Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

«Ивановский государственный политехнический университет»

(ИВГПУ)

Кафедра ВПМСИТ УЦСГЕН

Введение в VC++ MFC

УЧЕБНОЕ ПОСОБИЕ

по дисциплинам

«Прикладное программирование в информационных системах»,
«Программная реализация средств дизайна»

для студентов дневной формы обучения
специальности 740100
«Информационные технологии в дизайне»

Иваново 2014

Учебное пособие предназначено для студентов, изучающих основы программирования в MS VC++ системе. Данное учебное пособие направлено на необходимость приобретения студентами некоторых подходов к решению задач разработки программного обеспечения в VC++.

Данное учебное пособие является введением в изучение концепций программирования VC++ MFC и некоторых подходов к программной реализации средств дизайна средствами этой системы.

Составитель:
Доцент кафедры ВПМСИТ, к.т.н., доцент Д. Д. Ветчинин

Научный редактор:
Зав. кафедрой ВПМСИТ д.т.н., профессор Н. А. Коробов

Задание 1

1. Создать каталог Lesson_1 в директории Lesson_C++\MFC. Запустить IDE VC++.

2. VC++6.0. Через команды File – New и окно диалога New с активной вкладкой Projects выбрать вариант создания проекта - MFC AppWizard(exe). VC++2008. Проект – Создать. Тип проекта – MFC – приложение – MFC.

3. VC++6.0. В поле Project Name ввести имя проекта StringOut_1, в поле Location (расположение) выбрать директорию Lesson_C++, два других параметра оставить установленными по умолчанию и запустить AppWizard. VC++2008. Аналогично. Директорию для решения (Solution) создавать не нужно.

4. VC++6.0. В окне диалога AppWizard – Step 1 элемент OptionButton - Single document установить в True. Опции окна AppWizard - Step 2 оставить неизменными и без внесения изменений передвигайтесь к этапу AppWizard - step 6. VC++2008. Мастер приложений MFS (AppWizard), тип приложения (application type) – один документ (single document).

5. В качестве домашнего задания составьте список файлов проекта с описанием их назначения, указанием содержащихся в базовых файлах основных классов. AppWizard создаёт файл ReadMe.Txt в котором содержится требуемая краткая информация.

6. На основе проведённого в п.5 анализа содержимого проекта Вам необходимо выделить в нём четыре основные части: объект приложения, объект главного окна, объект документа, объект вида.

Объект приложения.

Объект приложения, находящийся в файлах [NameApp].h и [NameApp].cpp (файл [NameApp].h содержит определения констант, а также объявления переменных и методов ***класса),*** - то, что Widows запускает при старте программы. Когда этот объект начинает работу, он размещает на экране главное окно.

Объект главного окна.

Объект главного окна отображает на экране саму программу; в нем находится меню, заголовок окна и панель инструментов. Объект главного окна отвечает за все, что происходит вокруг того места, где работает программа (где она рисует, выводит текст и т.д.). Рабочая зона программы называется ***клиентской областью*** окна; например, текст выводится в клиентской области текстового редактора. За работу с клиентской областью

отвечает объект вида.

Объект вида.

Объект вида предназначен для работы с ***клиентской областью*** — местом, где обычно отображаются в соответствующем формате данные программы (например, текст, при работе с текстовым редактором). На самом деле объект вида представляет собой окно, которое накладывается поверх клиентской области. Объект вида отображает данные, хранящиеся в объекте документа.

Объект документа.

В *объекте документа* хранятся данные программы. Почему эти данные нельзя хранить в объекте вида? Можно, особенно в однодокументном приложении. По стандарту предполагается, что данных довольно много, причем не все они отображаются в клиентской области. Концепция MS Visual C++ предлагает сохранение всех данных в объекте документа, а затем отображение объектом вида только тех данных, которые попадают в клиентскую область объекта вида. В соответствии со стандартом этой концепции для одного документа можно иметь несколько одновременно открытых видов.

Взаимосвязь этих четырех составляющих проекта MFC выглядит следующим образом

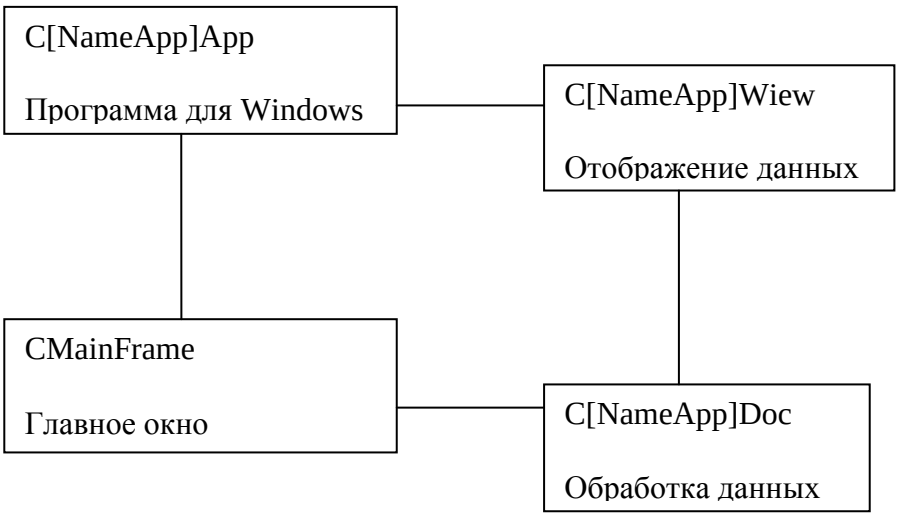

Рис. 1. Структура проекта MFC

7. Разместите объект out_string в объекте документа. Для этого, прежде всего необходимо объявить out_string в заголовочном файле документа StringOut_1Doc.h:

```
// StringOut_1Doc.h : interface of the CStringOut_1Doc class
(// StrOut1Doc.h : интерфейс класса CStrOut1Doc)

#pragma once

class CStrOut1Doc : public CDocument
{
protected: // создать только из сериализации
        CStrOut1Doc();
        DECLARE_DYNCREATE(CStrOut1Doc)

//**********************
        CString out_string;
//**********************

// Атрибуты
public:
.
.
.
```

8. Инициализируйте объект out_string в конструкторе класса документа, расположенном в файле StringOut_1Doc.cpp следующей программной строкой:

```
// StrOut1Doc.cpp : реализация класса CStrOut1Doc
//

#include "stdafx.h"
#include "StrOut1.h"

#include "StrOut1Doc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CStrOut1Doc

IMPLEMENT_DYNCREATE(CStrOut1Doc, CDocument)

BEGIN_MESSAGE_MAP(CStrOut1Doc, CDocument)
END_MESSAGE_MAP()


// создание/уничтожение CStrOut1Doc

CStrOut1Doc::CStrOut1Doc()
{
    // TODO: добавьте код для одноразового вызова конструктора
```

```
//*********************************************************
    out_string = "Ф.И.О. - Программная реализация средств дизайна";
//*********************************************************

}

CStrOut1Doc::~CStrOut1Doc()
{
.
.
.
```

9. Теперь необходимо обратится к документу из объекта вида. Для этого в классе вида нужно использовать фрагмент кода для получения указателя на объект документа с использованием метода GetDocument() класса вида. Указатель имеет имя (присвоил AppWizard) – pDoc. Фрагмент кода, необходимый для определения такого обращения:

```
// StrOut1View.cpp : реализация класса CStrOut1View
//

#include "stdafx.h"
#include "StrOut1.h"

#include "StrOut1Doc.h"
#include "StrOut1View.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CStrOut1View

IMPLEMENT_DYNCREATE(CStrOut1View, CView)

BEGIN_MESSAGE_MAP(CStrOut1View, CView)
    // Стандартные команды печати
    ON_COMMAND(ID_FILE_PRINT, &CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, &CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW,
    &CView::OnFilePrintPreview)
END_MESSAGE_MAP()

// создание/уничтожение CStrOut1View

CStrOut1View::CStrOut1View()
{
    // TODO: добавьте код создания
```

```
}

CStrOut1View::~CStrOut1View()
{
}

BOOL CStrOut1View::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: изменить класс Window или стили посредством изменения
    //  CREATESTRUCT cs

    return CView::PreCreateWindow(cs);
}

// рисование CStrOut1View

void CStrOut1View::OnDraw(CDC* pDC) //********** !!!! ????
{
    CStrOut1Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;
    // TODO: добавьте здесь код отрисовки для собственных данных
    //( TODO: add draw code for native data here)
    //************************************
        pDC->TextOut(0, 0, pDoc->out_string);
    //************************************

}
.
.
```

Т.о., обращение к out_string в объекте документа может иметь вариант реализации pDoc->out_string. На практике при таком маленьком объёме данных вряд ли целесообразно их размещение в объекте документа. Более оптимальным является их непосредственное размещение в объекте вида. Исключением является задача с необходимостью размещения информации на внешней памяти с последующим чтением.

10. Подготовительный этап сохранения и загрузки документа (внешняя память). В файле документа StringOut_1Doc.cpp имеется метод Serialize(). Методу Serialize() передается объект с именем ar, с ним можно работать точно так же, как и с потоком cout. В частности, для сохранения объекта out_string при записи данных на диск и для его последующего чтения при загрузке данных с диска в исходный код метода Serialize() следует добавить две строки программного кода. Найдите метод Serialize() и добавьте программный код:

```
void CStringOut_1Doc::Serialize(CArchive& ar)
{
        if (ar.lsStoring())
        {
        // TODO: добавте код для сохранения данных
        ar<< out_string;
        } else
        {
        // TODO: добавте код для загрузки данных
        ar<< out_string;
        }
}
```

В программе AppWizard есть несколько встроенных команд меню для работы с файлами: Save, Save as и Open. Система C/C++ берет на себя почти всю работу по сохранению файла на диске, но для задания последующего сохранения нашего объекта out_string необходимы вышеуказанные изменения в ней.

Задание 2

Напоминаем, что для вывода небольших информационных фрагментов при отсутствии необходимости их сохранения во внешней памяти, вывод можно организовать непосредственно через объект вида.

1. По методике задания 1 создайте проект StringOut_2 и разместите его той же директории, что и предыдущий проект.

2. Можно добавить предложенный ниже фрагмент кода в метод OnDraw() класса CStringOutView (в среде VC++6.0.) для отображения данных без использования объекта документа. Программа вызывает метод OnDraw(), когда ей требуется вывести что-либо в клиентской области программы (например, при запуске программы, при свертывании и восстановлении окна или при перемещении другого окна, закрывающего часть клиентской области).

```
Void CStringOutView(CDC* pDC)
{
    CString out_string = "Ф.И.О. - Программная реализация средств дизайна";
    CStringOut1Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: добавте код для отображения данных
    pDC->TextOut(0, 0, out_string);
}
```

VC++9.0. В этой среде возможность такого программного варианта разработчиками MS заблокирована.

В этой среде Вам необходимо будет ввести три, а не две командные строки:

- определение переменной класса CString в заголовочном файле объекта вида;
- инициализация определённой ранее переменной строковым литералом в конструкторе объекта вида;
- вывод изображения в методе OnDraw().

Примечание: Для редактирования метода в окне просмотра Visual C++ можно перейти на вкладку ClassView, раскрыть список методов класса CStringOutView и активизировать метод OnDraw(). Можно организовать вход на редактирование метода и через соответствующий файл объекта.

Задание 3

Отладка программ. Установка точек прерывания. Пошаговое выполнение программы. Считывание значений переменных.

1. По методике задания 1 создайте однодокументную (SDI) программу Debug_1 и сохраните его в директории Lesson_2.

2. Включите массив test в заголовочный файл документа:

```
// Debug_1Doc.h : interface of the CDebug_1Doc class
      .
      .
      .
// Attributes
public:
      int test[10];
      .
      .
      .
```

3. Числовые значения заносятся в массив в конструкторе документа:

```
// Debug_1Doc.cpp : implementation of the CDebug_1Doc class
.
.
.
CDebug_1Doc::CDebug_1Doc()
{
      test[0] = 1;
      test[1] = 2;
      test[2] = 3;
      test[3] = 4;
      test[4] = 5;
}
.
.
.
```

4. С помощью редактора меню вставьте новое меню Run между File и Edit, единственная команда этого меню – Average. Вариант последовательности действий: выбрать вкладку ResourceView проводника проекта; “войти в содержимое” файла ресурсов, раскрыть папку Menu и активизировать IDR_MAINFRAME двойным щелчком. То же самое можно проделать через обозреватель решений (Solution): в папке Файлы ресурсов активизировать файл Debug_1.rc и далее повторить действия, аналогичные первому варианту. После этого необходимо интуитивно разобраться в принципах работы MenuEditor (действия Click, DoubleClick, Drag&Drop стандартны для GUI MSW – интуитивно понятного для пользователя).

5. Используйте ClassWizard и создайте в программе обработчик для команды Average. VC++6.0. Вариант действий: открыть ClassWizard через контекстное меню команды Average или из меню окна; в поле ClassName выбрать объект вида; в поле Messages выбрать Command; активизировать элемент AddFunction. VC++2008. В контекстном меню команды Average командой Добавить обработчик событий (Add messages) запустить Мастер сообщений и в его окне диалога согласиться с предложенными опциями. Результат действий – сгенерированный мастером код:

```
// Debug_1View.cpp : implementation of the CDebug_1View class
.
.
.
void CDebug_1View::OnRunAverage()
{
        // TODO: Add your command handler code here

}
```

6. Добавьте в созданную заготовку обработки сообщения программный код для вычисления среднего арифметического:

```
void CDebug_1View::OnRunAverage()
{
   CDebug_1Doc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);

   float Sum;
   float Average;

   for(int index = 1; index < 5; index++)
   {
        Sum += pDoc->test[index];
   }

   Average = Sum / (float) 5.0;

// VC++6.0.

   // OutString.Format("Среднее арифметическое пяти чисел равно: %.3f",Average);

// VC++2008.

   OutString.Format(_T("Среднее арифметическое пяти чисел равно: %.3f"),Average);

   Invalidate();
}
```

7. Объявите объект OutString в заголовочном файле объекта вида (может быть логичнее это сделать до выполнения пункта 6):

```
// Debug_1View.h : interface of the CDebug_1View class
.
.
.
protected: // create from serialization only
        CDebug_1View();
        DECLARE_DYNCREATE(CDebug_1View)
        Cstring OutString;
      .
      .
      .
```

8. В конструкторе объекта вида задайте ввод в строку OutString подсказки, выводимой при запуске программы:

```
CDebug_1View::CDebug_1View()
{
        OutSring = "Выполните команду меню Run|Average"
}
```

9. В методе OnDraw() объекта вида определите вывод подсказки:

```
// рисование CDebuggingView

void CDebuggingView::OnDraw(CDC* pDC) //******** !!!! ????
{
        CDebuggingDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        if (!pDoc)
                return;
        // TODO: добавьте здесь код отрисовки для собственных
данных
        //*************************
        pDC->TextOut(0,0,OutString);
        //*************************
}
```

10. Проверьте выполнение программы и оцените результат. Что в нём не так?

11. Установка точки прерывания. Поместите точку прерывания в начале метода OnRunAverage(), в первой строке цикла for:

```
void CDebug_1View::OnRunAverage()
```

```
{
    CDebug_1Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    float Sum;
    float Average;

●   for(int index = 1; index < 5; index++)
    {
        Sum += pDoc->test[index];
    }

    Average = Sum / (float) 5.0;

// Помещение результата в строку OutString
    OutString.Format(_T("Среднее арифметическое пяти чисел равно: %.3f"),Average);

// Объявление вида недействительным для вывода строки на экран
    Invalidate();
}
```

VC++6.0. Команда Insert/Remove Breakpoint – кнопкой или F9. VC++2008. Insert/Remove Breakpoint – лучше просто щелчком кнопкой мыши на курсоре «служебного столбца» редактора кода.

Примечания: 1. В программе может быть установлено несколько точек прерывания. 2. Нельзя устанавливать Breakpoints на строках программы, декларирующих переменные. 3. После остановки программы на точке прерывания возможно пошаговое перемещение по её коду (F10 - VC++6.0.). 4. Точку прерывания можно установить перед запуском программы или во время остановки программы на другой точке.

12. Запустите программу Debug_1 в отладчике командой Build – Start Debug – Go (VC++6.0.). VC++2008. Начать отладку (Buld) или F5. В окне приложения подайте команду Run, изучите принципы работы отладчика при пошаговом выполнении кода метода OnRunAverage() от установленной точки прерывания. VC++6.0. Откройте окна средств отладки Variables и Watch (View – Debug Windows). В окне Watch может быть размещена любая переменная отлаживаемого кода (отслеживание её значения при перемещении по коду). Эти средства отладки (можно пользоваться и ToolTipText в программном коде) должны помочь обнаружению двух ошибок в коде метода OnRunAverage().

VC++6.0. Для определения значения выражения во время выполнения программы возможен его ввод в диалоговом окне QuickWatch (открывается одноименной командой меню Debug). Можно и просто перетащить имя переменной из кода в окно Watch.

Найдите соответствующие средства отладки в VC++2008.

13. Исправьте две найденные “неточности” в заданном программном коде и проверьте результат работы исправленной программы. Очевидно, что правильным результатом является вывод программой строки: Среднее арифметическое пяти чисел равно: 3.000.

14. Рекомендуем Вам сформировать конспект отчёта по результатам итогового выполнения задания 3 с целью формирования письменного ответа на экзаменационный вопрос (ГЭ): Отладка программ в MS VC++, примеры.

Задание 4

Ввод символов с клавиатуры, их форматирование и отображение в клиентской области.

1. Создайте однодокументное (SDI) приложение MFC с именем Keyboard и сохраните проект в директории Lesson_3.

2. Подготовьте в объекте документа буфер для хранения данных, вводимых с клавиатуры. Сохранённые в буфере данные будут затем отображаться в объекте вида. Буфер в концепции реализации данного задания фактически представляет собой объект класса MFC CString. Каждый вновь введённый символ добавляется к строке, хранящейся в данном объекте. Объявите эту строковую переменную под именем StrData в заголовочном файле документа:

```
// KeyboardDoc.h : interface of the CKeyboardDoc class
        .
        .
        .
class CKeyboardDoc : public CDocument
{
protected: // create from serialization only
        CKeyboardDoc();
        DECLARE_DYNCREATE(CKeyboardDoc)
        CString StrData;
        .
        .
        .
```

Проинициализируйте объект StrData пустой строкой в конструкторе объекта документа.

3. Для чтения нажатых клавиш необходимо добавить метод – обработчик сообщений Windows в класс вида CKeyboardView и связать его с сообщением Windows WM_CHAR (нажатие клавиши). VC++6.0. В окне классов (ClassView) открыть свойства CKeyboardView. В списке Messages найдите сообщение WM_CHAR и активизируйте его выбор двойным щелчком. В результате в списке появится метод OnChar(), двойной щелчок на активизированной строке метода позволит перейти к коду метода OnChar() в файле KeyboardView.cpp. VC++9.0. В Окне классов (ClassView) выделить класс CKeyboardView. Открыть Окно свойств (Properties) и выбрать его режим – Сообщения (Messages). Найдите сообщение WM_CHAR, активизируйте его строку, раскройте список и подайте команду Добавить (Add) сообщение OnChar() – после этого производится переход к коду функции OnChar() в файле KeyboardView.cpp.

4. Для обработки сообщения в методе OnChar() можно добавить свой

код (handler – вручную), можно использовать метод OnChar() базового класса CView. Введите «свой» код для сохранения введённых символов в объекте StrData:

```
// CKeyboardView message handlers
(// обработчики сообщений CKeyboardView)

void CKeyboardView::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
        // TODO: Add your message handler code here and/or call default
1 ⟶     CKeyboardDoc* pDoc = GetDocument();
2 ⟶     ASSERT_VALID(pDoc);
3 ⟶     pDoc->StrData += wchar_t(nChar); // VC++6 - char(nChar);

        CView::OnChar(nChar, nRepCnt, nFlags);
}
```

Объект StrData принадлежит документу и для работы с ним необходимо получение указателя pDoc на объект документа посредством метода GetDocument() класса вида (строка 1). Строка 2 определяет макрос проверки правильности ссылки полученного указателя на документ. Введённый символ передаётся методу OnChar() в параметре nChar, строка 3 задаёт добавление к строке StrData введённого символа. Выполнением пунктов 2 – 4 определены ввод символов с клавиатуры и их сохранение в буфере объекта документа.

5. Для перерисовки обновлённой строки вызовите метод Invalidate() объекта вида, вставив в программный текст пункта 4 оператор вызова этого метода:

```
        .
        .
4 ⟶     Invalidate();
        .
        .
```

Метод Invalidate() задаёт перерисовку объекта вида методом OnDraw().

6. Перейдите к методу OnDraw() объекта вида и введите в тело этого элемента – функции класса необходимые для вывода текстовой строки программные коды:

```
// CKeyboardView drawing (// рисование CKeyboardView)

void CKeyboardView::OnDraw(CDC* pDC)
{
	CKeyboardDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	if (!pDoc)
		return;

	// TODO: add draw code for native data here
	// TODO: добавьте здесь код отрисовки для собственных данных

	pDC->TextOut(0, 0, pDoc->StrData);
}
```

VC++6.0. В списке Overrides найдите сообщение OnDraw и активизируйте его выбор двойным щелчком.

Аналогично предыдущим заданиям вывод строки в методе OnDraw() реализуется через переданный этому методу указатель pDC, ссылающийся на объект класса MFC (класс CDC). Конкретнее, вызывается метод TextOut() этого класса. Класс CDC предназначен для работы с контекстами устройств – системной памяти, выделяемой для выполнения графических операций. Объекты Класс CDC содержат десятки методов, предназначенных для рисования в контексте устройства.

7. VC++6.0. Протестируйте созданный проект (Build) и его работоспособность (Execute Program).

6. По аналогии с предыдущим заданием рекомендуем Вам сформировать конспект отчёта по материалам задания 4 и результатам его выполнения. Целью формирования такого конспекта является подготовка к письменному ответу на экзаменационный вопрос (ГЭ): MFC VC++ - работа с клавиатурой, примеры. Следует отметить, что конспект может быть дополнен материалами и итогами выполнения следующего задания (задание 5).

Задание 5

Ввод символов с клавиатуры, действия по позиционированию отображения символьного объекта в клиентской области.

1. По методике задания 1 создайте однодокументную (SDI) программу с именем KeyCenter и сохраните проект в директории Lesson_3.

2. Полностью повторите пункты 2 – 5 предыдущего задания (задание 4) для сохранения введённого символьного объекта в буфере документа и подготовки его прорисовки в клиентской области.

3. Программные код пункта 6 предыдущего задания необходимо модернизировать для вывода графического объекта посередине клиентской области. Для этого в теле метода OnDraw() объекта вида введите следующие программные строки:

```
// CKeyCenterView drawing

void CKeyCenterView::OnDraw(CDC* pDC)
{
        CKeyCenterDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        // TODO: add draw code for native data here

1 ➡     CRect rect;
2 ➡     GetWindowRect(&rect);

3 ➡     int x = rect.Width()/2;
4 ➡     int y = rect.Height()/2;

5 ➡     CSize size = pDC->GetTextExtent(pDoc->StrData);

6 ➡     x -= size.cx/2;
7 ➡     y -= size.cy/2;

8 ➡     pDC->TextOut(x, y, pDoc->StrData);
}
```

4. Отладьте проект, проверьте его работоспособность и попробуйте осмыслить проделанную работу. Операторы сформированного Вами кода имеют следующее назначение:

Строка 1. Создаётся объект класса CRect с именем rect, используемый для хранения размеров прямоугольника. В проекте KeyCenter этот прямоугольник – клиентская область. После выполнения оператора объект rect содержит размеры клиентской области.

Строка 2. Методу GetWindowRect() передаётся указатель на объект rect. Метод GetWindowRect() является функцией класса CWnd – базового класса окна. Класс MFC CView является «наследником» класса CWnd. Класс вида

проекта CKeyCenterView является производным от класса MFC CView.

Строки 3, 4. Переменным x и y присваиваются половинные значения ширины и высоты прямоугольника клиентской области соответственно. Для этого используются методы Width() и Height() класса CRect. Т.е., итогом выполнения операторов 3 и 4 является сохранение координат центра клиентской области в системе (x, y).

Строка 5. Используется метод GetTextExtent() класса CDC (класс контекста устройства). Текст, вводимый с клавиатуры в данном проекте, выводится именно в контексте устройства (как и в предыдущих заданиях). Методу GetTextExtent() в строке 5 передаётся текстовая строка (pDoc->StrData). Этот метод передаёт размеры строки объекту с именем size класса MFC CSize.

Строки 6, 7. Класс CSize содержит переменные cx и cy, содержащие размеры строки. Изменение размеров строки в зависимости от количества введённых символов с соответствующим изменением значений системы координат (x, y) и определяют строки 6 и 7.

Строка 5. Вывод текстового объекта по определённым ранее координатам методом TextOut().

5. В качестве самостоятельного задания попытайтесь ввести в программу обработку нажатия базовых служебных клавиш.

Например, для перевода строки (обработка клавиши Enter) необходимо сравнение введённого символа с символом перевода строки (“\r”). Необходимо, так же определение высоты отображаемой на экране строки и, в случае обнаружения необходимости перехода на новую строку, прибавление высоты строки к координате y.

Удаление символа из строкового объекта с последующей его перерисовкой в области вида может быть реализовано средствами класса CString.

6. По итогам освоения материалов данного задания и по аналогии с предыдущими заданиями рекомендуем Вам дополнить конспекты отчётов по материалам заданий 1 – 4.

Задание 6

Создание курсора в клиентской области, управление курсором.

1. По методике задания 1 создайте однодокументную (SDI) программу с именем Cursor и сохраните проект в новой директории Lesson _4.

2. Как и в предыдущем задании повторите пункты 2 – 5 задания 4 для сохранения введённого символьного объекта в буфере документа и подготовки к его прорисовке в клиентской области (повторение пройденного – одна из основ учения, рекомендуем Вам повторить эти действия «вручную – по памяти»).

3. Создайте в объекте вида логическую переменную CursorCreat (строка 2) для определения наличия созданного курсора и объект класса CPoint (строка 1) с именем CursorPosition (класс CPoint содержит две переменные - x и y, для хранения значений координат контейнера (контекста вывода) изображений):

```
// CursorView.h : interface of the CCursorView class
        .
        .
        .
class CCursorView : public CView
{
protected: // create from serialization only
        CCursorView();
        DECLARE_DYNCREATE(CCursorView)
1 ➡     CPoint CursorPosition;
2 ➡     boolean CursorCreat;
        .
        .
        .
```

4. В конструкторе вида присвойте переменной CursorCreat значение false. Для создания курсора и включения его в объект вида задайте следующие операторы в теле функции OnDraw() объекта вида:

```
void CCursorView::OnDraw(CDC* pDC)
{
  CCursorDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);

  if (!pDoc)
        return;
```

```
        // TODO: добавьте здесь код отрисовки для собственных данных
        //***********************************************

1 ⟶     if(!CursorCreat) {
2 ⟶         TEXTMETRIC textmetr;

3 ⟶         pDC->GetTextMetrics(&textmetr);

4 ⟶         CreateSolidCaret(textmetr.tmAveCharWidth/8, textmetr.tmHeight);
5 ⟶         CursorPosition.x = CursorPosition.y =0;

6 ⟶         SetCaretPos(CursorPosition);
7 ⟶         ShowCaret();
8 ⟶         CursorCreat = true;
          }

9 ⟶       pDC->TextOut(0, 0, pDoc->StrData);

10 ⟶      CSize size = pDC->GetTextExtent(pDoc->StrData);

11 ⟶      HideCaret();

12 ⟶      CursorPosition.x = size.cx;
13 ⟶      SetCaretPos(CursorPosition);

14 ⟶      ShowCaret();
      }
```

Операторы сформированного в этом пункте задания кода имеют следующее назначение:

Строка 1. Если курсор не создан, выполняется последующий блок операторов по его созданию.

Строка 2. Структуре типа TEXTMETRIC присваивается имя textmetr. Структура TEXTMETRIC состоит из 18-ти полей типа LONG (например, 0 // - tmHeigt; 7 // tm – tmWeight; 11 // tm – tmFirstChar). Объекты этой структуры класса вида предназначены для хранения установленных характеристик шрифта.

Строка 3. Для заполнения объекта textmetr структуры TEXTMETRIC вызывается метод GetTextMetrics() класса CDC (класс для работы с контекстом устройства).

Строка 4. Для создания курсора вызывается метод CreateSolidCaret(). В данном проекте этот метод принимает от объекта textmetr два параметра. Первый задаёт ширину курсора в 8 раз меньшую, чем ширина символов, второй параметр задаёт высоту курсора равную высоте символов (методы CreateGrayCaret(), ShowCaret(), SetCaretPos(), HideCaret() так же предназначены для работы с курсором).

Строка 5. Задаются нулевые координаты вновь созданного курсора (см.

пункт 3 задания). Положение курсора в нашей программе хранится в объекте класса CPoint. Класс CPoint содержит две переменные x и y для хранения значений координат.

Строки 6, 7, 8. Позиция курсора задаётся методом SetCaretPos(), отображение курсора на экране задаётся методом ShowCaret(), создание и установка курсора переводит состояние флага CursorCreat в true. В результате курсор будет отображён на экране в заданной позиции (0, 0).

Строка 9. Как и предыдущих заданиях используется метод TextOut() класса CDC для прорисовки текстового объекта. Этими действиями задаётся ввод символов с клавиатуры. После этого нужно запрограммировать перемещение курсора по позиции каждого вновь введённого символа.

Строка 10. Оператор заполняет объект класса CSize с именем size информацией о координатах конца строки, получаемой вызовом метода GetTextExtent().

Строка 11. Перед выводом курсора в конце строки его изображение скрывается вызовом метода HideCaret().

Строка 12. Переменной x объекта CursorPosition присваивается координата конца текстовой строки на экране.

Строка 13. Метод SetCaretPos() определяет перемещение курсора в новую позицию (в данном проекте – конец строки).

Строка 14. Метод ShowCaret() задаёт отображение курсора на экране.

5. Отладьте проект, проверьте его работоспособность и попробуйте осмыслить проделанную работу.

6. Необходимо отметить, что перемещение курсора по тексту может задаваться и соответствующими клавишами. В качестве самостоятельного задания попытайтесь ввести в программу обработку нажатия базовых служебных клавиш в контексте отображения курсора на соответствующей позиции. Необходимо также отметить ещё один стандарт GUI Windows, подлежащий обработке Вашими программными кодами: при потере окном фокуса необходимо задание скрытия курсора, а при восстановлении фокуса окна – его отображение на прежней позиции.

7. По итогам освоения материалов данного задания и по аналогии с предыдущими заданиями возможно дополнение конспектов отчётов по материалам заданий 1 – 5. На основе освоения материалов задания подготовлены материалы для освоения принципов работы с мышью в MS VC++.

Задание 7

Работа с мышью, скрытие (отображение) курсора при потере (получении) фокуса окном программы.

1. По методике задания 1 создайте однодокументную (SDI) программу с именем Mouse и сохраните проект в директории Lesson_4.

2. Как и в предыдущем задании повторите пункты 2 – 5 задания 4 для сохранения введённого символьного объекта в буфере документа и подготовки к его прорисовке в клиентской области.

3. Повторите действия пунктов 3, 4 задания 6 по созданию курсора.

4. С помощью ClassWizard свяжите с сообщением Windows WM_KILLFOCUS новый метод – обработчик события (присвоенное ему ClassWizard имя – OnKillFocus()). VC++9.0. В Окне классов (ClassView) выделить класс CMouseView. Открыть Окно свойств (Properties) и выбрать его режим – Сообщения (Messages). Найдите сообщение WM_KILLFOCUS, активизируйте его строку, раскройте список и подайте команду Добавить (Add) сообщение OnKillFocus() – после этого производится переход к коду функции OnKillFocus() в файле MouseView.cpp. Добавьте в эту функцию команду скрытия курсора – вызов метода HideCaret():

```
void CMouseView::OnKillFocus(CWnd* pNewWnd)
{
        CView::OnKillFocus(pNewWnd);

        // TODO: Add your message handler code here
        HideCaret();
}
```

5. Аналогично добавьте метод OnSetFocus() для обработки сообщения WM_SETFOCUS и включите в него команду для отображения курсора при получении фокуса:

```
void CMouseView::OnSetFocus(CWnd* pOldWnd)
{
        CView::OnSetFocus(pOldWnd);
        // TODO: Add your message handler code here
        ShowCaret();
}
```

6. Объявите переменные для хранения координат курсора в заголовочном файле вида:

```
// MouseView.h : interface of the CMouseView class
        .
        .
        .
protected: // create from serialization only
        CMouseView();
        DECLARE_DYNCREATE(CMouseView)
        CPoint CursorPosition;
        boolean CursorCreat;
        int x, y;
        .
        .
        .
```

7. С помощью ClassWizard (по аналогии с предыдущими пунктами) свяжите с сообщением Windows WM_LBUTTONDOWN метод – обработчик события (присвоенное ему ClassWizard имя – OnButtonDown()) и добавьте в тело этой функции следующий программный код:

```
void CMouseView::OnLButtonDown(UINT nFlags, CPoint point)
{
        // TODO: Add your message handler code here and/or call default
1       x = point.x;
2       y = point.y;

3       CMouseDoc* pDoc = GetDocument();
4       ASSERT_VALID(pDoc);
5       pDoc->StrData.Empty();

6       Invalidate();

        CView::OnLButtonDown(nFlags, point);
}
```

Введённые программные строки обрабатывают событие – щелчок мышью и имеют следующее назначение:

Строки 1, 2. Методу OnLButtonDown() передаются два параметра. Параметр nFlags – информация о состоянии служебных клавиш (например, он может принимать значения MK_CONTROL, MK_SHIFT). Параметр Point, объект класса CPoint содержит текущие координаты указателя мыши. Т.е., строки 1 и 2 определяют сохранение текущих координат указателя мыши в переменных x и y.

Строки 3, 4. Их назначение рассмотрено в предыдущих заданиях.

Строка 5. Вызывается метод очистки строкового объекта Empty() класса CString. Т.е., после щелчка мышью, введённый ранее текст должен удаляться из объекта класса.

Строка 6. Вызывается метод Invalidate() объекта вида для объявления текущего состояния вида недействительным и подготовки последующего

отображения курсора в новом месте (метод OnDraw()).

8. В конструкторе вида присвойте переменной CursorCreat значение false. Для создания курсора в заданной позиции и его перемещения в конец строки после ввода символов определите в теле функции OnDraw() объекта вида набор операторов аналогичный предыдущему заданию, за исключением изменённых строк 9, 12 и добавленной между строкой 12 и последующими строками дополнительной строки:

```
void CCursorView::OnDraw(CDC* pDC)
{
        .
        .
        .
9 ➡     pDC->TextOut(x, y, pDoc->StrData);
        .
        .
        .
12 ➡    CursorPosition.x = x + size.cx;
13 ➡    CursorPosition.y = y;
        SetCaretPos(CursorPosition);

        ShowCaret();        }
```

Считаем, что операторы данного пункта в дополнительных комментариях не нуждаются, т.к., по логике своего функционирования подобны набору операторов соответствующей функции предыдущего задания.

9. Отладьте проект, проверьте его работоспособность и попробуйте осмыслить проделанную работу.

10. Рекомендуем Вам сформировать конспект отчёта по материалам задания 7, результатам его выполнения и других заданий. Цель формирования конспекта обозначена ранее. Экзаменационный вопрос (ГЭ): MFC VC++ - работа с мышью, примеры.

Задание 8

Диалоговые окна, создание класса диалогового окна, добавление элементов в диалоговое окно, связывание переменных класса с элементами диалоговых окон, связывание методов с элементами диалоговых окон.

1. Создайте однодокументный (SDI) проект на основе библиотеки MFC с именем Dialog1 и сохраните проект в директории Lesson_5.

2. В редакторе меню включите в меню File команду Show Dialog… (Диалог). Для этого активизируйте окно Ресурсы (например, вкладкой ResourceView), раскройте папку Menu и активизируйте ресурс IDR_MAINFRAME двойным щелчком. После создания команды меню в окне свойств задайте её идентификатор (ID) – ID_FILE_DIALOG. Затем включите в программный код объекта вида метод для обработки данной команды – OnFileDialog(), используя ClassWizard. Для его запуска через контекстное меню созданной команды (Диалог) подайте команду Добавить обработчик событий (Add method). В окне диалога нужно установить следующие опции: ClassName – CDialog1View, Messages – COMMAND – Double Click и т.д.

3. Для создания диалогового окна подайте команду Insert – Resource IDE VC++ (команду Добавить ресурс можно подать через контекстное меню окна ресурсов или через команду Проект меню окна среды разработки). Выбором из списка окна Insert Resource пункта Dialog и подачей команды New (Создать) запустите редактор диалоговых окон Dialogs. «Заготовке» диалогового окна по умолчанию присвоен идентификатор IDD_DIALOG1, как и меню, окно диалога отнесено к ресурсам Windows – программ.

4. Используя панель Controls (Панель элементов) и стандарт Drag&Drop, разместите элементы Button и EditBox (Edit Control) на «форме» проектируемого окна диалога. Измените свойство Caption объекта Button1 на что-либо более осмысленное, например, «Нажмите». Идентификаторы по умолчанию IDC_BUTTON1 и IDC_EDIT1 можно и не изменять. После указанных действий диалоговое окно представляет собой ресурс, хранящийся в файле ресурсов dialog1.rc. Фактически это текстовый файл, определения которого никак не связаны с объектами проекта. Пример внешнего вида интерфейса диалогового окна приложения показан на рисунке 2.

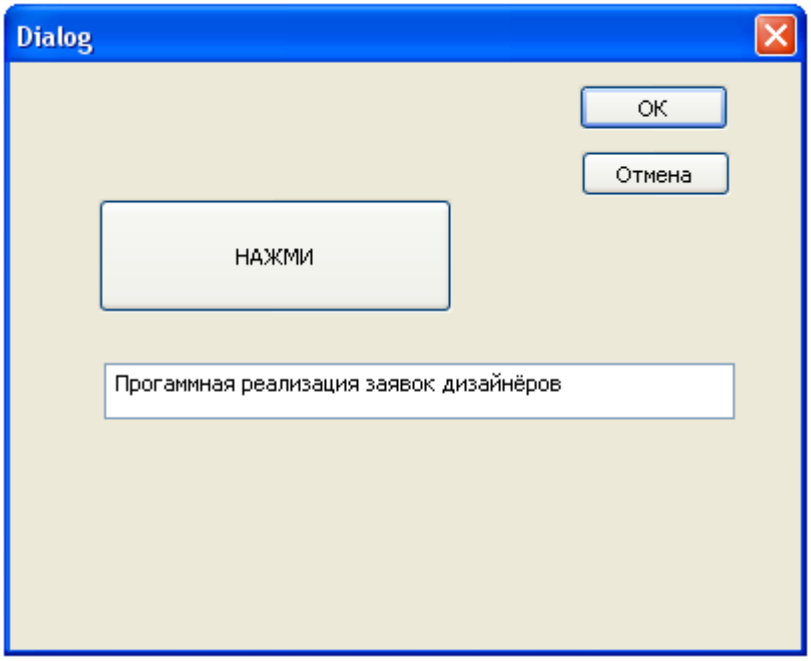

Рис. 2. Пример диалогового окна приложения

5. Для создания объекта, связанного с ресурсом (например, созданное диалоговое окно), с целью использования методов данного ресурса, необходимо создание нового класса диалогового окна. Для его создания запустите ClassWizard, подав команду Добавить класс (AddClass) или через контекстное меню «формы» окна диалога или через команду Проект меню окна среды разработки. В текстовых полях мастера классов MFC нужно задать следующие параметры: в поле Name (Имя класса) - имя создаваемого класса Dlg, Base Class - базовый класс CDialog, имена заголовочного файла и файла источника – по умолчанию, остальные параметры – аналогично. Подтвердите создание класса Dlg нажатием соответствующей кнопки.

6. Теперь необходимо создать обработчик сообщения – нажатие кнопки IDC_BUTTON1 опять же используя ClassWizard в контекстной взаимосвязи. Для этого могут быть использованы несколько вариантов действий.

Первый из них. Активизируйте ресурс IDD_DIALOG1 (если он у Вас на данный момент не активен), выделите на «форме» окна диалога объект IDC_BUTTON1 и откройте окно свойств. В окне Properties выберите вкладку Events (События элемента управления), из списка событий выберите сообщение BN_CLICKED, раскройте список справа от него и подайте команду Добавить (Add). В результате в программном коде реализации класса Dlg будет сформирована функция void Dlg::OnBnClickedButton1() для реализации метода, вызываемого программой при нажатии в диалоговом окне кнопки IDC_BUTTON1.

Второй вариант. В окне классов выделить класс Dlg, открыть окно свойств, выбрать его вкладку Events, раскрыть список идентификатора

IDC_BUTTON1, выбрать требуемое сообщение (Messag), конечные действия аналогичны предыдущему варианту.

Обратите внимание. Через последовательность этих действий и окно свойств можно переходить к требуемому фрагменту программного кода.

7. Для работы с текстом, вводимым в текстовое поле необходимо создание переменной соответствующего класса и её связывание с этим классом (аналогично и для других элементов диалоговых окон).

Чтобы создать такую переменную нужно подать команду Добавить переменную (Add variable) либо через контекстное меню, либо через меню команды Проект, при активизированной «форме» диалогового окна (ещё лучше выделить требуемый элемент, в нашем случае - IDC_EDIT1) или выделенном классе Dlg в окне классов. После реализации этих действий будет запущен ClassWizard, причём для его действия - Add Member Variable выбран класс Dlg. Если перед открытием ClassWizard не выбраны опции: Переменная элемента управления – True; Тип элемента управления – EDIT; ИД - IDC_EDIT1, установите их.

Задайте имя переменной m_text (m_ - префикс в имени переменной в венгерском стиле, обозначающий переменную класса). ***ВНИМАНИЕ***: в Category должно быть выбрано Value, а в Variable type – CString. После подачи команды подтверждения создания и связывания переменной класса можно проверить её наличие в окне классов (класс Dlg), а её свойства – в окне свойств.

8. Задайте программные коды тела функции OnBnClickedButton1() для вывода значения переменной m_text в текстовое поле:

```
// Dlg.cpp : implementation file (файл реализации)
        .
        .
        .
// Dlg message handlers (обработчики сообщений Dlg)


void Dlg::OnBnClickedButton1()
{
        // TODO: Add your control notification handler code here
        (// TODO: добавьте свой код обработчика уведомлений)
1 ➡     m_text = "Диалоговое окно|Edit Text|Объект вида";
2 ➡     UpdateData(false);
}
```

В строке 1 переменной m_text присваивается значение. В строке 2 вызывается метод UpdateData() для обновления текстового поля. Его вызов с параметром false заносит в текстовое поле значение переменной m_text, т.е. IDC_EDIT1 = m_text. . Его вызов с параметром true присваивает переменной m_text содержимое текстового поля значение, т.е. m_text = IDC_EDIT1.

Обмен информацией между переменной m_text и элементом IDC_EDIT1 реализуется в специальном методе, включённом ClassWizard в класс диалогового окна:

```
void Dlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(Dlg)
        DDX_Control(pDX, IDC_EDIT1, m_text);
        //}}AFX_DATA_MAP
}
```

9. Аналогично проделанному в пункте 6 свяжите метод OnBnClickedOk() с кнопкой OK (её идентификатор IDOK) и добавьте в метод строку кода 3:

```
void Dlg::OnBnClickedOk()
{
        // TODO: Add extra validation here
        (// TODO: добавьте свой код обработчика уведомлений)
3 ⟶     UpdateData(true);
        OnOK();
}
```

Оператор строки 3 задаёт присвоение переменной m_text содержимого текстового поля, т.е. вызов UpdateData(true) означает: m_text = IDC_EDIT1. Т.о., задано обновление содержимого m_text при нажатии кнопки OK. Вызов метода CDialog::OnOK() закрывает диалоговое окно и возвращает значение IDOK.

10. Для того чтобы класс вида мог взаимодействовать с членами созданного Вами класса Dlg (в первую очередь – отображение диалогового окна из объекта вида), необходимо включить в класс вида Dlg.h – заголовочный файл класса Dlg (необходимо «проинформировать» класс вида о наличии класса Dlg и, тем самым, предоставить доступ к его открытым членам). Добавьте в файл Dialog1View.cpp строку 4:

```
// Dialog1View.cpp : implementation of the CDialogView class
(//реализация класса CDialog1View)

#include "stdafx.h"
#include "Dialog.h"

#include "DialogDoc.h"
#include "DialogView.h"
4 ⟶ #include "Dlg.h"
        .
        .
```

После этого можно использовать класс Dlg в классе вида для отображения диалогового окна на экране и работы класса вида с членами

класса Dlg.

11. Для отображения диалогового окна на экране используйте созданный в пункте 2 задания метод OnFileDialog(). (если метод не создан – исправьте эту ошибку, перейти в программный код функции можно различными вариантам, в том числе, рассмотренными в пункте 6). Введите в тело функции, реализующей метод следующие программные строки:

```
void CDialog1View::OnFileDialog()
{
        // TODO: Add your command handler code here
        // TODO: добавьте свой код обработчика команд
5       Dlg dlg;
6       int result = dlg.DoModal();

7       if(result ==IDOK) {
8               CDialog1Doc* pDoc = GetDocument();
9               ASSERT_VALID(pDoc);

10              pDoc->StrData = dlg.m_text;
11              Invalidate();  }
}
```

Назначение операторов:

Строка 5. Создаётся объект класса Dlg с именем dlg.

Строка 6. Отображение диалогового окна на экране методом DoModal() класса CDialog. Класс CDialog в MFC является базовым для диалоговых окон, т.е. класс Dlg. наследует класс CDialog. Метод DoModal() вызывает модальное диалоговое окно и возвращает целое значение, в нашем случае сохраняемое в переменной result.

Строка 7. Проверка нажатия кнопки OK. Если нажата кнопка OK, задаётся считывание содержимого переменной m_text.

Строка 8. Содержимое переменной m_text хранится в документе – получение указателя на документ.

Строка 10. В объект StrData помещается текст из переменной m_text. (***ВНИМАНИЕ***: объект StrData Вами ещё не создан – в AutoListMember он отсутствует, см. пункт 12).

Строка 11. Вызов метода OnDraw() методом Invalidate() объекта вида – «перерисовка» объекта вида, т.е. вывод строки в клиентской области.

12. Создайте в документе объект StrData класса CString (строка 12) и инициализируйте его в конструкторе (строка 13):

```
// Dialog1Doc.h : interface of the CDialog1Doc class
        .
        .
        .
class CDialog1Doc : public CDocument
{
protected: // create from serialization only
        CDialogDoc();
        DECLARE_DYNCREATE(CDialog1Doc)

// Attributes
public:
12 ➡    CString StrData;
        .
        .
        .
// Dialog1Doc.cpp : implementation of the CDialog1Doc class
(// Dialog1Doc.cpp : реализация класса CDialog1Doc)
        .
        .
// CDialog1Doc construction/destruction
(// создание/уничтожение CDialog1Doc)

CDialog1Doc::CDialog1Doc()
{
        // TODO: add one-time construction code here
        (// TODO: добавьте код для одноразового вызова конструктора)

13 ➡    StrData = "";
}
        .
        .
```

13. Задайте отображение текста в клиентской области в методе OnDraw() (строка 14):

```
void CDialog1View::OnDraw(CDC* pDC)
{
   CDialog1Doc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
        // TODO: добавьте здесь код отрисовки для собственных данных
14 ➡  pDC->TextOut(0, 0, pDoc->StrData);
}
```

14. В случае удачного построения проекта протестируйте его работоспособность. Команда Файл – Диалог должна открыть диалоговое

окно. Нажатие на кнопку Нажми – вывод строки «Диалоговое окно|Edit Text|Объект вида» в текстовое поле. В текстовое поле можно ввести произвольный текст. Нажатие на кнопку OK – закрытие диалогового окна и вывод текста в клиентскую область.

15. Рекомендуем Вам сформировать конспект отчёта по материалам заданий 8, 9. Цель формирования конспекта - экзаменационный вопрос (ГЭ): MFC VC++ - создание класса диалогового окна, связывание методов и переменных с элементами диалоговых окон, примеры.

Задание 9

Диалоговое окно в качестве главного, использование набора элементов управления в диалоговом окне.

1. Создайте однодокументный (SDI) проект на основе библиотеки MFC с именем Dialog2 и сохраните проект в директории Lesson_ 5. При этом должна быть установлена опция Мастера приложений MFC (AppWizard) Тип приложения (Application type) – На базе диалоговых окон (Dialog based). В результате базовым классом для созданной программы становится класс CDialog. В отличие от предыдущего задания от Вас не требуется создание нового класса диалогового окна, т.к. AppWizard создал этот класс во время генерации программного кода.

2. Создаваемая Вами учебная программа будет имитировать выбор комплектации поставляемого программного обеспечения. Для создания интерфейса программы проведите «этап визуального проектирования». Для его реализации должен быть открыт «конструктор формы диалогового окна» (редактор диалоговых окон). Он будет открыт «по умолчанию» после выполнения пункта 1 задания. Если Вы случайно закрыли данную вкладку окна разработки, активизируйте ресурс IDC_DIALOG2_DIALOG.

На «этапе визуального проектирования» в первую очередь удалите элемент Static Text IDC_STATIC1, затем разместите на «форме» диалогового окна следующий набор элементов управления:

- 2 элемента Group Box. Идентификатор первого из них – IDC_DISTRIB, его свойство Подпись (Caption) – Установка, статическая граница – True. Идентификатор второго – IDC_COMPONENT, Caption - Компоненты, статическая граница – True.

ПРИМЕЧАНИЕ (ВНИМАНИЕ): Вы вовсе не обязаны использовать предложенные Вам в заданиях идентификаторы – можете использовать более осмысленные на Ваш вкус, но при этом необходимо учитывать риск путаницы с программными кодами при выполнении заданий.

- 4 элемента Radio Button. Их нужно разместить в «контейнере» - элементе Group Box – IDC_DISTRIB. Их ID и Caption следующие: IDC_RADIO_DEMO – “Демо”; IDC_RADIO_LEARN – “Учебная”; IDC_RADIO_CHOICE – “Выборочная”; IDC_RADIO_FULL – “Полная”.

ПРИМЕЧАНИЕ (ВНИМАНИЕ): после размещения этих элементов Вам целесообразно освоить действия по их форматированию – групповое выделение, Format (Формат).

6 элементов Check Box. Разместите их в элементе Group Box – IDC_COMPONENT. Вам предлагаются такие идентификаторы и подписи этих элементов: IDC_CHECK_DEMO_EXE – “Demo.exe”; IDC_CHECK_DEMO_MDB - “Demo.mdb”; IDC_CHECK_PROG_EXE – “Prog.exe”; IDC_CHECK_LEARRN_MDB – “Learn.mdb”; IDC_CHECK_PROG_MDB – “Prog.mdb”; IDC_CHECK_HELP –

“HELP.hlp”. ”.

ПРИМЕЧАНИЕ (ВНИМАНИЕ): после размещения этих элементов целесообразно использовать действия по их форматированию – групповое выделение, Format (Формат).

2 элемента Static Text . Их ID и Caption соответственно IDC_STATIC1, «Система» и IDC_STATIC2, «Цена».

1 элемент Combo Box . Его идентификатор – IDC_COMBO_W. Заполните его свойство Data (Данные) списком Windows 2000;Windows XP;Windows Vista;Windows 7.

1 элемент Edit Text . Его идентификатор – IDC_EDIT_PRICE.

Пример результатов «визуального проектирования» показан на рисунке 3.

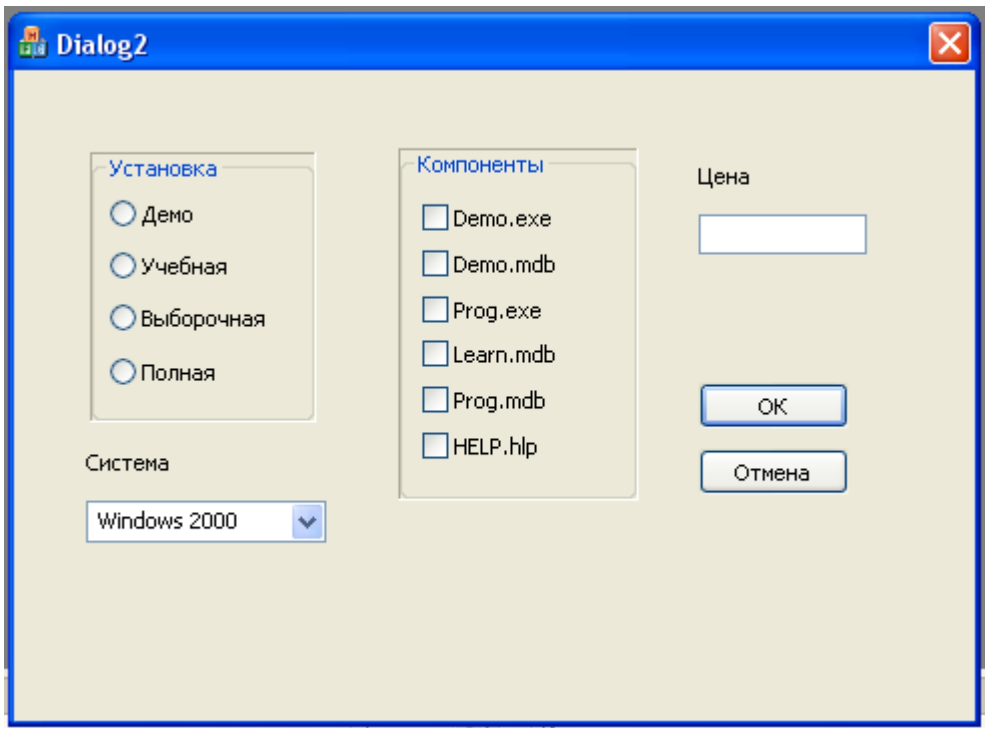

Рис. 3. Пример интерфейса учебного приложения

3. Теперь, по аналогии с заданием 8, необходимо добавить в проект объектные переменные для оптимизации взаимодействия с требуемыми элементами управления с программным кодом.

НАПОМИНАНИЕ: пожалуй, наиболее удобным способом инициализации добавления объектных переменных является использование команды Добавить переменную контекстного меню выбранного в Редакторе диалоговых окон элемента управления.

Для выполнения задания нужно создать переменные для всех шести элементов Check Box. В нашем случае требуется представление переменной

не всего элемента управления в целом, а только его отдельного свойства – значения. Поэтому выберите Категория (Category) – Value, соответственно, Тип переменной (Variable type) – BOOL. В примере выполнения задания использованы следующие имена переменных - членов класса элемента управления Check Box:

Объектные переменные элементов упрвления

Таблица 9.1

Идентификатор переменной	ИД элемента управления
m_CheckDemoExe	IDC_CHECK_DEMO_EXE
m_CheckDemoMdb	IDC_CHECK_DEMO_MDB
m_CheckProgExe	IDC_CHECK_PROG_EXE
m_CheckLearnMdb	IDC_CHECK_LEARRN_MDB
m_CheckProgMdb	IDC_CHECK_PROG_MDB
m_CheckHelp	IDC_CHECK_HELP

Далее создайте переменную - член m_ComboW класса CComboBox, представляющую комбинированное поле IDC_COMBO_W в целом. Т.е. должны быть установлены параметры: Category – Control, Variable type – CComboBox.

В завершение работы по определению объектных переменных задайте переменную-член класса CString для текстового поля IDC_EDIT_PRICE. Переменная элемента управления – True; Тип элемента управления – EDIT; ИД - IDC_EDIT_PRICE, имя переменной m_txtPrice, Category -Value, Variable type – CString

4. Чтобы при первом появлении комбинированного поля на экране в его текстовом поле отображалось строковое значение необходимо выбрать из заданного Вами списка одну из строк. Задайте это в методе инициализации данных диалогового окна:

```
BOOL CDialog2Dlg::OnInitDialog()
{
        CDialog::OnInitDialog();

        // TODO: добавьте дополнительную инициализацию
        (// TODO: Add extra initialization here)
        m_ComboW.SetCurSel(0);

        return TRUE;  // возврат значения TRUE, если фокус не передан
        элементу управления (// return TRUE  unless you set the focus to
        a control)
}
```

5. Теперь начнём подготовку к программной обработке данных программы. Для этого добавьте в проект свой собственный класс командой любого из меню Добавить – Класс (Add – Class), тип класса – C++ класс. В примере выполнения задания задано имя созданного класса – Price.

Возможно, особой необходимости в использовании такого класса в данной программе и нет – можно обойтись программными кодами, внедрёнными в классы, созданные мастером приложений. Но, во-первых, ориентация на классы – основной стиль C++ и только он и поддерживается AppWizard MFC. Во-вторых, мы создаём собственный класс в приложении MFC в учебных целях, хотя бы для того, чтобы вспомнить, что это такое и как с этим работать.

Класс Price создан – вводим в его файлы программные коды:

// Price.h: заголовочный файл

```
#pragma once

class Price
{
private:
	float PriceList[6];
	float WinList[4];
	bool Check1,Check2,Check3,
		Check4,Check5,Check6;
	int Check7;
public:
	float OutFullPrice;
	Price();
	~Price(void);
	float FullPrice(bool,bool,bool,bool,bool,bool,int);
};
```

Закрытые члены-данные класса предназначены:

- элементы PriceList[6] – для размещения цен шести составляющих программного обеспечения;
- элементы WinList[4] – для размещения ценовых коэффициентов под операционную систему;
- Check1 – Check7 – входные параметры функции-члена класса;

Открытый член-метод класса float FullPrice предназначен для расчета цены ПО в зависимости от выбора его составляющих.

// Price.cpp: файл реализации

```
#include "StdAfx.h"
#include "Price.h"

Price::Price()
{
	OutFullPrice = 0;
	PriceList[0]=499.00; PriceList[1]=300.50;
	PriceList[2]=2500.00; PriceList[3]=200.99;
	PriceList[4]=1850.50; PriceList[5]=1530.00;
	WinList[0]=1.4; WinList[1]=1.2;
```

```
        WinList[2]=1.5; WinList[3]=1.0;
}

Price::~Price(void)
{
}
float Price::FullPrice(bool Check1,bool Check2,bool Check3,
                       bool Check4,bool Check5,bool Check6,int Check7)
{
        if (Check1) OutFullPrice += PriceList[0];
        if (Check2) OutFullPrice += PriceList[1];
        if (Check3) OutFullPrice += PriceList[2];
        if (Check4) OutFullPrice += PriceList[3];
        if (Check5) OutFullPrice += PriceList[4];
        if (Check6) OutFullPrice += PriceList[5];

        switch (Check7) {
                case 0:         OutFullPrice *= WinList[0]; break;
                case 1: OutFullPrice *= WinList[1]; break;
                case 2: OutFullPrice *= WinList[2]; break;
                case 3: OutFullPrice *= WinList[3]; break;
        }
        return OutFullPrice;
}
```

6. Аналогично проделанному Вами при выполнении п. 6 Задания 7, создайте обработчики сообщений BN_CLICKED всех элементов управления Radio Button и Check Box. Добавьте также обработчик сообщения CBN_SELCHANGE элемента Combo Box.

7. В тела функций – обработчиков сообщения BN_CLICKED элементов управления Radio Button введите следующие программные коды:

```
// Dialog2Dlg.cpp : файл реализации
//

#include "stdafx.h"
#include "Dialog2.h"
#include "Dialog2Dlg.h"
1 ➡ #include "Price.h"
.
.
.
void CDialog2Dlg::OnBnClickedRadioDemo()
{
    // TODO: добавьте свой код обработчика уведомлений
2 ➡ m_CheckDemoExe = true;
    m_CheckDemoMdb = true;
    m_CheckProgExe = false;
    m_CheckLearnMdb = false;
    m_CheckProgMdb = false;
7 ➡ m_CheckHelp = false;

8 ➡ Price p;
9 ➡ m_txtPrice.Format(_T("%.2f"), p.FullPrice(m_CheckDemoExe,
                                    m_CheckDemoMdb,m_CheckProgExe,
            m_CheckLearnMdb,m_CheckProgMdb,m_CheckHelp,
                                    m_ComboW.GetCurSel()));
10 ➡ UpdateData(false);
}
```

Назначение Строки 1 должно быть очевидным – включение в файл реализации диалогового окна программного класса C++.

Строка 2 - Строка 7 – присвоение значений BOOL объектным переменным элементов Check Box.

Строка 8 – создание объекта класса Price с идентификатором p.

Строка 9 – вывод отформатированного в набор символов числового значения цены в текстовое поле через его объектную переменную. Числовое значение цены формируется в вызове открытой функции-члена объекта p класса класса Price. При этом функция принимает в качестве аргументов 6 значений BOOL объектных переменных элементов Check Box, а седьмым аргументом INT она принимает индекс выбранной строки из списка элемента Combo Box. Вызов метода GetCurSel() класса CComboBox производится через переменную m_ComboW, представляющую комбинированное поле IDC_COMBO_W.

Строка 10 – вызов метода UpdateData() для обновления состояний флажков и текстового поля. Напоминаем, что его вызов с параметром false задаёт состояние флажков и заносит в текстовое поле значения, соответствующие значениям объектных переменных этих элементов.

Три последующие функции обработки сообщений организованы по

аналогичной логической схеме.

```
void CDialog2Dlg::OnBnClickedRadioLearn()
{
    // TODO: добавьте свой код обработчика уведомлений
    m_CheckDemoExe = false;
    m_CheckDemoMdb = false;
    m_CheckProgExe = true;
    m_CheckLearnMdb = true;
    m_CheckProgMdb = false;
    m_CheckHelp = false;

    Price p;
    m_txtPrice.Format(_T("%.2f"), p.FullPrice(m_CheckDemoExe,
                                m_CheckDemoMdb,m_CheckProgExe,
                m_CheckLearnMdb,m_CheckProgMdb,m_CheckHelp,
                                m_ComboW.GetCurSel()));
    UpdateData(false);
}


void CDialog2Dlg::OnBnClickedRadioChoice()
{
    // TODO: добавьте свой код обработчика уведомлений
    m_CheckDemoExe = false;
    m_CheckDemoMdb = false; *
    m_CheckProgExe = false;
    m_CheckLearnMdb = false;
    m_CheckProgMdb = false;
    m_CheckHelp = false;

    Price p;
    m_txtPrice.Format(_T("%.2f"), p.FullPrice(m_CheckDemoExe,
                                m_CheckDemoMdb,m_CheckProgExe,
                m_CheckLearnMdb,m_CheckProgMdb,m_CheckHelp,
                                m_ComboW.GetCurSel()));
    UpdateData(false);
}

void CDialog2Dlg::OnBnClickedRadioFull()
{
    // TODO: добавьте свой код обработчика уведомлений
    m_CheckDemoExe = true;
    m_CheckDemoMdb = true;
    m_CheckProgExe = true;
    m_CheckLearnMdb = true;
    m_CheckProgMdb = true;
    m_CheckHelp = true;

    Price p;
    m_txtPrice.Format(_T("%.2f"), p.FullPrice(m_CheckDemoExe,
                                m_CheckDemoMdb,m_CheckProgExe,
```

```
                    m_CheckLearnMdb,m_CheckProgMdb,m_CheckHelp,
                                                m_ComboW.GetCurSel()));
    UpdateData(false);
```

8. В тела функций – обработчиков сообщения BN_CLICKED элементов управления Check Box введите следующие программные коды:

```
void CDialog2Dlg::OnBnClickedCheckDemoExe()
{
    // TODO: добавьте свой код обработчика уведомлений
    m_CheckDemoExe = true;
    Price p;
    m_txtPrice.Format(_T("%.2f"), p.FullPrice(m_CheckDemoExe,
                                m_CheckDemoMdb,m_CheckProgExe,
                    m_CheckLearnMdb,m_CheckProgMdb,m_CheckHelp,
                                m_ComboW.GetCurSel()));
    UpdateData(false);
}

void CDialog2Dlg::OnBnClickedCheckDemoMdb()
{
    // TODO: добавьте свой код обработчика уведомлений
    m_CheckDemoMdb = true;
    Price p;
    m_txtPrice.Format(_T("%.2f"), p.FullPrice(m_CheckDemoExe,
                                m_CheckDemoMdb,m_CheckProgExe,
                    m_CheckLearnMdb,m_CheckProgMdb,m_CheckHelp,
                                m_ComboW.GetCurSel()));
    UpdateData(false);
}

void CDialog2Dlg::OnBnClickedCheckProgExe()
{
    // TODO: добавьте свой код обработчика уведомлений
    m_CheckProgExe = true;
    Price p;
    m_txtPrice.Format(_T("%.2f"), p.FullPrice(m_CheckDemoExe,
                                m_CheckDemoMdb,m_CheckProgExe,
                    m_CheckLearnMdb,m_CheckProgMdb,m_CheckHelp,
                                m_ComboW.GetCurSel()));
    UpdateData(false);
}

void CDialog2Dlg::OnBnClickedCheckLearnMdb()
{
    // TODO: добавьте свой код обработчика уведомлений
    m_CheckLearnMdb = true;
    Price p;
    m_txtPrice.Format(_T("%.2f"), p.FullPrice(m_CheckDemoExe,
                                m_CheckDemoMdb,m_CheckProgExe,
                    m_CheckLearnMdb,m_CheckProgMdb,m_CheckHelp,
```

```
                                        m_ComboW.GetCurSel()));
    UpdateData(false);
}

void CDialog2Dlg::OnBnClickedCheckProgMdb()
{
    // TODO: добавьте свой код обработчика уведомлений
    m_CheckProgMdb = true;
    Price p;
    m_txtPrice.Format(_T("%.2f"), p.FullPrice(m_CheckDemoExe,
                                        m_CheckDemoMdb,m_CheckProgExe,
                    m_CheckLearnMdb,m_CheckProgMdb,m_CheckHelp,
                                        m_ComboW.GetCurSel()));
    UpdateData(false);
}
void CDialog2Dlg::OnBnClickedCheckHelp()
{
    // TODO: добавьте свой код обработчика уведомлений
    m_CheckHelp = true;
    Price p;
    m_txtPrice.Format(_T("%.2f"), p.FullPrice(m_CheckDemoExe,
                                        m_CheckDemoMdb,m_CheckProgExe,
                    m_CheckLearnMdb,m_CheckProgMdb,m_CheckHelp,
                                        m_ComboW.GetCurSel()));
    UpdateData(false);
}
```

Назначение операторов аналогично назначению операторов, откомментированных ранее. С точки зрения программного дизайна предложенные Вам десять обработчиков сообщений выглядят не слишком эстетично. В этой связи возникает вопрос о формировании задания на самостоятельную работу – если Вы стремитесь получить оценку, отличную от «удоволетворительно», попробуйте оптимизировать программное решение примера данного задания применительно к этому пункту.

9. В тело функции – обработчика сообщения CBN_SELCHANGE элемента Combo Box введите следующий программный код:

```
void CDialog2Dlg::OnCbnSelchangeComboW()
{
    // TODO: добавьте свой код обработчика уведомлений
    Price p;
    m_txtPrice.Format(_T("%.2f"), p.FullPrice(m_CheckDemoExe,
                                        m_CheckDemoMdb,m_CheckProgExe,
                    m_CheckLearnMdb,m_CheckProgMdb,m_CheckHelp,
                                        m_ComboW.GetCurSel()));
    UpdateData(false);
}
```

Назначение операторов этого этапа выполнения задания должно быть очевидным, равно как и логика работы программы.

10. Напоминаем Вам о рекомендации п.15 задания 8 - формирование конспекта отчёта по материалам заданий 8, 9. Цель формирования конспекта - экзаменационный вопрос (ГЭ): MFC VC++ - создание класса диалогового окна, связывание методов и переменных с элементами диалоговых окон, примеры.

Задание 10

Принципы создания элементарного графического редактора. Методы реализации стандартных графических шаблонов и операций. Сохранение графических изображений в файлах.

1. Создайте однодокументный (SDI) проект с именем Paint и сохраните его в директории Lesson_6.

2. Откройте редактор меню. Для этого активизируйте окно Ресурсы (например вкладкой Resource View), раскройте папку Menu и активизируйте ресурс IDR_MAINFRAME двойным щелчком.. Добавьте в меню окна программы команду Tools и снабдите её пятью вложенными командами со следующими надписями и идентификаторами: Рисованная кривая (ID_TOOLS_DRAW), Линия (ID_TOOLS_LINE), Прямоугольник (ID_TOOLS_RECTANGLE), Овал (ID_TOOLS_ELLIPSE), Заливка (ID_TOOLS_FILL).

3. Откройте редактор панели инструментов – делается это по аналогии с редактором меню через идентификатор в папке Toolbar Resource View. Добавьте на панель инструментов пять новых кнопок. При начале формирования рисунка на активизированном шаблоне кнопки редактор добавляет новый шаблон в панель. Перетаскивание за границы панели – удаление кнопки, перетаскивание внутри панели – группировка кнопок. Пример итога работы по формированию требуемой в данном задании панели инструментов показан на рисунке 4.

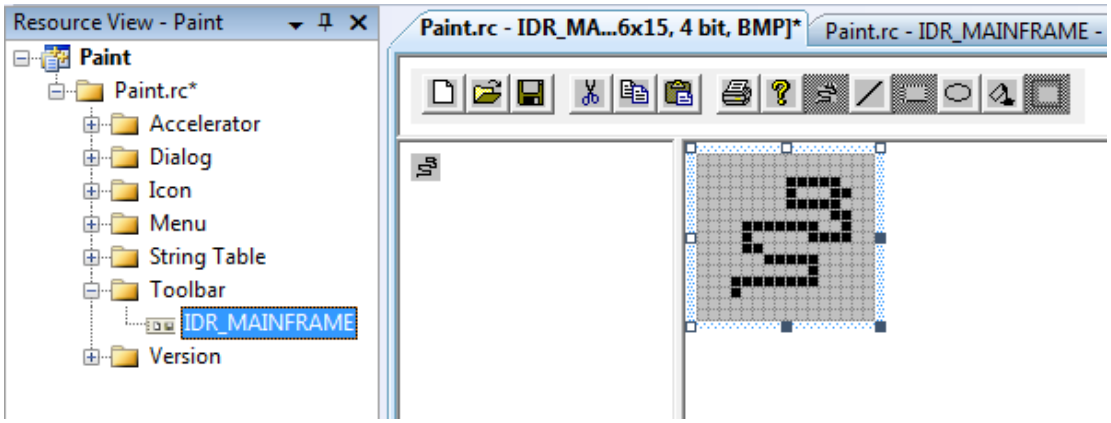

Рис. 4. Фрагмент работы с редактором Toolbar

ВНИМАНИЕ: желательно, чтобы порядок расположения заданных Вами кнопок соответствовал порядку расположения соответствующих команд меню, так будет просто удобнее работать в дальнейшем.

Завершите процесс создания панели инструментов программы связыванием кнопок панели с командами меню, которым они соответствуют.

Для этого кнопкам нужно присвоить соответствующие идентификаторы. Например, первой добавленной Вами кнопке необходимо присвоить идентификатор ID_TOOLS_DRAW, аналогия для четырёх последующих кнопок должна быть очевидна.

4. После выбора пользователем одного из пяти режимов рисования должен быть установлен соответствующий флаг – тип графического вывода (фигура). Далее должны быть считаны координаты указателя мыши: начальная точка – нажатие кнопки мыши, конечная точка – отпускание кнопки.

Объявите необходимые флаги, объекты для хранения координат и новый метод для сброса всех флагов (может активироваться только один режим рисования) в заголовочном файле класса вида:

```
class CPaintView : public CView
	{
	protected: // create from serialization only
		CPaintView();
		DECLARE_DYNCREATE(CPaintView)

			CPoint StartPoint; //объект начальной ккординаты
			CPoint DrawPoint; //объект координаты Drag&Drop
			CPoint EndPoint; //объект конечной координаты
			boolean FlagDraw; //флаг режима - рисованная кривая
			boolean FlagLine; //флаг режима - линия
			boolean FlagRectangle; //флаг режима - прямоугольник
			boolean FlagEllipse; //флаг режима - эллипс
			boolean FlagFill; //флаг режима - заполнение фигуры

			void AllFlagsFalse(); //объявление метода - сброс всех
флагов
		.
		.
		.
	}
```

5. Задайте действия метода AllFlagsFalse() – сброс всех флагов выбора режима в программном коде объекта вида:

```
	// CPaintView message handlers
	void CPaintView::AllFlagsFalse()
	{
		FlagDraw = false;
		FlagLine = false;
		FlagRectangle = false;
		FlagEllipse = false;
		FlagFill = false;
	}
```

Вызовите этот метод в конструкторе класса вида для отключения всех режимов рисования при запуске программы:

```
CPaintView::CPaintView()
{
	AllFlagsFalse();
}
```

6. Установите связи между флагами режимов и элементами интерфейса (команды меню, кнопки). Используя ClassWizard, свяжите метод с каждой командой меню (кнопкой панели инструментов). Последовательность действий должна быть известна по результатам выполнения предыдущих заданий.

VC++6.0. Редактор меню – активизация требуемой команды, её контекстное меню – вызов ClassWizard; выбор Class Name – CPaintView; двойной щелчок на пункте COMMAND в списке Message; подтверждение предлагаемого ClassWizard наименования; подтверждение создания метода (в том числе, возможным переходом в код функции метода).

VC++9.0. Редактор меню – контекстное меню созданной команды (например, Рисованная кривая) - команда Добавить обработчик событий (Add Event Handler). В списке Class list выбрать CPaintView, в списке Message type выбрать пункт COMMAND - подтвердить предлагаемое Event Handler Wizard наименования. Подтверждение создания метода (в том числе, возможным переходом в код функции метода) нажатием кнопки Add and Edit.

Результатом этих действий должно быть формирование пяти методов с двумя операторами: сброс флагов всех режимов; установление одного флага выбранного режима:

```
void CPaintView::OnToolsDraw()
{
	AllFlagsFalse();
	FlagDraw = true;
}

void CPaintView::OnToolsLine()
{
	AllFlagsFalse();
	FlagLine = true;
}

void CPaintView::OnToolsRectangle()
{
	AllFlagsFalse();
	FlagRectangle = true;
}
```

```
void CPaintView::OnToolsEllipse()
{
	AllFlagsFalse();
	FlagEllipse = true;
}

void CPaintView::OnToolsFill()
{
	AllFlagsFalse();
	FlagFill = true;
}
```

7. Для завершения формирования интерфейса программы необходима визуализация выбранного режима. Использование метода SetCheck() определит выделение команды активного режима в меню Tools и выделение кнопки этого режима на панели инструментов.

VC++6.0. При помощи ClassWizard свяжите с каждой командой метод обновления. Для этого свяжите обработчик команды с сообщением UPDATE_COMAND_UI.

VC++9.0. При помощи Event Handler Wizard свяжите с каждой командой метод обновления. Для этого свяжите обработчик команды с сообщением UPDATE_COMAND_UI. Результат проделанной работы должен выглядеть следующим образом:

```
void CPaintView::OnUpdateToolsDraw(CCmdUI* pCmdUI)
{
	pCmdUI->SetCheck(FlagDraw);
}

void CPaintView::OnUpdateToolsLine(CCmdUI* pCmdUI)
{
	pCmdUI->SetCheck(FlagLine);
}

void CPaintView::OnUpdateToolsRectangle(CCmdUI* pCmdUI)
{
	pCmdUI->SetCheck(FlagRectangle)
}

void CPaintView::OnUpdateToolsEllipse(CCmdUI* pCmdUI)
{
	pCmdUI->SetCheck(FlagEllipse);
}

void CPaintView::OnUpdateToolsFill(CCmdUI* pCmdUI)
{
	pCmdUI->SetCheck(FlagFill);
}
```

8. Обработка нажатия кнопки мыши в клиентской области – начала рисования, задание начальной точки фигуры.

VC++6.0. С помощью ClassWizard добавьте в программу метод OnButtonDown() (WM_LBUTTONDOWN) и сгенерируйте в нём код для сохранения координат начальной точки.

VC++9.0. Добавьте в программу метод OnButtonDown(). Для этого в окне классов (ClassView) выделите класс CPaintView и откройте окно свойств. В окне Properties выберите вкладку Messages, из списка методов выберите WM_LBUTTONDOWN и сгенерируйте в нём код для сохранения координат начальной точки:

```
void CPaintView::OnLButtonDown(UINT nFlags, CPoint point)
{
        StartPoint.x = point.x;
        StartPoint.y = point.y;

        CView::OnLButtonDown(nFlags, point);
}
```

9. Определите в программе рисование линий (установлен флаг - FlagLine). Рисование линии завершается при отпускании кнопки мыши, поэтому необходимо добавить в программу метод OnLButtonUp() и вставить в тело функции программный код для рисования линий:

```
void CPaintView::OnLButtonUp(UINT nFlags, CPoint point)
{
1       EndPoint.x = point.x;
2       EndPoint.y = point.y;

3       CClientDC* pDC = new CClientDC(this);

4       if(FlagLine) {
5               pDC->MoveTo(StartPoint.x, StartPoint.y);
6               pDC->LineTo(EndPoint.x, EndPoint.y);
        }
        CView::OnLButtonUp(nFlags, point);
}
```

Назначение операторов:

Строки 1, 2. Конечные координаты линии сохраняются в объекте EndPoint.

Строка 3. Начиная с этой строки, наблюдаются отличительные особенности вывода графики в клиентскую область по отношению ко всем предыдущим заданиям, в которых вызывалась функция Invalidate() – полное обновление изображения в методе OnDraw(). При этом удалялись бы все предыдущие изображения. Т.е., для отображения всего «нарисованного» пользователем во времени потребовалось бы периодическое сохранение всего изображения и обновление контекста устройства при каждом новом изменении вызовом метода OnDraw(). Запоминание изображения и его обновление в OnDraw() удобно реализовывать с использованием ***метафайлов***. Их использование является темой последующих заданий, поэтому на данном этапе ограничимся «рисованием» каждой новой графической фигуры по событию – отпускание кнопки мыши.

Этот пример демонстрирует вариант получения контекста устройства для объекта вида в произвольный момент, без использования метода OnDraw(). Для этого используется класс CClientDC() – наследник класса CDC, имеющий, в отличие от родителя, простой и удобный конструктор для создания контекста устройства.

В строке 3 конструктору класса CClientDC() передаётся указатель на текущий объект вида ключевым словом this. В результате получен указатель на объект контекста pDC, и вывод графики в это устройство можно осуществлять точно так же, как и в методе OnDraw().

Строка 4. Проверка состояния флага – рисование линии. Если он установлен (true), последующими двумя строками формируется изображение линии.

Строка 5. Перемещение к начальной точке линии методом MoveTo() класса CClientDC().

Строка 6. Изображение линии до конечной точки методом LineTo().

10. Рисование прямоугольников и эллипсов реализуется аналогично рисованию линии. Для этого нужно добавить в тело функции пункта 9 два блока операторов:

```
if(FlagRectangle) {
        pDC->SelectStockObject(NULL_BRUSH);
        pDC->Rectangle(StartPoint.x,StartPoint.y,EndPoint.x,EndPoint.y);
}
if(FlagEllipse) {
          pDC->SelectStockObject(NULL_BRUSH);
        pDC->Ellipse(StartPoint.x,StartPoint.y,EndPoint.x,EndPoint.y);
 }
```

Задание 11

Формирование графических операций. Использование метафайлов при работе с изображениями. Обновление изображений при работе с графикой. Сохранение графических изображений в файлах. Работа с графическими файлами.

1. Модифицируйте программный код пункта 10 предыдущего задания следующим образом:

```
        if(FlagRectangle) {
1 ⟶         pDC->SelectStockObject(NULL_BRUSH);
            pDC->Rectangle(StartPoint.x,StartPoint.y,EndPoint.x,EndPoint.y); }
        if(FlagEllipse) {
2 ⟶         pDC->SelectStockObject(NULL_BRUSH);
            pDC->Ellipse(StartPoint.x,StartPoint.y,EndPoint.x,EndPoint.y); }
        if(FlagFill) {
3 ⟶         pDC->SelectStockObject(BLACK_BRUSH);
4 ⟶         pDC->FloodFill(StartPoint.x,StartPoint.y,RGB(0,0,0)); }
        delete pDC;
    }
```

Назначение операторов:

Строки 1, 2. Выбор в текущем контексте устройства пустой кисти (NULL_BRUSH) методом SelectStockObject() класса CClientDC. Внутренняя область замкнутой фигуры по умолчанию заливается текущим цветом фона – команда отменяет заливку. В системе, кроме стандартных кистей, имеются стандартные перья (например, NULL_PEN).Собственные перья и кисти можно создать с помощью классов CPen, CBrush.

Строка 3. Выбор чёрной кисти (BLACK_BRUSH).

Строка 4. Вызов метода FloodFill(), закрашивающего фигуру текущей кистью. В данном варианте закрашивается область, внутри которой пользователь щёлкает мышью. Для задания цвета границы в методе FloodFill() используется макрос RGB.

2. Добавьте в объект вида обработчик перемещения мыши OnMouseMove(). Для этого в окне классов (ClassView) выделите класс CPaintView и откройте окно свойств. В окне Properties выберите вкладку Messages, из списка методов выберите WM_MOUSEMOVE. Затем внедрите в его тело нижеприведённый код:

```
void CPaintView::OnMouseMove(UINT nFlags, CPoint point)
{
1 ➔     CClientDC* pDC = new CClientDC(this);

2 ➔     if((nFlags && MK_LBUTTON) && FlagDraw) {
3 ➔         pDC->MoveTo(StartPoint.x, StartPoint.y);
4 ➔         pDC->LineTo(point.x, point.y);
5 ➔         StartPoint.x = point.x;
6 ➔         StartPoint.y = point.y; }

7 ➔     delete pDC;

        CView::OnMouseMove(nFlags, point);
}
```

Назначение программных строк:

Строка 1. Конструктору класса CClientDC() передаётся указатель на текущий объект вида ключевым словом this. В результате получен указатель на объект контекста устройства pDC.

Строка 2. Проверка включения соответствующего режима рисования и удерживания нажатой левой кнопки мыши при перетаскивании. Параметр nFlags метода OnMouseMove() сравнивается с константой Visual C++ MK_LBUTTON.

Строки 3-6. Рисование линии от начальной точки до текущей с назначением текущей точки начальной.

3. Включите в заголовочный файл документа указатель pMFDC на контекст устройства для метафайла:

```
// PaintDoc.h : interface of the CPaintDoc class
    .
    .
    .
// Implementation
public:
    virtual ~CPaintDoc();
➔   CMetaFileDC* pMFDC;
    .
    .
    .
```

Метафайлом называется размещаемый в памяти объект, поддерживающий собственный контекст устройства. Все операции, реализованные с контекстом устройства можно продублировать в метафайле. Для повторного вывода формируемого изображения (например, при обновлении) можно считать информацию из метафайла. Работа с метафайлами определена методами класса CMetaFileDC.

4. Создайте новый объект класса CMetaFileDC и реализуйте его в конструкторе документа вызовом метода Create():

```
CPaintDoc::CPaintDoc()
    {
        pMFDC = new CMetaFileDC();
        pMFDC->Create();
    }
```

5. Задайте дублирование формируемых пользователем графических операций в метафайле. Для этого необходимо при каждом вызове метода для контекста устройства вида вызывать аналогичный метод для контекста метафайла:

```
void CPaintView::OnLButtonUp(UINT nFlags, CPoint point)
{
    CPaintDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    EndPoint.x = point.x;
    EndPoint.y = point.y;

    CClientDC* pDC = new CClientDC(this);

    if(FlagLine) {
        pDC->MoveTo(StartPoint.x, StartPoint.y);
        pDC->LineTo(EndPoint.x, EndPoint.y);
        pDoc->pMFDC->MoveTo(StartPoint.x, StartPoint.y);
        pDoc->pMFDC->LineTo(EndPoint.x, EndPoint.y); }
    if(FlagRectangle) {
        pDC->SelectStockObject(NULL_BRUSH);
        pDC->Rectangle(StartPoint.x,StartPoint.y,EndPoint.x,EndPoint.y);
        pDoc->pMFDC->SelectStockObject(NULL_BRUSH);
        pDoc->pMFDC-
>Rectangle(StartPoint.x,StartPoint.y,EndPoint.x,EndPoint.y); }
    if(FlagEllipse) {
        pDC->SelectStockObject(NULL_BRUSH);
        pDC->Ellipse(StartPoint.x,StartPoint.y,EndPoint.x,EndPoint.y);
        pDoc->pMFDC->SelectStockObject(NULL_BRUSH);
        pDoc->pMFDC-
>Ellipse(StartPoint.x,StartPoint.y,EndPoint.x,EndPoint.y); }
    if(FlagFill) {
        pDC->SelectStockObject(BLACK_BRUSH);
        pDC->FloodFill(StartPoint.x,StartPoint.y,RGB(0,0,0));
        pDoc->pMFDC->SelectStockObject(BLACK_BRUSH);
        pDoc->pMFDC->FloodFill(StartPoint.x,StartPoint.y,RGB(0,0,0)); }
    delete pDC;
}
```

```
void CPaintView::OnMouseMove(UINT nFlags, CPoint point)
{
    CPaintDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    CClientDC* pDC = new CClientDC(this);

    if((nFlags && MK_LBUTTON) && FlagDraw) {
        pDC->MoveTo(StartPoint.x, StartPoint.y);
        pDC->LineTo(point.x, point.y);
        pDoc->pMFDC->MoveTo(StartPoint.x, StartPoint.y);
        pDoc->pMFDC->LineTo(point.x, point.y);
        StartPoint.x = point.x;
        StartPoint.y = point.y; }

    delete pDC;
}
```

6. Определите обновление изображения в методе OnDraw() воспроизведением метафайла:

```
void CPaintView::OnDraw(CDC* pDC)
{
    CPaintDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

1   HMETAFILE MetaFileHandle = pDoc->pMFDC->Close();
2   pDC->PlayMetaFile(MetaFileHandle);

3   CMetaFileDC* ReplacementMetaFile = new CMetaFileDC();
4   ReplacementMetaFile->Create();
5   ReplacementMetaFile->PlayMetaFile(MetaFileHandle);

6   DeleteMetaFile(MetaFileHandle);
7   delete pDoc->pMFDC;
8   pDoc->pMFDC = ReplacementMetaFile;
}
```

Назначение командных строк:

Строки 1, 2. Закрытие метафайла, получение его логического номера.

Строки 3, 4, 5. Создание нового метафайла и занесение в него содержимого предыдущего метафайла по его логическому номеру.

Строки 6, 7, 8. Удаление предыдущего метафайла, создание нового метафайла для последующего обновления

7. VC++6.0. Используя ClassWizard, свяжите обработчики с командами меню File – New, Save, Open. Генерируемые ими события необходимы для организации создания нового документа программы, сохранения изображения в файле и загрузки метафайлов.

VC++9.0. Свяжите обработчики с командами меню File – New, Save, Open. Для этого через контекстное меню команды (File – New) подайте команду - добавить обработчик событий (Add Event Handler). В списке Class list выберите класс CPaintView, в списке Message type выберите: пункт COMMAND: подтверждение предлагаемого Event Handler Wizard наименования; подтверждение создания метода (в том числе, возможным переходом в код функции метода) нажатием кнопки Add and Edit. Результатом должно быть формирование трех методов.

8. Для команды сохранения (не забудьте соответствующую кнопку – её идентификатор) код метода может быть сформирован следующим образом:

```
void CPaintView::OnFileSave()
{
    CPaintDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    HMETAFILE MetaFileHandle = pDoc->pMFDC->Close();
    CopyMetaFile(MetaFileHandle,  _T("paint.wmf"));

    CMetaFileDC* ReplacementMetaFile = new CMetaFileDC();
    ReplacementMetaFile->Create();
    ReplacementMetaFile->PlayMetaFile(MetaFileHandle);

    DeleteMetaFile(MetaFileHandle);
    delete pDoc->pMFDC;
    pDoc->pMFDC = ReplacementMetaFile;
}
```

9. Загрузка графического фала – метод OnFileOpen(). Сгенерируйте в этом методе следующую последовательность операторов:

```
void CPaintView::OnFileOpen()
{
    CPaintDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    HMETAFILE MetaFileHandle = pDoc->pMFDC->Close();
    CopyMetaFile(MetaFileHandle, _T("paint.wmf"));

    CMetaFileDC* ReplacementMetaFile = new CMetaFileDC();
```

```
    ReplacementMetaFile->Create();
    ReplacementMetaFile->PlayMetaFile(MetaFileHandle);

    DeleteMetaFile(MetaFileHandle);
    delete pDoc->pMFDC; //**********
    pDoc->pMFDC = ReplacementMetaFile;

    Invalidate();
 }
```

10. Для создания нового документа сгенерируйте нижеприведённый код, определяющий (используется метод Invalidate()) создание нового изображения в «очищенном» окне и в незаполненном метафайле:

```
void CPaintView::OnFileNew()
{
    CPaintDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    CMetaFileDC* ReplacementMetaFile = new CMetaFileDC();
    ReplacementMetaFile->Create();

    delete pDoc->pMFDC;
    pDoc->pMFDC = ReplacementMetaFile;

    Invalidate();

}
```

Задание 12

Методика использования встроенных в программу VC++ файловых средств называется сериализацией данных. Т.е., сериализацией можно назвать процессы записи/чтения объекта с использованием постоянного носителя информации. В большинстве программ, созданных на VC++, вся работа с данными производится в документах. Поэтому типичный вариант использования сериализации ориентирован на работу с объектом документа. Например, включение в метод Serialise() объекта документа кода для сериализации данных автоматически определяет работу файловых команд в меню File.

1. Скопируйте из директории Lesson_3 директорию с файлами проекта Keyboard в новую папку Lesson_7.

Напоминаем принцип работы программы проекта Keyboard. Программа получает от пользователя символы с клавиатуры, сохраняет их в объекте StrData объекта документа и отображает их в клиентской области с использованием метода OnDraw() объекта вида.

2. Задайте сохранение данных программы (объекта StrData) на диск и загрузку их с диска. Во встроенном методе Serialise() класса документа (файл KeyboardDoc.cpp) добавьте следующие командные строки:

```
void CKeyboardDoc::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
1 ⟶             ar << StrData;
        }
        else
        {
2 ⟶             ar >> StrData;
        }
}
```

В строках 1 и 2 методу Serialise() передаётся ссылка на адрес объекта ar класса CArchive. Принцип работы с объектом ar аналогичен работе с потоками cout и cin (см. задание 1). Строками 1 и 2 вызывается метод IsStoring() объекта ar для записи на диск и загрузки с диска соответственно. Сериализация строки StrData Вами подготовлена – можно продолжить организацию сохранения и считывания объекта во/с внешнюю(ей) память(и).

3. Сформируйте вывод диалогового окна Windows с предложением о сохранении изменений. Для этого необходимо вызвать в методе OnChar() метод объекта документа SetModifiedFlag():

```
void CKeyboardView::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
        CKeyboardDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        pDoc->StrData += wchar_t(nChar);
        Invalidate();
        pDoc->SetModifiedFlag();

        CView::OnChar(nChar, nRepCnt, nFlags);
}
```

4. Поэкспериментируйте с записью объекта в файл и загрузкой объекта из файла, имя файла задаётся произвольно, например string.dat.

5. Определите создание нового документа командой File-New. При подаче этой команды необходимо, прежде всего, удалить старое содержимое объекта StrData. Сформируйте это в методе OnNewDocument() объекта документа:

```
BOOL CKeyboardDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;
        StrData = "";
        return TRUE;
}
```

***ВНИМАНИЕ*:** начиная с пункта 6 задания 12, вы начинаете изучать приёмы сохранения произвольных объектов. Соответственно и проект очень желательно создать новый.

6. Изучение сериализации нестандартных объектов начните с создания проекта SDI под именем serial в папке текущего задания. В этом проекте необходимо создать собственный класс CText (рисунок 5).

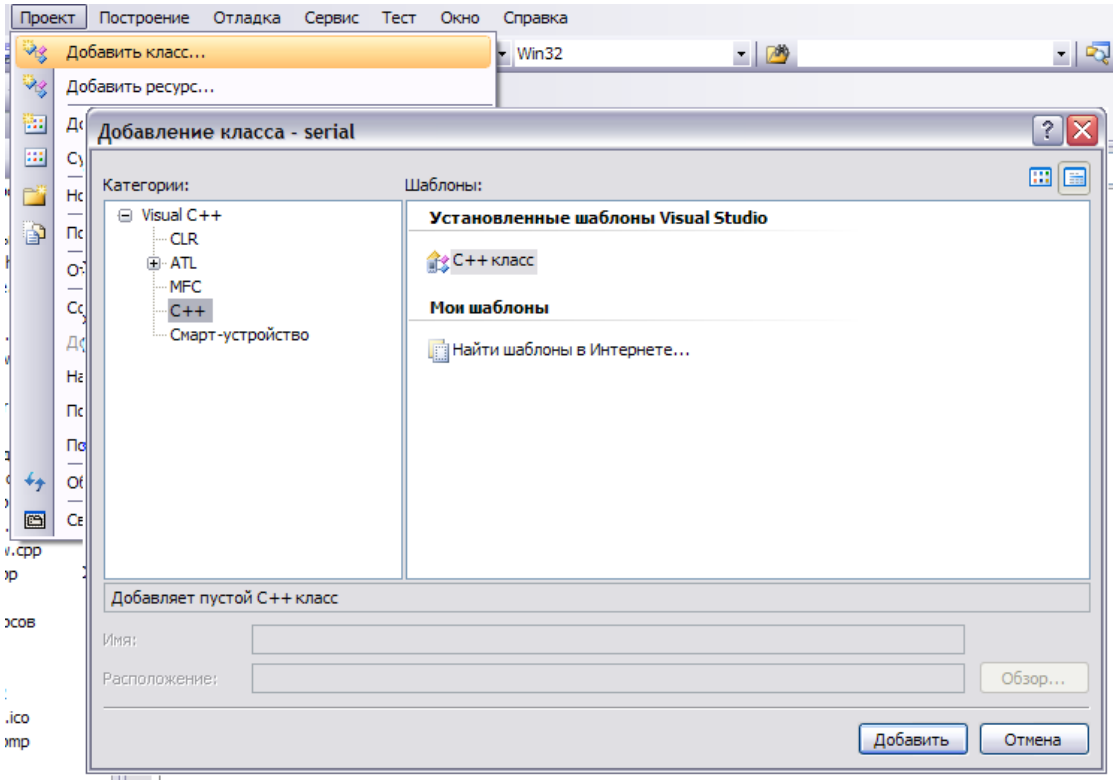

Рис. 5. Добавление класса в проект

В заголовочном файле этого класса CText.h определите его методы как встроенные (inline):

```
class CText {
private:
        CString strData;
public:
        CText(){strData = CString("");}
        void AddText(CString text){strData += text;}
        void DrawText(CDC* pDC){pDC->TextOut(0,0,strData);}
        void ClearText(){strData = "";}
};
```

7. Включите в заголовочный файл объекта документа ссылку на новый файл и создайте объект класса CText с именем TextObject:

```
// serialDoc.h : interface of the CSerialDoc class
//
1 ➞ include "CText.h"
    .
    .
    .
// Attributes
public:
2 ➞     CText TextObject;
    .
```

.
.

8. Аналогично предыдущему проекту определите сохранение вводимых с клавиатуры символов в методе OnChar(). Для этого в метод введите командную строку, которая задаёт ввод инициализированных символов во внутренний объект TextObject. При этом используется метод AddText(), унаследованный данным объектом от класса – родителя CText:

```
void CSerialView::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
        CSerialDoc* PDoc = GetDocument();
        ASSERT_VALID(pDoc);

        pDoc->TextObject.AddText(CString(wchar_t(nChar))); //***!!!***
        Invalidate();
        CView::OnChar(nChar, nRepCnt, nFlags);
}
```

9. По аналогии с предыдущим проектом содержимое TextObject выводится методом DrawText() в методе OnDraw() объекта вида:

```
void CSerialView::OnDraw(CDC* pDC)
{
        CSerialDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        pDoc->TextObject.DrawText(pDC);
}
```

Сгенерированная Вами программа должна выводить текстовый объект аналогично программам проектов keyboard. Но, делает она это через использование методов собственного, нового класса.

10. Попробуйте воспользоваться уже освоенным в данном задании набором операторов для выполнения сериализации – сохранения и считывания текстового объекта (см. п.2 задания12):

```
void CKeyboardDoc::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
1               ar << StrData;
        }
        else
        {
2               ar >> StrData;
        }
}
```

При попытке компиляции (Build) такого кода средой MS VC++ должно

быть сгенерировано сообщение об ошибке – для объекта StrData использование операторов передачи в поток недопустимо. Для выхода из этой ситуации необходимо определить способ выполнения сериализации для класса CText. При этом необходимо запрограммировать процесс сериализации в классе CText для обеспечения возможности сериализации объектов этого класса.

11. Для устранения этой, вовсе не очевидной для Вас ошибки, внесите следующие операторы в определение класса:

```
1 ➡  class CText : public CObject {
     private:
           CString StrData;
2 ➡        DECLARE_SERIAL(CText);
     public:
           CText(){StrData = CString("");}
           void AddText(CString text){StrData += text;}
           void DrawText(CDC* pDC){pDC->TextOut(0,0,StrData);}
           void ClearText(){StrData = "";}
3 ➡        void Serialize(CArchive& archive);
     };
```

Строка 1. Класс CText объявляется производным (наследование) от класса MFC CObject. Класс CObject – одна из основ в иерархии классов MFC.

Строка 2. Включение в определение класса макроса VC++, объявляющего методы необходимые при сериализации.

Строка 3. Переопределение метода Serialize() базового класса CObject (наследование и переопределение).

12. Для реализации новой версии переопределённого метода Serialize() необходимо в файле реализации CText.cpp определить метод Serialize():

```
     #include "stdafx.h"
     #include "serialDoc.h"

     void CText::Serialize(CArchive& archive)
     {
1 ➡        CObject::Serialize(archive);
           if(archive.IsStoring()){
                 archive << StrData;
           }
           else{
                 archive >> StrData;
           }
     }
2 ➡  IMPLEMENT_SERIAL(CText, CObject, 0)
```

Строка 1. Вызов метода Serialize() базового класса CObject.

Строка 2. Добавление макроса, содержащего дополнительные методы сериализации MS VC++.

13. Теперь для объектов класса CText можно выполнять сериализацию. Для выполнения сериализации сформированного Вами объекта NextObject, вызовите его метод Serialize() внутри метода Serialize() объекта документа:

```
void CSerialDoc::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
		// TODO: добавьте код сохранения
	}
	else
	{
		// TODO: добавьте код загрузки
	}
	//**************************
	TextObject.Serialize(ar);
	//**************************
	TextObject.Serialize(ar);
 }
```

14. Протестируйте разработанное приложение. Команды Сохранить и Открыть должны работать, команда Создать – нет. Как и при выполнении предыдущего проекта можно добавить коды, определяющие обязательность диалога при закрытии приложения – «сохранить…?».

15. Рекомендуем Вам сформировать конспект отчёта по материалам задания 12. Цель формирования конспекта - экзаменационный вопрос (ГЭ): MFC VC++ - сериализация объектов, примеры.

Задание 13

Работа с файлами. Выполнение стандартных файловых операций в VC++.

1. Создайте с помощью AppWizard проект на базе диалогового окна с именем Files и сохраните его в директории Lesson_8.

2. Разместите в главном диалоговом окне элементы управления: два текстовых поля и кнопку (рисунок 6). Свяжите переменную m_text1 с верхним текстовым полем, а переменную m_text2 – с нижним (***ВНИМАНИЕ:*** Category – Value; Type - CString).

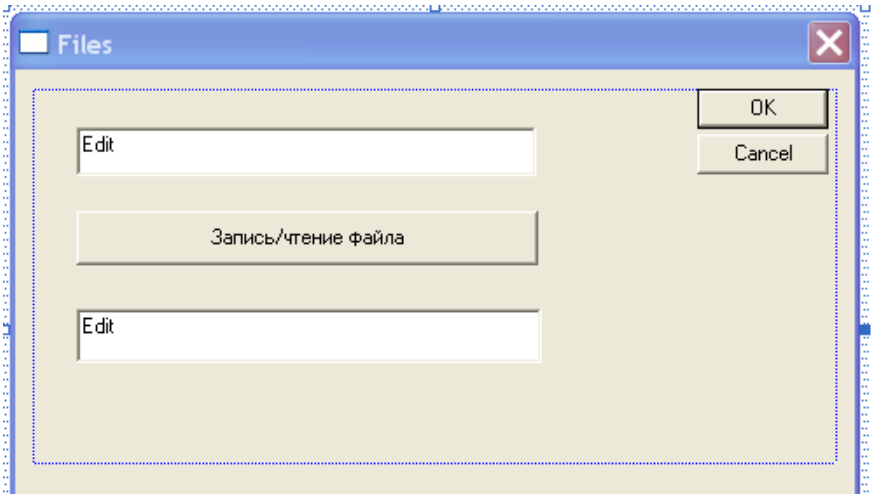

Рис. 6. Пример интерфейса приложения

3. Выделите память для хранения символьных (строковых) данных. Для этого в заголовочном файле объекта – главное окно диалога необходимо декларировать два массива:

```
// FilesDlg.h : header file
    .
    .
    .
// Implementation
protected:
        HICON m_hIcon;
1       char OutString[4][20];
2       char InString[20];
    .
    .
    .
```

ВНИМАНИЕ: предлагаемый вариант работы с тестовыми данными и их записи/чтения из файла имеет учебную ориентацию. В данном варианте

используется разбивка текста на записи. Т.е., каждая строка интерпретируется как запись – отдельный информационный объект, отдельно же заносимый в файл и считываемый из него. Конечной идеей такого подхода является изучение принципов работы с файловым указателем. Естественно, можно решить стандартную задачу данные – файл без разбивки файла на записи.

Строка 1. Декларируется массив для хранения четырёх строк, каждая из которых состоит из 20 символов.

В нашем случае «полной» строкой пусть будет – «Изучение работы с файлами», т.е., в массиве OutString[] схема хранения данной строки может быть представлена следующим образом:

OutString[]	“Изучение	”
OutString[]	“работы	”
OutString[]	“с	”
OutString[]	“файлами	”.

Строка 2. Декларируется двадцати-символьный массив InString[] для сохранения считанной из файла записи.

4. Заполнение «входного» массива реализуйте в методе OnInitDialog() (CFilesDlg):

```
BOOL CFilesDlg::OnInitDialog()
{
        CDialog::OnInitDialog();
1 ➡     strcpy(OutString[0], "Изучение ");
        strcpy(OutString[1], "работы ");
        strcpy(OutString[2], "с ");
4 ➡     strcpy(OutString[3], "файлами ");

5 ➡     m_text1 = CString(OutString[0]) + CString(OutString[1] +
                        CString(OutString[2]) + CString(OutString[3];
6 ➡     UpdateData(false);
.
.
.
}
```

Строки 1-4. Заполнение символьных массивов с использованием стандартной функции языка С strcpy().

Строки 5, 6. Заполнение верхнего текстового поля строкой «Изучение работы с файлами».

5. Используя ClassWizard, включите в программный код обработчик сообщения о нажатии кнопки OnButton1() и внедрите в него следующую последовательность операторов:

```
            void CFilesDlg::OnButton1()
            {
1 ➡         CFile OutFile(L"TextData.dat", CFile::modeCreate | CFile::modeWrite);
            for(int i = 0 ; i < 4; i++){
3 ➡               OutFile.Write(OutString[i], 20);}
4 ➡         OutFile.Close();

5 ➡         CFile InFile(L"TextData.dat", CFile::modeRead);
            for(i = 0 ; i < 4; i++) {
7 ➡               InFile.Seek(20 * i, CFile::begin);
8 ➡               int NumberChar = InFile.Read(InString, 20);
9 ➡               m_text2 += CString(InString); }
10 ➡        UpdateData(false);
11 ➡        InFile.Close();
            }
```

Строка 1. Создаётся объект OutFile класса VC++ CFile. Этот объект определяет создание файла TextData.dat и открывает его для записи. Во втором аргументе объекта OutFile определяются константы, передаваемые конструктору класса CFile. Префиксы CFile:: обозначают принадлежность констант классу CFile. Константы объединены оператором “|”, в итоге задаются режимы создания нового файла (CFile::modeCreate) и открытия файла только для записи (CFile::modeWrite). Список констант класса CFile достаточно обширен.

Строка 3. В теле оператора цикла в файл записываются четыре символьных строки посредством метода Write() класса CFile. При этом каждая строка трактуется как запись, разбиение файла на записи может оптимизировать доступ к нему.

Строка 4. Закрытие файла методом Close() класса CFile. Итогом последовательности операторов Строка 1 – Строка 4 должно являться создание на дисковом пространстве файла TextData.dat и его заполнение данными.

6. Рассмотрим последовательное чтение символьных строк из файла, определённое Вами в теле метода предыдущего задания.

Строка 5. Создаётся новый объект InFile класса VC++ CFile. Этот объект определяет открытие файла TextData.dat в режиме чтения. Константа CFile::modeRead – открытие файла только для чтения.

Строка 7. В цикле перебираются четыре символьные строки файла. Длина каждой строки фиксирована и равна 20 символам. Оператор строки задаёт установку файлового указателя в начало каждой последующей по

циклу строки методом Seek() класса CFile. Файловый указатель – фрагмент (индексируемый) файла, с этого фрагмента начинается последующая работа с файлом, в частности, методы Write() и Read() класса CFile выполняют свои действия с позиции файлового указателя. Метод Seek() позволяет переместить файловый указатель в любое место файла. Константа CFile::begin в вызове метода Seek() задаёт перемещение файлового указателя в новую позицию, положение которой задано первым аргументом метода относительно начала файла. Ещё возможны следующие точки отсчёта: CFile::current и CFile::end.

Строка 8. В массив InString методом Read() последовательно заносятся четыре записи. Метод Read() возвращает количество фактически считанных символов, присваиваемое в операторе строки 8 целочисленной переменной NumberChar. Еще раз отметим учебный характер разбиения на записи в нашем примере. Можно было бы считать данные одним оператором, использующим метод Read() со вторым аргументом, равным 80 (для нашего примера).

Строка 9. Четыре считанные из файла записи заносятся во второе текстовое поле.

Строка 10. Блокировка работы с данными

Строка 11. Закрытие файла.

7. Пример настоящего задания имеет несколько искусственный характер, в качестве самостоятельного задания возможно построение другого варианта выполнения задания, демонстрирующего работу с файлами вне объекта документа (без использования сериализации). По материалам данного задания может быть сформирован конспект ответа на вопрос: MS VC++ - операции по работе с файлами, примеры. Кроме того, может быть дополнен ответ на вопрос, связанный с использованием диалоговых окон.

Задание 14

В этом задании продолжим работу над созданным ранее графическим редактором (задания 10, 11 – директория Lesson_6 – проект Paint). Перед началом работы, видимо, целесообразно скопировать проект Paint в папку Lesson_9.

1. Создайте новый указатель мыши для объекта вида. Для этого используйте вариант создания указателя мыши при создании вида в методе PreCreateWindow(), имеющем место в классе вида:

```
BOOL CPaintView::PreCreateWindow(CREATESTRUCT& cs)
{
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs

        cs.lpszClass = AfxRegisterWndClass(CS_DBLCLKS,
                                AfxGetApp()->LoadStandardCursor(IDC_CROSS),
                                (HBRUSH)(COLOR_WINDOW+1),
                                AfxGetApp()->LoadIcon(IDR_MAINFRAME));
        return CView::PreCreateWindow(cs);
}
```

В результате изменяется содержимое одной из информационных структур окна, передаваемых параметром cs, новый указатель мыши задаётся в виде IDC_CROSS. При этом изменение курсора производится на уровне класса окна Windows.

2. Теперь зададим пользовательскую возможность слежения за контуром фигуры в процессе её создания. Естественно, должен быть выбран режим рисования и нажата левая кнопка мыши.

Для того, чтобы проводить линию от начальной точки (StartPoint) к текущей (DrawPoint) нужно стирать устаревшее изображение и прорисовывать вариант изображения на данный момент от начальной точки к текущему положению указателя мыши.

Для стирания линий (фигуры тоже линии) от точки начала изображения до точки, предшествующей следующему перемещению на пиксель, необходимо запоминать позицию этой предшествующей точки. Как раз для такого запоминания в программу Paint (ещё в п.4 задания 10) введён объект DrawPoint класса CPoint. В этом объекте предполагается сохранять текущие координаты указателя мыши при вызовах функции OnMouseMove().

При нажатии левой кнопки мыши занесём в объект DrawPoint текущие координаты курсора мыши:

```
void CPaintView::OnLButtonDown(UINT nFlags, CPoint point)
{
        // TODO: Add your message handler code here and/or call default
        StartPoint.x = point.x;
```

```
        StartPoint.y = point.y;

        DrawPoint.x = StartPoint.x; //*******
        DrawPoint.y = StartPoint.y; //*******

        CView::OnLButtonDown(nFlags, point);
    }
```

Код для имитации «растягивания фигур» нужно включить в тело функции - обработчика события OnMouseMove():

```
    void CPaintView::OnMouseMove(UINT nFlags, CPoint point)
    {
        // TODO: Add your message handler code here and/or call default
        CPaintDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
1       int oldMode;
        CClientDC* pDC = new CClientDC(this);

        if((nFlags && MK_LBUTTON) && FlagDraw) {
            pDC->MoveTo(StartPoint.x, StartPoint.y);
            pDC->LineTo(point.x, point.y);
            pDoc->pMFDC->MoveTo(StartPoint.x, StartPoint.y);
            pDoc->pMFDC->LineTo(point.x, point.y);
            StartPoint.x = point.x;
            StartPoint.y = point.y; }

        if((nFlags && MK_LBUTTON) && FlagLine){
2           oldMode = pDC->GetROP2();
3           pDC->SetROP2(R2_MERGEPENNOT);
4           pDC->MoveTo(StartPoint.x, StartPoint.y);
5           pDC->LineTo(DrawPoint.x, DrawPoint.y);
6           pDC->MoveTo(StartPoint.x, StartPoint.y);
7           pDC->LineTo(point.x, point.y);
8           DrawPoint.x = point.x; //***********
9           DrawPoint.y = point.y; //***********
10          pDC->SetROP2(oldMode);
        }

        if((nFlags && MK_LBUTTON) && FlagRectangle){
            CClientDC dc(this);
            oldMode = pDC->GetROP2();
            pDC->SetROP2(R2_MERGEPENNOT);
            pDC->SelectStockObject(NULL_BRUSH);
            pDC->Rectangle(DrawPoint.x, DrawPoint.y, StartPoint.x,
            StartPoint.y);
            pDC->Rectangle(StartPoint.x, StartPoint.y, point.x, point.y);
            DrawPoint.x = point.x; //***********
            DrawPoint.y = point.y; //***********
            pDC->SetROP2(oldMode);
        }
```

```
        if((nFlags && MK_LBUTTON) && FlagEllipse){

                CClientDC dc(this);
                oldMode=pDC->GetROP2();
                pDC->SetROP2(R2_MERGEPENNOT);
                pDC->SelectStockObject(NULL_BRUSH);
                pDC->Ellipse(DrawPoint.x, DrawPoint.y, StartPoint.x, StartPoint.y);
                pDC->Ellipse(StartPoint.x, StartPoint.y, point.x, point.y);
                DrawPoint.x = point.x; //***********
                DrawPoint.y = point.y; //***********
                pDC->SetROP2(oldMode);

        }

11 ➡    delete pDC;
12 ➡    CView::OnMouseMove(nFlags, point);
    }
```

Строка 1. Объявление переменной int (oldMode) для запоминания предыдущего растрового режима вывода изображений. Растровый режим вывода изображений или бинарные растровые операции задают относительный вид изображения в контексте устройства (относительно контейнера и предыдущих заданных параметров). Например: бинарный растровый режим (R2) R2_BLACK – пиксель всегда имеет чёрный цвет, R2_MERGEPENNOT – цвет писеля = (NOT цвет экрана) OR цвет пера, R2_NOT – цвет пикселя – инвертированный цвет экрана.

Строка 2. Переменной oldMode присваивается значение предыдущей бинарной растровой операции через указатель (cDC) на контекст устройства (CDC) и функцию класса CDC - GetROP2() – отдать код операции.

Строка 3. Установка режима бинарной растровой операции.

Строки 4, 5. Стирание устаревших на данный момент линий.

Строки 6 - 10. Прорисовка «итоговых» линий в исходном режиме, определяемом функцией pDC->SetROP2(oldMode).

Строками 1 - 10 определяются заданные функциональные возможности «графического редактора» при формировании изображения линии.

Строки последующего программного текста предназначены для формирования аналогичных действий при работе с прямоугольниками и эллипсами. Аналогичность этой программной логики ранее рассмотренной Вам должна быть очевидна.

Строка 11. Освобождается память, выделенная для хранения текущего объекта вида. Выделялась она передачей указателя pDC конструктору класса CClientDC с использованием ключевого слова this. Для этого использовался оператор CClientDC* pDC = new CClientDC(this); Класс CClientDC является

производным от класса CDC (контекст устройства) и позволяет выводить изображение в объект вида в произвольный момент, без использования функции OnDraw().

Строка 12. Функция OnMouseMove класса CView (объекта вида) принимает (и их значения заносятся в память) действующие на данный момент параметры: режим формирования изображения и координаты. Передаются они функции стандартно – списком аргументов списка её параметров.

Задание 15

В этом задании продолжаем усовершенствование нашего графического редактора. Наверное, Вы уже догадались, что для комфортного продолжения дальнейшей работы желательно иметь резервную копию предыдущего работоспособного проекта. Ну а естественным желанием пользователя Вашей программы будет желание выбора цвета изображения. Исследуем учебный пример реализации этой функции программы.

1. Добавьте в проек Paint диалоговое окно для выбора цвета рисования (комада Project – Add Resource - Dialog).

Расположите на диалоговом окне по три элемента ActiveX: Static Text, Slider Control, Edit Text.

Предлагаемые имена идентификаторов этих девяти элементов: IDC_STATIC_R, IDC_STATIC_G, IDC_STATIC_B, IDC_SLIDER_R, IDC_SLIDER_G, IDC_SLIDER_B, IDC_EDIT_R, IDC_EDIT_G, IDC_EDIT_B.

Идентификатор диалогового окна IDD_DIALOG1 в версии MS MC++9.0 изменению как бы не подлежит (свойство Name заблокировано). В примере реализации задания предлагается изменение ID на IDD_DIALOG_RGB (это можно сделать через свойство ID). В примере также предлагается отмена заголовка окна диалога (Title bar). На рисунке 7 показан пример выполнения этого пункта задания.

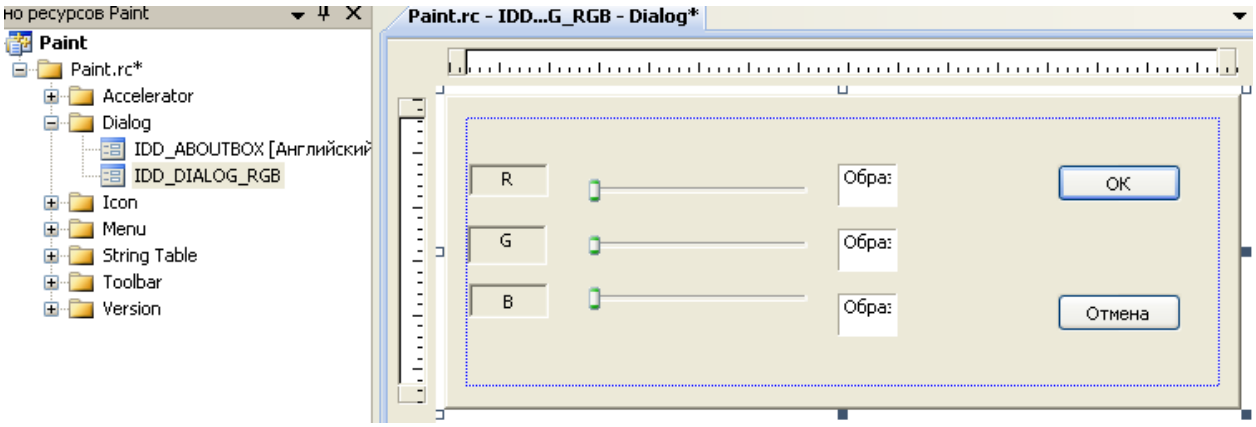

Рис. 7. Примерный вариант окна диалога IDD_DIALOG_RGB

2. Добавьте в проект класс диалогового окна с именем DlgRgb (Project – Add Class – MFC class: доступ – Public: базовый класс – CDialog: идентификатор ресурса - IDD_DIALOG_RGB).

3. Добавьте команду меню Tools – RGB и присвойте ей идентификатор (предлагается примером выполнения задания) ID_TOOLS_RGB. Изложенное

в данном пункте представлено рисунком 8. Через контекстное меню команды RGB в редакторе команд подайте команду Добавить обработчик события (Add Event Handler). Результатом должно быть генерирование функции void CPaintView::OnToolsRgb() в объекте вида (command, Class – CPaintView).

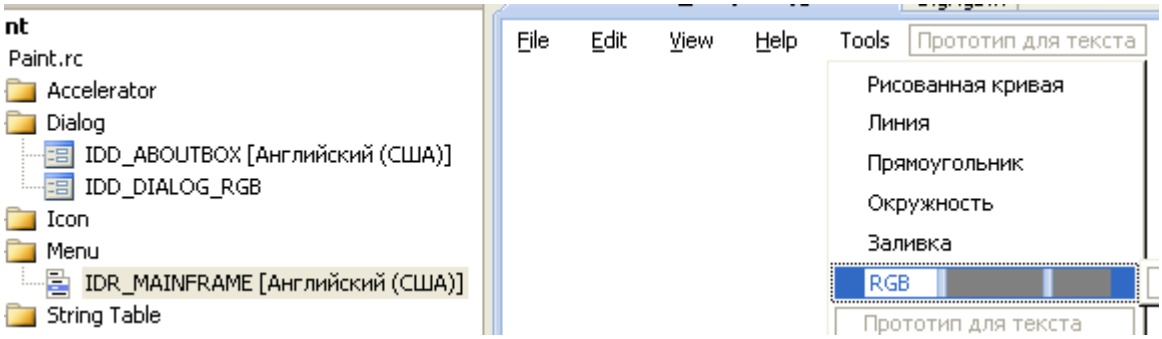

Рис. 8. Предлагаемые действия в редакторе меню

4. Добавьте кнопку в панель инструментов и присвойте ей такой же идентификатор, что и для соответствующей команды меню - ID_TOOLS_RGB. ***ВНИМАНИЕ:*** Это обязательное требование Wizard. Рисунком 9 представлен пример выполнения этого пункта задания.

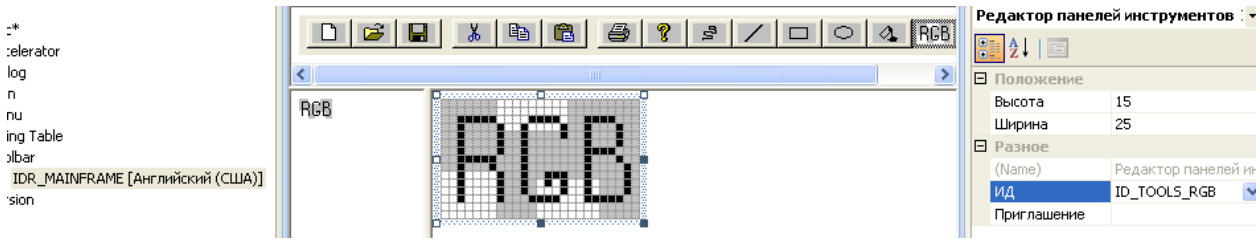

Рис. 9. Предлагаемые действия в редакторе Toolbar

Для открытия созданного диалогового окна необходимо сгенерировать следующий программный код:

```
void CPaintView::OnToolsRgb()
{
        // TODO: добавьте свой код обработчика команд
1       DlgRgb rgb; //**************
2       int result = rgb.DoModal(); //******
}
```

Строка 1. Объявление объекта rgb класса DlgRgb.

Для Вас должно быть естественным, что перед использованием этого оператора необходимо включение в данный файл (CPaintView) заголовочного файла класса DlgRgb директивой препроцесора #include "DlgRgb.h"

Строка 2. Задаёт отображение диалогового окна на экране методом DoModal() (т.е., как модальное окно). Метод возвращает целое значение, которое сохраняется в переменной int result .

5. Создайте для шести элементов ActiveX диалогового окна (Slider Control, Edit Text) объектные переменные. В примере выполнения задания использовано следующее соответствие элементов ActiveX (представлены ИД) и имён их объектных переменных:

- IDC_SLIDER_R – m_slider_R;
- IDC_SLIDER_G – m_slider_G;
- IDC_SLIDER_B – m_slider_B;
- IDC_EDIT_R – m_edit_R;
- IDC_EDIT_G – m_edit_G;
- IDC_EDIT_B – m_edit_B.

Напоминаем, что для добавления переменной в MFC VC++9 проще всего воспользоваться командой Добавить переменную (Add variable) контекстного меню элемента в окне редактора ресурсов для активизации соответствующего мастера. В диалоговом окне Wizard необходимо выбрать следующие опции для объектных переменных ActiveX Edit Text: Категория (Category) - Value, т.е., Тип переменной (Type) – CString, Имя переменной (Name) – логика присвоения идентификатора предложена ранее (например, m_edit_R). Для объектных переменных ActiveX Slider Control: Категория (Category) - Control, т.е., Тип переменной (Type) – CSliderCtrl, Имя переменной (Name) – логика присвоения идентификатора аналогична (например, m_slider_R).

6. Инициализируйте элементы диалогового окна в методе OnInitDialog(). Если данного сообщения нет в списке сообщений (Messages) класса DlgRgb, то необходимо выполнить следующие действия (естественно, класс DlgRgb по прежнему должен быть выделен в окне классов): в окне свойств выбрать опцию Переопределить (Overrides), в списке сообщений базового класса CDialog выделить OnInitDialog, в раскрывающемся списке его поля выбрать и подать команду Добавить (Add). В результате метод OnInitDialog() будет унаследован классом диалогового окна DlgRgb и Вы перейдёте в его программный код, который нужно модифицировать следующим образом:

```
BOOL DlgRgb::OnInitDialog()
{
        CDialog::OnInitDialog();
        // TODO:  Добавить дополнительную инициализацию
        m_slider_R.SetRangeMin(0,false);
        m_slider_R.SetRangeMax(255,false);
```

```
        m_slider_G.SetRangeMin(0,false);
        m_slider_G.SetRangeMax(255,false);
        m_slider_B.SetRangeMin(0,false);
        m_slider_B.SetRangeMax(255,false);
        m_edit_R = "0";
        m_edit_G = "0";
        m_edit_B = "0";
        UpdateData(false);
        return TRUE;
    }
```

Методами SetRangeMin() и SetRangeMax() задаются граничные значения ползунков. Второй параметр этих методов установлен в false – отказ от перерисовки после изменения интервала.

7. При перемещении бегунка слайдер генерирует сообщение WM_HSCROLL (WM_VSCROLL – вертикальные ползунки). Добавьте в программу метод OnHScroll(), вызываемый этим сообщением. В тело функции OnHScroll() внесите следующие программные строки:

```
void DlgRgb::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    if(nSBCode==SB_THUMBPOSITION) {
        if(DYNAMIC_DOWNCAST(CSliderCtrl,pScrollBar)==(&m_slider_R)) {
            m_edit_R.Format(_T("%1d"), nPos);
            UpdateData(false);
        }
        if(DYNAMIC_DOWNCAST(CSliderCtrl,pScrollBar)==(&m_slider_G)) {
            m_edit_G.Format(_T("%1d"), nPos);
            UpdateData(false);
        }
        if(DYNAMIC_DOWNCAST(CSliderCtrl,pScrollBar)==(&m_slider_B)) {
            m_edit_B.Format(_T("%1d"), nPos);
            UpdateData(false);
        }
    }
    else {
    CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
    }
}
```

Первый параметр метода OnHScroll() – код операции прокрутки, который может принимать стандартные значения сообщений о прокрутке в Windows. Примеры некоторых из них: SB_LINELEFT – прокрутка влево, SB_LINERIGHT – прокрутка вправо, SB_THUMBPOSITION – прокрутка в задаваемую позицию, SB_THUMTRACK – перемещение бегунка в заданную позицию.

Второй параметр метода OnHScroll() – значение новой позиции бегунка.

Третий параметр метода OnHScroll() – указатель на элемент, от которого поступило сообщение.

Т.о., оператор if верхнего уровня отслеживает прокрутку одного из элементов в задаваемую позицию. Если поступило сообщение SB_THUMBPOSITION, производится последовательный анализ того, какой из трёх слайдеров прислал это сообщение. Анализ проводится на основании значения указателя pScrollBar, переданного одним из элементов методу OnHScroll(). Естественно, значение указателя – адрес объекта.

DYNAMIC_DOWNCAST(*class*, *pointer*) – макрос библиотеки MFC (заголовочный файл afx.h), преобразующий переданный ему указатель *pointer* к типу *class*. В нашем случае базовым классом объектов – "ползунки" является CSliderCtrl.

Теперь должно быть очевидным, что в условиях вложенных структур if сравниваются адреса объектов. Первый операнд условий – адрес объекта, переданный методу OnHScroll() сообщением Wndows WM_HSCROLL, с обязательным преобразованием типа переданного указателя к типу указателя на объект данного класса. Второй операнд условий – адрес объекта, возвращаемый объектными переменными слайдеров.

В теле структур if с использованием метода Format() класса CString задано отображение чисел в соответствующих текстовых полях. Эти числовые данные в диапазоне 0..255 предоставляются функцией OnHScroll() через её локальную переменную (параметр) nPos.

ВЫВОДЫ: 1. Создано программное обеспечение для выбора цвета в естественной для программиста форме.

2. Данное программное обеспечение носит учебный характер – экономнее было бы использование одного из готовых элементов ActiveX, предназначенных для выбора цвета (но, такие элементы выбора кода цвета изображения через «цветовой куб», наверное, широко не распространены).

3. В нашем примере, разумнее с точки зрения профессионального разработчика было бы создание элемента ActiveX, реализующего функции нашего диалогового окна (и не только уже рассмотренные). Но, такая разработка довольно сложна для данного этапа обучения.

4. Должно быть очевидным, что в нашем приложении на настоящем этапе разработки отсутствуют следующие функциональные возможности:

- вывод изображения цвета на диалоговое окно в естественной для пользователя форме;
- ввод выбранного кода цвета изображения в базовую программу формирования графических изображений;
- запоминание выбранного кода цвета изображения на время работы всей программы.

5. Создание этих функциональных возможностей и определит содержание следующих двух заданий.

Задание 16

В этом задании реализуется вывод изображения цвета на диалоговое окно в естественной для пользователя форме. Это можно было бы сделать через использование штатных графических фреймов MFC. В данном задании предлагается создание собственного элемента ActiveX, предназначенного для вывода изображения цвета. Во-первых, создание элемента ActiveX целесообразно в учебных целях. Во-вторых, использование графических фреймов определяет гораздо большее потребление ресурсов системы (это про профессиональную практику разработки ПО).

ВНИМАНИЕ: ещё раз напоминание – резервное копирование уместно. Может, было бы разумным создание новой директории, перенесение в неё работающего проекта предыдущего задания и, только после этого, внесение в него предлагаемых заданием изменений.

1. Задайте создание нового проекта. Тип проекта – элемент управления ActiveX библиотеки MFC. Имя проекта (в примере выполнения задания) – colorX. Расположение (в ходе реализации последовательности заданий – Lesson_9_3). Все остальные параметры проекта можно не регулировать, т.е., командой Finish (Готово) необходимо подтвердить создание элемента управления ActiveX colorX.

2. В заголовочном файле элемента ActiveX colorXCtrl.h (идентификатор – пример реализации задания) введите два оператора:

```
#pragma once
// colorXCtrl.h: объявление класса элемента управления ActiveX для
CcolorXCtrl.

// CcolorXCtrl: про реализацию см. colorXCtrl.cpp.

class CcolorXCtrl : public COleControl
{
                .
                .
                .
// Переопределение (Overrides)
public:
        virtual void OnDraw(CDC* pdc, const CRect& rcBounds, const CRect&
        rcInvalid);
        virtual void DoPropExchange(CPropExchange* pPX);
        virtual void OnResetState();

// Реализация (Implementation)
protected:
        ~CcolorXCtrl();
1 ⟶     CRect boxColor;
2 ⟶     int Rr, Gg, Bb;
                .
                .
```

Строка 1. Объявление объекта boxColor класса MFC CRect – выделение прямоугольной области в объекте вида.

Строка 2. Объявление трёх переменных типа int для манипулирования кодировкой цвета изображения.

В конструкторе (файл реализации – colorXCtrl.cpp) инициализируйте эти переменные нулями:

```
// CcolorXCtrl::CcolorXCtrl - Конструктор

CcolorXCtrl::CcolorXCtrl()
{
        InitializeIIDs(&IID_DcolorX, &IID_DcolorXEvents);
        // TODO. Инициализируйте здесь данные экземпляра элемента
        //управления.
        // TODO: Initialize your control's instance data here.
        Rr = Gg = Bb = 0; //*************************
}
```

3. Теперь изменим изображение, выводимое нашим элементом в контейнер, заменив программные строки, предлагаемые Wizard MFC на следующие:

```
// CcolorXCtrl::OnDraw - функция рисования (Drawing function)

void CcolorXCtrl::OnDraw(CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
        if (!pdc)
                return;

        // TODO. Замените следующий код собственным кодом рисования.
        // TODO: Replace the following code with your own drawing code.
 1 ⟶
        pdc->FillRect(rcBounds,
        CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH)));
 2 ⟶
        boxColor = CRect(rcBounds.left, rcBounds.top, rcBounds.right,rcBounds.bottom);
 3 ⟶
        pdc->Rectangle(&boxColor);
 4 ⟶
        pdc->FillSolidRect(&boxColor, RGB(Rr, Gg, Bb));
}
```

Строка 1. Через указатель pdc вызывается метод OnDraw для вывода через контекст устройства прямоугольника rcBounds, в пределах которого будет сформировано изображение элемента ActiveX.

ВНИМАНИЕ: это единственная оставшаяся строка, сформированная Wizard MFC. Содержащийся в ней перечень действий избыточен для нашей

задачи, т.е., она может быть значительно упрощена. С другой стороны, отмеченная избыточность не мешает реализации поставленной задачи – поэтому строка оставлена (Вы, может быть, сформируете иной вариант реализации?). Но, в любом случае, Вам придётся разобраться в том, что в этой строке использовано. Необходимые для этого комментарии приведены ниже.

FillRect. Метод рисования в контексте устройства класса CDC – рисование прямоугольника, закрашиваемого определяемой в качестве второго параметра метода кистью.

rcBounds. Первый принимаемый методом FillRect параметр, объект класса CRect, содержащий логические координаты прямоугольника.

CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH)). Второй принимаемый методом FillRect параметр – это идентификация кисти, которой должен быть заполнен прямоугольник. CBrush – должно быть очевидным, что это класс MFC, предназначенный для создания объектов, реализующих заполнение областей объекта вида заданными цветовыми композициями.

CBrush::FromHandle - Returns a pointer to a CBrush object when given a handle to a Windows HBRUSH object. (Возвращает указатель на объект класса CBrush при получении «**хендлера**» на объект Windows HBRUSH).

HBRUSH – встроенный тип данных Windows. Handle to a brush. This type is declared in WinDef.h as follows: typedef HANDLE HBRUSH («Хендлер» на кисть. Этот тип декларирован в WinDef.h как тип, определяющий «хендлер» на объект типа кисть Windows HBRUSH;

GetStockObject. The GetStockObject function retrieves a handle to one of the stock pens, brushes, fonts, or palettes. (Stock – род.). Получение «хендлера» на объект? (afxctl.h - MFC ActiveX Controls). Функциональное назначение рассматриваемой синтаксической конструкции может быть сформулировано следующим образом – задание в качестве кисти стандартной кисти Windows – белой. Стандарты Windows хранятся, естественно, программными блоками, имеющими машинные адреса. Имеет место таблица размещения по адресам таких программных блоков. Обращение к блокам осуществляется через номер записи (ключевое поле)? «Хендлер» MS в настоящее время – тип данных для обращения к табличной служебной информации Windows по ключевым полям БД?

WHITE_BRUSH – стандарт кисти Windows – белая.

СПРАВОЧНОЕ:

1. Handle. Приемлемые с точки зрения программирования варианты перевода: рукоять; удобный случай, предлог; делать (*что-л.*) руками; перебирать, перекладывать; управлять, регулировать; обращаться к *кому-л.*, *чему-л.*.

2. Понятие Handle зависит от версии Windows. В старых версиях Windows, handle - это АДРЕС внутренней структуры Windows. В структуре описан тип объекта, его физическое размещение в памяти, и т.п. - служебное. В новых версиях, это уже и не адрес, а номер все той же структуры в одной

из таблиц объектов Windows?

3. «Хендлер» - перевод одного из русскоязычных программистов - Patch, в смысле – не всё в программировании есть окна. Т.е., имеется программное обращение, и обслуживает его некий блок. Рукоятка - ссылка на объект (этого блока) в памяти, предоставляемый и узнаваемый платформой.

Т.е., по сути, «Хендлер» - это идентификатор программного блока, представляющий из себя или адрес, или номер блока в одной из таблиц, в записи которой, естественно, хранится и адрес?

РЕЗЮМЕ: 1. Командная строка 1 является заготовкой для реализации «самодельной» кисти Windows, но использует более чем стандартную. 2. Эта заготовка (см. «заготовку» ActiveX MS VC++6.0) используется в MS C++ по меньшей мере с 1998 г. 3. MS – «великие путаники», с определённой точки зрения.

Строка 2. Созданному нами объекту boxColor класса CRect методом класса CDC CRect передаются логические параметры прямоугольника, определённые пользователем при визуализации элемента ActiveX (Строка 1).

Строка 3. Контекст устройства CDC через указатель pdc и метод Rectangle получает адрес объекта boxColor и выводит его в обект вида. Т.е. прямоугольник boxColor полностью «захватывает» область, выделенную для rcBounds.

Строка 4. CDC::FillSolidRect - this member function to fill the given rectangle with the specified solid color (Эта функция – член класса используется для заполнения данного прямоугольника сплошным цветом.). Первый параметр функции – адрес объекта, второй её параметр – цвет заполнения объекта.

4. На заключительном этапе разработки нашего элемента ActiveX необходимо создание его метода, который вызывался бы контейнером элемента для изменения его цвета.

В MS VC++6.0 и версиях MS VC++.net более ранних версий чем версия 2008 года, AppWizard ActiveX поддерживали технологию OLE Automation ActiveX создания COM компонентов ActiveX. При выборе программистом опции Automation мастер приложения генерировал заданные методы и свойства. В MS VC++9.0 (комплектация Professional Edition) такая поддержка зачем то отсутствует. Возможно - забыли, возможно – «навредничали», возможно – где-то «глубоко скрыли». По нашей версии оестествлён второй вариант с целью затруднения работы начинающих и осознаному использованию программистом механизмов MFC (можно использовать OnCommand, OnCmdMsg?).

Предлагаем Вам вариант создания требуемого метода создаваемого элемента «вручную» (hendler). Причём, для этого проще всего использовать в качестве образца коды, сгенерированные AppWizard ActiveX MS VC++6.0 при задании ему Automation – AddMethod.

Откройте программный текст файла colorX.idl и введите в него

следующую программную строку:

```
// colorX.idl: источник библиотеки типов для проекта элементов
// управления ActiveX.

// Этот файл будет обработан компилятором MIDL для
// создания библиотеки типов (colorX.tlb), которая станет ресурсом в
// colorX.ocx.

#include <olectl.h>
#include <idispids.h>

[ uuid(414A4928-D337-43F7-8A5A-FFDC2B8E7DD6), version(1.0),
  helpfile("colorX2.hlp"),
  helpstring("Модуль элементов ActiveX colorX"),
  control ]
library colorXLib
{
        importlib(STDOLE_TLB);

        //  Первичный интерфейс диспетчеризации для CcolorXCtrl

        [ uuid(BEF75F35-16A8-4E7F-9452-4AD2DCB23AED),
          helpstring("Интерфейс отправки для colorX Control")]
        dispinterface _DcolorX
        {
                properties:
                methods:

                        [id(1)] void outColor(short R, short G, short B);

                        [id(DISPID_ABOUTBOX)] void AboutBox();
        };
                .
                .
                .
```

Этой строкой в интерфейсе диспетчеризации заявлен метод с именем outColor, который должен принимать при вызове три параметра типа short. Этот метод имеет идентификатор 1. Если бы был второй метод, он должен был бы получить идентификатор 2, с тем же синтаксисом и т.д. В файле colorX.idl содержится исходный код языка описания объектов для библиотеки типов элемента управления.

Далее откройте программный текст файла colorXCtrl.h и введите в него программные строки:

```
                .
                .
                .
        // Схемы сообщений
                DECLARE_MESSAGE_MAP()

1  ⟶           afx_msg void outColor(short R, short G, short B);

        // Схемы подготовки к отправке
                DECLARE_DISPATCH_MAP()

                afx_msg void AboutBox();

        // Схемы событий
                DECLARE_EVENT_MAP()

        // Подготовка к отправке и ИД событий
        public:
                enum {
                //{{AFX_DISP_ID(COutColorCtrl)
2  ⟶           dispidOutColor = 1L,
                //}}AFX_DISP_ID
                };
        };
```

Строка 1. Объявление прототипа функции outColor(), принимающей три параметра типа short и отдающей void. Предшествующее void ключевое сочетание afx_msg по аналогии с «классическим» C++ можно интерпретировать по синтаксису как virtual. И в этом случае у MS VC++ MFC имеется некоторая «загадочность». «Загадочность» (путаница?) заключается в использовании трёх понятийных аббревиатур: **AFX, MFC, FCL.**

Microsoft Foundation Classes (MFC) – библиотека фирмы MS, реализованная на C++ и предназначенная для оптимизации разработки GUI - приложений для MS Windows. В библиотеку MFC входит набор классов, поддерживающих создание каркаса приложения. Разработку каркаса приложения и базовых составляющих приложения выполняют программные конструкции – Wizards (мастера - в переводе для профессиональной литературы, а буквальный перевод – волшебники, маги, колдуны). Программисту (по мнению MS) необходимо генерировать собственный код только в специфических местах и случаях. Добавление собственного кода приложения к каркасу приложения возможно двумя способами.

Первый способ – использование механизма наследования. Основные программные конструкции каркасу приложения представлены в виде классов, наследуемых от библиотечных классов. В этих классах определено множество виртуальных функций, которые могут переопределяться (доопределяться – поскольку, во многих случаях необходим вызов функции

базового класса). При переопределении (доопределении) этих функций программист может добавлять выполнение в эти моменты своего кода.

Второй способ – добавление обработчиков оконных событий. При этом мастер создаёт в оконном каркасе приложения - специализированные массивы – карты (оконных) сообщений (message map). Они, по стандарту (MS), содержат пары «ИД сообщения - указатель на обработчик». Описание всех обработчиков стандартных сообщений Windows можно найти в файле <afxwin.h> в объявлении класса CWnd. Командные сообщения Windows от меню, командных клавиш и кнопок панелей инструментов обрабатываются макросом:

ON_COMMAND (id, nemberFn). Этот макрос в качестве параметров использует идентификатор команды id и произвольное имя обработчика команды memberFn. Прототип обработчика сообщения должен быть описан в классе программиста (в нашем примере – класс CcolorXCtrl) и иметь вид:

afx__msg void memberFn(); в нашем примере – memberFn представляет идентификатор outColor с заданными нами параметрами этой функции. Обработчик команды — memberFn — будет вызван, только если в оконную процедуру поступит сообщение, которое в параметре wParam (в нашем случае 1 или 1L) содержит идентификатор команды, совпадающий с id.

Класс CObject является базовым для большинства классов библиотеки MFC. Класс CCmdTarget является базовым для всех классов, которые работают с сообщениями. Из всех компонентов этого класса (в рамках нашего исследования) интересен один – функция – член OnCmdMsg:

```
virtual BOOL CCmdTarget: :OnCmdMsg (
        UINT nD,
        int nCode,
        void* pExtra,
        AFX_CMDHANDLERINFO *pHandlerilfo)
```

Эта функция устанавливает маршрут и распределяет командные сообщения. Если внимательно просмотреть поставляемые MS исходные тексты, то можно разглядеть, что это единственная функция (не считая оконной процедуры по умолчанию), которая вызывается в оконной процедуре. Это означает, что именно она занимается поиском соответствующих обработчиков сообщений.

В качестве параметров функция принимает: nD — идентификатор команды; nCode— код извещения; pExtra — определяется значением nCode; pHandlerInfo — обычно равен NULL, в противном случае функция сама заполняет элементы pTargetw pmf структуры pHandlerInfo. Если функция нашла адресата, который обработает сообщение, то она возвращает true , в противном случае - false. Т.о., основным назначением OnCmdMsg является распределение команды по другим объектам. В принципе, эту функцию можно ***переопределить***, однако при этом на Вас перекладывается задача

указания следующего получателя команды, чтобы не прервалась цепочка передачи сообщений.

Application Framework Extensions (AFX) – аббревиатура программной концепции «оконных классов», присвоенная программистами MS на ранних стадиях разработки этой концепции соответствующим программным конструкциям. Чуть позже эти программные конструкции было решено поименовать фундаментальнее - **MFC** (видимо, это было решением руководящего состава MS). Релиз MFC 1.0 (Microsoft C/C++ 7/0) состоялся в 1992 г. Но, до сих пор «замечательнейшей» особенностью MFC является использование префикса "afx" (несмотря на то, что есть MFC «выше» 10). Он используется в именах многих функций, макросов, в названии стандартного заголовочного файла «stdafx.h» и соответствующего файла реализации «stdafx.cpp». Большинство функций MFC являются виртуальными и для их переопределения «программирующему» в MFC приходится использовать префикс afx как синоним virtual (точнее – иметь это в виду).

FCL (Framework Class Library) – библиотека классов системы .NET.Framework. Концепция заявлена MS в 2000/2001 г.г. По этой концепции FCL – это логическое продолжение идеи MFC, охватывающее большинство систем программирования. Термин FCL, похоже, не прижился (и был потихоньку выведен из обихода через год, два). Так, например, версия MS Visual C++ 2010 имеет MFC компонент MFC 10.0.30319.1 (mfc100.dll), .NET 4.0 (апрель 2010). Префикс "afx" в соответствующих программных конструкциях MFC используется до сих пор (традиции программистов MS?).

MFC Wizards (маг, волшебник MFC). После проведённого краткого справочного обзора рассмотрим программные строки, сформированные MFC Wizards для примера решения нашего задания. Строка, предшествующая Строке 1 , рассматриваемого фрагмента кода: DECLARE_MESSAGE_MAP().

Макрос DECLARE_MESSAGE_MAP располагается в конце объявления класса, использующего карту сообщений (в нашем примере – класс CcolorXCtrl). Карта сообщений (message map) представляет собой механизм пересылки сообщений и команд Windows в объекты приложения, реализованного на базе MFC. Она предоставляет альтернативу многочисленным операторам switch, используемым в традиционных Windows - программах для обработки сообщений, посылаемых окну. Такие карты преобразуют сообщения Windows в функции соответствующих классов, которые их обрабатывают. Карта сообщений реализована на основе идеи виртуальных функций C++, но имеет дополнительные системные и программные «уточнения».

Каждый класс, который может получить сообщение, должен иметь свою карту сообщений для того, чтобы соответствующим образом обрабатывать сообщения.

СИНТАКСИС: карта сообщений должна определяться вне какой-либо функции или объявления класса. Она также не может размещаться внутри C - блока. Для определения карты сообщений используются три макроса:

BEGIN_ MESSAGE_MAP, END_MESSAGE_MAP и DECLARE_MESSAGE_MAP. Макрос DECLARE_MESSAGE_MAP располагается в конце объявления класса, использующего карту сообщений. Структура карты сообщений достаточно проста и представляет собой набор макросов (операторов), заключенных в специальные так называемые "операторные скобки" ClassWizard:

//{{AFX_MSG_ MAP(CTheClass) и //{(AFX_MSG_MAP).

В карте сообщений «нашего приложения» определён только один прототип вызываемой по сообщению функции:

```
afx_msg void outColor(short R, short G, short B);
```

Сообщение – это «повод» для вызова функции (метода), связанной с этим сообщением? Но возможно и декларирование собственных методов в карте диспетчиризации (диспетчерские схемы) DECLARE_DISPATCH_MAP()? Можете проверить этот вариант разработки приложений при наличии соответствующей заинтерисованности.

Строка 2. Оператор enum *Name* {,…,} в языке С задаёт перечисление. В данном случае (MFC) определяется перечисление диспетчерских идентификаторов и идентификаторов событий. Закоментированный набор символов

//{{AFX_DISP_ID(COutColorCtrl) и //}}AFX_DISP_ID ,

так же представляет собой “операторные скобки” ClassWizard. Перечисление не имеет имени в реализации ClassWizard MFC. Первый член перечисления dispidOutColor = 1L имеет уникальное имя dispidOutColor (dispid, видимо ассоциатив с ID диспетчера?) и ему присваивается идентификатор – целочисленная 1 (L – long?). Если бы мы использовали второй метод, после запятой в теле enum нужно было бы вписать dispid*Name2* = 2L (может, от использования префикса dispid можно и отказаться? – попробуете?).

ВОПРОС: может быть, для создания метода попробовать воспользоваться переопределением OnCommand или OnCmg? (в обще-то, подобный вопрос задавался чуть раньше.).

В заключение «ручной работы» в программном тексте файла colorXCtrl.cpp нужно внедрить программные строки 1, 2, 3, что и отображают нижеприведённые фрагменты программного текста.

```
                .
                .
                .
// Схема подготовки к отправке

BEGIN_DISPATCH_MAP(CcolorXCtrl, COleControl)
        //*****************************************************
1 ⟶     DISP_FUNCTION(CcolorXCtrl, "outColor", outColor, VT_EMPTY, VTS_I2
VTS_I2 VTS_I2)
        //*****************************************************

        DISP_FUNCTION_ID(CcolorXCtrl, "AboutBox", DISPID_ABOUTBOX,
AboutBox, VT_EMPTY, VTS_NONE)

END_DISPATCH_MAP()
                .
                .
                .
```

Строка 1. Назначение макроса BEGIN_DISPATCH_MAP (CcolorXCtrl, COleControl), открывающего тело таблицы (карты) диспетчиризации, после ранее изложенного учебного материала должно быть понятным, равно как и назначение макроса END_DISPATCH_MAP(). Прототип фунции – члена, фомирующей метод обработки сообщения, имеющий место в карте сообщений, должен иметь следующий набор параметров.

DISP_FUNCTION(*theClass*, *pszName*, *pfnMember*, *vtRetVal*, *vtsParams*)

Defines an OLE automation function in a dispatch map.

Parameters

theClass

Name of the class.

pszName

External name of the function.

pfnMember

Name of the member function.

vtRetVal

A value specifying the function's return type. VT_EMPTY (void), VT_I2 (short), VT_I4 (long).

vtsParams

A space-separated list of one or more constants specifying the function's parameter list.

```
                .
                .
                .
void CcolorXCtrl::outColor(short R, short G, short B)
{
        // TODO: Add your dispatch handler code here

2 ⟶     Rr = R; Gg = G; Bb = B;
                .
                .
                .

3 ⟶     Invalidate();

}
```

Строка 2. Присвоение принятых методом outColor() параметров внутренним переменным элемента управления. Естественно, параметры передаются методу при его вызове.

Строка 3. Перерисовка элемента управления.

5. Для изменения внешнего вида значка элемента ActiveX воспользуйтесь соответствующим ресурсом – Ресурсы – Bitmap – ресурс IDB_COLORX. Дизайн значка элемента – элемент Вашей специальности.

6. Проведите построение – отладку элемента управления. На заключительном этапе построения (Build) предлагаем выбрать режим - Отладочный файл – Отмена (хотя можно попробовать и другие варианты). При успешном построении «созданный Вами» элемент ActiveX будет зарегистрирован в Вашей системе.

7. Теперь хотелось бы протестировать созданный элемент ActiveX. В MS VC++6.0 имеется утилита MS AciveX Control Test Container. Запускается она одноимённой командой из меню Tools окна среды разработки. Файл этой программы – TSTCON32.EXE. Найти его можно в директории: …\Microsoft Visual Studio\COMMON\Tools\... (Например, скопировать и использовать для наших целей).

В MS VC++.net более ранних версий, чем версия 2008 утилита тестирования элементов ActiveX - TstCon32.exe размещалась в директории …\Common7\Tools… . В MS VC++9.0.net комплектации Professional эта удобная программа зачем-то отсутствует?

Для тестирования элемента ActiveX задания 16 использовался TSTCON32.EXE. Можете повторить. Работа с подобной утилитой должна быть интуитивно понятной.

Задание 17

В этом задании предлагается продолжение работы, начатой в заданиях 15 и 16 – работа с цветом изображения в учебном графическом редакторе. А именно, программно реализуются следующие функциональные возможности приложения: индикация заданного цвета; ввод выбранного кода цвета изображения в базовую программу формирования графических изображений; запоминание выбранного кода цвета изображения на время работы всей программы.

1. Добавим созданный в задании 16 элемент ActiveX в проект Paint (резервное копирование, по-прежнему уместно). Естественно, перед добавлением элемент ActiveX должен быть открыт экземпляр проекта Paint.

Для такого добавления необходимо выбрать Ресурсы – IDD_DIALOG_RGB – и активизировать данный ресурс. Для того, чтобы добавить элемент ActiveX в проект необходимо подать команду меню окна среды Сервис – Выбрать элементы панели элементов. То же самое можно организовать через контекстное меню выбранной группы (группу можно создать и свою) Панели элементов – командой Выбрать элемент. Выберите вкладку COM – компоненты и найдите «свой» элемент ActiveX по имени и растровому изображению. Элемент CheckBox соответствующей строки списка установите в true, проверьте, подтвердите выбор командой OK.

Разместите экземпляр элемента controlX Control стандартным способом в диалоговом окне IDD_DIALOG_RGB. В окне свойств элемента измените его ИД на IDC_COLORX.

Создайте для элемента IDC_COLORX объектную переменную – m_colorX (контекстное меню элемента – команда Добавить переменную – ввести только один параметр – имя переменной).

Теперь можно воспользоваться методом элемента controlX – outClor и программно организовать вывод изображения выбранного цвета на экран через созданный в задании 16 элемент ActiveX. Тем самым, мы реализуем выполнение первого из трёх пунктов необходимых функциональных возможностей нашего приложения, сформулированных в п.4 выводов задания 15 (они продублированы во введении к данному заданию).

Поскольку все эти три пункта программно формируются практически в одних и тех же программных блоках, в целях компактизации текста методики выполнения работы рассмотрим примеры кодового конструирования этих пунктов совместно. Т.е., не в порядке задания программных кодов необходимых, необходимых для выполнения функционала конкретного пункта, а в порядке наполнения всеми нужными программными кодами (реализующих не одну функциональную возможность) одного из модифицируемых файлов проекта.

2. В файле PaintView.h определите три переменные:

```
        .
        .
        .
public:
    CPaintDoc* GetDocument() const;

        int r, g, b;
        .
        .
        .
```

Их можно определить и под квалификатором Protected (комментарии к такому варианту приведены после небольшого проведённого расследования – см. п.14 данного задания).

3. В конструкторе (файл PaintView.cpp) эти три переменные должны быть инициализированы нулями:

```
CPaintView::CPaintView()
{
    // TODO: add construction code here
    AllFlagsFalse();
    r = g = b = 0;
}
```

4. В код функции – обработчика сообщения OnLButtonUp (файл PaintView.cpp) введите следующие программные строки:

```
void CPaintView::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default

1   CBrush cb;
2   cb.CreateSolidBrush(RGB(r, g, b));

    CPaintDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    EndPoint.x = point.x;
    EndPoint.y = point.y;

    CClientDC* pDC = new CClientDC(this);
```

```
3 ⟶     CPen cp;
4 ⟶     cp.CreatePen(PS_SOLID, 1 , RGB(r, g, b));
5 ⟶     pDC->SelectObject(&cp);

        if(FlagLine) {
                pDC->MoveTo(StartPoint.x, StartPoint.y);
                pDC->LineTo(EndPoint.x, EndPoint.y);
                pDoc->pMFDC->MoveTo(StartPoint.x, StartPoint.y);
                pDoc->pMFDC->LineTo(EndPoint.x, EndPoint.y);
        }
        if(FlagRectangle) {
                pDC->SelectStockObject(NULL_BRUSH);
                pDC->Rectangle(StartPoint.x, StartPoint.y, EndPoint.x,EndPoint.y);
                pDoc->pMFDC->SelectStockObject(NULL_BRUSH);
    pDoc->pMFDC->Rectangle(StartPoint.x, StartPoint.y, EndPoint.x, EndPoint.y);
        }
        if(FlagEllipse) {
                pDC->SelectStockObject(NULL_BRUSH);
                pDC->Ellipse(StartPoint.x, StartPoint.y, EndPoint.x, EndPoint.y);
                pDoc->pMFDC->SelectStockObject(NULL_BRUSH);
    pDoc->pMFDC->Ellipse(StartPoint.x, StartPoint.y, EndPoint.x, EndPoint.y);
        }
        if(FlagFill) {
6 ⟶             pDC->SelectObject(cb);
7 ⟶
    pDC->FloodFill(StartPoint.x, StartPoint.y, RGB(r, g, b));
8 ⟶             pDoc->pMFDC->SelectObject(&cb);
9 ⟶             pDoc->pMFDC->FloodFill(StartPoint.x, StartPoint.y, RGB(r, g, b));
        }
        delete pDC;

    //      CView::OnLButtonUp(nFlags, point); //**Kill****
    }
```

Строка 1. Создание объекта cb класса CBrush в объекте вида. CBrush – класс MFC для создания «собственной» кисти (в частности, заданное цветовое заполнение определённой области контекста устройства).

Строка 2. Вызов метода CreateSolidBrush() класса CBrush для объекта cb в объекте вида.

Метод CreateSolidBrush(*crColor*) класса CBrush инициализирует кисть со специализируемым сплошным цветовым заполнением. В качестве параметра цвета может быть использован макрос RGB – WINDOWS.h.

Строка 3. Создание объекта cp класса CPen в объекте вида. CPen – класс MFC для создания «собственного» пера (реализующего, в частности, заданный цвет линии).

Строка 4. Вызов метода CreatePen() класса CPen для объекта cp.

Метод CreatePen() принимает три параметра: стиль, толщину линии,

цвет. В нашем случке задаются: PS_SOLID – предпочтителен при заполнении линии сплошным цветом (стандартный стиль?); 1 – толщина линии – 1 пиксель; RGB – WINDOWS.h.

Строка 5. Контексту устройства методом SelectObject() передаётся объект cp класса CPen (указатель – адрес).

Строка 6. Контексту устройства методом класса CDC контекста устройства SelectObject() передаётся объект cb класса CBrush (система указатель – адрес).

Строка 7. Вызывается метод класса CDC FloodFill() – заполнение зоны дисплея заданным цветом.

Строки 8, 9. Дублирование операций с контекстом устройства, определяемых строками 6, 7, в метафайле.

5. В код функции – обработчика сообщения (файл PaintView.cpp) OnMouseMove добавьте следующие программные строки:

```
void CPaintView::OnMouseMove(UINT nFlags, CPoint point)
{
        // TODO: Add your message handler code here and/or call default
        CPaintDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        int oldMode;
        CClientDC* pDC = new CClientDC(this);

1 ⟶     CPen cp;
2 ⟶     cp.CreatePen(PS_SOLID, 1, RGB(r, g, b));
3 ⟶     pDC->SelectObject(&cp);


        if((nFlags && MK_LBUTTON) && FlagDraw) {
                pDC->MoveTo(StartPoint.x, StartPoint.y);
                pDC->LineTo(point.x, point.y);
                pDoc->pMFDC->MoveTo(StartPoint.x, StartPoint.y);
                pDoc->pMFDC->LineTo(point.x, point.y);
                StartPoint.x = point.x;
                StartPoint.y = point.y; }

        if((nFlags && MK_LBUTTON) && FlagLine){
                oldMode = pDC->GetROP2();
4 ⟶            pDC->SetROP2(R2_NOTXORPEN);
                pDC->MoveTo(StartPoint.x, StartPoint.y);
                pDC->LineTo(DrawPoint.x, DrawPoint.y);
                pDC->MoveTo(StartPoint.x, StartPoint.y);
                pDC->LineTo(point.x, point.y);
                DrawPoint.x = point.x;
                DrawPoint.y = point.y;
                pDC->SetROP2(oldMode);
```

```
            }

            if((nFlags && MK_LBUTTON)&&FlagRectangle){
                  CClientDC dc(this);
                  oldMode = pDC->GetROP2();
5 ⟶               pDC->SetROP2(R2_NOTXORPEN);
                  pDC->SelectStockObject(NULL_BRUSH);
      pDC->Rectangle(DrawPoint.x, DrawPoint.y, StartPoint.x, StartPoint.y);
                  pDC->Rectangle(StartPoint.x, StartPoint.y, point.x, point.y);
                  DrawPoint.x = point.x;
                  DrawPoint.y = point.y;
                  pDC->SetROP2(oldMode);
            }

            if((nFlags && MK_LBUTTON)&&FlagEllipse){

                  CClientDC dc(this);
                  oldMode=pDC->GetROP2();
6 ⟶               pDC->SetROP2(R2_NOTXORPEN);
                  pDC->SelectStockObject(NULL_BRUSH);
                  pDC->Ellipse(DrawPoint.x, DrawPoint.y, StartPoint.x, StartPoint.y);
                  pDC->Ellipse(StartPoint.x, StartPoint.y, point.x, point.y);
                  DrawPoint.x = point.x;
                  DrawPoint.y = point.y;
                  pDC->SetROP2(oldMode);

            }

            delete pDC;
            //CView::OnMouseMove(nFlags, point);
      }
```

Строки 1 - 3. Строки аналогичны строкам, откомментированным в предыдущем пункте данного задания.

Строки 4, 5, 6. CDC::SetROP2() – метод устанавливает в контексте устройства бинарный растровый режим. Режим R2_NOTXORPEN – цвет пикселя = NOT (цвет пера XOR цвет экрана). Функциональное назначение использования операции то же, что и в задании 14 (см. строки 1 – 12) – стирание «старой» линии и прорисовка «новой» заданного цвета.

6. Добавьте в проект функцию (файл PaintView.cpp) – обработчик сообщения OnToolsRgb() и внесите в тело этой функции следующие программные строки:

```
void CPaintView::OnToolsRgb()
{
        // TODO: добавьте свой код обработчика команд
1       DlgRgb rgb;
2       int result = rgb.DoModal();

3       if(result == IDOK) {

4               r = rgb.rD;
                g = rgb.gD;
                b = rgb.bD;

7               Memory mw;
8               mw.inMem(r, g, b);
        }

}
```

Строка 1. Создаётся объект rgb класса диалогового окна DlgRgb.

Строка 2. Переменной result присваивается результат, возвращаемый методом DoModal() класса CDialog (если нажата кнопка OK – IDOK).

Строка 3. Если нажата кнопка OK – выполняется блок операторов тела условия.

Строки 4 - 6. Переменным цветовой кодировки объекта вида присваиваются значения соответствующих переменных диалогового окна.

Строка 7. Создаётся объект mw класса Memory.

ВНИМАНИЕ: 1. Переменные rD, gD, bD в классе DlgRgb Вами ещё не определены (см. п.8 задания). 2. Класс Memory Вами ещё не создавался (см. п.14 задания). Естественно, попытка отладки проекта на этом этапе его реализации без предварительного выполнения указанных пунктов, вызовет ошибки. 3. Это же относится и к строке 8 данного пункта и к пункту 7 задания.

Строка 8. Вызывается функция - член класса Memory inMem(r, g, b), унаследованная объектом этого класса mw.

7. Должно быть очевидным, что директивы препроцессора файла PaintView.cpp для работоспособности программных кодов п.6 должны выглядеть следующим образом:

```
#include "stdafx.h"
#include "Paint.h"

#include "PaintDoc.h"
#include "PaintView.h"
#include "DlgRgb.h"
#include "Memory.h"
        .
        .
        .
```

8. В классе DlgPgb объявите три переменные (файл DlgPgb.h):

```
class DlgRgb : public CDialog
{
        DECLARE_DYNAMIC(DlgRgb)

public:
        DlgRgb(CWnd* pParent = NULL);   // стандартный конструктор
        virtual ~DlgRgb();
        short rD, gD, bD;
                .
                .
                .
```

9. Файл DlgRgb.cpp должен иметь следующее «начало»:

```
        // DlgRgb.cpp: файл реализации
        //

        #include "stdafx.h"
        #include "Paint.h"
        #include "DlgRgb.h"
1       #include "Memory.h"

        // диалоговое окно DlgRgb

        IMPLEMENT_DYNAMIC(DlgRgb, CDialog)

        DlgRgb::DlgRgb(CWnd* pParent /*=NULL*/)
                : CDialog(DlgRgb::IDD, pParent)
                , m_edit_R(_T(""))
                , m_edit_G(_T(""))
                , m_edit_B(_T(""))
        {
2               rD = gD = bD = 0;
        }
                .
                .
                .
```

Строка 1 – стандарт C++. Строка 2 – инициализация переменных цветовой кодировки нулями так же является обязательной в данной реализации.

10. Добавьте в проект (файл DlgRgb.cpp) функцию – обработчик сообщения OnBnClickedOk(), её программный код должен иметь следующий вид:

```
void DlgRgb::OnBnClickedOk()
{
        // TODO: добавьте свой код обработчика уведомлений
        UpdateData(true);
        OnOK();
}
```

11. Модифицируйте функцию OnInitDialog() (файл DlgRgb.cpp):

```
BOOL DlgRgb::OnInitDialog()
{
        CDialog::OnInitDialog();

        // TODO:  Добавить дополнительную инициализацию
        m_slider_R.SetRangeMin(0, false);
        m_slider_R.SetRangeMax(255, false);
        m_slider_G.SetRangeMin(0, false);
        m_slider_G.SetRangeMax(255, false);
        m_slider_B.SetRangeMin(0, false);
        m_slider_B.SetRangeMax(255, false);

1       Memory md;

2       rD = md.outMem(0);
3       gD = md.outMem(1);
4       bD = md.outMem(2);

5       m_edit_R.Format(_T("%1d"), rD);
6       m_edit_G.Format(_T("%1d"), gD);
7       m_edit_B.Format(_T("%1d"), bD);
        UpdateData(false);

8       m_colorX.outColor(rD, gD, bD);

9       m_slider_R.SetPos(rD);
10      m_slider_G.SetPos(gD);
11      m_slider_B.SetPos(bD);

        return TRUE;  // return TRUE unless you set the focus to a control
        // Исключение: страница свойств OCX должна возвращать
        //значение FALSE
}
```

Строка 7. Создаётся объект mw класса Memory.

ВНИМАНИЕ: Класс Memory Вами ещё не создавался (см. п.14 задания), т.е. попытка построения на данном этапе вызовет ошибки.

Строки 2 - 4. Переменным цветовой кодировки класса DlgRgb присваиваются значения, хранящиеся в массиве объекта mw класса Memory. Обращение к элементам этого массив – через вызов функции – члена outMem(*index*).

Строки 5 - 7. В элементы Edit Control класса DlgRgb выводятся текстовые значения, отформатированные из числовых значений, хранящихся в переменных цветовой кодировки.

Строка 8. Вызов метода outColor(, ,) элемента ActiveX, созданного Вами в предыдущем задании (через объектную переменную m_colorX, созданную ранее для представления объекта этого элемента). Результат – изменение цвета выделенной для элемента зоны на дисплее.

Строки 9 - 11. Слайдеры выбора цвета класса DlgRgb устанавливаются в позиции, соответствующие значениям, хранящимся в переменных цветовой кодировки класса DlgRgb. Обращение к ним осуществляется через соответствующие объектные переменные, при этом вызывается метод SetPos() класса CSlider.

12. Модифицируйте OnHScroll() (файл DlgRgb.cpp) следующим образом:

```
void DlgRgb::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
        // TODO: добавьте свой код обработчика сообщений или вызов
        //стандартного

        if(nSBCode==SB_THUMBPOSITION){

if(DYNAMIC_DOWNCAST(CSliderCtrl,pScrollBar)==(&m_slider_R)){
                        m_edit_R.Format(_T("%1d"), nPos);
                        UpdateData(false);
1 ⟶                     rD = nPos;
                }

if(DYNAMIC_DOWNCAST(CSliderCtrl,pScrollBar)==(&m_slider_G)){
                        m_edit_G.Format(_T("%1d"), nPos);
                        UpdateData(false);
2 ⟶                     gD = nPos;
                }

if(DYNAMIC_DOWNCAST(CSliderCtrl,pScrollBar)==(&m_slider_B)){
                        m_edit_B.Format(_T("%1d"), nPos);
                        UpdateData(false);
3 ⟶                     bD = nPos;
                }
```

```
4 ➡   m_colorX.outColor(rD, gD, bD);

            }
            else {
            CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
            }
      }
```

Строки 1 - 3. Одной из трёх переменных цветовой кодировки класса DlgRgb присваиваентся значение, установленное только одним, соответствующим ей слайдером.

Строка 4. Строка аналогична строке 8 предыдущего пункта задания.

13. Добавьте в проект (файл DlgRgb.cpp) функции – обработчики сообщений – «Изменение состояния» (OnEnChangeEdit) трёх элементов EditText цветовой кодировки OnEnChangeEdit…():

```
      void DlgRgb::OnEnChangeEditR()
      {
            // TODO:  Добавьте код элемента управления
1 ➡   UpdateData(true);
2 ➡   rD = _wtoi(m_edit_R);

3 ➡   if((rD > 255)||(rD < 0))
4 ➡         {rD = 0; m_edit_R = _T("");
5 ➡          UpdateData(false);
            }
6 ➡   m_slider_R.SetPos(rD);
7 ➡   m_colorX.outColor(rD, gD, bD);
      }

      void DlgRgb::OnEnChangeEditG()
      {
            // TODO:  Добавьте код элемента управления
            UpdateData(true);
            gD = _wtoi(m_edit_G);

            if((gD > 255)||(gD < 0))
                  {gD = 0; m_edit_G = _T("");
                   UpdateData(false);
            }
            m_slider_G.SetPos(gD);
            m_colorX.outColor(rD,gD,bD);
      }

      void DlgRgb::OnEnChangeEditB()
      {
            // TODO:  Добавьте код элемента управления
```

```
        UpdateData(true);
        bD = _wtoi(m_edit_B);

        if((bD > 255)||(bD < 0))
                {bD = 0; m_edit_B = _T("");
                 UpdateData(false);
        }
        m_slider_B.SetPos(bD);
        m_colorX.outColor(rD,gD,bD);
}
```

Строка 1. В комментариях, пожалуй, уже не должна нуждаться.

Строка 2. Переменной rD (объект DlgRgb – красный цвет) присваивается значение, преобразованное из текста в целочисленное значение функцией MS C++ _wtoi().

Функция MS C++ (UTF-16) _wtoi() является аналогом функции atoi() «классического» C++ (стандарт ANSI/ISO - 1999 г.), хранящейся в библиотеке с заголовочным файлом <cstdlib>. MFC VC++6 библиотеки <cstdlib> поддерживал и использовал, VC++ 9 – не поддерживает (видимо, MS решил использовать свои фирменные программные конструкции "под Windows"). Зато классы VC++ 9 не нуждаются во включении вышеупомянутого (или всех подобных?) заголовочного файла – УРА?!

Строки 3 - 5. Если введённое в Edit Control числовое значение выходит за границы диапазона цветовой кодировки – задаём обнуление переменной цветовой кодировки объекта DlgRgb и соответствующего текстового поля.

Строка 6. Слайдер устанавливается в значение переменной цветовой кодировки объекта DlgRgb.

Строка 7. В комментариях, наверное, не должна нуждаться. Осмысленно создана и откомментирована ранее.

Два последующих фрагмента программного кода двух методов ввода с клавиатуры кодировок Blue и Green имеют назначение, аналогичное рассмотренному фрагменту, поэтому назначение их программных строк в комментариях также не нуждается.

14. Добавьте в проект новый класс. Это может быть класс MFC, но, может быть лучше (или – всё равно?) – класс C++? Имя класса C++ предлагаемого примера – Memory. Для чего в предлагаемом учебном примере выполнения задания использован этот класс? Просто-напросто - для «запоминания» выбранной через диалоговое окно цветовой кодировки.

Банальное решение этого вопроса в MFC C++ оказывается не однозначным. «Классические» варианты C++ в MFC заблокированы. Во всяком случае - два из них (третий – «просто не работает»).

Первый вариант – переменные r, g , b объекта вида, определённые как public , сохраняют за собой ячейки памяти на всё время выполнения всего

проекта. Т.е., они сохраняют значения, и после закрытия диалогового окна выбора цвета. Но, доступ к ним (членам – данным объекта вида) из класса DlgRgb заблокирован (зачем, почему? – оестествлён принцип минимума привилегий?). Попробуете?

Второй вариант – можно было бы объявить переменные r, g , b как extern:

```
extern int r, g, b;
```

Тогда по концепции «классического» C++ возможно их дублирование и использование в объектах других классов. В MFC этот класс памяти extern не поддерживается (по крайней мере – в объекте вида?).

Третий вариант – Можно объявить в объекте DlgRgb указатель на базовый класс вида (создание указателя на объект вида заблокировано):

```
CView *pcv;
```

и попробовать им воспользоваться в этом же объекте (приведённая синтаксическая конструкция не рекомендуется прграммирующими на «классическом» C++):

```
m_edit_R.Format(_T("%1d"), ((CPaintView *)pcv)->r);
```

Система MFC C++ 9.0 эту синтаксическую конструкцию пропускает и обрабатывает, но в период runtime работает не по тем адресам.

Более простые обращения к членам – данным объекта вида не пропускаются вовсе.

Может быть, воспользоваться объектом документа? Попробуйте.

Поэтому, исходя из нашего предположения, сформулированного в одном из предыдущих заданий – MFC C++ ориентирован на работу с классами (прежде всего – со своими), выбираем вариант программного формирования запоминания цветовой кодировки с использованием собственного класса. В этом классе должны быть члены – данные для хранения цветовой кодировки и два члена – функции для её записи и считывания (сервисные функции).

Программный текст заголовочного файла (Memory.h) нашего класса «памяти» с предлагаемым идентификатором Memory выглядит следующим образом:

```
#pragma once

class Memory
{
1 ⟶     static int arrMem[3];
public:
        Memory(void);
        ~Memory(void);

2 ⟶     void inMem(int R, int G, int B);
3 ⟶     int outMem(int);
};
```

Программный код файла реализации (Memory.cpp) этого класса

```
                #include "StdAfx.h"
                #include "Memory.h"

4 ➡             int Memory::arrMem[3];

                Memory::Memory(void)
                {
                }

                Memory::~Memory(void)
                {
                }
5 ➡             void Memory::inMem(int R, int G, int B)
                {
6 ➡                     Memory::arrMem[0] = R; arrMem[1] = G; arrMem[2] = B;
                }

7 ➡             int Memory::outMem(int index)
                {
8 ➡                     return Memory::arrMem[index];
                }
```

Строка 1. В определении класса Memory декларируем static int arrMem[3] с доступом private. Использование квалификатора private – соответствие принципам C++ ограничения доступа к данным, использование квалификатора static – сохранение принятых членами-данными значений при выходе объекта класса из области действия.

Строка 2. Определение прототипа функции inMem(), не возвращающей значения и, принимающей в списке параметров три целочисленных аргумента. Назначение этой сервисной функции – запись цветовой кодировки в ячейки памяти.

Строка 3. Определение прототипа функции outMem(), возвращающей значение типа int и, принимающей в качестве параметра аргумент типа int. Назначение функции – считывание одного из трёх параметров цветовой кодировки. Функция так же является сервисной.

Строка 4. Создаётся массив arrMem[3] класса Memory.

Строка 5. Определение функции inMem() класса Memory, принимающей в списке параметров три целочисленных аргумента (переменные R, G, B определяются как локальные переменные функции в её определении).

Строка 6. Блок операторов тела функции inMem(). Этими операторами трём элементам массива arrMem[] объекта класса Memory присваиваются значения локальных переменных функции R, G, B, переданные им в качестве аргументов при вызове функции. Назначение функции inMem() – принять

параметры цветовой кодировки и разместить их для хранения в объекте класса Memory (в элементах массива arrMem[]).

Строка 7. Определение функции outMem() класса Memory, принимающей в списке параметров один целочисленный аргумент (переменная index определяется так же как локальная переменная функции в её определении).

Строка 8. Оператор тела функции outMem (). Им задаётся возврат функцией одного из трёх значений цветовой кодировки в вызывающую функцию программу. При этом вызывающая функция задаёт в качестве аргумента индекс 0 – 2 (0 – красный, 1 – зелёный, 2 – голубой).

ВЫВОДЫ: 1. Создано программное обеспечение – учебный графический редактор, обеспечивающий минимальный набор функциональных возможностей.

2. Приведённые программные решения носят учебный же характер и ни в коей мере не претендуют на оптимальность и завершённость.

3. Рассмотрены некоторые особенности использования и реализации фирмой MS платформы и концепций C++.net в MFC.

4. После просмотра и возможного освоения приведённых примеров решения этой частной задачи программной реализации средств дизайна Вам, наверное, будет более понятным весь комплекс проблем, связанных с решением подобных задач.

Задание 18

Основы работы с базами данных с использованием VC++ MFC.

1. Создайте директорию для нового проекта. По нашей оценке – это десятая директория, соответствующая реализации заданий данного учебного пособия. Т.е., логично присвоение ей имени, например, Lesson_10. Для получения начальных навыков по работе с БД в MFC Вам, прежде всего, необходимо наличие БД. Создайте однотабличную БД ядра MS Jet 4.0, используя оболочку MS Access. Пусть наша с Вами таблица будет предназначена для размещения в ней списка студенческой группы. Т.е., в примитиве, она должна содержать четыре поля: ключевое поле – индекс (тип поля - счётчик); три текстовых поля, соответствующие Ф.И.О. В примере выполнения данного задания эта таблица имеет имя - tbl_Student. Заполните эту таблицу данными по Вашей группе и сохраните файл БД в корневой директории данного задания. В примере выполнения задания 18 этот файл поименован как bd1.mdb.

2. Используем AppWizard MFC (MS – мага) для создания проекта, реализующего начальные принципы работы с БД,

Создайте новый проект MFC, имя проекта примера выполнения задания – bd, дислокация – каталог Lesson_10. Установите опцию Application Type в Single Document. Выберите опцию Database Support и один из её CheckButton's – Database view without file support переведите в состояние true. Client type – OLE DB.

Теперь Вам нужно подключиться к источнику данных по технологии OLE DB. Для этого кнопкой Data Source откройте диалоговое окно выбора канала передачи данных. Выберите в качестве провайдера – Microsoft Jet4.0 OLE DB Provider и, либо кнопкой – Далее, либо вкладкой – Соединение, перейдите к диалоговому окну сведений для подключения к БД. Через соответствующий элемент TextBox или диалоговое окно выбора файла определите файл Вашей БД в качестве источника данных OLE DB. Выполните проверку подключения к БД,

Окно диалога Select Database Object позволит Вам выбрать единственную таблицу Вашей БД из группы Tables (напоминаем, в приводимом примере, она именуется - tbl_Student). Завершить диалог с Wizard MFC можно и кнопкой Finish, можно и последовательными переходами кнопкой Next.

ВНИМАНИЕ: на данной стадии освоения предлагаемой темы Вам не надо бы неосознанно изменять не отмеченные в этом задании опции.

3. Диалог с «колдуном» (Wizard MFC 9.0) проведён и им на этом основании должен быть сформирован работоспособный проект. Выполните построение этого проекта и проанализируйте возникшие, вероятнее всего, ошибки. В нашем варианте исследования «работы магов MFC» - это проблемы с паролем (password), т.е., - проблемы с security (но, в диалоге с

«волшебником» выбрана опция – «без пароля»?). Сообщение компоновщика при этом выглядит примерно так:

lesson_10\bd\bdset.h(52) : fatal error C1189: #error : Security Issue: The connection string may contain a password

Попробуем исправить эту ситуацию «вручную» (“hendler”). В заголовочном файле класса настроек БД (в примере – bdSet.h) найдём и «закомментируем» строку 52:

//#error Security Issue: The connection string may contain a password

Проблема, вероятнее всего, успешно разрешится – проект будет построен, но его диалоговое окно, используемое в качестве основного, конкретной осмысленности не приобретёт. Для придания проекту конкретики по работе с нашей БД опять-таки требуется внедрение в созданное «волшебником MFC» «вручную». Проделаем эти операции на примитивном уровне в следующих пунктах задания.

4. Для начала сформируем требуемую нам «объектную область» объекта вида – главного диалогового окна проекта. Любым уже известным Вам способом войдите в Recource View. Далее раскрываем содержимое Dialog и активизируем редактор диалоговых окон на главном окне диалога (в примере решения – двойной щелчок на IDD_BD_FORM). Удалите «приглашающее» поле Static Text и расположите в «объектном поле» окна, требуемые по заданию элементы Edit Control. Осмысление их количества, расположения, оформления, идентификаторов, наличия пояснений – предмет Вашего творчества.

ВНИМАНИЕ: идентификаторы примера могут не совпадать с Вашими идентификаторами.

5. Создайте для полей Edit Control объектные переменные по методике ранее изученных работ. В примере выполнении задания имеет место соответствие между идентификаторами элементов Edit Control и идентификаторами соответствующих им объектных переменных, отображённое таблицей 18.1.

ВНИМАНИЕ: в нашем примере значения переменных имеют тип control. Но, значение Category изменено в Value (в первую очередь). Соответственно, «магом» Variable type установлено в CString. Max char – 51. Variable name – m_edit1 (2, 3).

Впрочем, вариант, при котором Вы всё созданное «колдуном MS» оставите без изменений, пройдёт так же хорошо, как и предлагаемый.

Объектные переменные элементов управления

Таблица 18.1

ИД элемента управления	Идентификатор переменной
IDC_EDIT1	m_edit1
IDC_ EDIT2	m_edit2
IDC_ EDIT3	m_ edit3

6. В тело функции, формирующей метод DoDataExchange объекта вида, введите модифицирующие этот метод строки 1 – 3. Они определяют перерисовку содержимого элементов Edit Control по уже привычной для Вас после освоения основ работы с диалоговыми окнами методике. Обратите внимание, что размер текстовых полей в БД примера ограничен пятьюдесятью символами.

```
void CbdView::DoDataExchange(CDataExchange* pDX)
{
	COleDBRecordView::DoDataExchange(pDX);
	// you can insert DDX_* functions, as well as SetDlgItem*/GetDlgItem* API calls
	//to link your database to the view
	// ex. ::SetDlgItemText(m_hWnd, IDC_MYCONTROL, m_pSet->m_MyColumn);
	// See MSDN and OLEDB samples for more information

	DDX_Text(pDX, IDC_EDIT1, m_edit1);
	DDV_MaxChars(pDX, m_edit1, 51);
	DDX_Text(pDX, IDC_EDIT2, m_edit2);
	DDV_MaxChars(pDX, m_edit2, 51);
	DDX_Text(pDX, IDC_EDIT3, m_edit3);
	DDV_MaxChars(pDX, m_edit3, 51);

	DDX_Text(pDX, IDC_EDIT1, m_pSet->m_Name1, 51); //*  <-- 1
	DDX_Text(pDX, IDC_EDIT2, m_pSet->m_Name2, 51); //*  <-- 2
	DDX_Text(pDX, IDC_EDIT3, m_pSet->m_Name3, 51); //*  <-- 3

}
```

Строки 1 – 3. Используется указатель m_pSet на набор записей, созданный AppWizard. Он позволяет обращаться к полям отсоединённого набора записей Recordset (Recordset - копия записей таблицы БД в primary memory) БД, определённой соответствующим подключением. Это обращение и использовано в строках 1 – 3 через объектные переменные полей набора записей (m_Name1, m_Name2, m_Name3), которые так же сформированы AppWizard.

7. Для отображения данных набора записей OLE DB в данном приложении MEC использован класс MEC COleDBRecordView. Объект представления записей, созданный из класса COleDBRecordView, позволяет отображать записи баз данных в элементах управления MFC. Представление

записей (в нашем случае это объект вида CbdView, наследующий класс COleDBRecordView) является диалоговым представлением формы, напрямую подключенным к объекту набора строк OLE DB, созданному из класса шаблонов CRowset.

В представлении отображаются поля объекта CRowset в элементах управления диалогового окна. Объект COleDBRecordView использует набор функций Dialog Data Exchange (DDX) и функцию навигации, встроенную в класс CRowset (MoveFirst, MoveNext, MovePrev и MoveLast) для автоматизации перемещения данных между элементами управления формы и полями набора строк. Представление COleDBRecordView отслеживает положение пользователя в наборе строк, чтобы представление записей могло обновлять пользовательский интерфейс, и предоставляет метод OnMove для обновления текущей строки.

По итогам выполнения этого задания Вы, наверное, должны сделать вывод о необходимости наличия высокого уровня профессионализма разработчика при работе с базами данных в VC++ (соответствие этого вывода действительности, впрочем, определяется уровнем Вашего интеллектуального благосостояния). Профессиональная организация работы с базами данных посредством VC C++ MFC требует профессиональной же ориентации в использовании иерархической системы множества специализированных классов библиотеки MFC.

Содержание

№ задания.	Наименование	Стр.
Задание 1.	Объект документа, объект вида – вывод текста	3
Задание 2.	Вывод текста без использования объекта документа	9
Задание 3.	Основы отладки приложений	10
Задание 4.	Ввод с клавиатуры - отображение в клиентской области	15
Задание 5.	Ввод с клавиатуры - позиционирование отображения символа	18
Задание 6.	Курсор в клиентской области	20
Задание 7.	Работа с мышью - курсор	23
Задание 8.	Диалоговые окна, создание класса диалогового окна	26
Задание 9.	Диалоговое окно в качестве главного	33
Задание 10.	Основы работы с «графикой»	43
Задание 11.	Графические файлы, метафайлы, обновление изображений	49
Задание 12.	Объект документа - сериализация	55
Задание 13.	Стандартные файловые операции в MFC VC++	61
Задание 14.	Графический редактор II	65
Задание 15.	Графический редактор III	69
Задание 16.	Графический редактор IV	74
Задание 17.	Графический редактор V	85
Задание 18.	Основы работы с базами данных	99

Учебное пособие по курсам «Прикладное программирование в информационных системах», «Программная реализация средств дизайна»
Введение в VC++ MFC
Составители: Дмитрий Давидович Ветчинин
Научный редактор: Николай Анатольевич Коробов
Печатается в авторской редакции

УП № 020305 от 28.11.99. Подписано в печать 07.02.2014.
Формат 1/16 60*84. Бумага писчая. Плоская печать.
Усл. печ. л. 7,42 Уч.-изд.7,062. Тираж 26 экз. Заказ № 329

РИО ИвТИ ИВГПУ
ЦОТ кафедры ВПМСИТ ИВГПУ
152000, г. Иваново пр. Шереметьевский, 21.