

Лабораторная работа 1

Часть 1. Создание простейшего приложения с использованием библиотеки MFC в среде MS Visual C++

Цель работы:

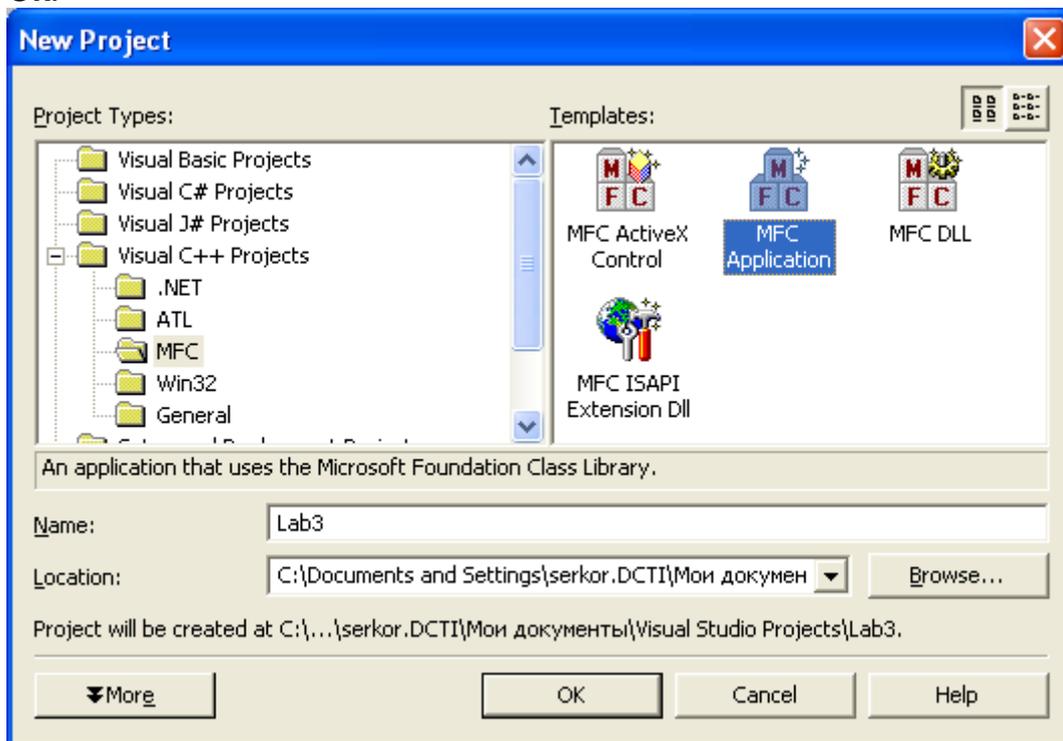
Познакомиться с работой мастера MFC ClassWizard, создать с его помощью простейшее приложение, основанное на диалоге, научиться добавлять в проект элементы управления (кнопка, окно редактирования, окно со списком).

Создание проекта

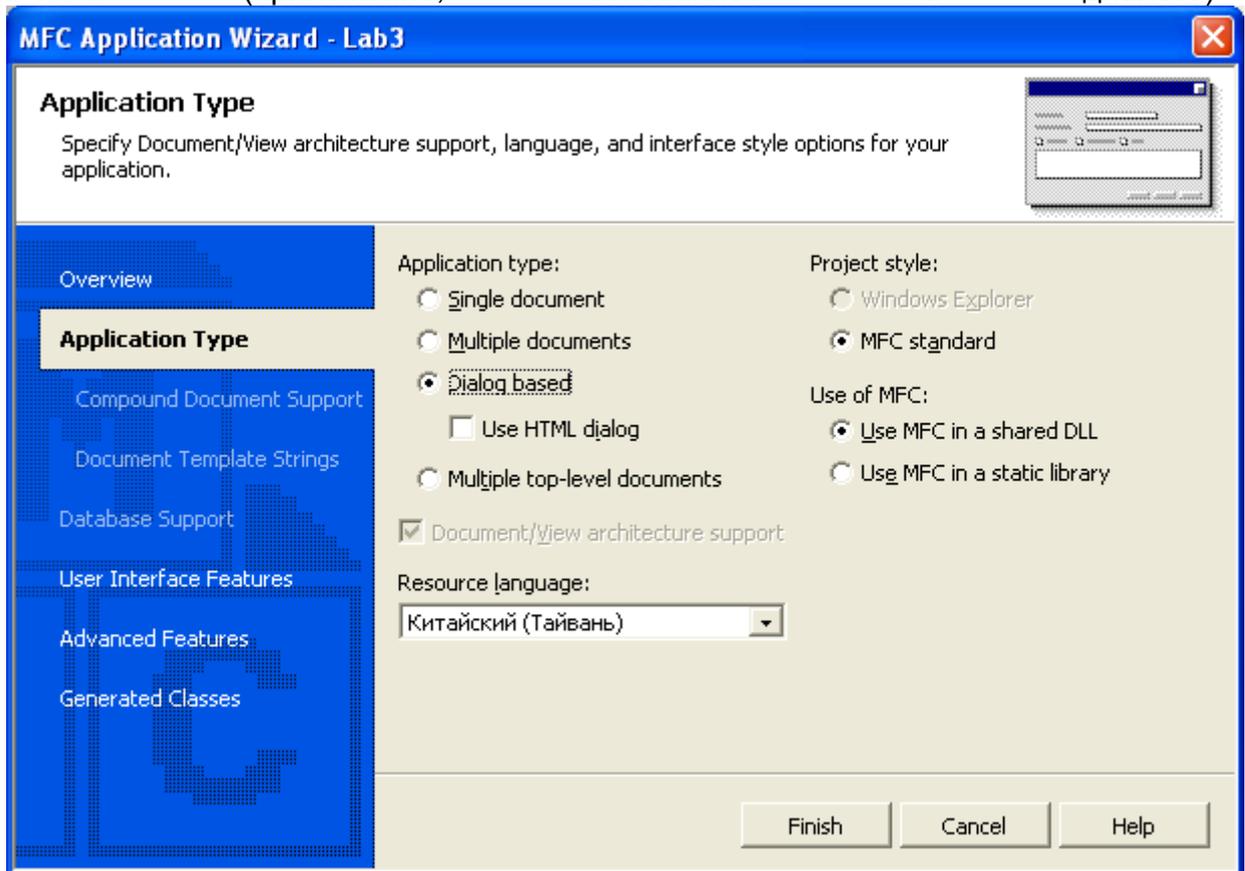
1. Выберите **File**→**New**

2. На закладке **Projects**

- Visual C++ Projects, MFC, MFC Application
- В поле **Name** введите требуемое имя проекта (в нашем случае **Lab1**). Это имя послужит названием для папки, которая будет создана в каталоге, определяемом значением поля **Location**, для хранения файлов проекта – **Ok**.



3. **Step1:** Раскройте пункт Application Type и выберите тип приложения - **Dialog based** (приложение, основанное на диалоге)



4. Если Вас всё устраивает (язык лучше установить английский), нажмите кнопку **Finish**.

Файлы проекта

- ⇒ **Lab3.vcproj** – этот файл содержит информацию о файлах проекта и используется для построения одного проекта, либо подпроекта.
- ⇒ **Lab3.sln** – файл решения (Solution), содержит информацию о проектах, включённых в рабочую область.
- ⇒ **Lab3.h** – Главный заголовочный файл проекта. Он включает другие заголовочные файлы, необходимые для генерации приложения, например, Resource.h. Так же в нем присутствует описание класса CLab1App – класса приложения.
- ⇒ **Lab3.cpp** – главный файл реализации. Содержит реализацию класса CLab1App.
- ⇒ **Lab1.rc** – список всех Microsoft Windows ресурсов, которые используются в программе. Он содержит иконки, bitmap'ы и курсоры, находящиеся в подкаталоге RES.
- ⇒ **Lab3.clw** – этот файл необходим ClassWizard'у для редактирования существующих или добавления новых классов, а так же для изменения карт сообщений и создания прототипов методов.
- ⇒ **res\Lab3.ico** – файл иконки, используемой как иконка приложения (левый верхний угол и TaskBar). Она включается главным файлом ресурсов Lab1.rc.
- ⇒ **res\Lab3.rc2** – в этом файле хранятся ресурсы, которые не могут быть напрямую отредактированы из среды Microsoft Visual C++.
- ⇒ **Lab3Dlg.h, Lab3Dlg.cpp** – файлы диалогового окна (далее *диалога*). Они содержат описание класса CLab1Dlg. Этот класс определяет основной диалог приложения. Шаблон этого диалога хранится в файле Lab1.rc.
- ⇒ **StdAfx.h, StdAfx.cpp** – файлы необходимые для генерации так называемых precompiled headers (PCH), а так же для подключения библиотек MFC.

⇒ **Resource.h** – Стандартный файл, содержащий константы - идентификаторы ресурсов проекта.

Редактирование проекта

Выбор того, что именно редактировать, осуществляется с помощью окон: **ClassView**, **ResourceView** и **Solution Explorer**. На закладке **ClassView** отображена иерархия классов проекта, на закладке **Solution Explorer** – иерархия файлов, а на **ResourceView** – отображены ресурсы проекта. Двойной щелчок на любом из пунктов – и вы уже редактируете выбранный объект.

Компиляция и запуск проекта

Перед компиляцией проекта выберите в меню **Build**→**Configuration Manager** конфигурацию проекта – Release (реализация) или Debug (отладка). Обычно компиляция происходит с помощью команды **Build**→**Build**, при этом компилируются только те файлы, которые еще не были откомпилированы и те, которые были отредактированы после предыдущей компиляции. Команда **Build**→**Rebuild Solution** перекомпилирует все файлы проекта, это бывает нужно, если файлы проекта были изменены вне VC.

Запуск приложения осуществляется с помощью команды **Debug**→**Start Without Debuggin**.

Попробуйте откомпилировать и запустить получившийся проект.

На данном этапе мы имеем практически пустое приложение, не несущее в себе никакой смысловой нагрузки. Теперь добавьте в окно диалога кнопку так, чтобы приложение реагировало на ее нажатие каким-либо сообщением.

Замечание: *теперь то и дело в тексте программы или в редакторе ресурсов вы увидите слова «TODO:» и какой-либо текст за ними. С помощью этого текста VC подсказывает вам, что нужно делать дальше.*

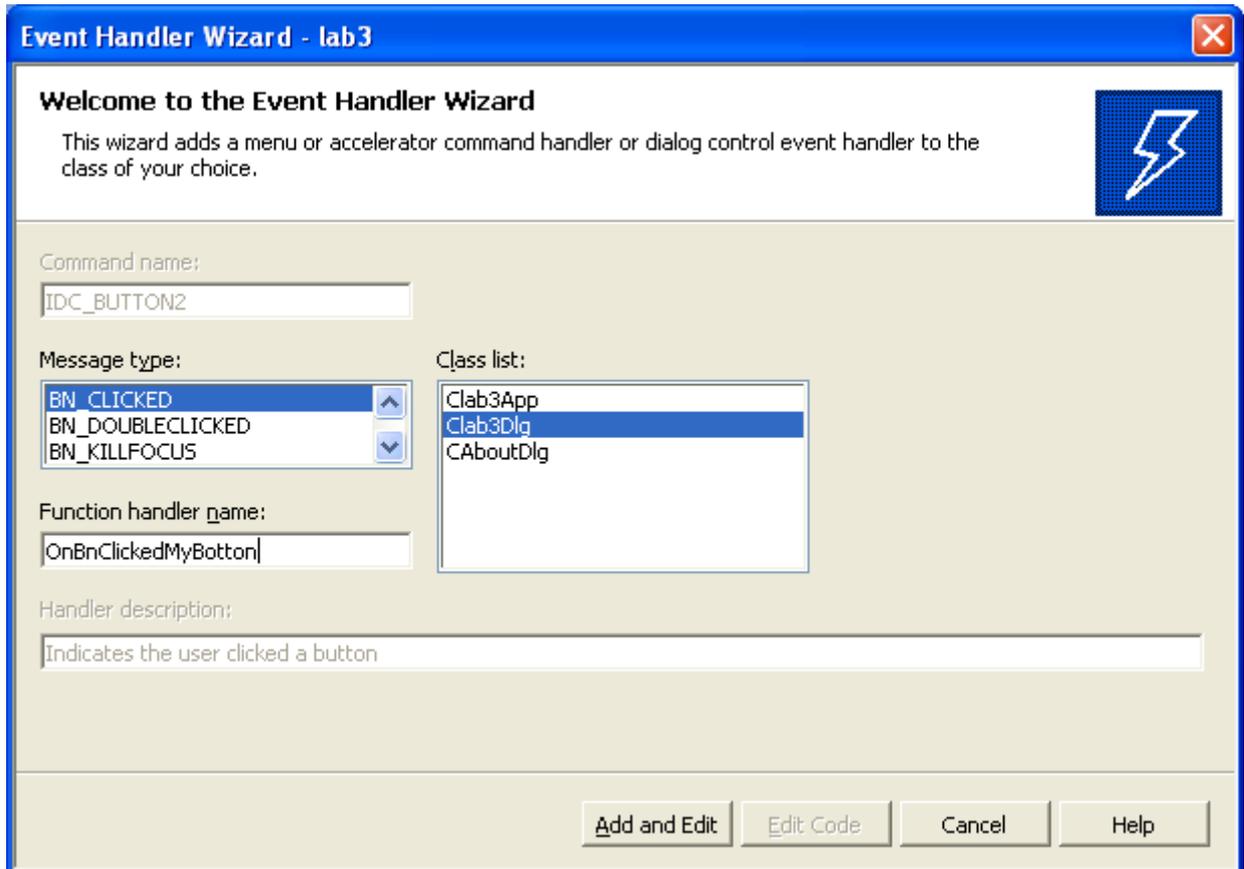
Создание кнопки

1. Активизируйте закладку Resource View.
2. Разверните ветвь Dialog, выделите элемент ветви – диалоговую панель IDD_LAB3_DIALOG и установите для неё русский язык (контекстное меню панели, пункт Properties)
3. Вызовите графический редактор для визуального редактирования диалоговой панели - сделайте двойной щелчок на строке IDD_LAB3_DIALOG .
4. В окне редактора удалите теперь нам ненужные
 - надпись *TODO: Place dialog controls here.*
 - кнопку Ok
 - кнопку Cancel
5. На панели **Tool Box** (Элементы управления) нажмите на иконку с изображением кнопки () и укажите место размещения кнопки в окне главного диалога. На окне диалога в этом месте должна появиться небольшая кнопка с текстом *Button1*.
6. Из контекстного меню этой кнопки выберите пункт *Properties*.
 - в поле Caption введите произвольное название, скажем, *Message*.
 - измените идентификатор с IDC_BUTTON1 на IDC_BUTTON_MSG. Конечно, ничего не будет страшного, если вы будете использовать идентификаторы, заданные по умолчанию, но намного лучше, если все названия в вашей программе будут говорящими. Однажды окажется, что, вернувшись позже к ра-

нее написанному материалу, вы потратите массу времени на то чтобы определить, что же такое IDC_BUTTON23 или IDC_EDIT18.

- Закройте окно свойств.

7. Из контекстного меню кнопки выберите пункт *Add Event Handler*.



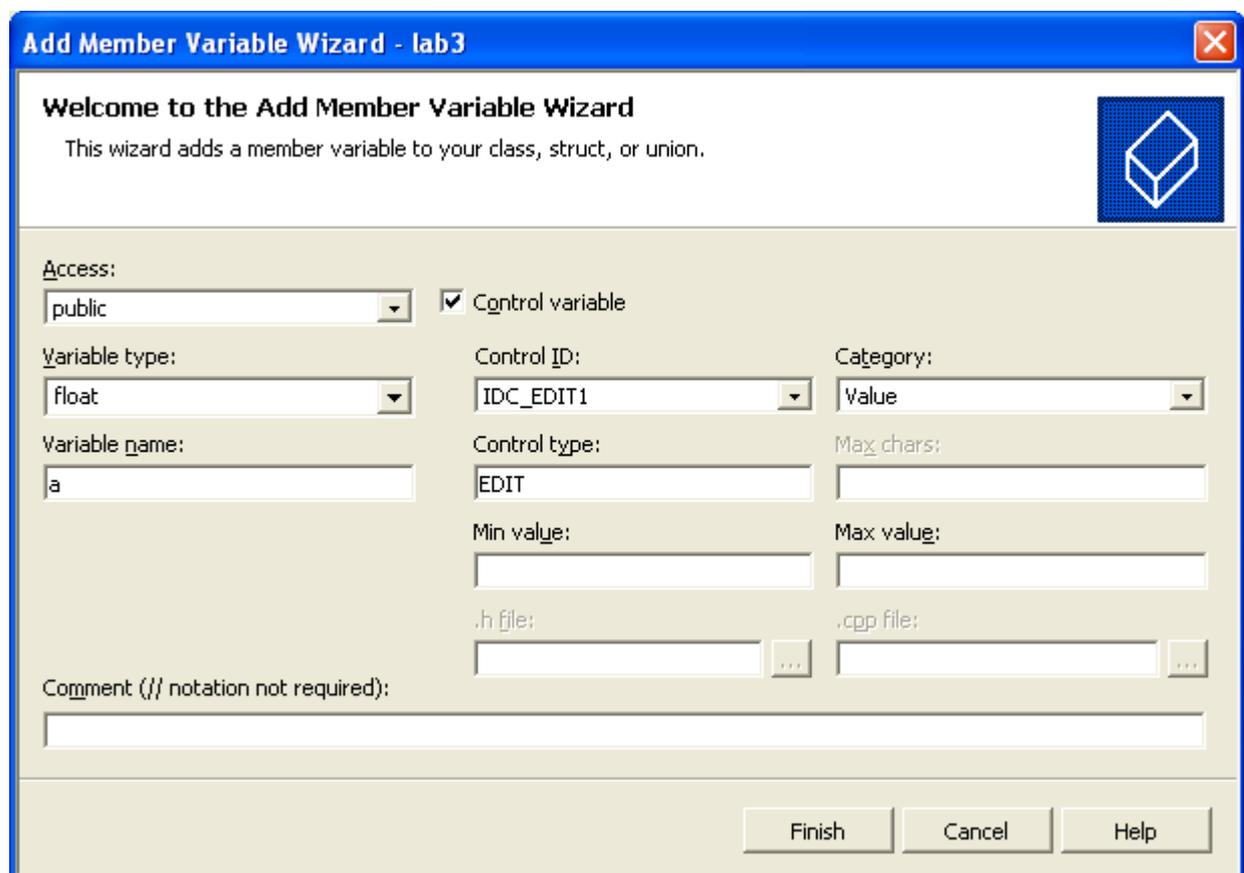
8. Если нужно изменить имя функции и нажмите *Add and Edit*.

9. Функция добавилась в класс *CLab3Dlg* (файл *Lab3Dlg.cpp*). Добавьте строку для вывода информации. Мы сделаем это с помощью функции *MessageBox*. Строка будет выглядеть так: `MessageBox("Hello, world!");` Понятно, что сообщение может быть любым.

Приложение готово. Запустите приложение. Согласитесь скомпилировать недостающие файлы, если предварительно проект вы не компилировали.

Добавление окна редактирования

1. Активизируйте закладку *ResourceView*.
2. Разверните ветвь *Dialog* и сделайте двойной щелчок на строке *IDD_LAB3_DIALOG*.
3. Аналогично пункту 5 **Создание кнопки**, разместите на диалоговой панели окно редактирования *Edit Box*. Через пункт меню *Properties*(свойства) определите имя (идентификатор, ID), присвоенное элементу; вероятно, оно будет *IDC_LAB3_EDIT*.
4. Введите в программу переменную, связанную с окном редактирования:
 - В контекстном меню поля ввода выберите *Add Variable...* (добавить переменную)



- в диалоговой панели *Add Member Variable* установите следующие значения: *Variable name* (имя переменной) – `m_Lab1Edit`, *Category* (категория) – `Value`, *Variable type* (тип переменной) – `CString` – `Ok`.
5. Добавьте кнопку, при нажатии на кнопку будет меняться регистр символов, введённых в строку редактирования. Установите надпись на кнопке – Регистр, идентификатор – `IDC_REG_BUTTON`.
 6. Аналогично п. 7 **Создание кнопки** создайте обработчик для сообщения `BN_CLICKED` :

```

UpdateData(TRUE);           // Обновление значения переменной,
                             // связанной с окном ввода

CString UpperValue;
UpperValue = m_Lab1Edit;
UpperValue.MakeUpper(); // Перевод символов строки в верхний
                        // регистр

m_Lab1Edit = UpperValue;
UpdateData(FALSE);         // Обновление информации в окне
                             // редактирования в соответствии
                             // с текущим значением связанной
                             // с окном переменной

```

Сохраните текст, откомпилируйте проект и выполните.

Справочный материал

Для чего нужен ClassWizard и как с ним работать

ClassWizard помогает обрабатывать сообщения Windows, посылаемые диалоговым окнам, элементам этих окон и т. п. Так же он дает возможность сопоставить элементам управления переменные. В рамках простого проекта его возможностей полностью хватает для реализации всех требований, предъявленных к программе. Однако не все сообщения можно обработать с помощью ClassWizard. В этом

случае нужно традиционно редактировать карту сообщений, как это делается в других средах разработки ПО.

Закладки ClassWizard, которые будут здесь рассмотрены, это *Message Maps* и *Member Variables*. Это основные страницы ClassWizard.

Message Maps

На этой странице производится добавление/удаление/редактирование функций-обработчиков сообщений. Это может быть реакция на щелчок мыши, ответ на ввод символа с клавиатуры, действие, выполняемое по таймеру и т. п.

В VC существует возможность редактировать два и более как независимых, так и связанных (ActiveX control и ActiveX container, например) проекта одновременно, поэтому для активизации одного из проектов существует выпадающий список (ComboBox) *Project*. Скорее всего, вы будете работать с одним проектом, и этот пункт вам не понадобится.

В поле *Class name* можно осуществить выбор класса, с функциями-обработчиками сообщений, которого вы хотите работать.

В списке *Object IDs* производится выбор элемента управления (по идентификатору), сообщения которого вы хотите обрабатывать.

В списке *Messages* отображаются сообщения, которые приходят к этому элементу. Набор этих сообщений различен, в зависимости от типа элемента.

В списке *Member functions* отображены все функции-обработчики сообщений класса, опять же, указанного в поле *Class name*.

Member Variables

На этой странице производится связывание переменных с элементами управления для класса проекта, указанных в самом начале страницы.

Переменные могут быть двух категорий: *элемент управления* и *значение*. Первая категория присутствует для всех видов элементов управления – в случае ее выбора, ассоциированная переменная будет объектом класса, к которому принадлежит элемент управления. Например, CStatic. Для второй категории все складывается немножко иначе. Для разных видов элементов, количество возможных типов, к которым может принадлежать ассоциированная переменная различно. Их может и не быть вообще (CButton), а может быть довольно много (CEdit).

Время, с которого можно использовать ассоциированную переменную, зависит от ее категории – переменную категории *значение* можно изменять уже в конструкторе, а категорию *элемент управления* только после того как с ней ассоциировано какое-нибудь окно (любой элемент управления это тоже окно).

Следует помнить, что для переменной категории *значение* необходимо вызывать функцию UpdateData для обновления связи. Если элемент управления должен быть установлен в значение, хранящееся в ассоциированной переменной, то эта функция должна быть вызвана с параметром **false**, в противном случае – с параметром **true**. Обычно, эта функция вызывается в начале и/или конце обработчика сообщения.

Как создаются обработчики событий (сообщений)

Один из таких обработчиков мы уже создали – это функция реакции на нажатие кнопки. По сути, обработчик события - это функция, вызываемая каждый раз, когда что-либо происходит. Список событий, на которые может реагировать элемент управления, можно посмотреть с помощью ClassWizard или выбрав пункт *Events...* из контекстного меню.

Класс CString

Класс CString состоит из последовательности символов переменной длины и предоставляет операторы и функции для преобразования строк. Объект CString представляет собой именно строку, а не указатель на строку.

Создание строки:

```
CString s1;  
CString s2("Строка");  
CString s3 = s2; //Копирование строки  
CString s4 = "Ещё строка";
```

Конкатенация строк:

```
CString s1="Строка ";  
CString s2 = "символов";  
CString s3 = s1 + s2;
```

Методы класса CString

GetLength	Возвращает число символов, хранимых в объекте (считая, что символ занимает 8 бит)
GetBuffer	Возвращает указатель на буфер объекта, что позволяет использовать его в обычных функциях работы со строками
ReleaseBuffer	Освобождает буфер, распределённый методом GetBuffer
Compare	Сравнивает две строки без учёта регистра букв
CompareNoCase	Сравнивает две строки с учётом регистра букв
GetAt	Возвращает символ в заданной позиции
SetAt	Устанавливает символ в заданную позицию
Find	Находит подстроку и возвращает индекс её первого символа
Format	Преобразует данные других типов в текст, подобно функции sprintf()
MakeLower	Преобразует все символы в строчные
MakeUpper	Преобразует все символы в прописные
MakeReverse	Обращает порядок символов в строке
Left	Выделяет заданное количество самых левых символов в строке
Right	Выделяет заданное количество самых правых символов в строке
Insert	Вставляет подстроку в строку
Delete	Удаляет указанные символы из строки

Элемент управления ListBox.

Список является одним из наиболее распространенных элементов управления. Окно со списком или список (List Box) – это модификатор информации, отображает информацию и одновременно служит для выбора информации. Для создания такого элемента в программе используется класс библиотеки MFC CListBox. Каж-

дый элемент списка – строка типа CString. Каждая строка имеет порядковый номер (индекс), нумерация строк начинается с нуля.

Программирование элемента ListBox:

Поместите элемент ListBox на диалоговую панель и настройте его свойства (идентификатор, стили, обратите внимание на то, установлена или нет сортировка элементов списка);

Свяжите с элементом, используя MFC ClassWizard, переменную типа Control (Элемент управления);

Подготовьте код программы, в котором будет использоваться список, для этого используйте методы класса CListBox.

Методы класса CListBox

AddString (CString)	добавление строки в окно ListBox
DeleteString (int)	удаление строки с заданным номером из списка
GetCurSel ()	возвращает номер выделенной строки списка
GetCount ()	возвращает количество записей в списке
GetText (int,CString)	в переменную типа CString записывает значение элемента списка с заданным номером (параметр типа int)
ResetContent ()	очистка окна списка

Пример функции, обрабатывающей нажатие кнопки. Функция добавляет новую строку в список:

```
void CLab1Dlg::OnListAdd()
{
    m_ListBoxVariable.AddString("Новая строка");
}
```

Функции *CListBox* работают с объектами *CListBox*. Значит, необходимо получить указатель на объект списка, что делается с помощью функции *GetDlgItem()*, являющейся членом класса *CWnd*:

```
CWnd *CWnd::GetDlgItem(int ItemIdentifier) const;
```

Эта функция возвращает указатель на объект, чей идентификатор передан как параметр. Если такой объект не существует, то возвращается 0. Необходимо понимать, что указатель, возвращаемый функцией, является временным. Он действителен только в пределах текущего обработчика.

Значение, возвращенное функцией, должно быть приведено к типу указателя на конкретный класс управления.

Задания для самостоятельной работы

Создать приложение, основанное на диалоге, выполняющее следующие функции:

- Ввод в окно редактирования строки (строк),
- Обработку строк(и) в соответствии с заданным вариантом,
- Вывод на экран результата через окно редактирования и окно ListBox.
- Очистка окна ListBox.

Для выполнения задания используйте объект класса CString.

Варианты заданий

№ варианта	Задание
1	Ввод строки и вывод на экран длины строки и 3-го символа

2	Ввод строки, добавление в начале строки символа 0 (ноль), в конце строки – первого символа исходной строки
3	Ввод строки, перевод её в нижний регистр, добавление новой строки к старой и вывод результата на экран
4	Ввод строки, удаление первого символа, вывод на экран получившейся строки и её длины
5	Ввод строки, замену первого символа на последний и последнего на первый
6	Ввод двух строк, вычисление суммы длин этих строк и вывод на экран
7	Ввод двух строк, определение разности длин этих строк и добавление этого значения (в виде подстроки) к первой строке
8	Ввод двух строк, объединение этих строк в одну, вывод на экран получившейся строки и её длины
9	Ввод двух строк, создание третьей строки, состоящей из первых символов введенных строк и вывод её на экран
10	Ввод двух строк, вставка второй строки в первую, начиная с третьего символа, вывод на экран получившейся строки и её длины
11	Ввод строки, ввод номера позиции в строке, вставка первого символа в строке в заданную позицию
12	Ввод двух строк, обмен первыми символами между строками, вывод на экран получившихся строк

Часть 2. Элементы управления MFC

Цель работы:

Познакомиться со стандартными диалоговыми окнами открытия и сохранения файла, с классами CFile и CFileDialog, элементами управления Group и Radio-Button, научиться работать с ресурсом MENU.

Задание

Разработать приложение с использованием библиотеки MFC, выполняющее следующие действия:

Добавление элементов в список через окно редактирования или с помощью радиокнопок,

Удаление произвольного элемента из списка,

Очистка окна списка,

Сохранение элементов списка в текстовом файле с произвольным именем,

Чтение строк из текстового файла в окно списка,

Требования к программе:

- Команды выполняются с помощью кнопок и/или меню.
- Диалоговое окно должно содержать следующие обязательные элементы: окно **ListBox**, окно редактирования, кнопки, радиокнопки, меню.
- Меню должно включать команды работы со списком, с файлами, команду **О программе...** и команду **Выход**.
- Сохранение элементов окна ListBox и чтение информации из файла выполнять по строкам. Предусмотреть возможность чтения/записи пустых строк, а также корректную обработку последней строки текстового файла, в которой может отсутствовать символ перевода строки.
- Для задания имени и места размещения файла использовать стандартные диалоговые панели CFileDialog.

Справочный материал

Создание меню

1. Добавить новый ресурс в окне Resource View – меню (ID – IDR_MENU1)
2. Открыть редактор меню двойным щелчком мыши и добавить в меню пункты (Файл, Справка) и подпункты (Выход, О программе...). Каждый подпункт получает идентификатор (ID_EXIT, ID_ABOUT)
3. Связать меню с классом диалогового окна через ClassWizard
4. Связать каждый конечный пункт меню с кодом программы через сообщение COMMAND:

```
ClassWizard⇒(ID_Object)ID_EXIT⇒(Message)COMMAND⇒(function)OnExit⇒  
добавить код: OnOK();
```

```
ClassWizard⇒(ID_Object)ID_ABOUT⇒(Message)COMMAND⇒(function)OnAbout⇒  
добавить код: CAboutDlg dlg;  
dlg.DoModal(); //Открытие модального окна About
```

5. Связать меню с диалоговой панелью: в окне Dialog Properties ввести в окно Menu идентификатор меню – IDR_MENU1

Класс CFile -

предназначен для работы с файловыми объектами

Создание объекта:

CFile MyFile;

Методы класса CFile:

Открытие файла:

Open(<имя файла>, <флаги открытия>)

Флаги открытия:

modeCreate – создать новый,

modeWrite – открыть существующий для записи,

modeRead - открыть существующий для чтения (см. примеры)

Запись строки указанного размера в текстовый файл:

Write(<строка>, <длина строки>)

Чтение заданного количества байт информации (параметр int) с диска в буфер; возвращает количество прочитанных байтов:

Read(<адрес буфера>, int)

Закрытие файла:

Close()

Пример работы с файлами:

Добавим в проект Lab1, использующий список ListBox, возможность записи и чтения данных, хранящихся в ListBox'е (переменная, связанная с окном списка – m_evList) . Для этого создадим две кнопки - *Save* и *Load*. Вот текст обработчиков нажатия на них:

```
void CLab1Dlg::OnButtonSave()
{
    CFile theFile;
    theFile.Open("ListBox.txt", CFile::modeCreate|CFile::modeWrite);

    CString textBuffer;

    for(int i = 0; i < m_evList.GetCount(); i++)
    {
        m_evList.GetText(i, textBuffer);
        textBuffer += '\n'; // Перевод строки
        theFile.Write(textBuffer, textBuffer.GetLength());
    }

    theFile.Close();
}
```

Здесь мы последовательно, строчка за строчкой, сохраняли данные из ListBox'а. Нужно помнить, что ListBox'у не нужны символы перевода каретки, в то время как в текстовом файле без них не обойтись.

```
void CLab1Dlg::OnButtonLoad()
{
    CFile theFile;
    if(!theFile.Open("ListBox.txt", CFile::modeRead)
    {
        MessageBox("File does not exists!");
        return;
    }

    m_evList.ResetContent(); // Очистка списка

    char currentChar;
    CString textBuffer;
    BOOL endOfFile = false;

    while(!endOfFile)
    {
        textBuffer = "";
        do
        {
```

```

        if(theFile.Read(&currentChar, 1) != 1) // EOF
        {
            endOfFile = true;
            break;
        }
        textBuffer += currentChar;
    }
    while(currentChar != '\n'); // Чтение до конца строки

    if(!endOfFile)
    {
        //Удаление символа '\n'
        textBuffer.Delete(textBuffer.GetLength()-1);
        m_evList.AddString(textBuffer);
    }
}
theFile.Close();
}

```

Здесь мы последовательно читаем символ за символом и анализируем их на наличие символа перевода каретки. Когда наступит конец файла, количество символов, прочитанных методом **Read**, будет равно нулю.

Класс CFileDialog -

предназначен для создания стандартных диалоговых панелей чтения или сохранения файлов.

Создание объекта:

CFileDialog SaveDlg(FALSE, "txt", FileName);

где первый параметр определяет назначение диалога: FALSE – для сохранения файла, TRUE – для открытия файла;
 второй параметр определяет тип файла по умолчанию;
 третий параметр задаёт строковую переменную, содержащую имя файла по умолчанию.

Методы класса CFileDialog:

Открытие окна диалога (модального): **DoModal()**

Метод возвращает значения IDOK или IDCANCEL в зависимости от того, какие кнопки были нажаты при закрытии диалоговой панели.

Получение полного имени выбранного файла (путь+имя+расширение):

GetPathName()

Получение имени и расширения выбранного файла:

GetFileName()

Получение имени выбранного файла без расширения:

GetFileTitle()

Получение расширения (типа) выбранного файла:

GetFileExt()

Пример использования стандартного диалога для сохранения файла:

```

CFile MyFile;
CString FileName = "op.txt";
CFileDialog SaveDlg(FALSE, "txt", FileName);
if (SaveDlg.DoModal() == IDOK)
{
    FileName = SaveDlg.GetPathName(); //Новое имя файла
    MyFile.Open(FileName, CFile::modeCreate|CFile::modeWrite);
        // Сохранение файла
    MyFile.Close();
}

```

Установка начального значения элементу управления

10. Открыть окно MFC ClassWizard , вкладка Message Maps;
11. Выбрать в окне Object IDs объект - диалоговую панель, содержащую элементы управления, CDialog;
12. Выбрать в окне Messages сообщение WM_INITDIALOG;
13. Щёлкнуть на кнопке Edit Code
14. Отредактировать функцию OnInitDialog:

```
BOOL CDialog::OnInitDialog()  
{  
    CDialog::OnInitDialog();  
  
    .....  
  
    // TODO: Add extra initialization here  
    m_TextEdit = "Начальный текст"; // Установка начального значения  
                                     // переменной, связанной с окном редактирования  
    UpdateData (FALSE);              // Обновление элемента управления на  
экране  
  
    return TRUE;  
}
```

Элемент управления CheckBox

флажок, предоставляет пользователю возможность выбора варианта вычислений, имеет два состояния: TRUE или FALSE (тип значения – BOOL).

Программирование элемента:

1. Установить элемент CheckBox в диалоговой панели;
2. Настроить свойства элемента;
3. Установить с помощью MFC ClassWizard переменную, связанную с элементом, категория переменной – значение (value);
4. Инициализировать элемент при создании диалогового окна в методе OnInitDialog: установить переменной элемента управления значение TRUE или FALSE:

m_CheckBox = TRUE;

Элемент управления GroupBox

группа элементов, предназначен для группировки других элементов управления, например, радиокнопок.

Элемент управления RadioButton

радиокнопка, или кнопка-переключатель, обычно используется в группе радиокнопок для выбора одного из нескольких исключаящих друг друга вариантов.

Программирование элемента:

1. Поместить в диалоговую панель элемент GroupBox;
2. Поместить в GroupBox несколько радиокнопок;
3. Настроить свойства радиокнопок: идентификаторы IDC_RADIO1, IDC_RADIO2 и т.д.;
4. подписи;
5. Для первой радиокнопки в окне Свойства установить флажок «Группа»;
6. Установить с помощью MFC ClassWizard переменную, связанную с элементом, категория переменной – значение (value), тип – int, при этом все кнопки будут иметь номера 0, 1, 2 и т.д.
7. Инициализировать элемент при создании диалогового окна в методе OnInitDialog: установить переменной элемента управления значение TRUE или FALSE:

m_RadioButton = 0;

Варианты самостоятельного задания

№	Содержание
1	В зависимости от значения флажка вводить строки в первое или второе окно со списком, перемещать строки из первого окна во второе и обратно. Сохранить в файле содержимое второго окна со списком. Для ввода списка из файла использовать первое окно.
2	Вводить строки в первое окно со списком, перемещать строки из первого окна во второе и обратно. Сохранять в файле содержимое первого или второго окна в зависимости от значения флажка. Для ввода списка из файла также использовать окно по выбору.
3	В зависимости от значения радиокнопок вводить строки в первое, второе или третье окно со списком. Сохранить в файле элементы третьего окна. Для ввода списка из файла использовать первое окно.
4	Вводить строки в первое окно со списком, копировать строки из первого окна во второе или в третье. Сохранять в файле содержимое первого, второго или третьего окна в зависимости от значения радиокнопок. Для ввода списка из файла использовать первое окно.

Лабораторная работа № 2

Создание приложения, использующего графические средства библиотеки MFC в среде MS Visual C++

Цель работы:

Познакомиться с классами CPaintDC, CDC, CPen, CBrush, CColorDialog и выполнением графических операций в диалоговой панели.

Задание:

Разработать программу рисования графика функции $f(x)$ на заданном интервале значений x .

Требования к программе:

Ввод данных организовать через специальное окно диалога;

Вывод графика выполнить в главной диалоговой панели;

График должен иметь следующие элементы оформления: оси, обозначения осей, линии сетки, метки значений, заголовков.

Программа должна обеспечивать выбор цвета для рисования графика (цвет линии и цвет фона).

Основа проекта – диалоговое приложение с использованием MFC.

Используемые средства:

5. Контекст устройства для рисования CPaintDC;
6. Средства рисования CPen, CBrush;
7. Графические примитивы (точки, линии, прямоугольники);
8. Стандартный диалог выбора цвета CColorDialog;

Справочный материал

Для отображения текста или графики в окне необходим объект контекста устройства, соответствующий окну или устройству вывода данных. При рисовании этот объект сохраняет выбранные средства и установленные атрибуты и предоставляет функции-члены для рисования точек, линий, прямоугольников и других фигур. Для выполнения графических операций используются объекты классов CDC (базовый для контекстов графических устройств), CWindowDC (используется для рисования в пределах всего окна, включая заголовок), CClientDC (используется для рисования только в клиентской области окна), CPaintDC (используется в обработчике сообщения WM_PAINT).

Перерисовка окна диалога производится при передаче ему сообщения WM_PAINT. Это сообщение передаётся окну системой при изменении его размеров, положения, при перекрытии окна другими окнами. Для обработки сообщения WM_PAINT с состав класса окна диалога включён метод OnPaint(), который вызывается при получении окном сообщения WM_PAINT и выполняет перерисовку окна. При этом для создания контекста экрана используется объект класса CPaintDC.

```
void CMyDialog::OnPaint()
{
    //Создание контекста для рисования в окне диалога
    CPaintDC PaintDC(this);
    //отобразите графику, используя 'PaintDC'
    . . .
}
```

Если сообщение WM_PAINT используется для вывода результатов в окно, необходим способ принудительной отправки этого сообщения из программы. Это осуществляется с помощью методов Invalidate или InvalidateRect класса CWnd, которые позволяют объявить соответственно клиентскую область окна или произвольную прямоугольную область окна поврежденными, т.е. требующими перерисовки, и послать сообщение WM_PAINT в очередь приложения. В клиентскую область окна входит вся поверхность окна, за исключением рамки и заголовка. Параметры (границы) клиентской области окна могут быть получены функцией GetClientRect класса CWnd:

```
CRect MyRect;
GetClientRect(&MyRect);
```

Параметры (границы) всего окна могут быть получены функцией `GetWindowRect` класса `CWnd`:

```
GetWindowRect(&MyRect);
```

Метод `Invalidate` перерисовывает всё окно и может вызываться без параметров

Метод `InvalidateRect` имеет следующий прототип:

```
void CWnd::InvalidateRect(LPRECT pRect, BOOL EraseBackground = TRUE);
```

Параметр `pRect` задает область, которую нужно перерисовать. Значение этого параметра, равное `NULL`, соответствует клиентской области окна. Параметр `EraseBackground` определяет, нужно ли перед отправкой сообщения закрасить окно цветом фона по умолчанию или оставить фон окна без изменения.

Графические операции в контексте `CPaintDC`:

Вывод строки:

```
void CMainWnd::OnPaint()
{
    CPaintDC dc(this);
    dc.TextOut( 200, 200,"TextOut Samples");
}
```

Вывод точки:

```
void CMainWnd::OnPaint()
{
    CPaintDC dc(this);
    dc.SetPixel(500,200,RGB(255,0,0));
}
```

Рисование дуги окружности:

```
void CMainWnd::OnPaint()
{
    CPaintDC dc(this);
    dc.Arc(200,200,100,100,400,400,10,10);
}
```

Замкнутая дуга:

```
void CMainWnd::OnPaint()
{
    CPaintDC dc(this);
    dc.Chord(250,250,100,100,400,400,10,10);
}
```

Эллипс:

```
void CMainWnd::OnPaint()
{
    CPaintDC dc(this);
    dc.Ellipse(450,450,50,150);
}
```

Линия:

```
void CMainWnd::OnPaint()
{
    CPaintDC dc(this);
    dc.MoveTo(200,200); // Установка курсора в заданную точку окна
    dc.LineTo(100,100); // Рисование линии от текущей позиции курсора до заданной точки
}
```

Пример работы с графикой

- Создаем новый проект с помощью MFC AppWizard (.exe), укажем имя проекта MiniDraw; тип создаваемого приложения Dialog Based.
- Переходим на закладку ResourceView и разворачиваем дерево MiniDraw Resources -> Dialogs
- Выбираем форму с идентификатором IDD_MINIDRAW_DIALOG. Помещаем на форму элемент управления GroupBox и задаем его идентификатор IDC_SAMPLE.
- В редакторе ресурсов выбираем из всплывающего меню пункт Insert Dialog. Получаем новую форму с идентификатором IDD_DIALOG1
- Щелкаем правой кнопкой мыши на форме IDD_DIALOG1 и выбираем ClassWizard – создаем новый класс для формы (Create a New Class) под названием CLineStyle.
- Устанавливаем на полученную форму IDD_DIALOG1 5 элементов управления RadioButton. Присваиваем идентификаторы элементам управления соответственно порядку: IDC_SOLID, IDC_DASH, IDC_DOT, IDC_DASHDOT, IDC_DASHDOTDOT. В свойстве IDC_SOLID обязательно ставим галочку Group. Все последующие элементы будут входить в данную группу элементов (пока не встретится следующий элемент с отмеченным атрибутом Group).
- Добавляем меню с идентификатором IDR_MENU1. Связываем меню с диалогом IDD_MINIDRAW_DIALOG. Используя ClassWizard, добавляем меню в существующий класс CMiniDrawDlg (Select an existing class). Добавляем пункты меню, присваиваем им идентификаторы и добавляем обработчики события WM_COMMAND.

Цвет фона (IDC_COLOR_CUSTOM)

Цвет линии (IDC_COLOR_LINE)

Тип линии (IDC_LINE_SELECT)

- В класс CMiniDrawDlg добавляем в раздел private переменные:

```
RECT m_RectSample; //Структура прямоугольной области
COLORREF Color,LineColor; //цвет кисти и пера
int PenStyle; //стиль пера
```

- В процедуре BOOL CMiniDraw6Dlg::OnInitDialog() добавляем следующие строки:

```
// TODO: Add extra initialization here
//Получаем координаты элемента управления GroupBox в системе координат окна
GetDlgItem(IDC_SAMPLE)->GetWindowRect(&m_RectSample);
//Переводим координаты экрана в координаты клиентской области
ScreenToClient (&m_RectSample);
//Уменьшаем область рисования(отступ от границы)
m_RectSample.bottom=m_RectSample.bottom-8;
m_RectSample.left=m_RectSample.left+8;
m_RectSample.right=m_RectSample.right-8;
m_RectSample.top=m_RectSample.top+8;
//Определяем начальные значения текущего цвета фона и линии
Color = RGB(30,30,30);
LineColor = RGB(200,200,0);
//Устанавливаем текущий стиль пера
PenStyle = PS_SOLID;
```

- В процедуре OnPaint комментируем строку //CDialog::OnPaint(); и добавляем следующий код:

```
//Создаем объект кисть
CBrush Brush (Color);//по умолчанию цвет черный
//Создаем объект перо
CPen Pen;
//Создаем перо методом CreatePen(стиль пера, толщина, цвет) .
//При толщине более 1 стиль ТОЛЬКО PS_SOLID
Pen.CreatePen(PenStyle,1,LineColor);
//Создаем объект контекста устройства
CPaintDC dc (this);
//Выбираем новое перо и сохраняем адрес старого пера в указателе OldPen
CPen* OldPen = dc.SelectObject(&Pen);
//Заполняем прямоугольную область методом FillRect(область RECT, кисть)
dc.FillRect(&m_RectSample,&Brush);
//Устанавливаем точку в заданном месте методом MoveTo(point)
dc.MoveTo(m_RectSample.left,(m_RectSample.bottom - m_RectSample.top)/2);

double x,y,z,k;
```

```

z=m_RectSample.top+(m_RectSample.bottom-m_RectSample.top)/2;
k=m_RectSample.left;
for (x=0;x<(m_RectSample.right-m_RectSample.left);x=x+5)
{
    y=sin(x);
    //Рисуем линию до следующей точки LineTo(point)
    dc.LineTo(x+k,y*100+z);
}
//Восстанавливаем старое перо
dc.SelectObject(OldPen);

```

- В обработчике пункта меню Цвет фона добавляем следующие строки:

```

void CMiniDraw6Dlg::OnColorCustom()
{
    // TODO: Add your command handler code here
    //Объект стандартного диалога выбора цвета
    CColorDialog ColorDialog;
    //Открытие диалога
    if (ColorDialog.DoModal()==IDOK)
    {
        //Присваиваем выбранный цвет
        Color=ColorDialog.GetColor();
        //Помечаем область для перерисовки
        InvalidateRect(&m_RectSample);
    }
}

```

- В обработчике пункта меню Цвет линии добавляем следующие строки:

```

void CMiniDraw6Dlg::OnColorLine()
{
    // TODO: Add your command handler code here
    CColorDialog ColorDialog;
    if (ColorDialog.DoModal()==IDOK)
    {
        LineColor=ColorDialog.GetColor(); //Определяем цвет линии
        InvalidateRect(&m_RectSample);
    }
}

```

- В файле LineStyle.h добавляем переменную в класс CLineStyle для хранения стиля линии (сплошная, пунктир и т.д.) и метода для доступа к переменной:

```

class CLineStyle : public CDialog
{
    // Construction
public:
    CLineStyle(CWnd* pParent = NULL);    // standard constructor
    int GetLineStyle();
private:
    int ps_style; //тип пера
    ....
}

....
int CLineStyle::GetLineStyle()
{
    return ps_style;
}

```

- В обработчике каждого элемента Radio button присваиваем соответствующий тип пера:

```

void CLineStyle::OnSolid()
{
    // TODO: Add your control notification handler code here
    ps_style=PS_SOLID;//сплошная линия
}

```

```

void CLineStyle::OnDash()
{
    ps_style=PS_DASH;//прерывистая
}
void CLineStyle::OnDot()
{
    ps_style=PS_DOT;//точки
}
void CLineStyle::OnDashdot()
{
    ps_style=PS_DASHDOT;//точки с тире
}
void CLineStyle::OnDashdotdot()
{
    ps_style=PS_DASHDOTDOT;//точки с двойным тире
}

```

- В обработчике пункта меню Тип линии добавляем следующие строки:

```

//Создание объекта диалоговой панели для выбора типа линии
    CLineStyle LineStyleDialog;
    if (LineStyleDialog.DoModal() == IDOK)
    {
        PenStyle = LineStyleDialog.GetStyle();
        InvalidateRect(&m_RectSample);
    }

```

- Тестирование работы программы

После запуска программы проверьте реакцию программы на выбор нового цвета, отрисовку линий различных типов. Очевидно, вы будете наблюдать, что пунктирные линии в промежутках не заполняются цветом фона. Чтобы промежутки между штрихами были окрашены в цвет фона, в методе OnPaint вызовите метод установки цвета фона:

```
dc.SetBkColor(Color);
```

Лабораторная работа № 3

Создание приложения на основе архитектура «Документ/представление» с использованием библиотеки MFC в среде MS Visual C++

Цель работы:

Познакомиться с классами библиотеки MFC CDocument (класс документа) и CView (класс представления), с методами создания SDI-приложения.

Задание:

Создать простейший текстовый редактор, с функциями записи в файл, чтения из файла, работы с буфером обмена, поиска и замены текста.

Создать простейший графический редактор на базе SDI-приложения. Редактор должен обеспечивать рисование нескольких фигур, сохранение параметров фигур в файле и восстановление фигур на экране из файла. Команды пользователя должны вызываться с помощью специальной панели инструментов (минимальное количество кнопок – 2). Функции редактора:

№	Состав команд
1	Выбор формы фигуры
2	Изменение цвета фигуры
3	Изменение типа линии
4	Включение/выключение границы фигуры и заливки

Справочный материал

Пример создания простейшего текстового редактора

7. Воспользуйтесь мастером AppWizard, чтобы сгенерировать исходные файлы, как в предыдущих примерах лабораторных работ. Во вкладке Project диалогового окна New введите имя проекта (MiniEdit) в поле name, а в поле location – путь к каталогу проекта. В окне мастера (Step 6) вместо задания всех значений по умолчанию необходимо выполнить следующее:
 - Выбрать имя класса CMiniEditView из списка, находящегося в верхней части диалогового окна.
2. В списке BaseClass выбрать CEditView.
8. Изучите предложенный мастером состав классов, файлов.
9. Откомпилируйте проект и проверьте выполнение редактором заложенных в него функций.
10. Откройте вкладку ResourceView в нижней части окна ProjectWorkspace, чтобы просмотреть ресурсы программы.
11. Отредактируйте меню программы – добавьте четыре новых команды в раздел меню Edit (под командой Paste). Идентификаторы команд и надписи приведены ниже. После добавления команды Select All следует добавить разделительный элемент. Чтобы вставить разделитель, отметьте опцию Separator в диалоговом окне Menu Item Properties вместо ввода идентификатора и надписи.

Идентификаторы команд	Надписи
ID_EDIT_SELECT_ALL	Select &All
ID_EDIT_FIND	&Find...
ID_EDIT_REPEAT	Find &Next\tF3
ID_EDIT_REPLACE	&Replace...

12. В надписи к команде Find Next указана функциональная клавиша F3 для быстрого выполнения команды. Такие клавиши называются горячими клавишами (акселераторами) и определяются редактором ресурсов Developer Studio. Чтобы определить эту клавишу выполните следующее:
 - В ResourceView выполните двойной щелчок на идентификаторе IDR_MAINFRAME в разделе Accelerator. Откроется окно горячих клавиш.
 - Выполните двойной щелчок на пустом поле в нижней части списка для добавления новых клавиш. Откроется диалоговое окно Accel Properties.

- Введите идентификатор ID_EDIT_REPEAT в поле ID. Идентификатор задан для команды Repeat пункта меню Edit.
 - Щелкните на Next Key Typed, а затем нажмите функциональную клавишу F3.
13. Следующий этап – изменение строкового ресурса программы для определения стандартного расширения файлов, отображаемых в диалоговых окнах Open и Save As. Для этого откройте редактор строк DeveloperStudio, выполнив двойной щелчок на элементе StringTable графа ResourceView. Первая строка в окне редактора имеет имя IDR_MAINFRAME. Она создана мастером AppWizard и содержит информацию, относящуюся к программе MiniEdit. Её содержимое такое:

```
MiniEdit\n\nMiniEd\n\n\nMiniEdit.Document\nMiniEd Document
```

14. Чтобы модифицировать строку, откройте диалоговое окно String Properties. Вставьте в строку после четвёртого символа '\n' текст MiniEdit Files (*.txt), отображаемый в окне «Тип файла» стандартной диалоговой панели Open/Save, затем вставьте текст .txt – стандартное расширение файлов, используемое по умолчанию. В результате получится строка:

```
MiniEdit\n\nMiniEd\nMiniEdit Files (*.txt)\n.txt\nMiniEdit.Document\nMiniEd Document
```

Пример создания графического редактора для рисования одной фигуры - окружности.

Создайте стандартное SDI-приложение с настройками по умолчанию, имя проекта - GrEdit. Добавим в класс документа CGrEditDoc переменные для хранения данных - координат центра окружности x и y.

```
int x,y;
```

Инициализируем элементы данных в методе OnNewDocument класса документа:

```
BOOL CGrEditDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)
    x = 200; y = 200;
    return TRUE;
}
```

Ввести код для рисования круга. Рисование выполняется методом OnDraw() класса представления. Метод вызывается при получении сообщения WM_PAINT. Это сообщение может быть вызвано функцией Invalidate:

```
void CGrEditView::OnDraw(CDC* pDC)
{
    CGrEditDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    CRect Rect;
        //Структура задает квадрат, в который будет вписана окружность
    Rect.left = p_Doc->x-20;           // Радиус окружности - 20
    Rect.right = p_Doc->x+20;
    Rect.top = p_Doc->y-20;
    Rect.bottom = p_Doc->y+20
    pDC->Ellipse(&Rect); // Указатель на контекст устройства
}
```

Откомпилируйте программу и убедитесь, что окружность появляется в заданном месте окна.

Добавьте в код класса CGrEditView функцию OnLButtonDown, обрабатывающую нажатие левой кнопки мыши: вызвать MFC ClassWizard; на странице Message Maps установить Class name – CGrEditView, Object ID – CGrEditView, Message – WM_LBUTTONDOWN, добавить функцию.

Отредактируйте функцию OnLButtonDown:

```
void CGrEditView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // Обновляем элементы данных класса документа
    CGrEditDoc* pDoc = GetDocument();
    pDoc->x = point.x; //Обновление элементов класса документа
    pDoc->y = point.y; // в соответствии с координатами точки, где была нажата мышь
}
```

```

        // Вызываем OnDraw( )
        Invalidate();
        // Установление флажка изменения документа – документ изменён
        pDoc->SetModifiedFlag(TRUE);

        CView::OnLButtonDown(nFlags, point);
    }

```

Откомпилируйте программу и убедитесь, что окружность теперь появляется в том месте окна, где был произведён щелчок левой кнопкой мыши.

Добавьте в проект возможность сохранять данные на диске или загружать с диска. Эти действия выполняет функция `Serialize` в классе документа:

```

void CGrEditDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
        ar << x << y; // Помещение данных в поток ar
    }
    else
    {
        // TODO: add loading code here
        ar >> x >> y; // Извлечение данных из потока ar
    }
}

```

Создание панели инструментов

9. Добавить ресурс `ToolBar`.

10. Указать в окне Свойства идентификатор ресурса – `IDR_TOOLBAR1`.

11. Двойным щелчком мыши на ресурсе `IDR_TOOLBAR1` включить графический редактор панели инструментов и создать там несколько кнопок. Настроить свойства каждой кнопки - присвоить идентификатор (`ID_ . .`). Окно свойств кнопки доступно с помощью двойного щелчка мышью на изображении кнопки в панели инструментов.

12. Добавить в класс `CMainFrame` указатель на панель инструментов (переменная `m_MyToolBar`).

13. Создать панель инструментов в методе `Create` класса главного окна `CMainWnd`:

```

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    .....
    //Создание панели инструментов
    m_MyToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE |
                        CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS |
                        CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

    // Загрузка ресурса панели инструментов
    m_MyToolBar.LoadToolBar(IDR_TOOLBAR1);
    .....
    return 0;
}

```

14. Связать кнопки панели инструментов с методами класса представления: вызвать `MFC Class Wizard`; на странице `Message Maps` установить `Class name` – `CGrEditView`, `Object ID` – идентификатор кнопки, `Message` – `COMMAND`, добавить функцию.

Класс `CArchive` и сериализация класса

Класс `CArchive` предназначен для хранения двоичных потоков данных. Чаще всего он используется для сериализации объектов. Сериализация – это способность объектов сохранять свои данные в байтовом потоке с последующим восстановлением. Чтение и запись переменных, относящихся к базовым типам данных, возможна с помощью перегруженных операторов `>>` и `<<` класса `CArchive`. Класс `CArchive` предоставляет также функции `Read()` и `Write()` для чтения и записи произвольных блоков данных, например, структуры или текстового буфера.

Функция `CArchive::Read()` имеет следующий вид:

```

UINT Read (void* lpBuf, UINT nMax);

```

Где `lpBuf` – адрес памяти для приема данных, а `nMax` – число читаемых байтов
Функция `CArchive::Write` имеет следующий вид:

```
void Write (const void* lpBuf, UINT nMax);
```

Где `lpBuf` – адрес записываемого блока данных, а `nMax` – число записываемых байтов. Функция `Write` записывает только необработанные байты из указанного источника. Она не форматирует данные и не добавляет информацию о классе.

Чтение и запись объектов возможна при помощи функции `Serialize()`, принадлежащей базовому классу библиотеки MFC `CObject`. Она входит в состав класса документа и выполняется автоматически, если были выбраны стандартные команды `Save`, `Save as`, `Open`.

Функция `Serialize()` автоматически добавляется в класс документа и имеет следующую структуру:

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

Здесь `ar` – объект-архив, в который помещаются данные документа для сохранения их в файле (`if (ar.IsStoring())`), или из которого извлекаются данные и помещаются в документ. В примере `GrEdit` функция сериализации была отредактирована в соответствии со структурой данных класса документа. В программе `MiniEdit` не нужно изменять функцию `Serialize` класса документа, поскольку при построении класса представления из класса `CEditView` мастер `AppWizard` автоматически добавляет требуемый код в `Serialize`.

Структура данных документа

Структура данных документа может быть произвольной. Обычно удобно создать специальный класс для хранения элемента данных, и в класс документа включить массив или список таких элементов.

В библиотеке MFC предусмотрен удобный шаблон класса для хранения данных массив `CArray`, который может наращивать свою длину по мере надобности, т.е. по сути дела представляющей собой список.

Создание массива `MyArr` элементов класса `CElem`:

```
CArray<CElem, CElem&> MyArr;
```

Первый параметр в шаблоне – тип элемента массива, второй – тип параметра массива, который используется для доступа к его элементам.

Методы класса `CArray`:

<code>int GetSize()</code>	Возвращает размер массива
<code>int Add()</code>	Добавляет элемент в конец массив, выделяет под элемент память, если это надо: <code>CElem elem;</code> <code>.....</code> <code>MyArr.Add(elem);</code>
<code>void RemoveAll()</code>	Удаляет все элементы из массив
<code>CElem GetAt(int nIndex)</code>	Возвращает элемент массива с заданным индексом: <code>CElem elem = GetAt(0);</code>
<code>void SetAt(int nIndex, CElem&)</code>	Помещает новое значение в элемент массива с заданным номером.

Доступ к элементу массива осуществляется с помощью `[]`: `MyArr[2]`.