Федеральное агентство по образованию

Уральский государственный технический университет – УПИ
имени первого Президента России Б. Н. Ельцина

Электронный аналог печатного издания

РАЗРАБОТКА ПРОГРАММ В СРЕДЕ MS VISUAL STUDIO

Методические указания по дисциплине
«Системное программирование»
для студентов специальности 230105 – Программное
обеспечение вычислительной техники
и автоматизированных систем

Екатеринбург
УГТУ – УПИ
2010

УДК 004.2:004.9

Составитель О. Л. Чагаева

Научный редактор С. И. Тимошенко, канд. техн. наук

РАЗРАБОТКА ПРОГРАММ В СРЕДЕ MS Visual Studio : методические указания по дисциплине «Системное программирование» / сост. О. Л. Чагаева. – Екатеринбург: УГТУ – УПИ, 2010. – 35 с.

Работа представляет собой введение в интегрированную среду разработки MS Visual Studio. Проводится обзор основных утилит и мастеров MS Visual Studio. Приводится пример создания приложения по шаблону с помощью мастера MFC Application Wizard.

Библиогр.: 3 назв. Табл. 3. Рис. 2.

Подготовлены кафедрой
«Программные средства и системы»
факультета ускоренного обучения

Оглавление

Введение

Визуальное программирование — способ создания программы путём манипулирования графическими объектами вместо написания ее текста.

Языки визуального программирования могут быть дополнительно классифицированы, в зависимости от типа и степени визуального выражения, на следующие типы:

- языки на основе объектов, когда визуальная среда программирования предоставляет графические или символьные элементы, которыми можно манипулировать интерактивным образом в соответствии с некоторыми правилами;
- языки на основе форм, когда программирование осуществляется помещением на специальные формы объектов и настройкой их свойств и поведения (*примеры: Delphi и C++ Builder фирмы Borland);*
- языки схем, основанные на идее «фигур и линий», где фигуры (прямоугольники, овалы и т. п.) рассматриваются как субъекты и соединяются линиями (стрелками, дугами и др.), которые представляют собой отношения *(пример: UML)*.

Естественно-визуальные языки имеют неотъемлемое визуальное выражение, для которого нет очевидного текстового эквивалента (например, графический язык G в среде LabVIEW).

Визуально-преобразованные языки являются невизуальными языками с наложенным визуальным представлением. Примером является среда Visual C++ для языка C++.

C++ является языком программирования общего назначения. Естественная для него область применения — системное программирование, понимаемое в широком смысле этого слова. Кроме того, C++ успешно используется во многих областях приложения, далеко выходящих за указанные рамки. Реализации C++ теперь есть на всех машинах, начиная с самых скромных микрокомпьютеров и заканчивая самыми большими супер-ЭВМ, и практически для всех операционных систем.

1. Интегрированная среда разработки MS Visual Studio. Понятие проекта и решения

Visual C++ является частью Microsoft Visual Studio — комплекта средств разработки приложений. Visual C++ — это интегрированная среда разработки, и все создаваемые с помощью нее приложения представляют собой проекты.

Проект — это набор взаимосвязанных исходных файлов, компиляция и компоновка которых позволяет создать исполняемую windows-программу или файл с расширением DLL.

Исходные файлы проекта хранятся в отдельном каталоге. Кроме того, проект часто зависит от внешних файлов, например, от подключаемых (include) и библиотечных файлов. В проекте Visual C++ взаимозависимости между отдельными компонентами описаны в текстовом файле проекта с расширением VCPROJ. А специальный текстовый файл решения с расширением SLN содержит список всех проектов данного решения.

Решение (Solution) — набор проектов, объединенных вместе, которые решают одну задачу.

Для того чтобы начать работу с существующим проектом, необходимо открыть в Visual C++ соответствующий SLN-файл. Типы файлов, создаваемых в проекте Visual C++, указаны в табл. 1.

Таблица 1

Типы файлов, создаваемых в проекте Visual C++

Расширение файла	Описание
APS	Поддержка просмотра ресурсов
BSC	Информация браузера
IDL	Файл на языке описания интерфейсов IDL
NCB	Поддержка просмотра классов
SLN	Файл решения
SUO	Поддержка параметров и конфигурации решения
VCPROJ	Файл проекта

Среда разработки Visual Studio предлагает множество инструментов для создания и настройки приложений любого типа. С её помощью можно:

- генерировать скелет приложения без написания кода вручную;
- открывать проект в нескольких различных режимах представления;
- редактировать файлы с исходным кодом и включаемые файлы;
- подключаться к внешним ресурсам (базам данных);
- разрабатывать визуальный интерфейс (меню, иконки, диалоговые окна);
- компилировать и связывать приложение;
- производить отладку приложения в процессе работы.

С технической точки зрения Visual C++ представляет собой один из инструментов Visual Studio. С помощью этой интегрированной среды можно использовать любые другие языки программирования, в том числе разработанные не Microsoft.

2. Утилиты и мастера MS Visual Studio

Написание программ Windows «с чистого листа» вручную требует много времени, причем большая его часть уходит на создание и отладку каркаса приложения. Если используются функции Win API, то требуется написать функции `WinMain` и `WndProc`, а также цикл обработки сообщений. Если же используется библиотека MFC — нужно написать собственный класс приложения, его метод `InitInstance` и класс окна. Среда MS Visual Studio предоставляет набор мастеров и утилит для автоматизации процесса создания каркаса приложения, и, тем самым, избавляет программиста от рутинной работы, которую необходимо проделывать при создании приложения Windows.

2.1. Мастера для создания проектов

1. **MFC Application Wizard (exe)** — мастер для создания проектов Windows-приложений на основе классов библиотеки MFC. Мастер предоставляет программисту богатый выбор настроек проекта. С его помощью можно создавать приложения с однодокументным, многодокументным или диалоговым интерфейсом. Однодокументное приложение позволяет пользователю работать только с одним файлом. Многодокументное приложение может одновременно предоставить работу с несколькими документами, каждым в собственном окне. Пользовательский интерфейс диалогового приложения представляет собой единственное диалоговое окно.

2. **MFC DLL Wizard** — мастер приложений, позволяющий создать структуру DLL, основанную на MFC. При помощи него можно определить характеристики будущей DLL.

3. Средство **ATL Project Wizard** позволяет создать элемент управления ActiveX или сервер автоматизации, используя новую библиотеку шаблонов ActiveX (ActiveX Template Library — ATL). Опции этого мастера дают возможность выбрать активный сервер (DLL) или исполняемый внешний сервер (exe-файл).

4. При помощи средства **Custom Wizard** можно создать пользовательские мастера AppWizard. Пользовательский мастер может базироваться на стандартных мастерах для приложений MFC или DLL, а также на существующих проектах или содержать только определяемые разработчиком шаги.

5. **Visual Studio Add-in Wizard** — мастер дополнений, позволяющий создавать дополнения к Visual Studio. Библиотека расширений DLL может поддерживать панели инструментов и реагировать на события Visual Studio.

6. **MFC ActiveX Control Wizard** — мастер элементов управления, реализующий процесс создания проекта, содержащего один или несколько элементов управления ActiveX, основанных на элементах управления MFC.

7. **Win32 Project Wizard** — мастер, позволяющий создать проект обычного Windows-приложения или динамически подключаемой библиотеки. Тип проекта определяется выбором соответствующих опций в диалоговых окнах

мастера. Проект создается незаполненным, файлы с исходным кодом следует добавлять в него вручную.

8. **Win32 Console Application Wizard** — мастер создания проекта консольного приложения. Проект консольного приложения создается пустым, предполагается добавление в него файлов исходного текста вручную.

2.2. Утилиты для редактирования проектов

1. **Утилита Class View**. Окно Class View открывается при выборе команды View→Class View и отображает дерево всех классов проекта с методами и полями. Чтобы увидеть код элемента, необходимо дважды кликнуть по нему. При внесении изменений в исходный текст содержимое окна Class View автоматически обновляется. За создание новых классов, добавление их в проект, создание виртуальных функций и функций обработчиков сообщений отвечает утилита Class View.

2. **Редактор ресурсов** используется для создания и редактирования ресурсов (меню, панелей управления, строк состояния, курсоров, диалогов и т. д.) в режиме WISIWIG (What you see is what you get — «Что видите, то и получаете»). Окно для просмотра ресурсов проекта открывается при выборе команды View→Resource. Для перехода в режим редактирования уже созданного ресурса необходимо дважды кликнуть по нему. Для создания нового ресурса необходимо вызвать контекстное меню (при помощи правой кнопки мыши) в окне просмотра ресурсов и выбрать пункт Add→Resource.

3. **Утилита Solution Explorer**. В Solution Explorer отображается структура всего решения. Окно Solution Explorer содержит древовидное представление элементов проекта, которые можно открывать по отдельности для модификации или выполнения задач по управлению. Для добавления нового элемента в проект необходимо в окне Solution Explorer щелкнуть правой кнопкой мыши по одному из внутренних узлов дерева и выбрать пункт Add.

Для настройки компиляции и компоновки проекта необходимо выбрать команду Project→Properties… или нажать правой кнопкой мыши на имя проекта в окне Solution Explorer и выбрать пункт Properties.

2.3. Утилита Class View

2.3.1. Создание нового класса

При помощи Class View можно добавить новый класс в проект, созданный на основе базовых классов. Утилита позволяет использовать в качестве базовых классов как классы каркаса MFC, так и собственные классы. Объекты, порожденные от класса `CcmdTarget`, могут обрабатывать сообщения Windows и команды, поступающие от меню, кнопок, акселераторов. Класс `CCmdTarget` и другие, наследованные от него классы, имеют таблицу сообщений (Message Map) — набор макрокоманд, позволяющий сопоставить сообщения Windows и команды метода класса.

Чтобы добавить класс, необходимо выполнить следующие действия:

1. Вызвать контекстное меню проекта в окне утилиты Class View (клик правой кнопкой мыши по имени проекта) и выполнить команду Add→Class.
2. Выбрать в появившемся диалоговом окне тип добавляемого класса (например, Categories: MFC. Templates: MFC class). Нажать Ok.
3. В появившемся окне ввести необходимые данные. Нажать Ok.

Полученная заготовка класса полностью работоспособна. Ее можно дополнить по своему усмотрению новыми методами и данными. Эту работу можно выполнить вручную, но удобнее воспользоваться услугами утилиты Class View. Для этого необходимо:

1. Вызвать контекстное меню соответствующего класса в окне утилиты Class View (клик правой кнопкой мыши по имени класса) и выполнить команду Add→Function или Add→Variable.
2. Следовать дальнейшим инструкциям мастера.

2.3.2. Редактирование классов с помощью Class View

С помощью Class View можно редактировать уже созданные классы, добавлять в них обработчики сообщений для наследников класса `CCmdTarget`, переопределять виртуальные функции. За счет использования Class View процедура редактирования собственного класса значительно ускоряется и уменьшается вероятность совершить ошибку во время объявления методов. Для добавления обработчиков или переопределения методов необходимо:

1. Вызвать контекстное меню соответствующего класса в окне утилиты Class View→Properties.
2. Во всплывающем окне нажать кнопку Events, Messages или Overrides в зависимости от того, что требуется сделать в данный момент (обработать событие, обработать сообщение или переопределить виртуальную функцию).

Class View позволяет не только добавить в класс новые методы, но и удалить их. Class View самостоятельно удалит объявление метода из прототипа класса и его тело из cpp-файла.

2.3.3. Включение в класс новых элементов данных

Class View позволяет включать в класс не только новые поля и методы, но и элементы данных, связанные с полями диалоговых панелей, форм просмотра и форм для просмотра записей баз данных и полей наборов записей. Class View использует специальные процедуры, чтобы привязать созданные им элементы данных к классам и полям диалоговых панелей. Эти процедуры носят названия «обмен данными диалоговой панели» и «проверка данных диалоговой панели» (Dialog Data Exchange и Dialog Data Validation — DDX/DDV). Чтобы привязать поля из наборов записей к переменным, используется процедура обмена данными с полями записей (Record Field Exchange — RFX).

3. СОЗДАНИЕ ПРИЛОЖЕНИЯ ПО ШАБЛОНУ С ПОМОЩЬЮ МАСТЕРА MFC Application Wizard

Благодаря мастеру MFC Application Wizard, среда разработки позволяет быстро создавать новые Windows приложения по шаблону. Разработчику достаточно ответить на ряд вопросов, касающихся того, какое приложение требуется создать, и исходные тексты приложения вместе с файлами ресурсов будут созданы. Эти тексты можно откомпилировать и получить готовый загрузочный модуль приложения, однако прикладную часть приложения должен написать программист.

Работа мастера MFC Application Wizard рассмотрена на примере создания программы с однодокументным интерфейсом с поддержкой технологии «документ–вид»:

1. Выполнить команду: File→New→Project.
2. В окне Project types выбрать MFC. В окне Templates выбрать MFC Application. Ввести имя "MyProg" проекта в строку Name. Снять флажок в пункте Create directory for solution. Нажать кнопку Ok.
3. Нажать кнопку Next. В следующем окне выбрать Application type: Single document. Снять флажок в пункте Use Unicode libraries.
4. Нажимать Next до тех пор, пока кнопка не будет заблокирована (в появляющихся окнах мастер предлагает добавить или удалить различные компоненты: поддержка баз данных, поддержка печати и т. п.).
5. В последнем окне можно увидеть список классов, которые будут созданы мастером.
6. Нажать кнопку Finish. Приложение готово. Можно скомпилировать и запустить.

   В указанной директории "MyProg" будут созданы файлы:

   - MyProg.vcproj — основной файл проекта;
   - MyProg.h — заголовочный файл приложения;
   - MyProg.cpp — исходный текст приложения;
   - StdAfx.h — заголовочный файл для стандартного «каркаса» приложения;
   - StdAfx.cpp — исходный текст стандартного «каркаса» приложения;
   - MainFrm.h — заголовочный файл главного окна;
   - MainFrm.cpp — исходный текст главного окна;
   - MyProgDoc.h — заголовочный файл документа;
   - MyProgDoc.cpp — исходный текст документа;
   - MyProgView.h — заголовочный файл вида;
   - MyProgView.cpp — исходный текст вида;

- Resource.h — файл с ресурсными константами;
- MyProg.rc — файл с ресурсами;
- MyProg.ncb — файл с информацией о представлении и взаимных связях;
- MyProg.sln — файл решения;
- res — каталог для ресурсов.

Программа будет состоять из четырех основных частей (рис. 1):

1. **Объект приложения** находится в файлах MyProg.h и MyProg.cpp. Это то, что Windows запускает при старте программы. Когда этот объект начинает работу, он размещает на экране главное окно.

2. **Объект главного окна** находится в файлах MainFrm.h и MainFrm.cpp и отображает главное окно программы: в нем находится меню, заголовок окна и панель инструментов. Рабочая зона программы называется клиентской областью окна.

3. **Объект документа** находится в файлах MyProgDoc.h и MyProgDoc.cpp и хранит данные программы.

4. **Объект вида** находится в файлах MyProgView.h и MyProgView.cpp и предназначен для работы с клиентской областью. Отображает данные, хранящиеся в объекте документа.

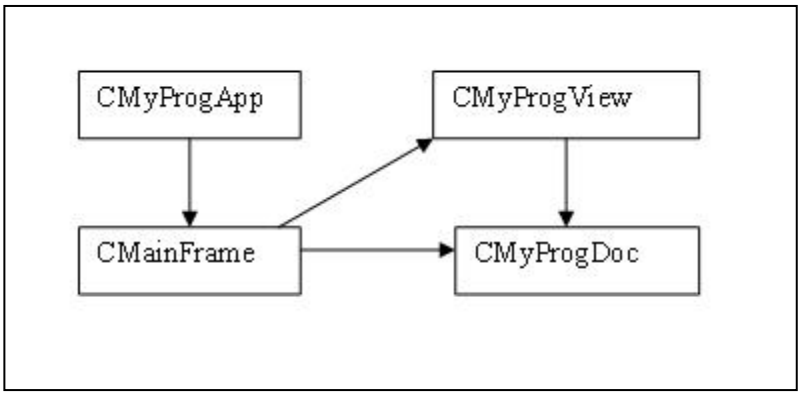

Рис. 1. Схема взаимосвязи частей программы

3.1. Вывод текста в окно программы

Создадим программу, которая будет считать символы с клавиатуры и отображать их в клиентском окне. Для этого будем обрабатывать сообщение Windows `WM_CHAR`. Свяжем с ним функцию `OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)`. Назовем программу Key. Будем создавать ее с помощью MFC Application Wizard, интерфейс SDI (Single Document Interface).

Этапы написания программы

1. Подготовка буфера для хранения символов

Определим в классе CKeyDoc, прототип которого содержится в заголовочном файле KeyDoc.h, строку-объект StringData класса MFC CString. Для этого добавим следующий код:

```
class CKeyDoc : public CDocument
{
  ……
  DECLARE DYNCREATE(CKeyDoc)
Public:
  CString StringData;
  ……
};
```

Проинициализируем переменную пустой строкой " ". Это делается в конструкторе объекта документа, расположенном в файле KeyDoc.cpp. Конструктор объекта документа CKeyDoc:

```
CKeyDoc::CKeyDoc()
{
  // TODO:
  StringData="";
}
```

2. Чтение символов с клавиатуры

При нажатии клавиши Windows посылает сообщение WM_CHAR. Его надо связать с методом OnChar(UINT nChar, UINT nRepCnt, UINT nFlags) объекта вида. Для этого используется утилита Class View:

- вызовем контекстное меню класса CKeyView (нажмем правой кнопкой по имени класса в окне утилиты Class View) и выберем пункт Properties;
- во всплывшем окне свойств нажмем на кнопку Messages;
- выберем в списке сообщений WM_CHAR. Раскроем список действий в пустом поле напротив названия сообщения нажатием левой кнопки мыши. Выберем единственный вариант <Add> OnChar. Мастер автоматически создаст обработчик OnChar и откроет файл KeyView.cpp для редактирования тела метода.

3. Сохранение символа в документе

Введенный символ находится в параметре nChar и его необходимо сохранить в строковом объекте StringData. Обработчик OnChar:

```
void CKeyView::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
   //Получение указателя на объект документа
   CKeyDoc *pDoc = GetDocument();
   //Проверка указателя
   ASSERT_VALID(pDoc);
```

```
   if(!pDoc)
      return;
   //Обновление строки символов
   pDoc->StringData += char(nChar);
   Invalidate();   //Вызов метода OnDraw
}
```

Макрос ASSERT_VALID проверяет, что полученный указатель действительно ссылается на документ (иначе формируется сообщение об ошибке).

4. Отображение текста

Вывод будем осуществлять в методе OnDraw(CDC* pDC) класса вида. Для вызова метода OnDraw используется функция Invalidate. Метод OnDraw:

```
void CKeyView::OnDraw(CDC *pDC)
{
  …
  pDC->TextOut(0,0,pDoc->StringData);
}
```

Скомпилируем и запустим программу. Усложним задачу. Потребуем, чтобы текст выводился в центре клиентской области окна. Для решения поставленной задачи изменим метод OnDraw. Измененный метод OnDraw:

```
void CKeyView::OnDraw(CDC* pDC)
{
  CKeyDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  // TODO: add draw code for native data here
  //Переменная для хранения размеров клиентской области
  CRect rect;
  //Получение размеров клиентской области
  GetWindowRect(&rect);
  //Задание координаты X центра клиентской области
  int x=rect.Width()/2;
  //Задание координаты Y центра клиентской области
  int y=rect.Height()/2;
  //Определение размеров выводимой строки в пикселях
  CSize size = pDC->GetTextExtent(pDoc->StringData);
  //Сдвиг координаты X на половину длины выводимой строки влево
  x -= size.cx/2;
  //Сдвиг координаты Y на половину высоты выводимой строки вверх
  y -= size.cy/2;
  //Вывод строки в центр клиентской области
  pDC->TextOut(x,y,pDoc->StringData);

}
```

3.2. Работа с курсором и мышью

Напишем программу, которая будет создавать курсор в выбранной произвольной точке клиентской области и выводить текст с указанной позиции. Создадим программу SDI под названием Mouse. Воспользуемся материалом предыдущего пункта: создадим обработчик сообщения `WM_CHAR`. Чтобы создать курсор, необходимо знать его размеры (обычно высота равна высоте символа, а ширина — 1/8 ширины символа). Курсор будем создавать в методе `OnDraw`. Чтобы определить размеры курсора, необходимо получить данные из структуры `TEXTMETRIC`. Для этого необходимо создать объект типа `TEXTMETRIC` и заполнить его поля, используя метод `GetTextMetrics(&tm)`. Но сначала добавим в прототип класса `CMouseView` следующие переменные:

```
void CMouseView::OnDraw(CDC* pDC)
{
   CMouseDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   if (!pDoc)
        return;
 //Переменная для хранения информации о шрифте
   TEXTMETRIC textmetric;
 //Получение информации о шрифте
   pDC->GetTextMetrics(&textmetric);
 //Получение размеров выводимой строки в пикселях
   CSize size=pDC->GetTextExtent(pDoc->StringData);
 //Создаём курсор
CreateSolidCaret(textmetric.tmAveCharWidth/8,textmetric.tmHeight);
 //Присваивание x  координаты конца
   CaretPosition.x=size.cx;
 //Присваивание y координаты
   CaretPosition.y=0;
 //Задаем позицию курсора
   SetCaretPos(CaretPosition);
 //Вывод на экран курсора
   ShowCaret();
   pDC->TextOut(0,0,pDoc->StringData);
}
```

Рассмотрим задачу вывода строки с места, на который указывает указатель мыши. Свяжем сообщение `WM_LBUTTONDOWN` с методом `OnLButtonDown(UINT nFlags, CPoint point)`. Параметр `Cpoint` (объект класса `CPoint`) содержит текущие координаты указателя мыши. Для очистки строкового объекта используется метод `Empty()` класса `CString`. Обработчик нажатия левой кнопки мыши `OnLButtonDown`:

```
void CMouseView::OnLButtonDown(UINT nFlags, CPoint point)
{
//TODO: Add your message handler code here and/or call default
//Запоминаем координату X места, где была нажата левая кнопка мыши
  x=point.x;
```

```
//Запоминаем координату Y места, где была нажата левая кнопка мыши
  y=point.y;
//Получаем указатель на объект документа
  CMouseDoc* pDoc = GetDocument();
//Проверяем правильность указателя
  ASSERT_VALID(pDoc);
//Очищаем выводимую строку
  pDoc->StringData.Empty();
//Вызываем OnDraw
  Invalidate();
  CView::OnLButtonDown(nFlags, point);
}
```

В методе класса `OnDraw` сделаем изменения:

```
//Присваивание x  координаты конца
  CaretPosition.x=x+size.cx;
  CaretPosition.y=y;
  ……
  pDC->TextOut(x,y,pDoc->StringData);
```

Скомпилируем и запустим приложение.

3.3. Панель инструментов, меню, акселераторы

Рассмотрим работу с меню с помощью редактора ресурсов на примере создания простой программы.

3.3.1. Создание меню

1. Создадим SDI-программу с названием Menu.
2. Открываем вкладку Resource View, идем по дереву Menu→Menu.rc→Menu. Делаем двойной клик на вкладке IDR_MAINFRAME.
3. Вставим пункт меню: File→Type Hear. Вводим название Print Welcome.
4. Вызовем свойства пункта меню. Для этого сделаем клик правой кнопкой по имени и выберем пункт меню Properties.
5. Запишем в поле ID: `ID_FILE_PRINTWELCOME`.
6. Закроем редактор меню и запустим программу.

В меню File появилась команда, но она не работает (не подключена). Для того чтобы программа реагировала на выбор только что созданного пункта меню, свяжем с ним обработчик.

1. В окне утилиты `Class View` вызовем свойства класса `CMenuView`.
2. В окне свойств нажмем кнопку Events.
3. Выберем в списке идентификаторов `ID_FILE_PRINTWELCOME` и раскроем ветку.
4. Раскроем список напротив поля `COMMAND` и выберем `"Add" OnFilePrintwelcome`.

Создадим переменную типа `CString` и инициализируем в объекте документа. Добавим следующий код в прототип класса `CMenuDoc`.

```
public:
  virtual ~CMenuDoc();
  CString StringData;
```

Проинициализируем переменную в конструкторе:

```
  StringData="";
```

Напишем код обработчика `OnFilePrintwelcome`:

```
void CMenuView::OnFilePrintWelcome()
{
  CMenuDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  if(!pDoc)
     return;
  pDoc->StringData="Добро пожаловать в меню.";
  Invalidate();
}
```

Добавим следующую строку в метод `OnDraw`:

```
  pDC->TextOut(0,0,pDoc->StringData);
```

Усложним задачу. Создадим полномасштабную ветку меню с собственными подменю. Добавим в меню пункт Demo.

1. Войдем в редактор меню.
2. Выделим Edit→Вызовем контекстное меню→Выберем Insert New→ Введем имя Demo.
3. Введем пункты: Grayed, Checked, Submenus.

Если в начало ввести &, то первая буква меню будет подчеркнута. Пусть это буква D, тогда Alt+D - это клавишный ускоренный вызов. Добавим подменю в пункте Submenus. Выделим пункт Submenus. В раскрывшейся слева вкладке Type Hear введем имя первого элемента подменю. Добавим в подменю еще один пункт. Получим:

- `Sub1`
- `Sub2`

3.3.2. Создание акселератора

Акселератор — сочетание клавиш для выбора команды меню. Чтобы добавить акселератор, сделаем следующее:

1. Откроем папку Accelerator в редакторе ресурсов. Дважды кликнем IDR_MAINFRAME.
2. Выделим последнюю пустую строку.
3. В появившемся списке в поле ID выберем строку `ID_SUBMENUS_SUB1`.
4. Выберем строку Ctrl из списка в поле Modifiers.
5. Выберем строку `VK_F5` из списка в поле Key.
6. Выход.
7. Добавить в название клавиши Sub2\tCtrl+F5.

3.3.3. Добавление кнопок на панель инструментов

Создадим кнопку на панели для команды Sub1 (то же самое можно сделать и для команды Sub2):

1. Откроем папку Toolbar в редакторе ресурсов. Дважды кликнем `IDR_MAINFRAME`.
2. Выделим пустую кнопку и сделаем рисунок.
3. Вызовем свойства нарисованной кнопки. Выберем в поле ID имя идентификатора `ID_SUBMENUS_SUB1`.

Добавим код для пункта Sub1. Для этого вызовем свойства класса `CMenuView` из утилиты `Class View`. Перейдем к списку событий Events. Раскроем вкладку `ID_SUBMENUS_SUB1` и добавим обработчик.

```
void CMenuView::OnSubmenusSub1()
{
  CMenuDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  if(!pDoc)
       return;
  pDoc->StringData="Меню 1";
  Invalidate();
}
```

3.4. Диалоговые окна. Работа с элементами управления: кнопками, текстовыми полями

3.4.1. Модальное диалоговое окно

Создадим программу, которая будет выводить диалоговое окно в ответ на выбор пользователем пункта Show Dialogs в меню File. В диалоговом окне создадим текстовое поле и кнопку. При нажатии на кнопку в текстовое поле должна заноситься символьная строка. Символьную строку изменим и при выходе из диалогового окна в клиентской области выведем измененную символьную строку.

Создание диалогового окна

1. Создадим SDI-программу Dialog.
2. Включим в меню File команду Show Dialogs и свяжем с обработчиком.
3. Войдем в редактор ресурсов. Щелкнем правой кнопкой по папке Dialog, выберем Insert Dialog. Среда переведет нас в режим визуального редактирования диалогового окна.
4. Вставим два элемента: кнопку и текстовое поле.
5. Изменим надпись на кнопке через свойства Properties: "Print String".

Создадим класс диалогового окна на основе только что созданного ресурса:

1. Project→Add Class…
2. В списке Categories: выберем MFC. В списке Templates: выберем MFC Class. Нажмем Add.
3. В поле Base class: выберем CDialog. В поле Dialog ID выберем идентификатор только что созданного ресурса.
4. Введем имя класса: CDlg и нажмем Finish.

Связывание обработчиков с элементами диалоговых окон

При нажатии на кнопку в текстовом поле должна появиться строка "Текст в диалоговом окне". Свяжем сообщение о нажатии на кнопку с обработчиком этого сообщения:

1. Вызовем свойства класса CDlg через Class View.
2. Перейдем к списку событий Events.
3. Выберем поле `ID_BUTTON1` и добавим обработчик `OnBnClickedButton1`.

Связывание переменных с элементами диалоговых окон

Visual C++ позволяет связывать переменные класса с элементами диалоговых окон. Создадим переменную m_text для хранения строки текстового поля с помощью мастера:

1. Вызовем контекстное меню класса CDlg через Class View. Выберем пункт Add→Add Variable…
2. Установим флажок Control Variable. Выберем в поле `Control ID: IDC_EDIT1`, в поле `Category: Value, Variable type: CString, Variable name: m_text`.
3. Finish.

Обработчик `OnButton1`:

```
void CDlg::OnBnClickedButton1()
{
  m_text="Текст в диалоговом окне";
  UpdateData(false);
}
```

Обмен информацией между переменной и элементом `IDC_EDIT1` осуществляется в специальном методе, включенном в класс диалогового окна:

```
void CDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  DDX_Text(pDX, IDC_EDIT1, m_text);
}
```

Код этого метода генерируется мастером, изменять его вручную не рекомендуется. Вызов метода `UpdateData(BOOL)` с параметром `false` заносит в текстовое поле значение переменной `m_text`. Вызов с параметром true присваивает переменной `m_text` содержимое текстового поля.

Переопределение метода для кнопки Ok

Переопределение необходимо, если поле редактировалось и необходимо сохранить текущее содержимое текстового поля. Для того чтобы переопределить метод, вызовем свойства класса `CDlg`, перейдем к списку виртуальных функций, нажав кнопку Overrides. Выберем метод `OnOk()`.

```
void CDlg::OnOK()
{
  UpdateData(true);
  CDialog::OnOK();
}
```

Отображение диалогового окна

Чтобы класс вида мог работать с членами класса `CDlg`, необходимо включить в него `Dlg.h` — заголовочный файл класса `CDlg`:

```
#include "Dlg.h"
```

В методе `OnFileShowdialogs()` создадим новый объект класса `CDlg` и отобразим его, пользуясь методом `DoModal()`. Далее добавим в класс документа переменную `StringData` типа `CString` и присвоим ей значение переменной `m_text`. Выведем значение переменной `StringData` в клиентском окне в левом верхнем углу.

Обработчик `OnFileShowdialogs`:

```
void CDialogView::OnFileShowdialogs()
{
   CDlg dlg;
   int result=dlg.DoModal();
   if(result==IDOK)
   {
        CDialogDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        if(!pDoc)
             return;
        pDoc->StringData=dlg.m_text;
        Invalidate();
   }
}
```

Добавим в метод `OnDraw` строку:

```
  pDC->TextOut(0,0,pDoc->StringData);
```

Использование диалогового окна в качестве главного

Создадим диалоговое окно в качестве главного и выведем в нем, нажав на кнопку, текстовое сообщение. Создадим программу `BaseDialog`.

1. На первом шаге работы мастера Application Wizard выбрать Dialog based (базовым классом для новой программы станет класс `CDialog`).

2. Нажать Finish.
3. Перейти в редактор ресурсов, откроем ресурс диалогового окна для редактирования.
4. Поместить кнопку и текстовое поле.
5. Связать кнопку с обработчиком (двойной клик по OnButton1).

Создавать класс диалогового окна в этом случае не надо. Его уже автоматически создал Application Wizard. Рассмотрим общий способ работы с элементами управления (будем рассматривать их как объекты):

1. Вызовем контекстное меню текстового поля (щелчок правой кнопкой мыши по элементу управления в редакторе ресурсов).
2. Выберем пункт Add Variable…
3. Выберем необходимый тип и введем имя переменной.
4. Нажмем Finish.

Обработчик OnButton1:

```
void CBaseDialogDlg::OnButton1()
{
  m_edit.SetWindowText(Cstring("Диалоговое окно!"));
}
```

3.4.2. Совместное использование флажков и переключателей

Флажок — управляющий элемент, который позволяет выбрать один или несколько вариантов. **Переключатели** осуществляют выбор одного из нескольких элементов.

Флажки

Создадим диалоговое окно с тремя флажками и текстовым полем, в котором будет отображаться информация о выбранном флажке. Создадим программу Maps на основе диалогового окна (Dialog based). В редакторе диалоговых окон удалим надпись "TODO …", добавим три флажка и текстовое поле. Флажки надо выстроить в виде столбца. Текстовое поле расположим внизу. Названия флажков — Flag 1, Flag 2, Flag 3.

Выравнивание элементов в редакторе диалоговых окон

Нажать на клавишу Ctrl и, не отпуская ее, щелкнуть на каждом из флажков. Последний из выбранных флажков будет служить «эталоном» (он выделен синими маркерами).

- Выравнивание по горизонтали (по левому краю):
  Контекстное меню->Align Lefts.
- Выравнивание по вертикали (расстояния между флажками будут одинаковыми):
  Контекстное меню -> Align Tops.

Связывание флажков с кодом программы

С помощью Class View каждый флажок свяжем со своим обработчиком (OnCheck1(), OnCheck2(), OnCheck3()). Создадим переменную m_text и свяжем ее с текстовым полем.

```
void CMapsDlg::OnCheck1()
{
  m_text="Выбран 1 флажок";
  UpdateData(false);
}
void CMapsDlg::OnCheck2()
{
  m_text=" Выбран 2 флажок ";
  UpdateData(false);
}
void CMapsDlg::OnCheck3()
{
  m_text=" Выбран 3 флажок ";
  UpdateData(false);
}
```

Переключатели

Создать диалоговое окно с тремя переключателями и текстовым полем, в котором будет отображаться информация о выбранном переключателе. Создать программу Radios на основе диалогового окна (Dialog based). В редакторе диалоговых окон удалить надпись "TODO …" , добавить три переключателя и текстовое поле. Переключатели надо выстроить в виде столбца, текстовое поле разместить внизу. Названия переключателей: Radio 1, Radio 2, Radio 3.

```
void CRadiosDlg::OnRadio1()
{
  m_text="Выбран  1 переключатель";
  UpdateData(false);
}
void CRadiosDlg::OnRadio2()
{
  m_text=" Выбран  2 переключатель";
  UpdateData(false);
}
void CRadiosDlg::OnRadio3()
{
  m_text=" Выбран  3 переключатель";
  UpdateData(false);
}
```

Совместное использование флажков и переключателей

Создать диалоговое окно с тремя флажками, тремя переключателями и текстовым полем, в котором будет отображаться информация о выбранном варианте. Использовать групповые поля. **Групповые поля** предназначены для

группировки элементов — как визуальной, так и функциональной. Все переключатели внутри группового поля работают совместно.

Создать программу `Choice` на основе диалогового окна (Dialog based). В редакторе диалоговых окон удалить надпись "TODO …" и добавить две группы флажков и переключателей и одно текстовое поле. Названия переключателей: `Math`, `Gum`, `Inf`, `Bio`, флажки: `Math, Phys, Hist, Bio`.

Создать для каждого переключателя свой обработчик — `OnRadio1(),…, OnRadio4()`. Для текстового поля ввести переменную `m_text`. Свяжем с флажками переменные, которые будут изменять состояния флажков. Эти переменные будут представлять весь элемент. Зададим нужное состояние флагов в обработчиках для каждого переключателя.

```
void CChoiceDlg::OnRadio1()
{
  m_check1=true;
  m_check2=true;
  m_check3=false;
  m_check4=true;
  m_text="85 ball";
  UpdateData(false);
}
void CChoiceDlg::OnRadio2()
{
  m_check1=false;
  m_check2=true;
  m_check3=true;
  m_check4=true;
  m_text="70 ball";
  UpdateData(false);
}
void CChoiceDlg::OnRadio3()
{
  m_check1=true;
  m_check2=true;
  m_check3=true;
  m_check4=false;
  m_text="83 ball";
  UpdateData(false);
}
void CChoiceDlg::OnRadio4()
{
  m_check1=true;
  m_check2=true;
  m_check3=false;
  m_check4=true;
  m_text="70 ball";
  UpdateData(false);


}
```

3.5. Списки. Комбинированные поля и бегунки

Список — управляющий элемент, представляющий пользователю перечень из нескольких строк (выделить — 1 клик, выбрать — 2 клика.) **Комбинированное поле** — сочетание текстового поля, раскрывающегося списка и кнопки, с помощью которой пользователь открывает список. **Бегунок** — управляющий элемент. Чаще всего применяется для ввода числовых величин (например, интенсивности цвета).

3.5.1. Списки

Создадим список и выведем выбранную строку в текстовое поле. Напишем программу Lists на основе диалогового окна, добавим список, текстовое поле и 2 надписи («Дважды щелкните по строке» и «Вы выбрали»). Управляющие элементы можно отображать и в обычном окне (недиалоговом). В этом случае нельзя воспользоваться редактором WISIWYG.

Создание объекта для работы со списком

1. Вызовем контекстное меню списка в редакторе диалогового окна и выберем пункт Add.
2. Variable.
3. В поле Category: выберем значение Control.
4. `Variable name: m_list.`
5. Finish.

Инициализация данных в списке

Для инициализации данных используем метод `OnInitDialog()` и метод `AddString(…)` объекта `m_list`. Внесем список группы.

```
BOOL CListsDlg::OnInitDialog()
{
  CDialog::OnInitDialog();
  m_list.AddString("Васильев Н.И.");
  …
  m_list.AddString("Яковлева Е.А.");
  …
}
```

Если сортировка не нужна, то надо сделать клик правой кнопкой мыши в редакторе диалоговых окон и вызвать свойства списка Properties→Sort = false.

Обработка двойных кликов в списках

При двойном клике заносится текст выбранной строки в текстовое поле. Создадим переменную `m_text` типа `CString`, связанную с текстовым полем, и обработчик `OnDblclkList1()`. Для определения выбранной строки будем использовать метод `GetCurSel()`. Это метод класса `CListBox`. Он возвращает индекс строки, которую пользователь дважды кликнул левой кнопкой

мыши. Содержимое строки получим при помощи метода GetText(…) класса CListBox.

```
void CListsDlg::OnDblclkList1()
{
  m_list.GetText(m_list.GetCurSel(),m_text);
  UpdateData(false);
}
```

3.5.2. Комбинированные поля

Создадим программу Combos на основе диалогового окна. Создадим комбинированное поле и текстовое поле. Добавим следующие переменные:

- m_combo типа CComboBox, связанную с элементом управления «Комбинированное поле»;
- m_text типа CString, связанную с текстовым полем.

Комбинированные поля инициализируются в методе OnInitDialog():

```
BOOL CCombosDlg::OnInitDialogs()
{
  CDialog::OnInitDialog();
  m_combo.AddString("Васильев Н.И.");
  …
  m_combo.AddString("Яковлева Е.А.");
  m_combo.SetCurSel(0);  //Выбор первой строки списка
  …
}
```

Программа должна сообщать, какую строку списка выбрал пользователь. Комбинированное поле генерирует событие CBN_SELCHANGE. Добавим обработчик этого события в класс диалогового окна.

Properties (Класс диалогового окна) → Events → IDC_COMBO1 → CBN_SELCHANGE → OnSelchangeCombo1()

Обработчик OnSelchangeCombo1:

```
void CComboDlg::OnSelchangeCombo1()
{
  m_combo.GetLBText(m_combo.GetCurSel(),m_text);
  UpdateData(false);
}
```

3.5.3. Прокрутка и использование бегунков

Создадим программу Slider на базе диалогового окна. Добавим текстовое поле, две надписи («координата бегунка» и «движение»), бегунок (ему присвоится идентификатор IDC_SLIDER1) и связанную с ним переменную m_slider типа CSliderCtrl. При инициализации бегунка необходимо задать его интервал. Эта величина определяет возможные позиции бегунка от крайнего левого до крайнего правого положения. В примере бегунок принимает

значения от 1 до 100. Эти значения задаются методами SetRangeMin(…) и SetRangeMax(…) класса CSliderCtrl. Второй параметр этих методов показывает, нужно ли перерисовывать ползунок после изменения интервала. Передавая значение false, мы отказываемся от перерисовки.

```
BOOL  CSliderDlg::OnInitDialog()
{
  CDialog::OnInitDialog();
  m_slider.SetRangeMin(1, false);
  m_slider.SetRangeMax(100, false);
//Связываем переменную с содержимым текстового поля и
//присваиваем значение 1
  m_text="1";
  UpdateData(false);
  …
}
```

Обработка сообщений бегунка

При перемещении бегунка элемент посылает сообщение WM_HSCROLL. Добавим его обработчик. Нам необходимо перехватывать сообщения с кодом SB_THUMBPOSITION, посылаемые при перемещении бегунка. Выведем в текстовом поле его новую позицию. Создадим переменную m_text, связанную с содержимым текстового поля, и присвоим ей значение параметра nPos обработчика OnHScroll(). Объект m_text относится к классу CString, а параметр nPos имеет целый тип. Для представления целого числа в виде текстовой строки воспользуется методом Format() класса CString.

```
void CSliderDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar*
pScrollBar)
{
  if(nSBCode == SB_THUMBPOSITION)
  {
    m_text.Format("%ld",nPos);
    UpdateData(false);
  }
  else CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

3.6. Сериализация. Работа с файлами

3.6.1. Сериализация объектов класса CString

Сериализация — процесс записи (чтения) объектов и данных на диск (с диска). Рассмотрим сериализацию как встроенных классов Visual C++ (например, CString), так и нестандартных. Если в программе отсутствует документ, на который можно возложить выполнение файловых операций (программы на базе диалоговых окон), то работают с классом MFC CFile. Рассмотрим программу Writer, которая будет записывать на диск введенную строку, а затем, по требованию пользователя, загружать ее из файла.

1. Создадим SDI программу Writer.
2. Создадим объект CString StringData и инициализируем ее.
3. Создадим обаботчик WM_CHAR OnChar().

```
void CWriterView::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
  //TODO:
  CWriterDoc *pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  pDoc->StringData += char(nChar);
  Invalidate();
}
```

Отобразим данные в OnDraw:

```
pDC->TextOut(0,0,pDoc->StringData);
```

Класс документа (файл WriterDoc.cpp) содержит встроенный метод `Serialize(CArchive& ar)`. В этом методе происходит сериализация объекта `StringData`. Методу `Serialize(CArchive& ar)` передается ссылка на объект `ar` класса `CArchive`. Работа с объектом `ar` практически не отличается от работы с потоками `cout` и `cin`.

```
if(ar.IsStoring()) ar<<StringData;
else ar>>StringData;
```

Чтобы сообщить приложению об изменении данных (заносим новый символ), вызовем в `OnChar(...)` метод объекта документа `SetModifiedFlag()`. Если будет сделана попытка выхода из программы без сохранения данных, то приложение выведет диалоговое окно с предложением сохранить данные.

Добавим строку в метод OnChar(...):

```
pDoc->SetModifiedFlag();
```

При создании нового документа следует стереть старое содержимое `StringData` и обновить вид программы новыми данными документа.

Метод OnNewDocument:

```
BOOL CWriterDoc::OnNewDocument()
{
  if (!CDocument::OnNewDocument())
    return FALSE;
  // TODO: add reinitialization code here
  // (SDI documents will reuse this document)
  StringData="";
  UpdateAllViews(NULL);
  return TRUE;
}
```

3.6.2. Сериализация нестандартных объектов

Создадим пользовательский класс и организуем сериализацию его объектов. Пусть класс содержит строковую переменную (типа `CString`). Кроме этого, в классе должны быть реализованы конструктор и три метода для работы со строками: `AddText()` — добавление текста в конец строки, `DrawText()` — вывод текста в констексте устройства и `ClearText()`— очистка (стирание содержимого) строки.

Алгоритм организации сериализации нестандартных объектов

1. Создать SDI-программу Serializer.
2. Добавить в проект заголовочный файл.
3. Включить в этот файл описание класса (наследуем его от CObject) — члены и методы класса.
4. Включить в заголовочный файл документа ссылку на новый файл.
5. Создать объект класса.
6. Использовать новый объект (его методы) в приложении.
7. Включить в определение класса макрос Visual C++ DECLARE_SERIAL, объявляющий методы, используемые в процессе сериализации.
8. Переопределить метод `Serialize()` класса `CObject.`
9. Для написания новой версии `Serialize()` добавить в проект новый файл *.cpp* и определить в нем метод `Serialize().`

Реализуем пользовательский класс `CData`, наследуемый от `CObject`. Для этого создадим заголовочный файл `Data.h` и включим его в проект.

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

Подключим `Data.h` в файл `SerializerDoc.h`, для этого добавим в начало файла строку:

```
#include "Data.h"
```

В классе документа введем переменную типа `CData`

```
…
public:
  CData DataObject;
…
```

Добавим в документ вида метод `OnChar(…)` и используем методы объекта `DataObject`.

```
void CSerializerView::OnChar(UINT nChar, UINT nRepCnt, UINT
nFlags)
{
 // TODO: Add your message handler code here and/or call default
  CSerializerDoc* pDoc=GetDocument();
  ASSERT_VALID(pDoc);
  if(!pDoc)
        return;
  pDoc->DataObject.AddText(CString(char(nChar)));
  Invalidate();
  CView::OnChar(nChar, nRepCnt, nFlags);
}
```

Метод OnDraw:

```
void CSerializerView::OnDraw(CDC* pDC)
{
  CSerializerDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  pDoc->DataObject.DrawText(pDC);
}
```

Добавим в проект новый файл Data.cpp и определим `Serialize(CArchive &archive)`.

```
#include "stdafx.h"
#include "SerializerDoc.h"

void CData::Serialize(CArchive& archive)
{
  // Вызов метода Serialize() базового класса (CObject)
  CObject::Serialize(archive);
  if(archive.IsStoring())archive<<data;
  else archive>>data;
}
IMPLEMENT_SERIAL(CData, CObject, 0)
```

Макрос `IMPLEMENT_SERIAL` содержит дополнительные методы, используемые Visual C++ для сериализации. Чтобы выполнить сериализацию для объекта `DataObject` класса `CData`, следует вызвать его метод `Serialize(…)` внутри метода `Serialize(…)` документа.

```
void CSerializerDoc::Serialize(CArchive& ar)
{
  DataObject.Serialize(ar);
  // if (ar.IsStoring())
  //{
    // TODO: add storing code here
  //}
  //else
  //{
    // TODO: add loading code here
//}
}
```

3.6.3. Работа с файлами. Класс CFile

Некоторые методы класса `CFile` перечислены в табл. 2. Режимы открытия файлов в конструкторе класса `CFile` приведены в табл. 3.

Таблица 2

Основные методы класса CFile

Метод	Назначение
Abort	Закрывает файл, игнорируя любые предупреждения и ошибки
Close	Закрывает файл и удаляет объект
GetLength	Получает длину файла
GetPosition	Получает текущую позицию файлового указателя
Open	Производит открытие файла с возможностью проверки ошибок
Read	Читает данные из файла с текущей позиции
Remove	Удаляет заданный файл
Rename	Переименовывает заданный файл
Seek	Перемещает файловый указатель в заданную позицию
SeekToBegin	Перемещает файловый указатель в начало файла
SeekToEnd	Перемещает файловый указатель в конец файла
SetLength	Изменяет длину файла
Write	Записывает данные в файл с текущей позиции

Таблица 3

Режимы открытия файлов в конструкторе класса CFile

Константа	Назначение
CFile::modeCreate	Создает новый файл
CFile::modeNoTruncate	Комбинируеся с modeCreate — если создаваемый файл уже существует, он не обрезается до нулевой длины
CFile::modeRead	Открывает файл только для чтения
CFile::modeReadWrite	Открывает файл для чтения/записи
CFile::modeWrite	Открывает файл только для записи
CFile::typeBinary	Устанавливает двоичный режим
CFile::typeText	Устанавливает текстовый режим со специальной обработкой пар символов конца/перевода строки

Задача

Написать программу, которая на базе диалогового окна при нажатии кнопки записывает в заданный файл на диске некоторый текст и считывает этот текст в тектовое поле в обратном порядке.

Создадим на базе диалогового окна программу `Filer`.

Создадим 4 текстовых записи. Длина каждой записи — 20 символов. Реализуем эти записи в программе как массив из 4 символьных строк `OutString`.

```
…
protected:
  HICON m_hIcon;
  char OutString[4][20];
…
```

Добавим в диалоговое окно необходимые управляющие элементы — два текстовых поля и кнопку. Для кнопки создадим функцию-обработчик. Создадим также две переменные (`m_text1` и `m_text2`) и свяжем их с текстовыми полями. Имя кнопки: Write and Read.

```
BOOL CFilerDlg::OnInitDialog()
{
  CDialog::OnInitDialog();
  strcpy(OutString[0], "Иллюстрация ");
  strcpy(OutString[1], "работы ");
  strcpy(OutString[2], "с ");
  strcpy(OutString[3], "файлами ");
  m_text1 = CString(OutString[0]) + CString(OutString[1]) +
CString(OutString[2]) + CString(OutString[3]);
  UpdateData(false);
}
```

Для записи данных в файл будем использовать класс `CFile`. При нажатии на кнопку будет создаваться и открываться для записи файл `data.txt`. Запись будем производить с помощью метода `Write(…)`, для чтения используем методы `Read(…)` и `Seek(…)` класса `CFile`.

```
void CFilerDlg::OnButton1()
{
  UpdateData(true);
  CFile OutFile("data.txt",CFile::modeCreate | CFile::modeWrite);
//Создание обекта
  OutFile.Write(m_text1.GetBuffer(),          m_text1.GetLength());
//Запись данных в файл
  OutFile.Close();  //Закрытие файла
  CFile InFile("data.txt",CFile::modeRead);
  ULONGLONG pos = InFile.SeekToEnd();
  while(pos != CFile::begin)
  {
    char buf;
    InFile.Seek(--pos,CFile::begin);
    int n = InFile.Read(&buf,1);
    m_text2+=CString(buf);
  }
  UpdateData(false);
  InFile.Close();
}
```

3.7. Графика. Работа с растровыми изображениями (фракталы)

Алгебраические фракталы

Создадим программу `Fractal` для построения алгебраических фракталов и рассмотрим на ее примере работу с растровыми изображениями на основе класса `CImage`. Алгебраические фракталы строятся на основе алгебраических формул, при этом по алгоритму производится последовательность преобразований исходных данных, и результаты очередного шага зависят от предыдущего. Будем строить фрактальное множество, заданное формулой:

$$z_{n+1} = z^{Zn}_{n} + c,$$

где z_n, c – комплексные числа, n – номер текущей итерации, $z_0 = 0$.

Для того чтобы получить изображение (фрактал), надо произвести определенное количество итераций для точки комплексной плоскости.

Алгоритм

1. Все точки последовательно перебираются и закрашиваются разными цветами в зависимости от номера итерации при выполнении следующего условия: модуль z_n больше или равен 2 и достигает заданного предела. Можно установить определенные цвета для закрашивания (в программе 16 цветов). Процесс продолжается до тех пор, пока не будут перебраны все точки.
2. Для того чтобы определить цвет каждого пикселя во множестве, необходимо выполнить ряд действий:
   - задать значение мнимой части параметра равным максимальному значению по мнимой оси, а значение действительной части параметра — равным минимальному значению по действительной оси;
   - выбрать текущий столбец пикселей;
   - для каждой строки текущего столбца пикселей увеличить счетчик итераций и проверить, не превышает ли модуль комплексного числа на данной итерации значение 2 и не достигнуто ли предельное количество итераций;
   - если хотя бы одно из этих условий не выполняется, нарисовать точку с текущими координатами, задаваемыми номерами столбца и строки, окрашенную цветом, зависящим от номера текущей итерации;
   - перейти к следующей точке столбца, уменьшив значение мнимой части;
   - перейти к следующему столбцу, увеличив значение действительной части.

Создание приложения

Напишем программу, реализующую данный алгоритм. Создадим SDI приложение без поддержки архитектуры «документ вид» (для этого необходимо снять соответствующий флажок во втором окне мастера MFC Application Wizard). Добавим в главное меню пункт Save для сохранения полученного фрактала. Объявление необходимых переменных в прототипе класса `CChildView` (заголовочный файл ChildView.h):

```
public:
  CImage imgOriginal;  //Объект класса CImage
  //Переменная для хранения размеров клиентской области
  CRect m_ClientRect;
  double m_LeftBound;
  double m_RightBound;
  double m_TopBound;
  double m_BottomBound;
  COLORREF m_ColorTable[16];  //Таблица цветов
  //Константа - максимальное число итерраций
  const unsigned int m_MaxIter;
```

Проведем необходимую инициализацию в конструкторе (исходный файл `ChildView.cpp`):

```
CChildView::CChildView(): m_MaxIter(256),
m_RightBound(0), m_LeftBound(-1.5),
m_TopBound(0.5), m_BottomBound(-0.5)
{
  //Создание таблицы цветов
  m_ColorTable[0] = RGB(128,0,0);
  m_ColorTable[1] = RGB(255,0,0);
  m_ColorTable[2] = RGB(255,128,128);
  m_ColorTable[3] = RGB(255,128,192);
  m_ColorTable[4] = RGB(255,0,255);
  m_ColorTable[5] = RGB(128,0,255);
  m_ColorTable[6] = RGB(0,0,255);
  m_ColorTable[7] = RGB(0,0,128);
  m_ColorTable[8] = RGB(0,128,128);
  m_ColorTable[9] = RGB(0,255,0);
  m_ColorTable[10] = RGB(128,255,128);
  m_ColorTable[11] = RGB(255,255,128);
  m_ColorTable[12] = RGB(255,255,0);
  m_ColorTable[13] = RGB(255,128,0);
  m_ColorTable[14] = RGB(255,128,64);
  m_ColorTable[15] = RGB(128,64,0);
}
```

Добавим в класс `CChildView` с помощью мастера обработчик изменения размеров окна `OnSize`.

```
void CChildView::OnSize(UINT nType, int cx, int cy)
{
  GetClientRect(&m_ClientRect);
}
```

Добавим в класс CChildView функцию для построения фрактала. В прототип класса (файл CChildView.h) добавим строку:

```
public:
  void BuildFractal();
```

Определим ее в файле ChildView.cpp Функция BuildFractal:

```
void CChildView::BuildFractal()
{
  imgOriginal.Destroy();
  imgOriginal.Create(m_ClientRect.right,m_ClientRect.bottom,24);
  double f_Dx = (m_RightBound - m_LeftBound)/m_ClientRect.right;
  double f_Dy = (m_TopBound - m_BottomBound)/m_ClientRect.bottom;
  Complex c0(m_LeftBound,m_TopBound);
  for(int i = 0; i < m_ClientRect.right; i++)
  {
  for(int j = 0; j < m_ClientRect.bottom; j++)
  {
    Complex z0(0,0);
    int n_Iter = 0;
    while(n_Iter < m_MaxIter && z0.Abs() < 4)
    {
      z0 = pow(z0,z0) + z0*z0*z0 + c0;
      n_Iter++;
    }
    imgOriginal.SetPixel(i,j,m_ColorTable[n_Iter%16]);
    c0 -= Complex(0,f_Dy);
  }
  c0 = Complex(c0.GetRe(),m_TopBound);
  c0 += f_Dx;
  }
}
```

В данной функции используются методы класса CImage:

- Destroy() — для удаления растрового изображения из объекта;
- Create(int nWidth, int nHeight, int nBPP) — для создания растрового изображения с размерами nWidth и nHeight и глубиной цвета nBPP;
- SetPixel(int x, int y, COLORREF color) — для закрашивания одного пикселя изображения.

Также используется класс Complex, его полный исходный код приведен в приложении. Чтобы его использовать, нужно добавить в программу с помощью утилиты Solution Explorer файлы Complex.h и Complex.cpp в проект и добавить в файл ChildView.cpp следующую строку:

```
#include "Complex.h"
```

Построение и вывод изображения на экран будем производить по двойному клику левой кнопкой мыши. Обработчик OnLButtonDblClick (файл CChildView.cpp):

```
void CChildView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
  BuildFractal();
  Invalidate();
  CWnd::OnLButtonDblClk(nFlags, point);
}
```

Обработчик OnPaint (файл CChildView.cpp):

```
void CChildView::OnPaint()
{
  CPaintDC dc(this); // device context for painting
  if(!imgOriginal.IsNull())
  imgOriginal.StretchBlt
(dc,0,0,imgOriginal.GetWidth(),imgOriginal.GetHeight(),SRCCOPY);
}
```

Функция `StretchBlt` копирует изображение объекта класса `CImage` в контекст устройства, переданный ей в качестве параметра. Чтобы сохранить полученное изображение, обработаем пункт меню Save. Обработчик `OnFileSave`:

```
void CChildView::OnFileSave()
{
  CString strFilter;
  HRESULT hResult;
//Строка с поддерживаемыми расширениями
  strFilter = "Bitmap image|*.bmp|JPEG image|*.jpg|GIF im-
age|*.gif|PNG image|*.png||";
//Создание стандартного диалога сохранения
  CFileDialog dlg(FALSE,NULL,NULL,OFN_HIDEREADONLY |
OFN_OVERWRITEPROMPT | OFN_EXPLORER,strFilter);
  hResult = (int)dlg.DoModal();  //Вызов диалога сохранения
  if (FAILED(hResult)) return;
  //Создание строки с путем и полным именем файла, в который будет
производиться сохранение
CString strFileName;
  CString strExtension;
  strFileName = dlg.m_ofn.lpstrFile;
  if (dlg.m_ofn.nFileExtension == 0)
  {
  switch (dlg.m_ofn.nFilterIndex)
  {
    case 1 : strExtension = "bmp"; break;
    case 2 : strExtension = "jpg"; break;
    case 3 : strExtension = "gif"; break;
    case 4 : strExtension = "png"; break;
    default : break;
  }
  strFileName = strFileName + '.' + strExtension;
```

```
}
//Сохранение
  hResult = imgOriginal.Save(strFileName);
  if (FAILED(hResult))
  {
    CString fmt;
    fmt.Format("Save image failed:\n%x", hResult);
    ::AfxMessageBox(fmt);
    return;
  }
}
```

Для сохранения изображения используется метод класса `CImage: Save`, которому необходимо передать строку с путем и именем файла. Для использования класса `CImage` и математических формул, необходимо добавить в файл `stdafx.h` строки:

```
#include <atlimage.h>
#include <cmath>
```

Программа готова. Скомпилируем и запустим. В результате должно получиться следующее изображение (рис. 2).

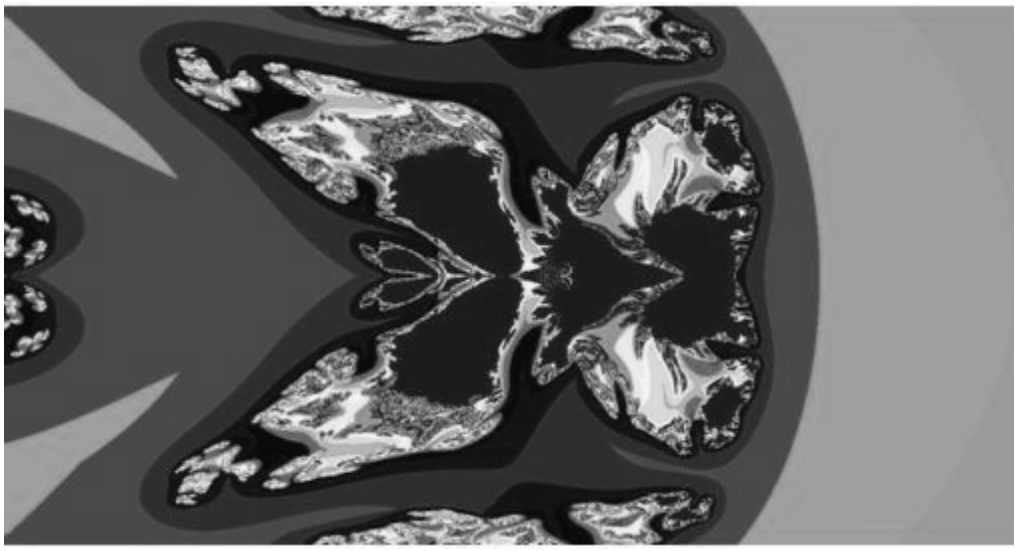

Рис. 2. Изображение после компиляции программы

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Давыдов В. Visual C++. Разработка Windows-приложений с помощью MFC и API-функций [Текст] / В. Давыдов. – Санкт-Петербург: BHV, 2008. – 567 с.
2. Довбуш Г., Хомоненко А. Visual C++ на примерах [Текст] / Г. Довбуш, А. Хомоненко. – Санкт-Петербург: BHV, 2007. – 528 с.
3. Рихтер Д., Назар К. Windows via C/C++. Программирование на языке Visual C++ [Текст] / Д. Рихтер, К. Назар. – Санкт-Петербург: Питер, 2009. – 896 с.

Учебное издание

РАЗРАБОТКА ПРОГРАММ В СРЕДЕ MS VISUAL STUDIO

Составитель Ольга Леонидовна Чагаева

Редактор Е. В. Рябая

Подписано в печать 12.01.2010 г. Формат 60x84 1/16.
Бумага 80 г/м ^{2}. Цифровая печать. Усл.п.л. 2,03.
Уч.-изд. л. 2,5. Тираж 100. Заказ

Редакционно-издательский отдел УГТУ – УПИ
620002, Екатеринбург, ул. Мира, 19
rio@fat.ustu.ru
Отпечатано в отделении полиграфии ИВТОБ
620002, Екатеринбург, ул. Мира, 19, ауд. И-120
Тел.: (375-41-43)
opivtob@mail.ustu.ru