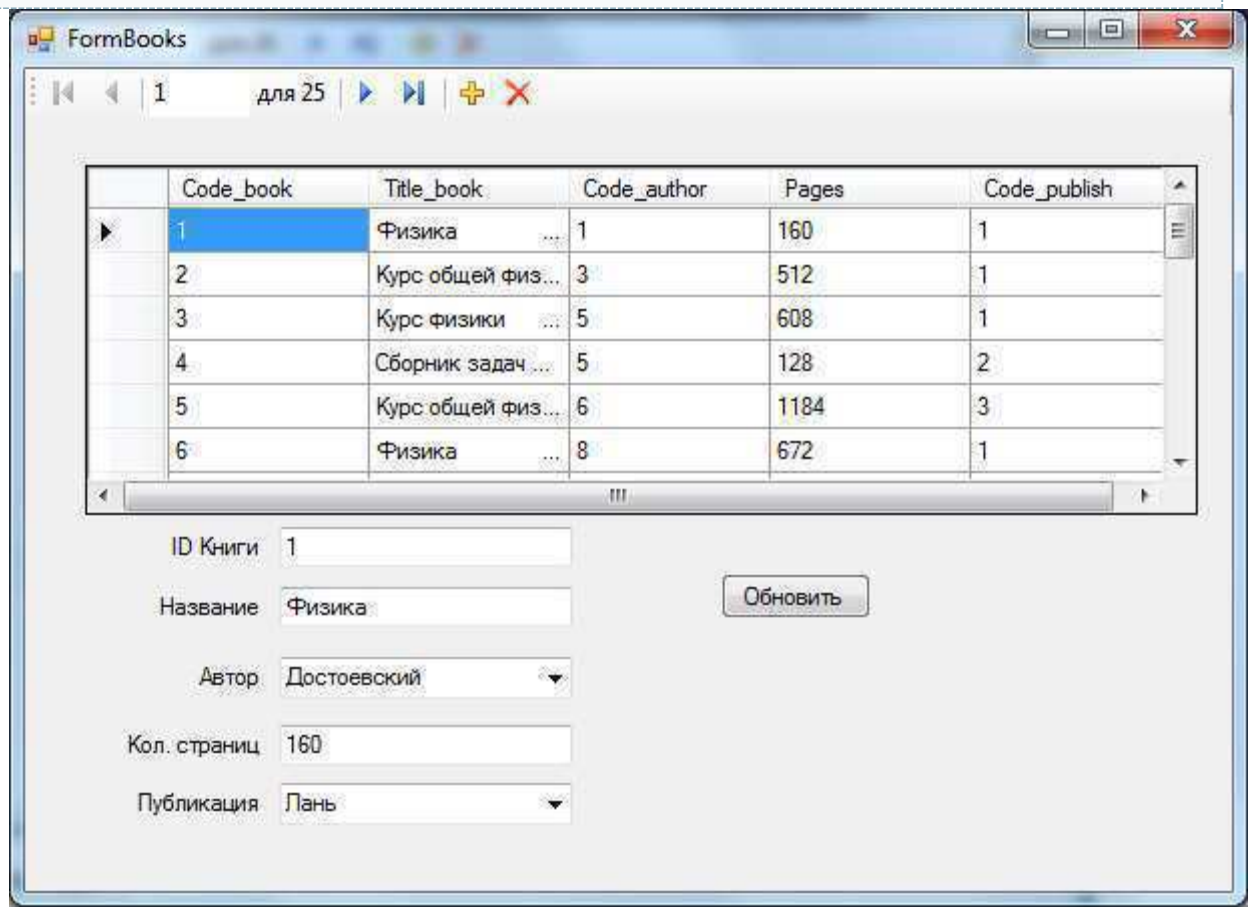


Рисунок 46 – Пример обновления данных

14. Аналогично для остальных форм добавить элементы типа TextBox.

15. Проверить работу приложения.

16. На форму FormBooks добавить 5 компонентов типа Button (рис. 18).

У 1-го компонента Button изменить свойства и метод:

Text Фильтр по текущему издательству;

В методе Click кнопки написать код:

```
int bb = dataGridView1.CurrentCell.RowIndex;
booksBindingSource.Filter = "Code_Publish = " +
dataGridView1[4,bb].Value;
```

У 2-го компонента Button изменить свойства и метод: Text Фильтр по текущему названию книги. В методе Click кнопки написать код:

```
int bb = dataGridView1.CurrentCell.RowIndex;
booksBindingSource.Filter = "Title_book = " +
dataGridView1[1, bb].Value;
```

У 3-го компонента Button изменить свойства и метод: Text Фильтр по текущему автору. В методе Click кнопки написать код:

```
int bb = dataGridView1.CurrentCell.RowIndex;
booksBindingSource.Filter = "Code_Author = " +
dataGridView1[0, bb].Value;
```

У 4-го компонента Button изменить свойства и метод: Text Фильтр по количеству книг. В методе Click кнопки написать код:

```
int bb = dataGridView1.CurrentCell.RowIndex;
booksBindingSource.Filter = "Pages = " +
dataGridView1[3, bb].Value;
```

У 5-го компонента Button изменить свойства и метод: Text Снять фильтр. В методе Click кнопки написать код:

```
booksBindingSource.Filter = "";
```

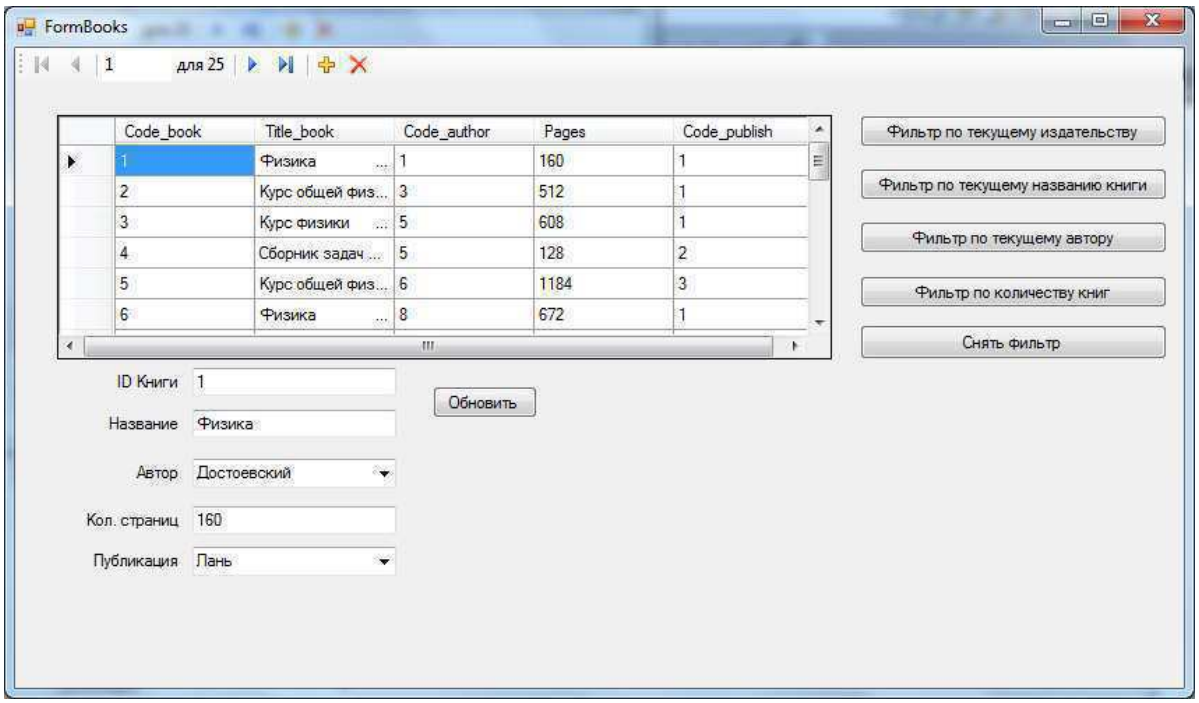


Рисунок 47 – Пример работы приложения

17. Аналогично для остальных форм добавить элементы типа Button, которые будут запускать фильтры по соответствующим значениям полей текущей записи.

18. Проверить работу приложения.

7.2. Задания для выполнения лабораторной работы

Разработать приложение с использованием подготовленных файлов баз данных. Расположить в приложении элементы управления и модификации данных в базе данных.

7.3. Список контрольных вопросов

1. Какие компоненты обеспечивают подключение в файлом БД?
2. Укажите порядок выполнения подключение в файлом БД?
3. Перечислите компоненты визуализации данных в файлах БД?
4. Каким образом осуществляется управления и модификации данных в базе данных?
5. В какой момент выполняется обновление данных в БД?

8. Лабораторная работа №9.

Разработка приложений для работы с базами данных

Цель работы:

Знакомство с подходами разработки приложений для работы с базами данных. Получение навыков разработки приложений баз данных

8.1. Теоретические сведения

При начале проектирования приложения следует определиться, какого типа создается приложение. Существуют два основных типа приложений баз данных:

- Системы поддержки решений. Такие системы используются управленческим персоналом. Необходимый уровень доступа к данным для такого приложения – просмотр данных только в режиме чтения.

- Системы обработки транзакций. Такие системы должны иметь возможность как читать, так и записывать данные в базу данных.

Рассмотрим порядок создания Учет договоров продажи компьютеров» как приложения обработки транзакций, т.к. пользователь должен иметь возможность просматривать и корректировать файлы БД.

После подготовительной работы по созданию модели предметной области, и построению блок-схемы основных функциональных процессов системы, следует организовать иерархию форм приложения.

Во-первых, следует продумать количество форм приложения и сгруппировать их по видам:

1. Главная форма
2.Формы ввода и редактирования данных

3. Формы просмотра (сетка)
4. Формы вспомогательные (пароля, справки, вопроса, ..)
Построить дерево форм приложения (рис.2).

Для достижения единообразного стиля оформления всех форм одного приложения и сокращений времени подготовки форм имеет смысл использовать возможности ООП – механизм наследования форм.

Прежде чем начать подготовку форм создадим форму модуля данных.

Модуль данных – это особый тип формы, предназначенный для содержания невидимых компонентов доступа к файлам базы данных. Эта форма будет содержать все таблицы базы данных и средства доступа к ним.

Сохранить созданный модуль в репозитарии объектов, для чего щелкнуть правой кнопкой мыши на модуле и выбрать «AddRepository». При сохранении выбрать вкладку DataModule.
Закрыть форму модуля данных. Теперь данных модуль может быть использован всегда, когда нужно получить доступ к базе данных

Достоинство использования формы модуля заключается в том, что эта форма позволяет установить формат предъявления данных в приложении, может быть сохранена в репозитарии объектов и повторно использована.

Преимуществом размещения невизуальных компонентов в модуле данных является то, что изменение значения любого свойства проявится сразу же во всех модулях, к которым подключен этот модуль данных. Кроме этого, все обработчики событий этих компонентов, т.е. вся логика работы с данными приложения собраны в одном месте, что тоже весьма удобно.

С помощью формы модуля данных выполнить настройку вида таблиц в приложении, для чего открыть форму модуля, выделить требуемую таблицу и дважды щелкнуть по ней мышью, в появившемся окне выделить требуемое поле и настроить его свойства (DisplayLabel –отражаемое имя; Width –ширина поля; маска ввода, например для поля телефон TditMask.

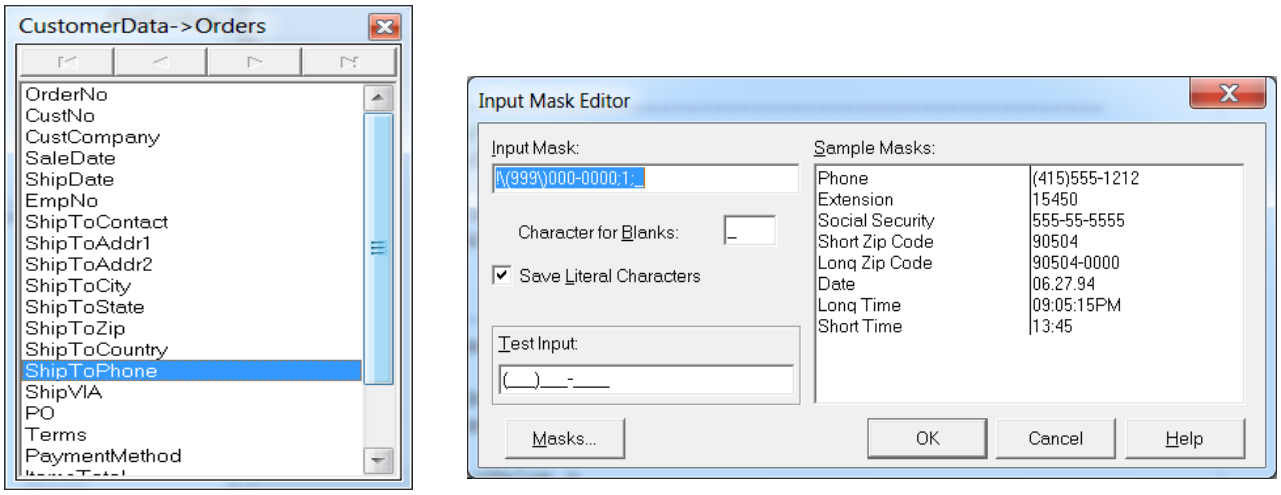


Рисунок 48 - Настройка свойств полей с использованием модуля данных приложения

Таким образом, выделяя каждое поле можно настроить его отображение в форме.

Использование модулей данных для размещения невизуальных компонент не является обязательным требованием при разработке приложений, но считается "хорошим тоном" в программировании.

2. **Создание иерархии форм приложения**

Создадим макет формы как образец для всех форм приложения. Для чего выполним команду File/New/Form. Изменим имя Name=mfAnyForm и установим свойство Position=poScreenCenter.

Все формы приложения будут иметь трехпанельную конструкцию -Расположим на форме три компанента Panel.

Настроим их следующим образом:

1. Panel 1 – Align=allTop
   Height=117
   Caption=””

2 . Panel 3 – Align=allBottom
   Height=39
   Caption=””

3 . Panel 2 – Align=allClient

Height=117
Caption=””

Сохранить заготовку в репозитарии, для чего щелкнуть правой кнопкой мыши на модуле и выбрать «AddRepository». При сохранении выбрать вкладку Forms.

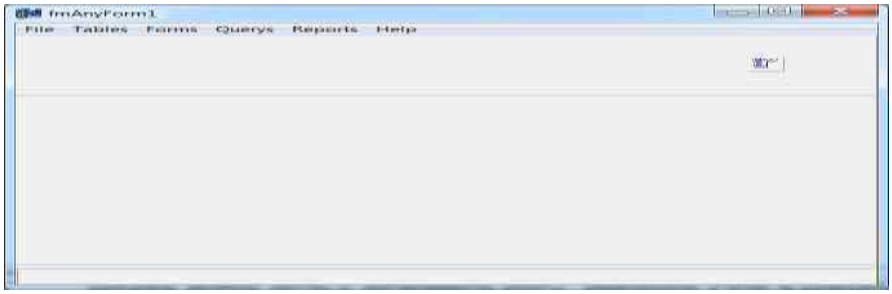


Рисунок 49 - Общий вид главной формы приложения

Послесоздания базовой формы можно начинать создавать отдельные формы приложения, для чего выпонить File/New/Other (Forms) и выбрать созданную заготовку mfAnyForm. При этом в диалогк выбора учтановит переключатель Inherit (наследовать!), при этом создается новая форма с именем mfAnyForm1(2,3,4..), унаследовавшая все особенности базовой формы. Изменить имя формы на ManeForm. Расположить на ней MainMenu, ToolBar, StatusBar.

Создать форму ввода и редактирования. Выбрать Panel3 и расположить на ней Panel4, Align=alRight, Bevel, Outer=bvNone, на панеле разместить три компонентв BtnButton, задать им тип Kind={ bkOk, bkCancel, bkHelp). Разметить на Panel3 компонент DBNavigator, прижав к левому краю панели (рис.4).

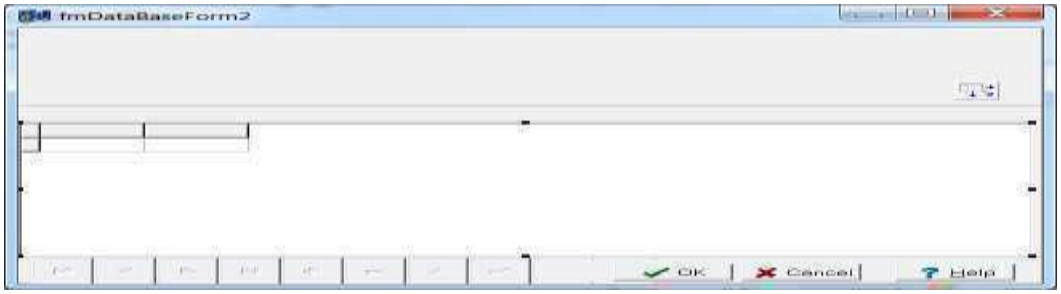


Рисунок 50 - Форма просмотра и редактирования данных приложения

8.2. Задания для выполнения лабораторной работы

Разработать приложение с использованием подготовленных файлов баз данных. Расположить в приложении элементы управления и модификации данных в базе данных.

8.3. Список контрольных вопросов

1. Какие компоненты обеспечивают подключение в файлом БД?
2. Укажите порядок выполнения подключение в файлом БД?
3. Перечислите компоненты визуализации данных в файлах БД?
4. Каким образом осуществляется управления и модификации данных в базе данных?
5. В какой момент выполняется обновление данных в БД?

9. Лабораторная работа №10.

Связывание приложений. Технология ActivX.

Цель работы:

Знакомство с технологией ActivX. Получение навыков разработки приложений с элементами ActivX.

9.1. Теоретические сведения

Спецификация *ActiveX* является составной частью технологии COM. Элементы *ActiveX* оформляются, как правило, в виде файлов динамических библиотек с расширением **.ocx**, которые затем регистрируются в реестре Windows и затем могут использоваться в программах, написанных на различных языках с использованием различных систем (Visual Basic, PowerBuilder, Delphi), а также в приложениях, работающих с Интернет.

Ниже рассмотрен простой пример создания элемента *ActiveX* на базе компонента *MonthCalendar*.

Создадим новый проект динамической библиотеки или библиотеки ActiveX (*ActiveX Library*). Затем выполним команду File|New|Other и на странице *ActiveX* выберем пиктограмму *ActiveX Control*. В выпадающем списке VCL Class Name появившегося окна ActiveX Control Wizard можно выбрать элемент VCL (стандартный, или созданный вами), который нужно оформить как *ActiveX*. В данном примере будет выбран класс *TMonthCalendar.* Также в этом окне можно определить имена файлов, которые будут созданы для элемента *ActiveX*.

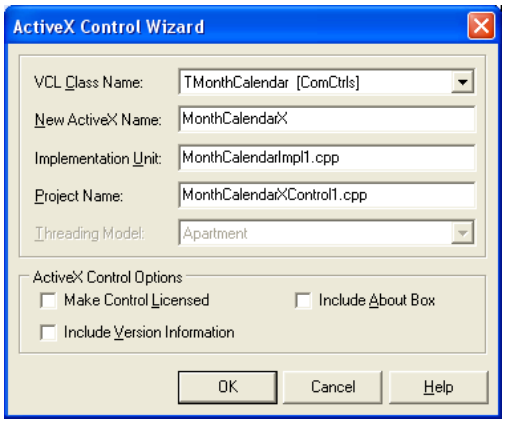


Рисунок 51 – Окно *ActiveX Control Wizard*

Индикатор Make Control Licensed обеспечивает генерацию лицензионного файла **.lic**. Если включить этот индикатор, то созданным элементом *ActiveX* потребители смогут пользоваться, только если лицензионный файл включен в переданный им завершенный проект. А использовать этот элемент в каких-то своих разработках они смогут, только если приобретут соответствующую лицензию и получат копию файла **.lic**. Индикатор Include Version Information обеспечивает включение в результирующий файл **.ocx** информации о версии. Индикатор Include About Box обеспечивает включение в проект дополнительной формы с информацией об элементе и его авторах.

Заполнив необходимую информацию, необходимо щелкнуть на кнопку ОК. Создаст все необходимые файлы, включая библиотеку типов. Далее нужно внести необходимые изменения в текст программы и выполнить команду Run|Register ActiveX server, чтобы зарегистрировать элемент в реестре Windows.

После регистрации компонента в системном реестре Windows его можно использовать в других проектах, которые можно создавать в других средствах разработки, на других языках программирования. Например, созданный раннее элемент ActiveX будет использоваться в приложении MS Word.

Необходимо создать в *Word* макрос, который вызывал бы форму, содержащую созданный календарь и три кнопки. При щелчке на первую кнопку в текущую позицию курсора в документе *Word* должна вставляться дата, которую пользователь выберет в ка-

лендаре. При щелчке на вторую - в документ должно заноситься число дней от текущей даты до даты, выбранной в календаре. А щелчок по третьей кнопке должен закрывать форму с календарем.

Выполним в *Word* команду Сервис|Макрос|Макросы. В открывшемся окне введем имя создаваемого макроса (например «Календарь») и щелкнем по кнопке Создать. Далее открывается окно Редактора Microsoft Visual Basic.

В окне Project нужно выделить вершину шаблона Normal, щелкнуть правой кнопкой мыши и выбрать во всплывшем меню раздел Insert|UserForm. Появится форма и панель ToolBox с компонентами, которые можно размещать на форме. Конечно, сначала в этой панели не будет компонента календаря.

Теперь нужно щелкнуть на ToolBox правой кнопкой мыши и выполнить команду Additional Controls. Появится список всех зарегистрированных элементов *ActiveX*. Включим индикатор у MonthCalendarX. Соответствующая пиктограмма появится на панели ToolBox. Разместим компонент на форме. В окне Property можно изменить какие-то его свойства.

Например, можно назвать форму **FCalendar** (свойство **Name** формы). Перенесем на форму с панели ToolBox три кнопки, расположим их примерно так, как показано на рис. 3.2, изменим надписи на них (свойство **Caption**), полезно также задать их свойства **ControlTipText** — надписи всплывающих ярлычков, позволяющие пользователю ориентироваться в назначении кнопок формы.

Сделаем поочередно двойные щелчки на кнопках и в появившихся заготовках обработчиков напишем соответствующие операторы языка Visual Basic. В результате код процедур будет иметь вид:

```
Private Sub CommandButtonl_Click()
  Selection = MonthCalendarXl.Date
End Sub
Private Sub CommandButton2_Click()
  Selection = MonthCalendarX1.Date – Date
```

```
End Sub
Private Sub CommandButton3_Click()
  End
End Sub
```

В окне Project совершим двойной щелчок на вершине создаваемого макроса (вершина «MyCalendar») и в открывшемся окне заготовки кода макроса добавим единственный оператор (см. рис. 52.):

```
FCalendar.Show  (modeless),
```

делающий форму видимой как немодальное окно. Последнее определяется параметром **modeless** и позволяет пользователю свободно переключаться между окном документа и формой.

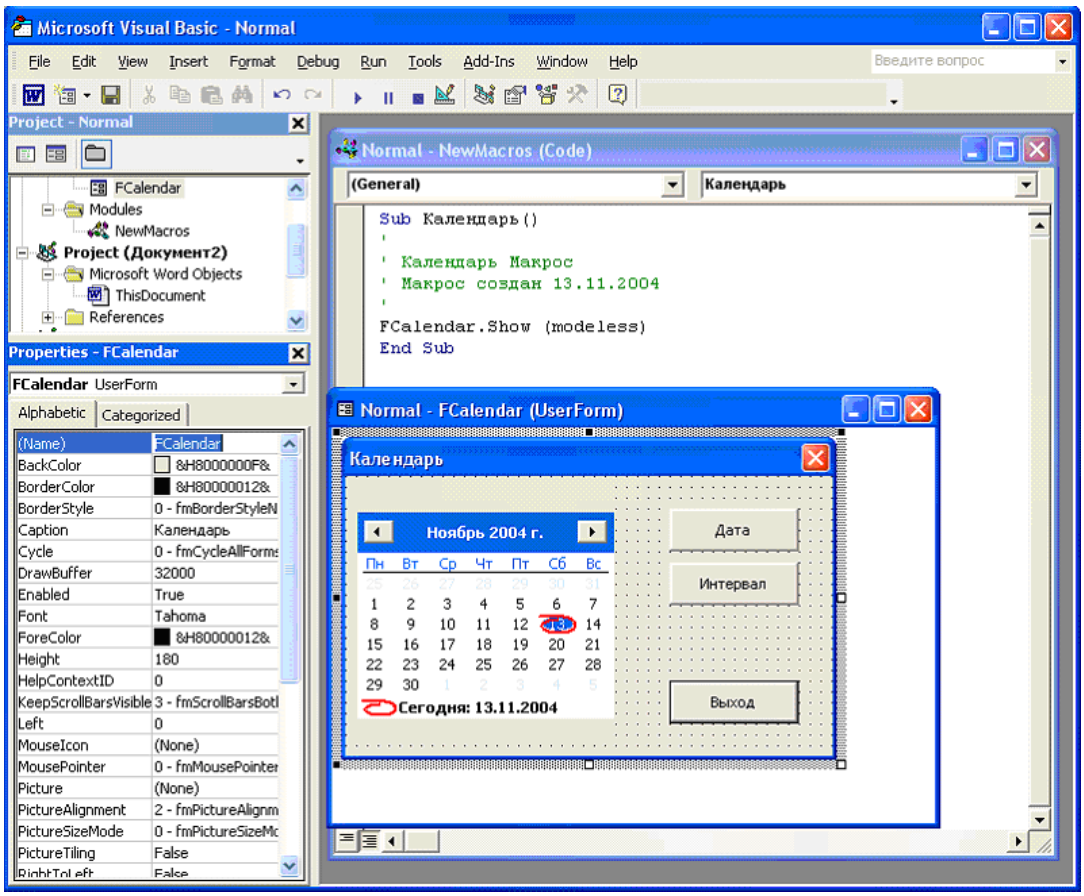


Рисунок 52 - Окно Редактора Microsoft Visual Basic

Сохраним созданный макрос, протестируем его и закроем редактор Visual Basic. Теперь для облегчения работы пользователя желательно задать макросу горячие клавиши. Для этого надо

выполнить в *Word* команду Сервис|Настройка, в открывшемся окне щелкнуть по кнопке Клавиатура, в открывшемся окне в списке Категории выбрать раздел Макросы, затем в панели Макросы выбрать макрос Календарь, перейти в окно Новое сочетание клавиш, нажать желательные клавиши (например, Ctrl-Alt-C), щелкнуть на кнопке Назначить и затем по кнопке Закрыть.

Теперь можно испытать макрос, использующий элемент *ActiveX*. Для этого необходимо открыть *Word* документ, написать какой-либо текст и нажать клавиши Ctrl-Alt-C. Появится спроектированная форма (рис. 53.). Выбирая в календаре дату и, нажимая соответствующую кнопку, можно заносить в текущую позицию курсора, выбранную дату или число дней между этой и текущей датой. В некоторых ситуациях подобный макрос может быть полезен, если часто приходится рассчитывать интервалы между различными датами. К тому же, очень легко усложнить элемент *ActiveX* и внести с помощью его библиотеки типов новые полезные свойства и методы.

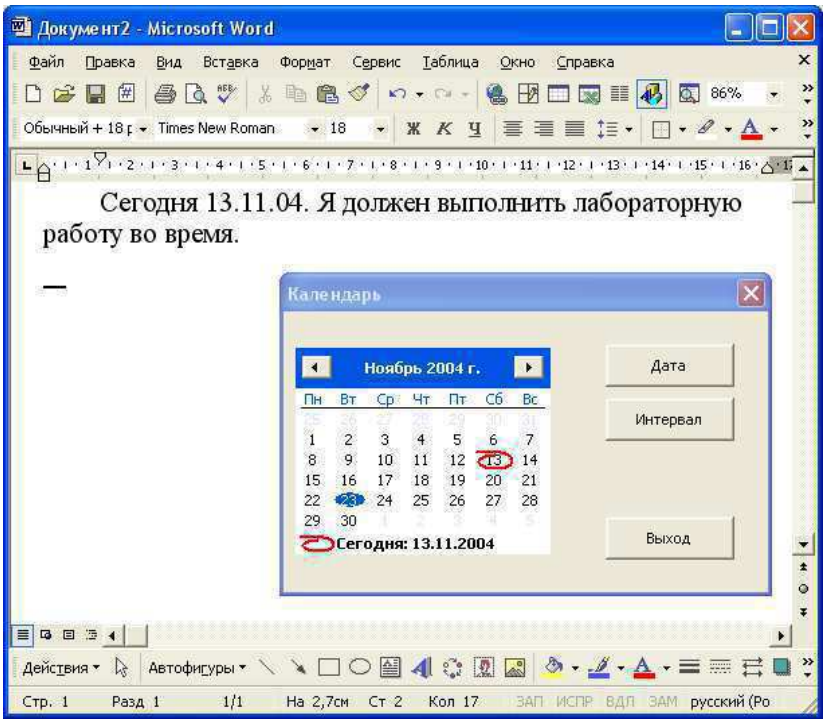


Рисунок 53 - Пример применения макроса, использующего элемент ActiveX

Рассмотренный выше пример показывает, как использовать созданный в элемент ActiveX в MS Word. Далее внимание будет уделено использованию готовых элементов ActiveX в приложениях C++ Builder.

Выполним команду Component|Import ActiveX Control. Откроется окно, представленное на рис. 54. Вверху окна расположен список всех зарегистрированных в системе элементов. Если требуется установить новый, еще не зарегистрированный элемент, необходимо нажать кнопку Add и в открывшемся окне найти файл **.ocx** устанавливаемого элемента.

Если выделить в списке какую-то строку, под списком появится полное имя соответствующего файла **.ocx**, а в окне Class names появится имя класса или нескольких классов, реализуемых данным элементом. Выпадающий список Palette page позволяет выбрать или задать новую страницу палитры компонентов, на которую будет установлен компонент. Окно Unit dir name содержит каталог модулей, использующих компонент *ActiveX*. Окно Search path содержит список путей, используемых при поиске файлов.

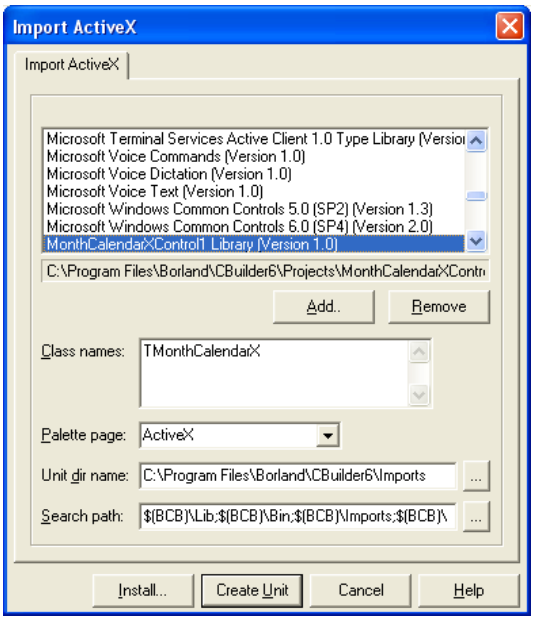


Рисунок 54- Окно импорта элемента ActiveX

Кнопка Create unit создает и открывает в окне Редактора Кода модуль элемента. При этом модуль не включается в текущий проект.

Кнопка Install (установка) открывает окно, в котором запрашивается имя имеющегося или нового пакета, в который должен устанавливаться регистрируемый компонент. Далее открывается окно Редактора Пакетов, в этом окне можно произвести установку компонента. В результате компонент появится в палитре компонентов на той странице, которая была показана на рис. 54. Теперь элемент ActiveX можно использовать в проектах как обычный VCL-компонент.

9.2. Задания для выполнения лабораторной работы

Разработать приложение с использованием подготовленных элементов ActivX.

9.3. Список контрольных вопросов

1. Какие компоненты называются элементами ActivX?
2. Укажите порядок создания элементов ActivX
3. Каким образом осуществляется взаимодействие приложения и элементов ActivX?
4. В чем преимущество элементов ActivX?

10. Лабораторная работа №11.

Контейнерные компоненты, технологии COM.

Цель работы:

Знакомство с технологией COM. Получение навыков разработки приложений с компонентами COM.

9.1. Теоретические сведения

Технология COM (Component Object Model — компонентная модель объектов) предоставляет возможность одной программе (клиенту) работать с объектом другой программы (сервера). COM — это модель объекта, которая предусматривает полную совместимость во взаимодействии между компонентами, написанными разными компаниями и на разных языках. При этом неважно, где выполняются программы: в одном потоке, в разных потоках, на одном или разных компьютерах.

С точки зрения COM приложение содержит несколько *объектов* (в частном случае может быть один объект). Каждый объект имеет один или несколько *интерфейсов*. В интерфейсе описаны методы объекта, к которым могут получить доступ внешние программы. Если интерфейсов несколько, каждый из них экспонирует некоторое подмножество методов, выполняющих однородные функции.

Каждый интерфейс имеет имя, как правило, начинающееся с символа «I», и GUID — глобальный уникальный идентификатор (Globally Unique Identifier). Идентификаторы, подобные GUID, создаются и используются не только для интерфейсов, но и для COM-компонентов, библиотек типов. Для интерфейсов GUID

называется IID. Каждый объект COM должен содержать реализацию интерфейса **IUnknown**. Этот интерфейс имеет всего три метода: **QueryInterface** — получение указателя на интерфейс по указанному IID, **AddRef** и **Release** — увеличение и уменьшение на 1 числа ссылок на объект.

Метод **Release** должен вызываться по окончании работы с интерфейсом, чтобы уведомить объект, что клиент больше не нуждается в данном объекте. **AddRef** используется только в тех случаях, когда один клиент передает ссылку на интерфейс другой программе или другому модулю свою программу. При достижении счётчика ссылок 0, COM-компонент удаляется из памяти.

Все остальные интерфейсы объектов являются наследниками **IUnknown** и наследуют те же три метода, добавляя, конечно, свои собственные. Так что сказанное выше применимо к любым интерфейсам. Среди этих интерфейсов-наследников необходимо отметить **IDispatch**. Это интерфейс, используемый клиентами серверов автоматизации OLE для доступа к методам и свойствам объекта.

IDispatch предоставляет клиентам и компонентам новый способ общения между собой. Вместо предоставления нескольких собственных интерфейсов, специфичных для его сервисов, компонент может обеспечить доступ к этим сервисам через один стандартный интерфейс, **IDispatch**.

Говоря попросту, **IDipatch** принимает имя функции и выполняет ее.

Наиболее интересны в этом интерфейсе функции **GetIDsOfNames** и **Invoke**. Первая принимает имя функции и возвращает ее диспетчерский идентификатор, или *DISPID*. *DISPID* — это длинное целое (LONG), идентифицирующее функцию. Идентификатор *DISPID* не уникален (за исключением данной

реализации **IDipatch**). У каждой реализации **IDipatch** имеется собственный *IID* (некоторые называют его *DIID*).

Для вызова функции контроллер автоматизации передает ее *DISPID* функции-члену **Invoke**. Последняя использует *DISPID* как индекс в массиве указателей на функции, что очень похоже на обычные интерфейсы COM. Однако сервер автоматизации не обязан реализовывать **Invoke** именно так. Простой сервер автоматизации может использовать оператор **switch**, который выполняет разный код в зависимости от значения *DISPID*. Именно так реализовывали оконные процедуры, прежде чем стала популярна MFC.

Способ действий **IDispatch::Invoke** напоминает вызов через виртуальную таблицу функций (*vtbl*). **Invoke** реализует набор функций, доступ к которым осуществляется по индексу. Таблица *vtbl* — массив указателей на функции, обращение к которым также идет по индексу. Однако в C++ *vtbl* статические, и компилятор работает только во время компиляции. Если программисту C++ необходимо порождать *vtbl* во время выполнения, он предоставлен самому себе. С другой стороны, легко создать универсальную реализацию **Invoke**, которая сможет легко адаптироваться для реализации самых разных сервисов.

Disp-интерфейсы

У реализации **IDispatch::Invoke** есть еще одно сходство с *vtbl*. Обе они определяют интерфейс. Набор функций, реализованных с помощью **IDispatch::Invoke**, называется диспетчерским интерфейсом (dispatch interface) или, короче, disp-интерфейсом (dispinterface). По определению, интерфейс COM — это указатель на массив указателей на функции, первыми тремя из которых являются **QueryInterface**, **AddRef** и **Release**. В соответствии с более общим определением, интерфейс — это набор функций и переменных,

посредством которых взаимодействуют две части программы. Реализация **IDispatch::Invoke** определяет набор функций, посредством которых взаимодействуют сервер и контроллер автоматизации. Как нетрудно видеть, функции, реализованные **Invoke**, образуют интерфейс, но не интерфейс COM.

На рис. 4.1. диспетчерский интерфейс представлен графически. Слева изображен традиционный интерфейс COM — **IDispatch** реализованный при помощи *vtbl*. Справа показан disp-интерфейс. Центральную роль в **disp**-интерфейсе играют *DISPID*, распознаваемые **IDispatch::Invoke**. На рисунке показана одна из возможных реализаций **Invoke** и **GetIDsOfNames**: массив имен функций и массив указателей на функции, индексируемые *DISPID*. Это только один способ. Для больших disp-интерфейсов **GetIDsOfNames** работает быстрее, если передаваемое ей имя используется в качестве ключа хеш-таблицы. Disp-интерфейсы реализуются с помощью **IDispatch** и не являются интерфейсами COM.

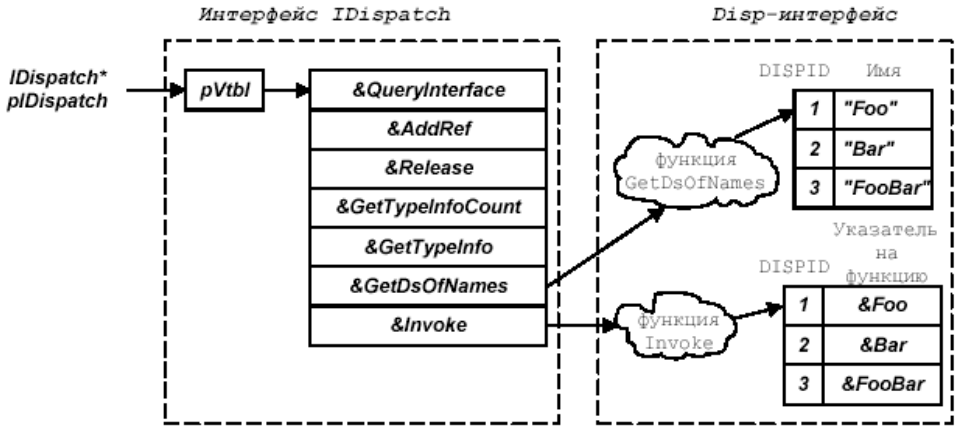


Рис. 4.1. Одна из возможных реализаций **IDispatch::Invoke**.

Дуальные интерфейсы

Метод, показанный на рис. 4.2., состоит в том, чтобы интерфейс COM, реализующий **IDispatch::Invoke**, наследовал не

IUnknown, а **IDispatch**. Так реализуют интерфейсы, называемые дуальными интерфейсами (dualinterface). Дуальный интерфейс — это **disp**-интерфейс, все члены которого, доступные через **Invoke**, доступны и напрямую через виртуальную таблицу функций *vtbl*.

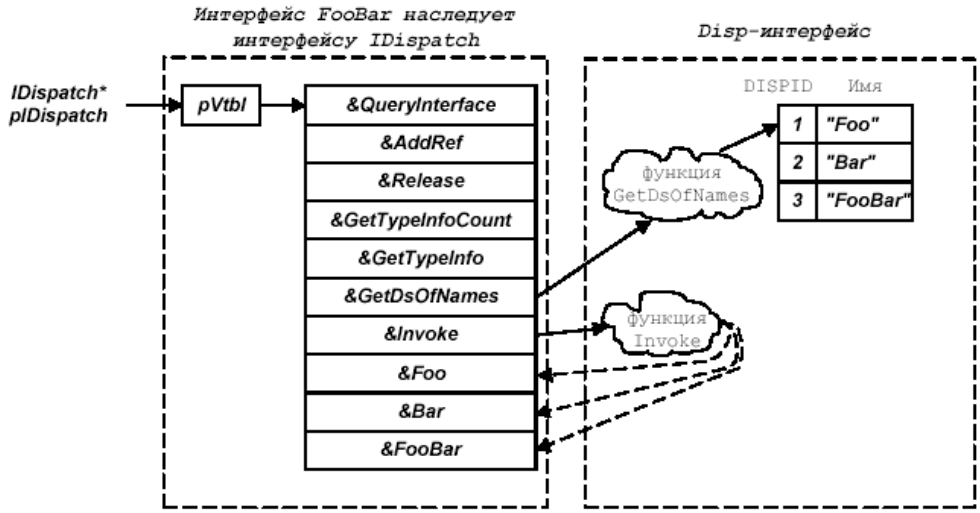


Рисунок 55 - Дуальный интерфейс — это интерфейс COM, который наследует **IDispatch**. Доступ к членам такого интерфейса возможен и через **Invoke**, и через **vtbl**.

Дуальные интерфейсы предпочтительны для реализации disp-интерфейсов. Они позволяют программистам на C++ работать через *vtbl*; такие вызовы не только легче реализовать на C++, но они и быстрее выполняются.

Макро- и микроинтерпретируемые языки также могут использовать сервисы компонентов, реализующих дуальные интерфейсы, применяя **Invoke** вместо вызова через *vtbl*. Программа на Visual Basic может работать с дуальным интерфейсом как с disp-интерфейсом, так и через *vtbl*.

Технология COM реализуется специальными библиотеками, включая OLE32.dll и OLEAut32.dll. Они содержат стандартные интерфейсы и API с функциями, обеспечивающими создание объектов COM и управление ими.

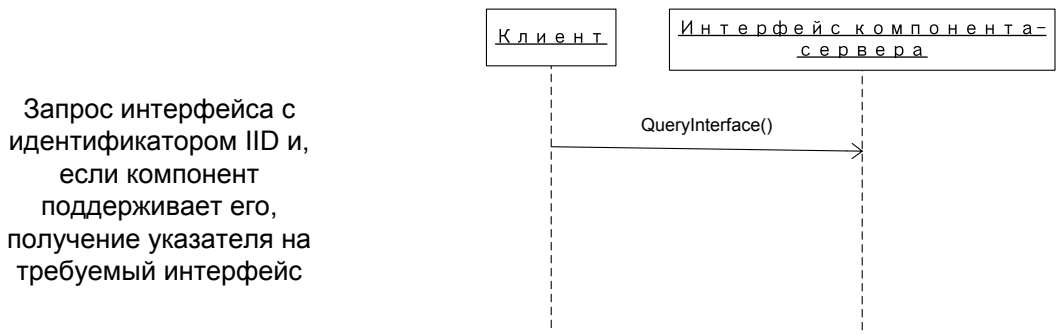
Так как технология COM не зависит от языка, в ней используются типы, отличные от других языков. Прежде всего, это относится к строкам, которые в разных языках описываются по-разному. В COM используется свой строковый тип — BSTR (Basic STRing). Он описывает строку, в начале которой указана ее длина. Поскольку длина строки известна, завершающего нулевого символа не требуется.

Создание COM – компонента

Клиент требует библиотеку COM создать COM-компонент с указанным CLSID вызовом функции **CoCreateInstance**.

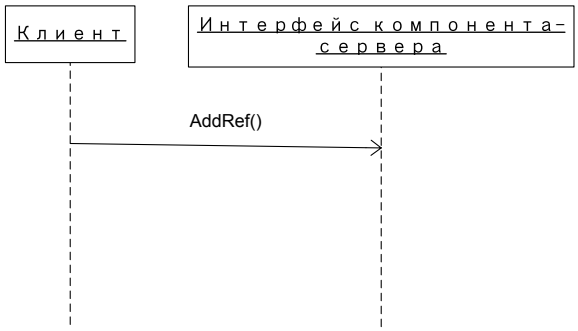
Библиотека COM ищет в ветке ветку HKEY_CLASSES_ROOT\Classes системного реестра требуемый CLSID и после успешного поиска загружает соответствующую CLSID динамическую библиотеку или исполняемый файл. Затем библиотека COM требует у загруженного файла создать экземпляр компонента и вернуть указатель на требуемый интерфейс. В случае успешного создания указатель на интерфейс компонента возвращается клиенту.

Запрос интерфейса:



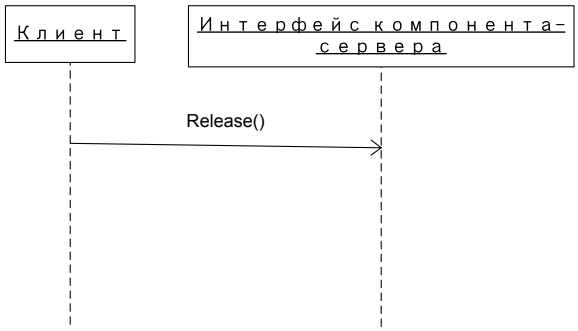
Передача интерфейса другому компоненту в качестве аргумента функции:

Перед тем, как передать указатель на интерфейс компонента другому компоненту или функции, клиент обязан увеличить счётчик использования компонента



Завершение работы с интерфейсом:

После того, как клиент закончил работу с интерфейсом компонента, он обязан уменьшить счётчик использования



10.2. Задания для выполнения лабораторной работы

Разработать приложение с использованием COM-компонета

10.3. Список контрольных вопросов

5. Какие компоненты называются COM?
6. Укажите порядок создания COM-компонета?
7. Каким образом осуществляется взаимодействие приложения и COM-компонета?
8. В чем преимущество компонента COM?

11. Лабораторная работа №12.

Разработка приложений с XML кодом

Цель работы:

Знакомство с технологией связывания приложений. Получение навыков разработки приложений с технологией связывания.

11.1. Теоретические сведения

XML (Extensible Markup Language) - это язык разметки, описывающий целый класс объектов данных, называемых XML-документами.

Язык разметки документов - это набор специальных инструкций, называемых тегами, предназначенных для формирования в документах какой-либо структуры и определения отношений между различными элементами этой структуры.

Теги языка (управляющие дескрипторы) служат в качестве инструкций для программы, производящей показ содержимого документа на стороне клиента. В самых первых системах для обозначения этих команд использовались символы “<” и “>”, внутри которых помещались названия инструкций и их параметры. Сейчас такой способ обозначения тегов является стандартным.

Примеры тегов: <note>, <background color=”red”>, </node>.

Различают открывающий и закрывающий теги. Открывающий тег определяет начало инструкции. Все данные, следующие за ним, являются параметрами данной инструкции.

Для завершения определения инструкции используется закрывающий тег.

Обозначение открывающего тега - <instruction-name>, закрывающего - </instruction-name >

Если инструкция не использует никаких данных, то отрывающий и закрывающий тег объединят в один <instruction-name/>. Следующие два примера тегов эквивалентны: <root/> и <root></root>.

11.1. Назначение XML

XML используется в качестве средства для описания грамматики других языков и контроля за правильностью составления документов. Т.е. сам по себе XML не содержит никаких тегов, предназначенных для разметки, он просто определяет порядок их создания. Таким образом, если, например, для обозначения элемента tree в документе необходимо использовать тег <tree>, то XML позволяет свободно использовать такой элемент в документах. Пример: <tree>appletree</tree>

Как видно, сам процесс создания XML-документа очень прост и требует лишь понимания тех задач, которые необходимо выполнить, используя XML в качестве языка разметки. Таким образом, у разработчиков появляется уникальная возможность определять собственные команды, позволяющие им наиболее эффективно определять данные, содержащиеся в документе. Автор документа создает его структуру, строит необходимые связи между элементами, используя те команды, которые удовлетворяют его требованиям, и добивается такого типа разметки, который необходим ему для выполнения операций просмотра, поиска, анализа документа.

Еще одним из очевидных достоинств XML является возможность использования его в качестве универсального языка запросов к хранилищам информации. Сегодня в глубинах W3C находится на рассмотрении рабочий вариант стандарта XML-QL (или XQL), который, возможно, в будущем составит серьезную конкуренцию SQL.

Кроме того, XML-документы могут выступать в качестве уникального способа хранения данных, который включает в себя одновременно средства для разбора информации и представления ее на стороне клиента. Для поиска требуемой информации предназначен язык XPath, для представления XML-данных в требуемом виде используется язык трансформаций XSLT.

XML позволяет также осуществлять контроль за корректностью данных, хранящихся в документах, производить проверки иерархических соотношений внутри документа и устанавливать единый стандарт на структуру документов, содержимым которых могут быть самые различные данные. Это означает, что его можно использовать при построении сложных информационных систем, в которых очень важным является вопрос обмена информацией между различными приложениями, работающими в одной системе. Создавая структуру механизма обмена информации в самом начале работы над проектом, менеджер может избавить себя в будущем от многих проблем, связанных с несовместимостью используемых различными компонентами системы форматов данных.

Таким образом, использование XML в качестве языка разметки документа позволяет:

- использовать единый унифицированный формат хранения документов, позволяющий избежать несовместимости различных форматов представления данных,
- осуществлять контроль за корректность данных, хранящихся в документах,
- использовать документы в качестве уникального способа хранения данных, который включает в себя одновременно средства для разбора информации и представления ее на стороне клиента.

Основной структурной единицей языка XML является элемент.

Элемент языка XML – совокупность открывающего и закрывающего тегов, соответствующих одной инструкции, а также параметров этой инструкции. Примеры элементов: <name>Sample</name>, <food-name>Apple</food-name> <price unit=”rub”>12</price></food>. Название инструкции в данном случае является **именем элемента**, значение, заключённое между открывающим и закрывающим тегом – **значением элемента**. В рассмотренных примерах значением для элемента с именем name является Sample. Элемент XML может содержать вложенные элементы, о чём свидетельствует второй пример.

Если провести аналогию XML с «человеческим языком», то можно считать, что элемент – имя существительное (характеризует предмет). Для обозначения имён прилагательных (признаков предмета) в XML используются **атрибуты**. Атрибуты включают в открывающий тег элемента после имени элемента. Синтаксис атрибута: имя-атрибута=”значение атрибута”. Примеры элементов с атрибутами: <ball color=”green”/>, <paper format=”A4” printer-kind=”laser;ink”>ALux</paper>.

Соглашения по обозначению имён элементов и атрибутов XML.

Имя может начинаться с буквы, символа подчеркивания или двоеточия.

После первого символа в имени могут быть буквы, цифры, знаки переноса, подчеркивания, точка или двоеточие.

Имена не должны содержать буквосочетания XML или вариаций на эту тему, поскольку все подобные имена защищены правами на интеллектуальную собственность консорциума W3C. Например, имя my-xml-tag будет недопустимым для обозначения элемента или атрибута XML.

Документ XML является хорошо оформленным, если соответствует всем синтаксическим правилам XML.

7.2. Структура XML документа

Рассмотрим структуру XML-документа на следующем примере.

```
<?xml version="1.0" encoding="utf-8" ?>
<group>
<name>ПЭ-00</name>
<students>
    <student sex="муж">
        <numbook>98123</numbook>
        <familyname>Иванов</familyname>
        <name>Пётр</name>
        <parentname>Сидорович</parentname>
    </student>
    <student sex="жен">
        <numbook>876456</numbook>
        <familyname>Петрова</familyname>
        <name>Нина</name>
        <parentname>Алексеевна</parentname>
    </student>
</students>
</group>
```

Любой документ XML должен начинаться с тега xml. Атрибут version определяет версию языка XML и на текущий момент всегда должен быть равен 1.0. Атрибут encoding является необязательным и

задает используемую при формировании документа кодировку. Возможные варианты кодировок для русскоязычных документов: utf-8, utf-16 (UNICODE), win-1251(кодировка Windows по умолчанию) и т.д.

Каждый документ XML должен содержать корневой элемент. **Корневой элемент** – единственный элемент документа, который включает в себя все остальные элементы документа и не входит в состав других элементов. В рассматриваемом примере корневым элементом является элемент group. Элемент group содержит элемент students, представляющий собой список студентов группы, каждый элемент которого – student – хранит информацию об одном студенте.

Иерархия элементов приведена на рисунке 57.

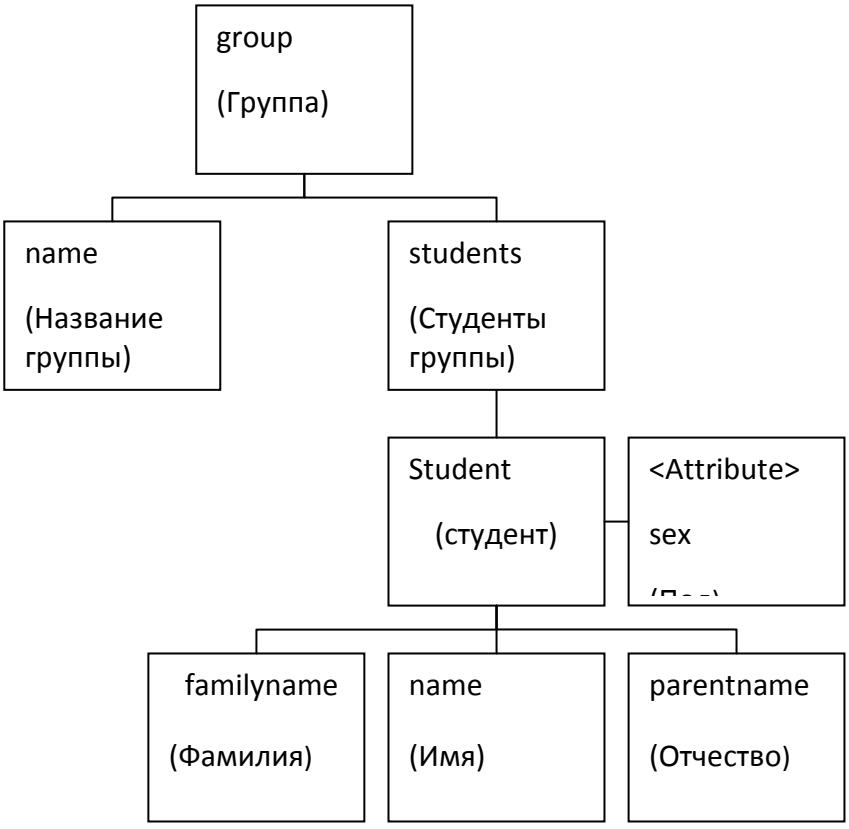


Рисунок 57- Иерархия элементов XML-документа

Откроем документ в Internet Explorer. Поскольку созданный документ является хорошо оформленным, то увидим следующее.

11.2. Задания для выполнения лабораторной работы

Разработать приложение с использованием XML-документа

10.3. Список контрольных вопросов

1. Какие компоненты называются XML-документом?
2. Укажите порядок создания XML-документа?
3. Каким образом осуществляется взаимодействие приложения и XML-документа?
4. В чем преимущество использования XML-документа

Список использованных источников

1. Ларман, К. Применение UML 2.0 и шаблонов проектирования/К. Ларман. - М.: Издательский дом «Вильямс», 2013. -736 с.
2. Иванова, Г.С. Технология программирования [Текст] : учебник / Г. С. Иванова. - М. : Кнорус, 2011. - 336 с. - ISBN 978-5-406-005 19-4.
3. Технология программирования [Электронный ресурс] : учебное пособие. - Тамбов : Издательство ФГБОУ ВПО «ТГТУ», 2013. - 173с.
4. Лафоре, Роберт. Объектно-ориентированное программирование в C++ [Текст] / Р. Лафоре. - 4-е изд. - СПб. [и др.] : Питер, 2012. - 928 с. : ил. - (Классика ComputerScience). - ISBN 978-5-4237-00 38-6.
5. Лапина, Татьяна Ивановна. Методы и технологии объектно-ориентированного программирования [Текст] : учебное пособие / Юго-Западный гос. ун-т ; Министерство образования и науки Российской Федерации, Юго-Западный государственный университет. - Курск : ЮЗГУ, 2011. - 131 с.
6. Архангельский, А. Я. Программирование в в VisualStudio [Текст] / А. Я. Архангельский. - М. : БИНОМ, 2000. - 1152 с. : ил. - Б. ц.
7. Адаптивный дизайн. Делаем сайты для любых устройств // Tim Kedlek, Спб.:Питер, Библиотека специалиста, 2013. 101-124 с.
8. C++. Объектно-ориентированное программирование. Практикум/Т.А. Павловская, Ю.А. Щупак. – СПб.:Питер, 2004.- 265с. – 3 экземпляра.
9. Голицына, Ольга Леонидовна. Программное обеспечение [Текст] : учебное пособие / О. Л. Голицына, Т. Л. Партыка, И. И. Попов. - 3-е изд., перераб. и доп. - М. : Форум, 2010. - 201 с. - ISBN 978-5-91134-3 76-7.
10. Визуальное программирование [Электронный ресурс] : методические указания по выполнению лабораторных работ / Юго-Зап. гос. ун-т; сост.: Т. Лапина, Курск, 2019. 97 е.: ил. 31, табл.4, Библиогр.: с. 23.
11. Электронная библиотека ЮЗГУ (http://www.lib.swsu.ru
12. http://www.edu.ru/ Федеральный портал Российское образование.

13. http://window.edu.ru/Электронная библиотека «Единое окно доступа к образовательным ресурсам».
14. Электронно-библиотечная система «Университетская библиотека online» (http://www.biblioclub.ru)
15. http://cbuilder.ru/http://www.atlants.ru
16. Электронно-библиотечная система «Университетская библиотека online» (http://www.biblioclub.ru)
17. Клиент-серверные технологии (http://www.sql.ru/)
18. Сайт центра «Информика»: http://www. (informika.ru);