МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра интеллектуальных информационных технологий

МЕТОДИЧЕСКОЕ ПОСОБИЕ

ОСНОВЫ СОЗДАНИЯ WINDOWS-ПРИЛОЖЕНИЙ В СИСТЕМЕ MICROSOFT VISUAL STUDIO C++ НА БАЗЕ БИБЛИОТЕКИ MFC. ИСПОЛЬЗОВАНИЕ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ. МЕНЮ

Часть 3

БРЕСТ 2010

УДК 681.3 (075.8)
ББК с57

Целью пособия является знакомство студентов с базовыми понятиями оконных приложений и каркасного программирования в системе Microsoft Visual Studio C++ .

Издаётся в 3-х частях. Часть 3.

Составители: Г.Л. Муравьев, доцент, к.т.н.,
С.В. Мухов, доцент, к.т.н.,
В.Н. Шуть, доцент, к.т.н.

Рецензент: доцент кафедры математического моделирования Брестского государственного университета им. А.С. Пушкина, к.т.н. Прописко Е.Е.

1. ОБЩИЕ СВЕДЕНИЯ

Графический интерфейс приложения предназначен обеспечивать интерактивный доступ к приложению со стороны пользователей. В ОС Windows для этого используются специальные графические объекты. Как правило, это окна различных типов, включая диалоговые окна, служащие подложкой, где могут располагаться различные элементы управления (ЭУ) и сами элементы управления.

Как при создании Windows-приложений, так и их графических интерфейсов широко используются классы библиотеки MFC, что ускоряет и упрощает разработку приложений. Фрагмент иерархии классов библиотеки MFC, используемых для создания приложений с графическим интерфейсом, представлен на рисунке ниже.

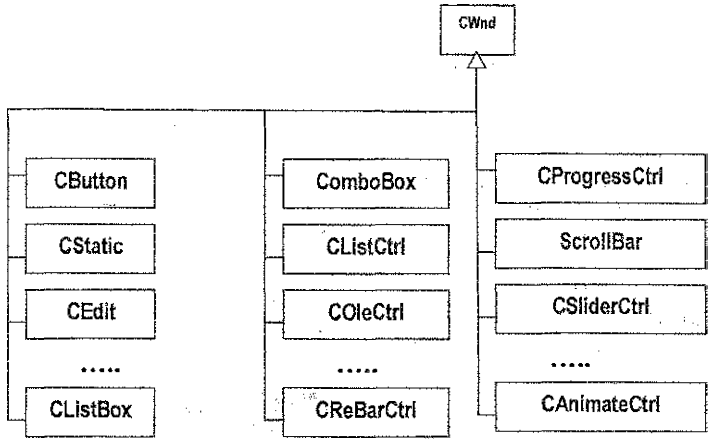

Рисунок 1 – Фрагмент иерархии классов MFC

2. ИСПОЛЬЗОВАНИЕ ДИАЛОГОВЫХ ОКОН

2.1. Общие сведения о списках. Класс CListBox

Назначение, общее описание. Список – это элемент управления, используемый в составе окна. Соответственно все адресуемые списку сообщения являются сообщениями его окна и включаются в его очередь сообщений, а "привязка" к конкретному списку производится с помощью параметра сообщения, содержащего ID_Списка.

Список хранит множество элементов, строк и поддерживает предопределенный набор действий над ними, а каждая строка имеет идентифицирующий номер – индекс, исчисляемый от нуля. Список хранимых элементов может, например, включать имена файлов, фамилий сотрудников и т.п. Пользователь может просматривать указанный список и делать свой выбор.

В списках с единичным выбором (single-selection list box) пользователь может выбрать только один элемент. В списках с множественным выбором (multiple-selection list box) может выделяться диапазон элементов. Результат выбора подсвечивается, а сам список посылает уведомляющее сообщение (notification message) родительскому окну.

В файле ресурсов каждый список описывается командой LISTBOX.

Функциональность элемента управления Windows типа список (list box) в MFC непосредственно поддерживается классом CListBox, точнее объектами класса CListBox (фрагмент диаграммы классов представлен ниже).

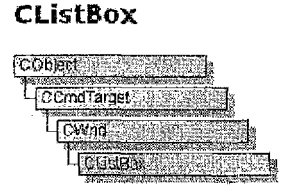

Рисунок 2 – Иерархия классов CListBox

Списки, как и другие элементы управления, могут создаваться как ресурс вне приложения на основе соответствующего оконного шаблона (dialog template) либо непосредственно программным путем – в самом приложении.

При работе со списками следует подключить #include <afxwin.h>

Состояние списка может меняться: под воздействием действий пользователя, событий; программно. Для этого список может как посылать, инициировать сообщения, так и принимать, выполнять команды. В первом случае действия пользователя вызывают информирующие сообщения, которые могут быть приняты обработчиком списка (его окна) и соответствующим образом обработаны, учтены. Во втором случае используются команды списка (методы и операторы класса CListBox), позволяющие влиять на состояние списка программно, например, из его обработчика. Фрагмент составляющих класса CListBox представлен на рисунке ниже.

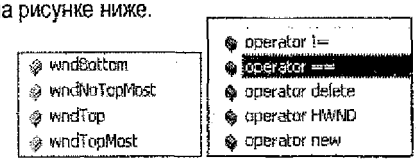

Рисунок 3 – Члены класса CListBox

Стили. Основные разновидности списка: список с единичным выбором; список с множественным выбором; список только для отображения текста и т.д. Полный перечень стилей списков приведен в Приложении 1.

Создание и удаление. Списки могут создаваться на основе соответствующего оконного шаблона (dialog template) либо непосредственно программным путем. В первом случае список создается визуально в редакторе ресурсов либо описывается в текстовом редакторе. В обоих случаях далее создается объект класса CListBox соответствующим конструктором CListBox.

При непосредственном создании вначале создается локальный объект, например, как

CListBox *МойСписок* ;

или динамический объект, например, как

CListBox* *УказательМойСписок* = **new CListBox**; .

Затем вызывается метод Create для создания самого элемента управления и "прикрепления" его к этому объекту. Метод может вызываться в конструкторе объекта.

В случае создания списка на базе оконного шаблона описывается переменная в классе родительского диалогового окна, затем используется средство DDX_Control

(DoDataExchange) для связывания переменной и элемента управления. Указанное выполняется мастером ClassWizard автоматически каждый раз, когда в класс диалогового окна добавляется такая переменная.

Если список создан как ресурс вне приложения на основе соответствующего оконного шаблона (dialog template), то соответствующий CListBox-объект автоматически разрушается, когда пользователь закрывает родительское диалоговое окно.

Если список создан программным путем внутри окна, то о его разрушении должен позаботиться программист в самом приложении. Например, если CListBox-объект создан методом new, то необходимо вызвать метод delete для его разрушения, когда пользователь закрывает родительское диалоговое окно.

Обработка сообщений. Для обработки уведомляющего сообщения (notification message), посылаемого родительскому окну (как правило, объекту класса, производного от CDialog) списком, необходимо ввести в карту сообщений окна макрокоманду со стандартной ON_COMMAND-сигнатурой вида

ON_Notification(*ID_Списка, ИмяОбработчикаСообщения* **)** ,

где ID специфицирует список-источник сообщения и добавить обработчик сообщения с прототипом

afx_msg void *ИмяОбработчикаСообщения* **( )** .

Посылаемые сообщения. Список сообщений можно получить в MSDN, а информацию о сообщении – путем поиска, например, по образцу LBN_, LBN_DBLCLK. Ниже дана общая характеристика типовым событиям и соответствующим уведомляющим сообщениям (notification message):

- сообщение LBN_DBLCLK посылается, если пользователь дважды "щелкает" на строке списка для ее выбора (при условии, что стиль списка LBS_NOTIFY);
- сообщение LBN_ERRSPACE посылается, если не хватает памяти для обслуживания запроса;
- сообщение LBN_KILLFOCUS посылается, если список теряет фокус ввода;
- сообщение LBN_SELCANCEL посылается, если выбор отменен (при условии, что стиль списка LBS_NOTIFY);
- сообщение LBN_SELCHANGE посылается, если выбор в списке меняется (при условии, что стиль списка LBS_NOTIFY). Оно не посылается, если выбор менялся как результат выполнения команды-метода CListBox::SetCurSel. Для списков с множественным выбором сообщение посылается при нажатии клавиш-стрелок, даже если выбор пользователя не меняется;
- сообщение LBN_SETFOCUS посылается, если список получает фокус ввода.

Члены класса. Основные методы класса приведены в Приложении 2 и подразделяются на:

- методы-конструкторы;
- методы инициализации;
- общие методы;
- методы списка с единичным выбором;
- методы списка с множественным выбором;
- строковые методы.

Прототипы наиболее часто используемых методов приведены ниже.

Общая схема создания списка. Включает следующий набор действий:

1) определение вида, типа списка, его свойств (стиля);

2) описание всех событий, связанных со списком, соответствующих сообщений, описание реакции на них (обработчиков), описание макрокоманд обеспечения чувствительности приложения (окна) к сообщениям списка. Т.е. выполнение действий, например, в следующей последовательности:

тип сообщения LBN_< *НазваниеСообщения* > ->

обработчик < *ИмяОбработчикаСообщения* > ->

выполняемые действия как < *СпецификацияОбработчикаСообщений* > ->

макрокоманда включения как ON_LBN_< *НазваниеСообщения*> (<*ID_Списка*>, <*ИмяОбработчикаСообщения* >);

3) описание списка в составе окна-подложки (диалогового окна) в файле ресурсов (командой описания LISTBOX), например, с помощью редактора ресурсов, и установка его свойств;

4) настройка класса окна (например, *ДИАЛОГОВОЕ_ОКНО*: **CDialog**), содержащего список, на работу с ним:

- в описание класса окна необходимо вставить прототипы всех функций-обработчиков сообщений списка (например, afx_msg void On< *ИмяОбработчикаСообщения* > ( ));
- необходимо описать каждый обработчик сообщений списка на основе < *СпецификацииОбработчикаСообщений* >;
- включить чувствительность списка к соответствующим сообщениям – в описание очереди сообщений окна со списком включить необходимые макрокоманды (например, ON_LBN_DBLCLK (< *ID_Списка* >, < *ИмяОбработчикаСообщения* >));
- при необходимости, выполнить инициализирующие действия относительно списка в функции OnInitDialog( ) соответствующего окна-подложки (например, задать начальное содержимое списка).

Общая схема работы со списком. Для работы со списком (это общее правило работы с элементами управления) необходимо получить на него указатель, используя функцию

CWnd *CWnd::GetDlgItem (int *ID_Списка* **) const** .

Через указатель далее можно вызывать необходимые методы, посылать списку команды. Например,

CListBox *pCListBox = (CListBox *) GetDlgItem (ID_CListBox) ,

где pCListBox (или lbptrCListBox) и есть указатель, через который вызываются все методы списка, например,

int i = pCListBox -> GetCount( );

int i = pCListBox -> GetCurSel( ).

Для инициализации содержимого списка в описании класса его родительского окна используется метод

OnInitDialog ( )

Общая схема использования

BOOL *ДИАЛОГОВОЕ_ОКНО*::**OnInitDialog( )**

{

CDialog::OnInitDialog ( );

Получить_указатель_на_список (например, **CListBox** **pListBox* **= (CListBox *) GetDlgItem(** *ID_Списка* **));**

Инициализировать_список (например, **pListBox -> AddString(** < ОднаСтрока*Списка* > **);**

.....

return TRUE;

};

2.2. Диалоговое окно со списком в качестве главного окна

ЗАДАНИЕ № 1. *Создать MFC-приложение на базе ТКП для ведения списка записей – фамилий.*

Исходный, а затем и модифицированный список фамилий должен отображаться в диалоговом окне в соответствующем списковом элементе управления (класса CListBox). Исходный список создается в момент инициализации диалогового окна последовательным добавлением к списку заранее определенных фамилий.

Пользователь может осуществить выбор (выделение) фамилии одинарным либо двойным щелчком "мыши".

Выбранную (выделенную) фамилию здесь можно: удалить; использовать для создания и добавления к списку новой фамилии.

Каждое из трех действий (выбор, удаление, добавление) должно сопровождаться выводом сообщения о его выполнении и требованием на подтверждение.

Добавление фамилии здесь производится в упрощенном варианте и состоит во вводе в список копии уже содержащейся фамилии, выбранной одинарным или двойным щелчком, с добавлением к ней числового номера (например, Иванов1, Иванов5, ...).

Для создания приложения необходимо выполнить следующие действия.

1. Спроектировать приложение.

1.1. Спроектировать интерфейс – диалоговое окно.

1.1.1. Разработать сценарии работы приложения.

Здесь в полях списка диалогового окна выводится перечень строк – фамилий.

Фамилию можно выбрать щелчком мыши (этому событию соответствует сообщение LBN_CHANGESEL), двойным щелчком (этому событию соответствует сообщение LBN_DBLCLK). Выбор пользователя должен сопровождаться его же подтверждением.

Выделенную фамилию можно удалить нажатием кнопки Удалить ФИО. Для этого используется метод

int DeleteString(UINT *nIndex*) .

Новую фамилию можно добавить нажатием кнопки Добавить ФИО. Здесь добавление фамилии будет состоять во вводе копии уже содержащейся фамилии (выбранной одинарным или двойным щелчком) с добавлением к ней соответствующего числового номера. Для добавления новой фамилии к списку используется метод

int AddString(LPCSTR *lpszStr*) .

Каждое из перечисленных выше действий сопровождается сообщением о его выполнении и требованием подтверждения.

Соответственно в приложении понадобятся обработчики:

- сообщений типа LBN_DBLCLK для вывода результата выбора;
- обработчики сообщений нажатия кнопок Удалить ФИО и Добавить ФИО, приводящие к соответствующей коррекции списка.

1.1.2. Определить состав графического интерфейса приложения, стили элементов интерфейса. Здесь в качестве главного окна будет использоваться диалоговое окно (поддерживаемое классом *MY_DIALOG*) со списком и двумя кнопками управления списком. Кроме этого, используется статичный элемент – рамка с названием "Управление списком". Примерный вариант интерфейса представлен ниже.

В случае попытки выполнения команды без выбора фамилии должно выводиться предупреждение, как показано на рисунке ниже.

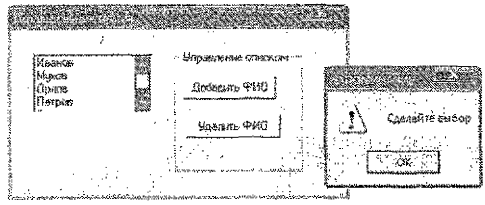

Рисунок 4 – Интерфейс приложения

Каждая выполняемая команда комментируется, как показано на рисунке.

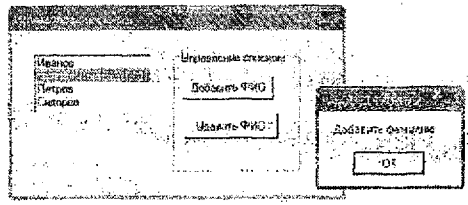

Рисунок 5 – Интерфейс приложения. Подтверждение команды

Выполнение каждой команды требует подтверждения

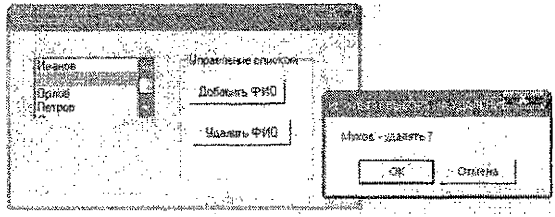

Рисунок 6 – Интерфейс приложения. Подтверждение выполнения

1.1.3. Специфицировать состав событий-сообщений при работе с диалоговым (главным) окном и реакций приложения на эти сообщения:

- событие "двойной щелчок на элементе списка" вызывает сообщение LBN_DBLCLK. Чувствительность окна, содержащего этот список, задается макрокомандой

ON_LBN_DBLCLK(*<ID_Списка>*, *<ИмяОбработчикаСообщения>* **).**

Реакция на сообщение – вывод выбранной фамилии. Поскольку используется пользовательский обработчик, то в очереди обрабатываемых сообщений нужно указать этот тип сообщений – в очередь сообщений вставляется макрокоманда, например,

ON_LBN_DBLCLK(*<ID_Списка>*, *ToDisplaySelectedFIO* **).**

Это означает, что сообщению соответствует обработчик с прототипом

afx_msg void *MY_DIALOG*:: *ToDisplaySelectedFIO* **( );**

- событие "нажатие кнопки Добавить ФИО" вызывает сообщение WM_COMMAND. Реакция на сообщение – добавление фамилии к списку. В очередь сообщений вставляется макрокоманда, например,

ON_COMMAND (*<ID_КнопкиДобавитьФИО>*, *ToAddFIO* **).**

Это означает, что сообщению соответствует обработчик с прототипом

afx_msg void *MY_DIALOG*:: *ToAddFIO* **( );**

- событие "нажатие кнопки Удалить ФИО" вызывает сообщение WM_COMMAND. Реакция – удаление фамилии. В очередь сообщений вставляется макрокоманда, например,

ON_COMMAND (*<ID_КнопкиУдалитьФИО>*, *ToDeleteFIO*).

Это означает, что сообщению соответствует обработчик с прототипом

afx_msg void *MY_DIALOG*:: *ToDeleteFIO* ( ).

1.2. Спроектировать классы приложения. Здесь используются следующие классы:

- класс myAPPLICATION: public CWinApp для создания экземпляра приложения;
- класс myDIALOG: public CDialog для создания объекта – диалоговое окно.

1.3. Спроектировать модульную структуру приложения. Здесь для представления программной части (за исключением ресурсов) используется один модуль, например, main.cpp.

2. Подготовить каркас MFC-приложения с файлом ресурсов. Для этого необходимо создать приложение на базе типового каркаса, создать файл описания ресурсов и подключить его к приложению.

3. Создать новый ресурс – диалоговое окно. Для этого необходимо в соответствии с видом графического интерфейса приложения (приведен выше) добавить ресурс – диалоговое окно, отредактировать его внешний вид, добавить необходимые элементы управления (список и кнопки), настроить свойства окна и его элементов. Примерное текстовое описание окна приведено ниже.

```
// Dialog
IDD_DIALOG1 DIALOG  DISCARDABLE  0, 0, 230, 103
STYLE  DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION  "Использование списка"
FONT  8, "MS Sans Serif"
BEGIN
   LISTBOX       IDC_LIST1, 15,15,81,37,LBS_SORT | LBS_NOINTEGRALHEIGHT |
                 WS_VSCROLL | WS_TABSTOP
   GROUPBOX      "Управление списком",IDC_STATIC,111,13,89,75
   PUSHBUTTON    "Добавить ФИО",IDC_BUTTON1,118,29,64,14
   PUSHBUTTON    "Удалить ФИО",IDC_BUTTON2,120,52,65,15
END
```

Здесь IDD_DIALOG1, IDC_LIST1, IDC_STATIC, IDC_BUTTON1, DC_BUTTON2 – дескрипторы диалогового окна, списка, рамки и двух кнопок соответственно.

4. Необходимо описать пользовательский класс MY_DIALOG для создания экземпляра главного окна, например, как

```
class myDIALOG : public CDialog
{
public:
      myDIALOG(char *DialogName, CWnd *Owner): CDialog(DialogName, Owner)
{     };
BOOL OnInitDialog( );
afx_msg void  OnToDisplaySelectedFIO( );
afx_msg void  OnToAddFIO( );
      afx_msg void OnToDeleteFIO( );
DECLARE_MESSAGE_MAP( )
};
```

Описать карту сообщений

```
BEGIN_MESSAGE_MAP(myDIALOG, CDialog)
    ON_LBN_DBLCLK(IDC_LIST1, OnToDisplaySelectedFIO)
    ON_COMMAND(IDC_BUTTON1, OnToAddFIO)
    ON_COMMAND(IDC_BUTTON2, OnToDeleteFIO)
END_MESSAGE_MAP( )
```

Описать метод OnInitDialog(), используемый здесь для инициализации списка – для задания его первоначального содержимого в момент вывода диалогового окна на экран.

```
BOOL myDIALOG::OnInitDialog( )
{
CDialog::OnInitDialog();
CListBox *PListBox = (CListBox *) GetDlgItem(IDC_LIST1);
PListBox -> AddString("Иванов");
PListBox -> AddString("Петров");
PListBox -> AddString("Сидоров");
PListBox -> AddString("Орлов");
PListBox -> AddString("Мухов");
        return TRUE;
};
```

Описать обработчики сообщений

```
afx_msg void myDIALOG::OnToDisplaySelectedFIO( )
{
char Str[80];
MessageBox( "Выбор двойным щелчком","РАБОТАЕТ ФУНКЦИЯ" );
CListBox *PListBox = (CListBox *) GetDlgItem( IDC_LIST1 );
int i = PListBox ->GetCurSel( );
PListBox ->GetText( i,Str );
strcat( Str," - ваш выбор" );
MessageBox( Str,"РЕЗУЛЬТАТ ВЫБОРА" );
};
afx_msg void myDIALOG::OnToAddFIO( )
{
static int j=0;
char Str[80];
        char Str1[80];
         char Str2[80];
j++;
MessageBox( "Добавить фамилию","РАБОТАЕТ ФУНКЦИЯ" );
CListBox *PListBox = (CListBox *) GetDlgItem(IDC_LIST1);
int i = PListBox -> GetCurSel( );
        if (i == LB_ERR )
{
MessageBox("Сделайте выбор","НАПОМИНАНИЕ", MB_OK | MB_ICONWARNING)
}
```

```
else
{
    PListBox ->GetText(i,Str);
    strcpy( Str1,Str );
    strcat( Str1," - ваш выбор" );
    MessageBox( Str1,"РЕЗУЛЬТАТ ВЫБОРА На ДОБАВЛЕНИЕ" );
    wsprintf( Str2,"%d",j );
    strcat( Str,Str2 );
    PListBox -> AddString( Str );
};
};

afx_msg void myDIALOG::OnToDeleteFIO( )
{
char Str[80];
MessageBox( "Удалить фамилию", "РАБОТАЕТ ФУНКЦИЯ" );
CListBox *PListBox = (CListBox *) GetDlgItem( IDC_LIST1);
      int i = PListBox -> GetCurSel( );
if  (i == LB_ERR )
{
 MessageBox( "Сделайте выбор","НАПОМИНАНИЕ", MB_OK | MB_ICONWARNING );
}
else
{
      i = PListBox -> GetCurSel( );
      PListBox ->GetText( i,Str );
      strcat( Str," - удалять ?" );
            if (MessageBox(Str,"РЕЗУЛЬТАТ ВЫБОРА НА УДАЛЕНИЕ",
                  MB_OKCANCEL) == IDOK)
      {
          PListBox -> DeleteString( i );
      };
};
};
```

5. Подключить диалоговое окно к приложению – оно должно запускаться автоматически при инициализации пользовательского приложения в качестве главного окна

```
class myAPPLICATION:public CWinApp
{
public:
BOOL InitInstance( );
};

BOOL myAPPLICATION::InitInstance( )
{
myDIALOG  TheDialog( ( LPTSTR ) IDD_DIALOG1,NULL );
TheDialog.DoModal( );
return TRUE;
} .
```

6. Откомпилировать и выполнить приложение.

2.3. Задания для самостоятельного выполнения

1. Модифицировать приложение, добавив отображение в диалоговом окне выбранной строки, номера выбранной строки, а также числа строк в списке.

2*. Модифицировать предыдущее приложение для удаления только фамилий, выбранных двойным щелчком.

3*. Модифицировать приложение в части добавления в список новой введенной фамилии. Для этого предусмотреть в том же диалоговом окне сегмент с полем ввода, полем отображения ввода и кнопками управления вводом. Разработать соответствующую диаграмму состояний UML.

4*. Модифицировать предыдущее приложение, добавив проверку новой фамилии на совпадение с уже имеющимися и на корректность написания фамилии.

5. Модифицировать предыдущее приложение, использовав для ввода новой фамилии отдельное диалоговое окно.

6. Модифицировать предыдущее приложение, обеспечив автоматическое сохранение списка в файле при закрытии приложения и загрузку списка из файла при запуске приложения. Использовать один и тот же файл.

7.* Модифицировать предыдущее приложение, обеспечив автоматическое сохранение списка при закрытии приложения в заданном файле и загрузку списка при запуске приложения из заданного файла.

8.* Модифицировать предыдущее приложение, обеспечив автоматическое сохранение списка в заданном файле и загрузку списка из заданного файла в любой момент работы приложения.

9. Модифицировать приложение, реализовав поддержку обработки списка с помощью соответствующего пользовательского класса.

9*. Добавить поиск строки в списке по заданному образцу.

10*. Модифицировать приложение, реализовав систему меню для поддержки всех операций.

ПРИМЕЧАНИЕ: пункты, помеченные *, выполняются по указанию преподавателя.

2.4. Программное создание элементов управления. Индикаторы. Класс CProgress

Индикатор – это элемент управления, используемый в составе окна. Функциональность индикатора поддерживается классом CProgressCtrl.

CProgressCtrl

Рисунок 7 – Иерархия классов

В файле ресурсов индикатор описывается командой PROGRESS.

В интерфейсе индикатор отображается полем – окном, заполняемым динамически, например, прямоугольниками. Количество прямоугольников может отражать состояние индикатора. А состояние индикатора соответственно может отображать степень выполнения какой-то задачи, процесса во времени. Можно считать, что с каждым индикатором связан счетчик его заполненности. Управление счетчиком поддерживается методами ти-

па **SetPos (int** *NewPos*), **SetRange (int** *Min* = **0, int** *Max* = **100), SetStep (int** *Amount* = **10), StepIt ( )** и др.

Наиболее часто используемые методы CProgressCtrl описаны ниже:

- создать индикатор

BOOL CProgressCtrl::Create(DWORD *Стиль*, **const RECT &***Размер*, **CWnd** * *Окно-Родитель*, **UINT** *ID_Индикатора*);

- установить начальное значение счетчика индикатора

int CProgressCtrl::SetPos (int *НачальноеСостояние*);

- установить диапазон значений счетчика индикатора

int CProgressCtrl:: SetRange (int *Min* = **0, int** *Max* = **100);**

- установить шаг изменения счетчика индикатора

int CProgressCtrl::SetStep (int *Шаг* = **10);**

- вывести следующий элемент индикатора (прямоугольник)

int CProgressCtrl::StepIt ( ).

Основные стили: PBS_SMOOTH для отображения индикатора с непрерывным заполнением вместо сегментированного по умолчанию; PBS_VERTICAL для вертикального отображения индикатора.

Общая схема использования индикатора включает следующий набор действий.

1. Проектирование индикатора. Включает:

- определение стиля, вида, свойств индикатора;

- определение и описание событий, связанных с индикатором, и соответствующих им сообщений. Описание реакции на сообщения, макрокоманд обеспечения чувствительности приложения (окна) к сообщениям.

2. Подключение заголовочного файла общих элементов управления

```
#include <afxcmn.h>
```

и инициализацию их в конструкторе окна, например,

```
ОКНО::ОКНО
{
   Create ( ... "Главное окно" ...);
   InitCommonControls ( );
   ......
}
```

или в конструкторе диалогового окна (если оно является главным), например,

```
class мойДИАЛОГ : public CDialog
{
   ......
public:
   мойДИАЛОГ ( char *ИмяОкна, CWnd *Родитель ): CDialog( ИмяОкна, Родитель )
   {
       InitCommonControls ( );
       .........
   };
   ........
   DECLARE_MESSAGE_MAP( )
};
```

3. Настройку класса окна (например, *мойДИАЛОГ*:**public CDialog**), содержащего индикатор, на работу с ним. Для этого необходимо:

- включить в описание класса окна новые закрытые члены, например, описать объект-индикатор и переменную, отображающую его состояние (счетчик), например,

```
CProgressCtrl  ЭтотИндикатор;
int            m_ProgPos;
```

- внести в описание класса окна прототипы всех функций-обработчиков сообщений, например,

```
afx_msg void On< ИмяОбработчикаСообщенийИндикатора > ( ) ;
```

- описать обработчики сообщений;
- включить чувствительность окна с индикатором к сообщениям соответствующими макрокомандами;
- выполнить инициализирующие действия относительно индикатора в функции OnInitDialog( ) соответствующего окна-подложки. Например, задать его начальное положение, вид, состояние и т.п. Например, как

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

    RECT     ОбластьИндикатора;
    ОбластьИндикатора.bottom = 60;
    ОбластьИндикатора.left  = 10;
    ОбластьИндикатора.right  = 150;
    ОбластьИндикатора.top  = 50;
    ЭтотИндикатор.Create ( WS_VISIBLE , ОбластьИндикатора, this , 2000 );
    ЭтотИндикатор.SetRange ( 0, 120 );
    ЭтотИндикатор.SetStep ( 5 );
    m_ProgPos = 0;
    return TRUE;
};
```

При работе с индикатором, как правило, контролируется такое событие, как его заполнение. Например, в цикле можно выполнять следующие действия (для каждого промежуточного события увеличивать заполненность индикатора, фиксировать новое состояние индикатора, при необходимости выполнять пользовательские действия, а в случае полной заполненности индикатора – и завершающие пользовательские действия).

```
ЭтотИндикатор.StepIt ( );
m_ProgPos  += Шаг;
<ДействияПользователя >
if  ( m_ProgPos == Max)
{
    <ЗавершающиеДействия Пользователя >
}
```

2.5. Индикаторы в составе диалогового окна

ЗАДАНИЕ № 2. *Создать приложение на базе ТКП с интерфейсом в виде диалогового окна, содержащего индикатор, управляемый кнопкой ОК.*

По нажатию кнопки ОК необходимо увеличивать значение внутреннего счетчика индикатора, выводить очередной прямоугольник в поле индикатора и контролировать значение счетчика. При достижении или превышении максимального значения (когда индикатор полностью заполняется) необходимо завершать работу приложения.

Установить диапазон значений счетчика индикатора от 0 до 120 единиц, шаг изменения счетчика при нажатии кнопки ОК в 5 единиц. Соответственно в поле индикатора при полном заполнении будет отображаться ровно 24 прямоугольника, а приложение при 24-м нажатии кнопки ОК должно завершить свою работу.

Кроме этого, работа приложения может быть прервана в любой момент соответствующей кнопкой системного меню либо пользовательской кнопкой Cancel.

Для создания приложения необходимо выполнить следующие действия.

1. Спроектировать приложение.

1.1. Спроектировать интерфейс – диалоговое окно.

1.1.1. Определить состав элементов управления окна и их вид. При необходимости описать окна и элементы управления на соответствующем языке описания ресурсов.

Здесь интерфейсные формы включают, как показано на рисунке ниже, одно диалоговое окно, выводимое при запуске приложения в роли главного. В состав окна входит горизонтально-расположенный индикатор, кнопки управления, окно содержит системное меню и системные кнопки.

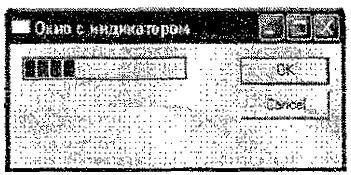

Рисунок 8 – Интерфейс приложения

1.1.2. Специфицировать состав событий-сообщений при работе с диалоговым окном и реакции приложения на эти сообщения. Основные события:

- событие "нажатие кнопки ОК" вызывает сообщение WM_COMMAND. Реакция на сообщение пользовательская – заполнение 1/24 индикатора, увеличение переменной-счетчика индикатора m_ProgPos на 5 единиц, контроль значения счетчика и при превышении значения 120 закрытие окна и приложения;
- событие "нажатие кнопки Cancel" вызывает сообщение WM_COMMAND. Реакция на сообщение системная – закрытие окна и приложения. Поэтому при описании ресурсов приложения этой кнопке следует задать системный дескриптор IDCANCEL, обеспечивающий вызов системного обработчика;
- аналогично автоматически поддерживаются действия над системными кнопками.

1.2. Спроектировать классы приложения. Здесь используются следующие классы, представленные на рисунке ниже:

- класс APPLICATION: public CWinApp для создания экземпляра приложения;
- класс MY_DIALOG: public CDialog для создания объекта – диалоговое окно;
- класс CProgressCtrl для создания объекта TheProgress – индикатор.

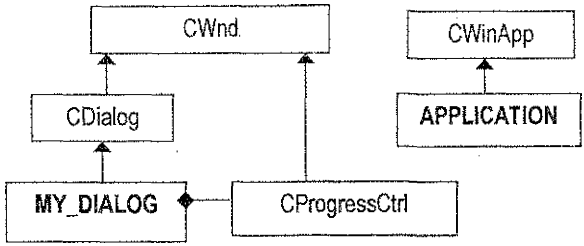

Рисунок 9 – Упрощенная иерархия классов приложения

Примерный состав классов приложения, их члены представлены ниже на рисунке.

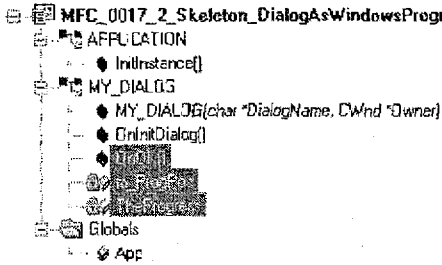

Рисунок 10 – Состав классов приложения

1.3. Спроектировать модульную структуру приложения. Здесь для представления программной части (за исключением ресурсов) используется один модуль, например, main.cpp

2. Подготовить каркас MFC-приложения с файлом ресурсов. Для этого создать типовой каркас приложения, создать файл описания ресурсов и подключить его к приложению. Подключить общие элементы управления директивой

#include <afxcmn.h> .

3. Создать новый ресурс – диалоговое окно. Для этого добавить ресурс к проекту, отредактировать его внешний вид, настроить свойства окна и элементов управления. Как видно из рисунка ниже, в редакторе ресурсов индикатор не описывается, однако для его программного вывода и расположения в окне оставлено место слева от кнопок.

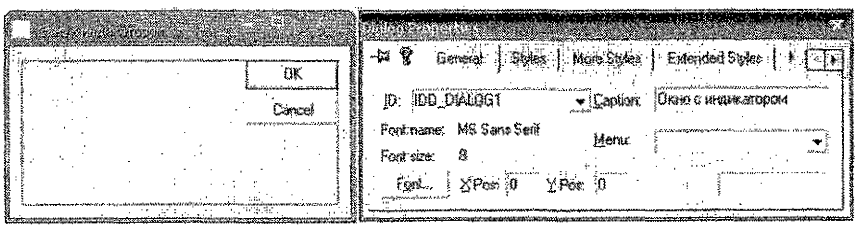

Рисунок 11 – Ресурс – диалоговое окно

Примерный вид соответствующего описания на языке ресурсов представлен ниже.

```
// Dialog
IDD_DIALOG1   DIALOG     DISCARDABLE    0, 0, 186, 90
              STYLE      WS_MINIMIZEBOX | WS_MAXIMIZEBOX | WS_POPUP |
                          WS_CAPTION | WS_SYSMENU   | WS_THICKFRAME
              CAPTION    "Окно с индикатором"
              FONT       8, "MS Sans Serif"
BEGIN
  DEFPUSHBUTTON  "OK",IDOK,129,7,50,14
  PUSHBUTTON     "Cancel",IDCANCEL,129,24,50,14
END
```

4. Описать пользовательский класс MY_DIALOG для создания экземпляра главного окна.

4.1. В описание класса MY_DIALOG следует добавить члены, необходимые для работы с индикатором. Это два закрытых члена

```
CProgressCtrl  TheProgress;
int            m_ProgPos;
```

где *TheProgress* – объект класса CProgressCtrl; а m_ProgPos – счетчик для фиксации состояния индикатора.

В открытую секцию класса надо добавить прототип функции BOOL OnInitDialog( ), необходимой для описания параметров индикатора и его инициализации, а также прототип afx_msg void OnOK( ) функции-обработчика события нажатия кнопки OK для воздействия на состояние индикатора.

В результате получим

```
class MY_DIALOG : public CDialog
{
    CProgressCtrl TheProgress;      // объект – индикатор
    int m_ProgPos;                  // состояние индикатора
public:
    MY_DIALOG ( char *DialogName, CWnd *Owner ): CDialog( DialogName, Owner )
    {
        InitCommonControls ( );
    };
    BOOL OnInitDialog ( );
    afx_msg void OnOK ( );
    DECLARE_MESSAGE_MAP( )
};
```

А остальные функции пока можно описать как функции-заглушки, например, как

```
BOOL MY_DIALOG::OnInitDialog( )
{
    CDialog::OnInitDialog( );
    return TRUE;
};
afx_msg void MY_DIALOG::OnOK ( )
{
    MessageBox( "OnOK","OnOK" );
};
```

4.2. Необходимо скорректировать карту сообщений

```
BEGIN_MESSAGE_MAP(MY_DIALOG, CDialog)
    ON_COMMAND (IDOK, OnOK)
END_MESSAGE_MAP( )
```

5. В классе приложения APPLICATION необходимо выполнить инициализацию общих элементов управления

```
BOOL APPLICATION::InitInstance( )
{
    InitCommonControls( );
     ...
}  .
```

6. В классе окна необходимо настроить индикатор (здесь объект TheProgress):

```
BOOL MY_DIALOG::OnInitDialog( )
{
    CDialog::OnInitDialog( );
    InitCommonControls( );
    // задание размеров индикатора
    RECT TheRect;
    TheRect.bottom = 60;
    TheRect.left = 10;
    TheRect.right = 150;
    TheRect.top = 50;
    // создание индикатора
    TheProgress.Create(WS_VISIBLE| WS_CHILD  WS_BORDER , TheRect, this , 2000) ;
    // задание параметров индикатора
    TheProgress.SetRange(0,120);
    TheProgress.SetStep(5);
    m_ProgPos = 0;
    return TRUE;
};  .
```

7. Описать обработчик

```
afx_msg void MY_DIALOG::OnOK( )
{
    TheProgress.StepIt( );
    m_ProgPos += 5;
    if (m_ProgPos >= 120)
          EndDialog( 0 );
};  .
```

8. Подключить диалоговое окно к приложению – оно будет запускаться автоматически при запуске приложения.

```
BOOL APPLICATION::InitInstance ( )
{
    InitCommonControls ( );
    MY_DIALOG     TheDialog ( ( LPTSTR ) IDD_DIALOG1, NULL );
```

```
    TheDialog.DoModal ( );
    return TRUE;
}
```

3. ИСПОЛЬЗОВАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО МЕНЮ

3.1. Общие сведения

Меню представляет собой набор пунктов и может рассматриваться как определенным образом организованное множество (система) кнопок, где каждому пункту меню соответствует кнопка. Кнопки-пункты объединяются в подменю. Пользователь может перемещаться от меню верхнего уровня к подменю нижних уровней. Кнопки, пункты меню бывают разных типов. Это, например, пункты типа:

- конечный пункт MENUITEM;
- "всплывающий" пункт POPUP (pop-up в Visual Studio C++);
- разделитель пунктов MENUBARBREAK (separator в Visual Studio C++).

При этом кнопки меню типа POPUP автоматически при нажатии обеспечивают переход к подменю более низкого уровня – другим кнопкам меню в соответствии с его структурой. Они поддерживают навигацию пользователя в системе меню.

А конечные кнопки типа MENUITEM работают как командные кнопки (например, как кнопки диалоговых окон OK, Cancel и т.д.). Их нажатие приводит к генерации сообщения типа WM_COMMAND, чувствительность к которому включается макрокомандой

ON_COMMAND(< *ID_пункта_меню* >, < *ИМЯ_обработчика_пункта_меню* >) ,

и, в конечном итоге, приводит к запуску соответствующего выбранному пункту меню обработчика.

Стили меню и пунктов меню приведены в Приложении 5.

Общая схема создания меню включает следующий набор действий.

1. Проектирование меню.

1.1. Описание общих свойств пользовательского меню. Например, описание названия меню, определение необходимости динамической загрузки меню или постоянного размещения в памяти и т.п.

1.2. Разработка структуры пользовательского меню, состава пуктов и их соподчинения. Для каждого пункта меню следует определить:

- тип пункта (MENUITEM, POPUP, MENUBARBREAK);
- свойства пункта, включая название пункта (caption) с отметкой при необходимости предваряющим символом & (например, F&ile) наличия горячей клавиши. Определить необходимость автоматической отметки выбранного пользователем пункта (свойство CHECKED), отсутствия подсветки пункта в случае отсутствия выбора (свойство GRAYED), временной неактивности, недоступности пункта для выбора (свойство INACTIVE);
- желаемую реакцию приложения. Для выбранного конечного пункта меню это набор действий, описываемых в обработчике. Для выбранного "всплывающего" пункта меню это активизируемое подменю и т.д.

2. Описание меню в файле ресурсов, например, с помощью редактора ресурсов аналогично тому, как, например, описываются ресурсы типа "окно".

3. Подключение меню к окну приложения. Например, через соответствующий параметр метода create.

4. Настройка класса окна приложения на работу с пользовательским меню.

4.1. В описание класса окна следует вставить прототипы (например, afx_msg void On<ИмяОбработчика> ( )) функций-обработчиков сообщений для всех событий типа "выбор конечного пункта меню".

4.2. Описать функции-обработчики сообщений события "выбор конечного пункта меню".

4.3. Включить чувствительность окна к сообщениям события "выбор конечного пункта меню" с помощью размещения в карте окна макрокоманд типа ON_COMMAND (<ID_ПунктаМеню>, < ИмяОбработчика >).

3.2. Приложения с пользовательским меню

ЗАДАНИЕ № 3. *Создать приложение на базе ТКП с пользовательским меню.*

При выборе каждого конечного пункта меню должно выводиться подтверждающее сообщение. Завершение приложения производится закрытием главного окна либо выбором первого пункта меню. Предполагается, что визуализация окна с меню должна происходить автоматически каждый раз при запуске приложения.

Приложение создается на базе типового каркаса MFC-приложения, а для создания ресурса – меню используется встроенный редактор ресурсов как альтернатива описанию меню текстом в файле ресурсов.

Для создания приложения необходимо выполнить следующие действия.

1. Спроектировать приложение.

1.1. Разработать интерфейс приложения. Здесь это главное окно с пользовательским меню, как показано на рисунке ниже.

1.1.1. Определить состав меню, свойства пунктов, их вид. Структура меню представлена на рисунке ниже. Меню включает главное и два подменю:

- меню верхнего уровня из двух пунктов (Пункт_1, Пункт_2);
- подменю Пункта_2 из двух пунктов (Пункт_2.1, Пункт_2.2);
- подменю Пункта_2.2 из одного пункта (Пункт_2.2.1).

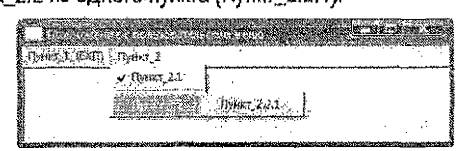

Рисунок 12 – Интерфейс приложения

Соответственно пункты меню Пункт_2, Пункт_2.2 – выпадающие пункты типа Pop-up, содержащие подпункты. А остальные пункты меню – конечные типа MenuItem. Для разделения пунктов меню Пункт_2.1 и 2.2 в подменю используется разделитель (пункт типа Separator). Для отметки выбора пункта меню Пункт_2.1 используется флажок (устанавливается свойство Checked).

1.1.2. Специфицировать состав событий-сообщений при работе с меню и реакций приложения на эти сообщения.

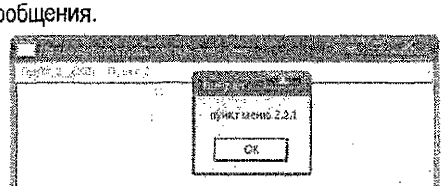

Рисунок 13 – Выбор пункта меню

События-сообщения при работе с меню здесь относятся к типу события "нажатие кнопки выбранного пункта меню мышью или с помощью "горячей" клавиши", что вызывает сообщение WM_COMMAND.

Поскольку здесь три конечных пункта меню, то будут использоваться обработчики, например, с прототипами

afx_msg void OnMenuItem_1 ( );
afx_msg void *OnMenuItem_2_1* ( );
afx_msg void *OnMenuItem_2_2_1* ( );

и соответствующие макрокоманды включения. По условию выбор каждого конечного пункта меню кроме первого (Пункт_1), сопровождается подтверждением. Выбор первого пункта должен приводить к закрытию окна и приложения.

1.2. Спроектировать классы приложения. Здесь используется типовой каркас с двумя классами:

- класс ПРИЛОЖЕНИЕ (**class** *APPLICATION*: **public CWinApp**) для создания экземпляра приложения;
- класс ОКНО (здесь **class** *WINDOW*: **public CFrameWnd**) для создания объекта – главное окно приложения.

Состав классов иллюстрируется рисунком ниже.

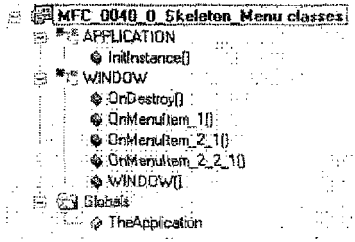

Рисунок 14 – Состав классов приложения

1.3. Спроектировать модульную структуру приложения.

2. Подготовить каркас MFC-приложения с файлом ресурсов.

2.1. Создать типовой каркас приложения.

2.2. Создать и подключить файл описания ресурсов к приложению посредством команды #include "resource.h". Соответственно после создания файла ресурсов *.rc в папке проекта появятся файлы main.rc, resource.h. В главном меню в пункте View активизируются пункты меню, связанные с работой с классами и ресурсами: ClassWizard, ID=ResourceSymbols, ResourceIncludes. Следует выполнить приложение и убедиться в его работоспособности.

3. Создать новый ресурс – меню.

3.1. Для этого надо добавить ресурс к проекту командой главного меню Insert-Resource-Menu и создать меню, пункты меню в редакторе ресурсов. Состав ресурсов представлен на рисунке ниже.

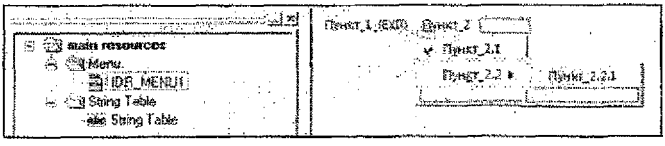

Рисунок 15 – Ресурсы приложения

3.2. Настроить свойства меню и его пунктов, используя окна свойств. Окно свойств меню представлено на рисунке ниже.

Рисунок 16 – Окно свойств меню

В окне свойств первого пункта меню (представлено на рисунке ниже) надо задать название пункта и установить ранее определенные свойства. В качестве дескриптора этого пункта задано – IDM_MenuItem_1.

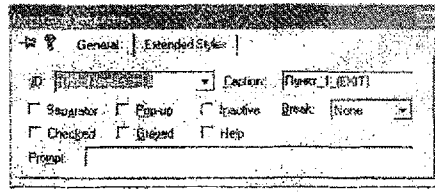

Рисунок 17 – Окно свойств пункта меню (Пункт_1)

Если задать значение в поле Prompt, то автоматически к ресурсам (в файл ресурсов main.rc) добавится ресурс типа StringTable, содержащий введенную строку.

Далее надо установить значения в окне свойств второго пункта меню (представлено на рисунке ниже). Пункт не является конечным, поэтому дескриптор отсутствует.

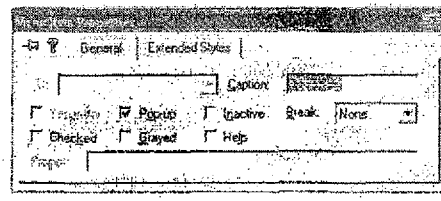

Рисунок 18 – Окно свойств пункта меню (Пункт_2)

Для выделения буквы горячей клавиши в названии пункта меню используется амперсанд, например, имя &Пункт_2 означает, что вызов пункта меню может также производиться комбинацией клавиш Alt-П.

Окно свойств пункта меню 2.1 представлено на рисунке ниже. Здесь для отметки пункта флажком устанавливается свойство Checked.

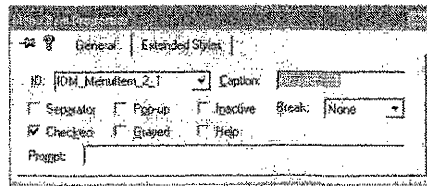

Рисунок 19 – Окно свойств пункта меню (Пункт_2.1)

Окно свойств разделителя (пункт Separator) пунктов меню 2.1 и 2.2 представлено на рисунке ниже.

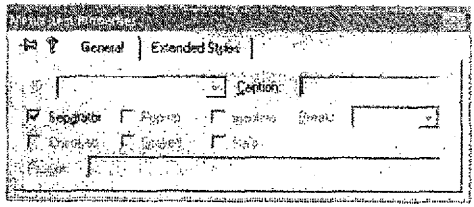

Рисунок 20 – Окно свойств пункта меню типа разделитель

Окно свойств других пунктов меню представлены на рисунках ниже.

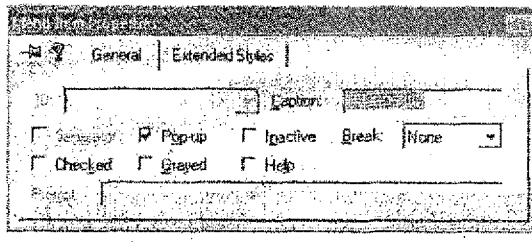

Рисунок 21 – Окно свойств пункта меню (Пункт_2.2)

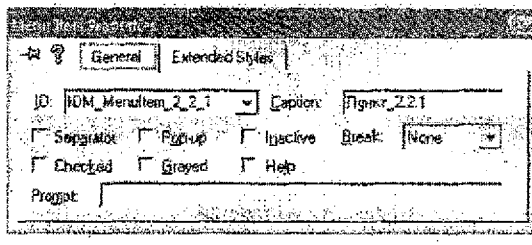

Рисунок 22 – Окно свойств пункта меню (Пункт_2.2.1)

Состав проекта приложения представлен на рисунке 23.

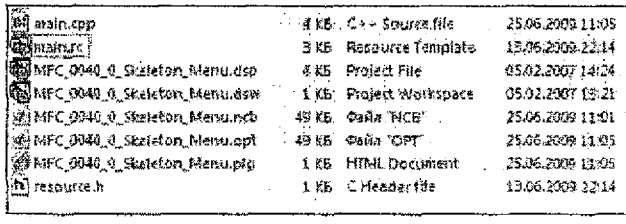

Рисунок 23 – Состав проекта приложения

При выполнении приложения файлы автоматически обновятся. Заголовочный файл описаний (resource.h) можно увидеть, использовав пункты главного меню View ID=ResourceSymbols.

Содержимое файла описаний показано на рисунке 24. Здесь видно, например, что к меню можно обращаться по числовому дескриптору 101 или используя его поименованный эквивалент IDR_MENU1.

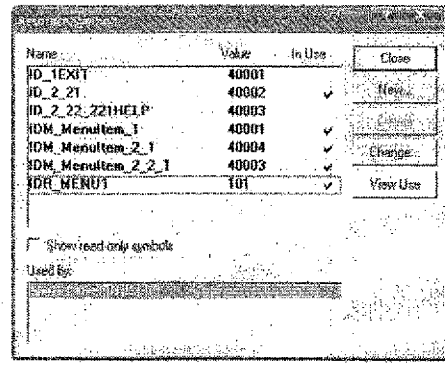

Рисунок 24 – Файл описаний

Если произвести компиляцию приложения в настоящем виде, то при выполнении приложения меню не будет визуализировано, так как оно, как ресурс, пока существует отдельно от приложения.

Содержимое ресурсного файла (здесь main.rc), находящегося в папке проекта, можно просмотреть, например, в текстовом редакторе Word. Фрагмент содержимого ресурсного файла для рассматриваемого приложения приведен ниже.

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
...
// Menu
IDR_MENU1              MENU DISCARDABLE
BEGIN
  MENUITEM      "Пункт_1_(EXIT)",  IDM_MenuItem_1
  POPUP         "&Пункт_2"
  BEGIN
    MENUITEM    "П&ункт_2.1",      IDM_MenuItem_2_1, CHECKED
    MENUITEM    SEPARATOR
    POPUP       "Пун&кт_2.2"
    BEGIN
      MENUITEM  "Пункт_2.2.1",     IDM_MenuItem_2_2_1
    END
  END
END  .
```

4. Далее следует подключить меню к приложению – к соответствующему окну приложения, используя 6-й параметр метода create.

Пусть имя ресурса "меню" (его дескриптор) определено в файле resource.h командой #define как

```
#define IDR_MENU1 101  ,
```

то есть можно использовать или номер 101, или его макроопределение IDR_MENU1. Для подключения меню в конструкторе класса ОКНО следует указать

```
ОКНО::ОКНО ( )
{
...
   Create ( NULL, "Приложение с пользовательским меню",
            WS_OVERLAPPEDWINDOW, rectDefault, NULL, (LPTSTR) 101 );
```

или

```
   Create ( NULL, "Приложение с пользовательским меню",
            WS_OVERLAPPEDWINDOW, rectDefault, NULL, (LPTSTR) IDR_MENU1 );
...
}
```

В результате меню будет загружаться автоматически при запуске приложения, однако пункты меню будут неактивными и изображаются "бледно".

5. Далее необходимо настроить класс ОКНО (здесь WINDOW) на работу с пользовательским меню.

5.1. В описание класса окна для каждого конечного пункта меню надо вставить прототипы указанных ранее функций-обработчиков сообщений события "выбор пункта меню".

5.2. Описать функции-обработчики. Например, обработчик первого пункта меню

```
afx_msg void WINDOW:: OnMenuItem_1 ( )
{
    MessageBox ( "пункт меню 1 ", " Выбран  " );
    SendMessage ( WM_CLOSE );
};
```

5.3. Включить чувствительность класса к сообщениям события "выбор пункта меню" путем внесения в карту сообщений окна следующих макрокоманд

```
BEGIN_MESSAGE_MAP(WINDOW,CFrameWnd)
    ON_COMMAND( IDM_MenuItem_1, OnMenuItem_1 )
    ON_COMMAND( IDM_MenuItem_2_1, OnMenuItem_2_1 )
    ON_COMMAND( IDM_MenuItem_2_2_1, OnMenuItem_2_2_1 )
END_MESSAGE_MAP()
```

6. Откомпилировать и выполнить приложение.

3.3. Приложения с пользовательским меню и диалоговыми окнами

ЗАДАНИЕ № 4. *Создать приложение с пользовательским меню, предназначенное для многократного ввода строк.*

При выборе пункта меню Enter должно выводиться диалоговое окно для ввода новой строки. При выборе пункта Display должна отображаться последняя введенная строка. Завершение работы приложения производится закрытием главного окна либо выбором пункта меню Exit.

Для создания приложения необходимо выполнить следующие действия.

1. Спроектировать приложение.

1.1. Спроектировать интерфейс приложения.

1.1.1. Разработать структуру окон, структуру меню, свойства пунктов меню, их вид. Здесь будут использованы:

- главное окно с пользовательским меню;
- диалоговое окно для ввода строки;
- окна сообщений для вывода строки и сообщений.

Интерфейс приложения представлен на рисунках ниже.

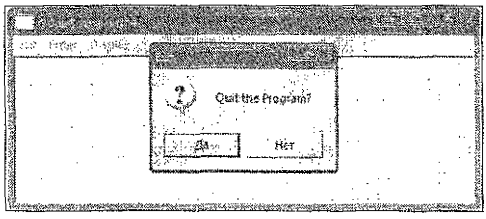

Рисунок 25 – Интерфейс приложения. Завершение работы

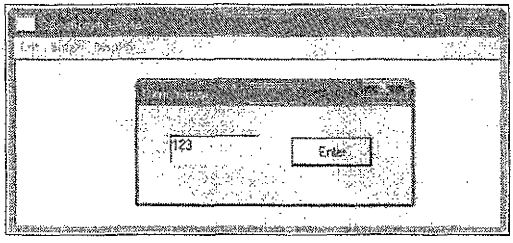

Рисунок 26 – Интерфейс приложения. Ввод строки

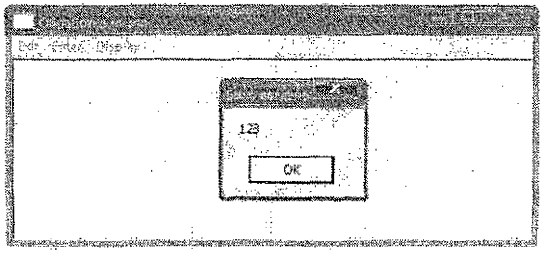

Рисунок 27 – Интерфейс приложения. Вывод строки

1.1.2. Специфицировать состав событий-сообщений.

При работе с меню это события типа "нажатие кнопки выбранного пункта меню", приводящие к сообщениям WM_COMMAND. Здесь три конечных пункта меню, соответственно будут использоваться обработчики с прототипами

afx_msg void *OnExit*( );
afx_msg void *OnEnter*( );
afx_msg void *OnDisplay*( );

и соответствующие макрокоманды включения.

При работе с диалоговым окном ввода это события нажатия кнопки Enter, так же приводящие к сообщениям WM_COMMAND.. Прототип обработчика

afx_msg void *OnEnter*( ); .

1.2. Спроектировать классы приложения. Здесь используются следующие классы:

- класс ПРИЛОЖЕНИЕ (class APPLICATION: public CWinApp) для создания экземпляра приложения;
- класс ГЛАВНОЕ_ОКНО (class WINDOW: public CFrameWnd) для создания объекта – главное окно приложения;
- класс ДИАЛОГОВОЕ_ОКНО (class DIALOG_WINDOW: public CDialog) для создания объекта – окно ввода.

Примерный состав классов представлен на рисунке ниже.

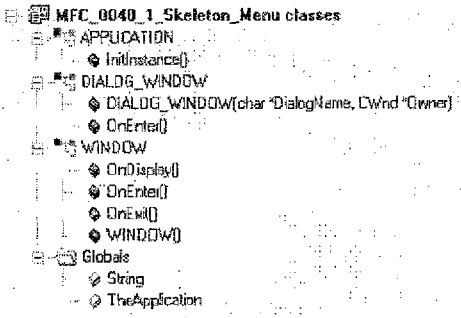

Рисунок 28 – Классы приложения

1.3. Спроектировать модульную структуру приложения (представлена на рисунке 29).

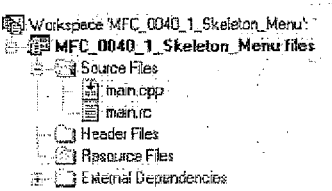

Рисунок 29 – Структура приложения

2. Необходимо создать каркас MFC-приложения с файлом ресурсов, обеспечить подключение заголовочных файлов и описание глобальной переменной для работы с данными

```
#include <afxwin.h>
#include <iostream>
#include "resource.h"
using namespace std;
char String[80];  .
```

3. Создать ресурсы приложения.

3.1. Создать ресурс – меню, настроить свойства меню и его пунктов, как показано на рисунках ниже.

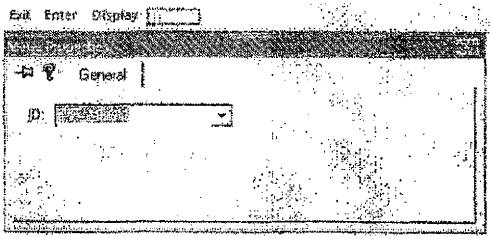

Рисунок 30 – Свойства меню

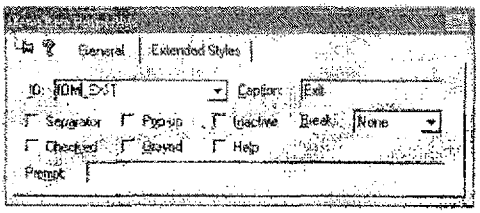

Рисунок 31 – Свойства пункта меню Exit

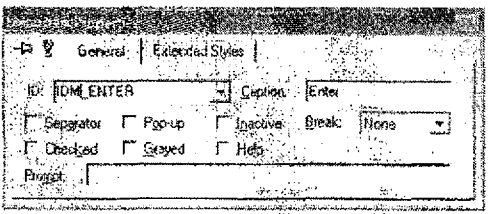

Рисунок 32 – Свойства пункта меню Enter

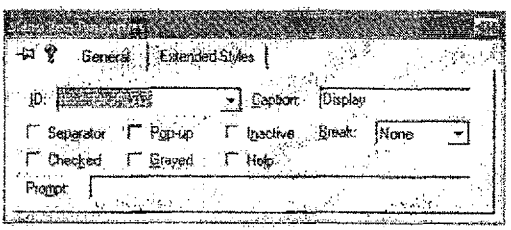

Рисунок 33 – Свойства пункта меню Display

3.2. Создать новый ресурс – диалоговое окно, настроить свойства окна и его элементов, как показано на рисунках ниже.

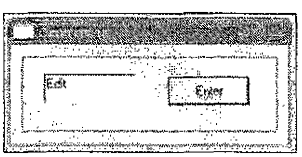

Рисунок 34 – Окно ввода

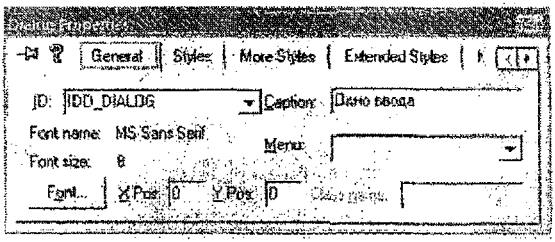

Рисунок 35 – Свойства окна ввода

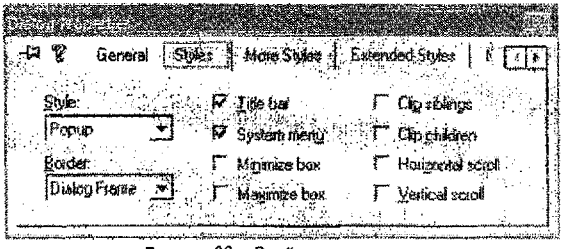

Рисунок 36 – Свойства окна ввода

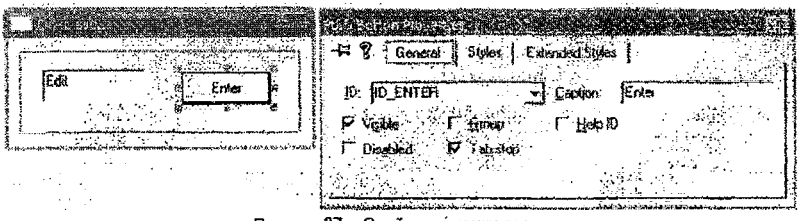

Рисунок 37 – Свойства кнопки ввода

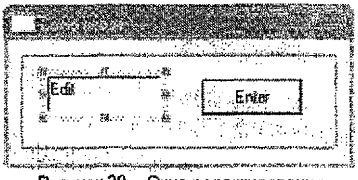

Рисунок 38 – Окно редактирования

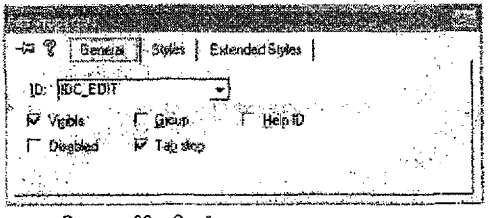

Рисунок 39 – Свойства окна редактирования

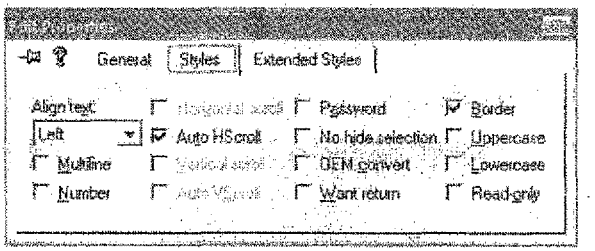

Рисунок 40 – Свойства окна редактирования

Состав проекта приложения, включая состав ресурсов, представлен на рис. 41 и 42. Фрагмент содержимого ресурсного файла resource.h описан ниже:

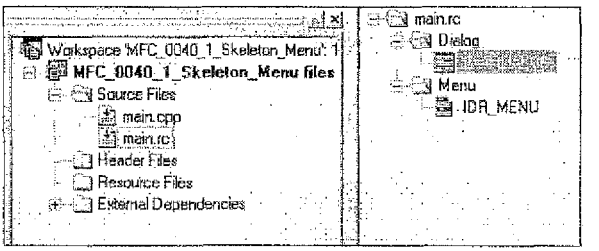

Рисунок 41 – Состав ресурсов

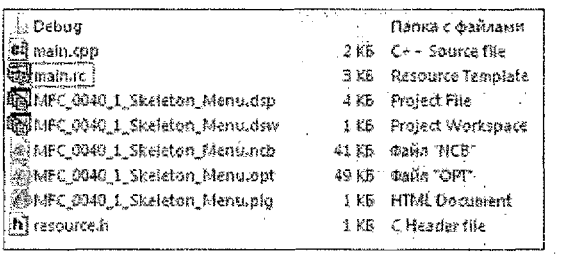

Рисунок 42 – Состав проекта

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
...
// Dialog
IDD_DIALOG DIALOG DISCARDABLE  0, 0, 171, 55
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Окно ввода"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "Enter",ID_ENTER,97,18,50,16
    EDITTEXT        IDC_EDIT,19,16,58,17,ES_AUTOHSCROLL
END
```

```
...
// Menu
IDR_MENU MENU DISCARDABLE
BEGIN
   MENUITEM "Exit",                IDM_EXIT
   MENUITEM "Enter",               IDM_ENTER
   MENUITEM "Display",             IDM_DISPLAY
END
...
```

4. Необходимо подключить меню (дескриптор IDR_MENU) к приложению – к главному окну приложения (класс ГЛАВНОЕ_ОКНО). Для этого в конструкторе класса ГЛАВНОЕ_ОКНО следует указать

```
ГЛАВНОЕ_ОКНО:: ГЛАВНОЕ_ОКНО ( )
{
...
      Create ( NULL, "Обработка числа", WS_OVERLAPPEDWINDOW, rectDefault,
            NULL, (LPTSTR) IDR_MENU );
...
}  .
```

5. Далее надо настроить классы приложения ГЛАВНОЕ_ОКНО и ДИАЛОГОВОЕ_ОКНО.

5.1. Для настройки класса ДИАЛОГОВОЕ_ОКНО (DIALOG_WINDOW) в него следует включить прототипы функций-обработчиков сообщений события "выбор пункта меню"

```
class ДИАЛОГОВОЕ_ОКНО: public CDialog
{
public:
    ДИАЛОГОВОЕ_ОКНО (char *DialogName, CWnd *Owner):
          CDialog (DialogName, Owner) {  };
    afx_msg void OnEnter ( );
    DECLARE_MESSAGE_MAP()
};  ,
```

описать функцию-обработчик, например, как

```
afx_msg void ДИАЛОГОВОЕ_ОКНО::OnEnter( )
{
    CEdit *ptr = (CEdit *) GetDlgItem(IDC_EDIT);
    ptr->GetWindowText(String, sizeof String-1);
};
```

и включить чувствительность класса к соответствующим сообщениям

```
BEGIN_MESSAGE_MAP(ДИАЛОГОВОЕ_ОКНО, CDialog)
    ON_COMMAND(ID_ENTER, OnEnter)
END_MESSAGE_MAP()  .
```

5.2. Для настройки класса ГЛАВНОЕ_ОКНО (WINDOW) на работу с пользовательс
меню в него следует включить прототипы функций-обработчиков сообщений соб
"выбор пункта меню"

```
class ГЛАВНОЕ_ОКНО: public CFrameWnd
{
public:
    ГЛАВНОЕ_ОКНО ( );
    afx_msg void OnExit( );
    afx_msg void OnEnter( );
    afx_msg void OnDisplay( );
    DECLARE_MESSAGE_MAP()
};
```

описать функции-обработчики, например,

```
afx_msg void ГЛАВНОЕ_ОКНО::OnEnter( )
{
    ДИАЛОГОВОЕ_ОКНО  MyDataEnterDialog((LPTSTR)IDD_DIALOG ,this);
    MyDataEnterDialog.DoModal( );
};

afx_msg void ГЛАВНОЕ_ОКНО::OnDisplay( )
{
    MessageBox(String,"Было введено: ");
};

afx_msg void ГЛАВНОЕ_ОКНО::OnExit( )
{
    int Answer;
    Answer = MessageBox("Quit the Program?","Exit", MB_YESNO
                        | MB_ICONQUESTION);
    if (Answer == IDYES)
        SendMessage(WM_CLOSE);
};
```

и включить чувствительность класса к соответствующим сообщениям

```
BEGIN_MESSAGE_MAP(ГЛАВНОЕ_ОКНО,CFrameWnd)
    ON_COMMAND(IDM_EXIT, OnExit)
    ON_COMMAND(IDM_ENTER, OnEnter)
    ON_COMMAND(IDM_DISPLAY, OnDisplay)
END_MESSAGE_MAP()
```

6. Создать класс ПРИЛОЖЕНИЕ (APPLICATION):

```
class ПРИЛОЖЕНИЕ : public CWinApp
{
public:
    BOOL InitInstance();
};
```

```
BOOL ПРИЛОЖЕНИЕ:: InitInstance()
{
    m_pMainWnd =  new WINDOW;
    m_pMainWnd -> ShowWindow(m_nCmdShow);
    m_pMainWnd -> UpdateWindow();
    return TRUE;
} ,
```

объект приложения

ПРИЛОЖЕНИЕ МоеПриложение; .

Далее необходимо приложение откомпилировать и выполнить.

3.4. Задания для самостоятельного выполнения

1. Создать приложение с пользовательским меню. Интерфейс приложения представлен рисунками 43-46.

Завершение работы приложения производится закрытием главного окна или выбором пункта меню Exit и затем Quit. Выбор пункта Help приводит к выводу окна с описанием приложения. Аналогично работает пункт Message box.

Пункты меню Buttons и Edit запускают соответствующие диалоговые окна – "Командные кнопки" и "Ввод данных". В окне" Командные кнопки" при выборе Первой или Второй кнопки выводится соответствующее сообщение, а нажатие кнопок ЭТО ВСЕ и ЗАКРЫТЬ ведет просто к завершению работы с этим диалоговым окном. В окне "Ввод данных" по нажатию кнопки ОК следует запомнить введенное значение, а нажатие кнопки CANCEL ведет к завершению работы с этим диалоговым окном (с требованием подтверждения завершения работы).

Предполагается, что визуализация окна с меню должна происходить автоматически каждый раз при запуске приложения.

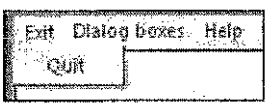

Рисунок 43 – Интерфейс приложения

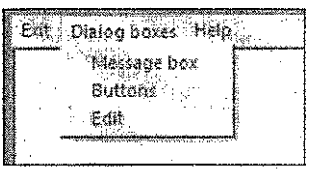

Рисунок 44 – Подменю Dialog boxes

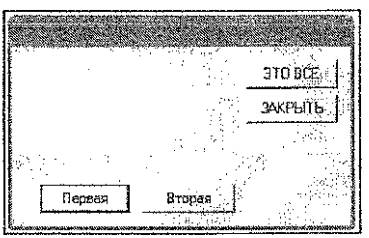

Рисунок 45 – Диалоговое окно "Командные кнопки"

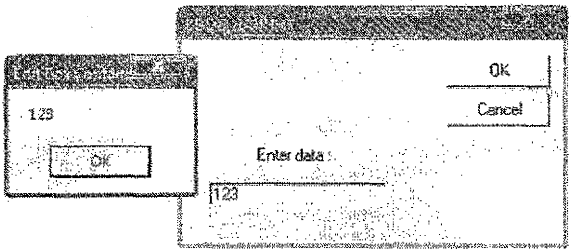

Рисунок 46 – Диалоговое окно "Ввод данных"

2*. Модифицировать предыдущее приложение для вывода справочных данных о приложении в элементе управления список с возможностью "прокручивания" текста.

3*. Модифицировать исходное приложение. В главном окне меню должно включать только два пункта Завершение и Работа. При выборе пункта Работа производить переход в следующее окно типа главное с меню как в исходном приложении. Однако при выборе пункта Quit должен происходить возврат в исходное окно.

4*. Модифицировать исходное приложение. Вызов окна "Ввод данных" и окна справки производить через окно "Командные кнопки".

ПРИМЕЧАНИЕ: пункты, помеченные *, выполняются по указанию преподавателя.

4. ПОРЯДОК ВЫПОЛНЕНИЯ

Цель работы:

- изучить технологию работы с пользовательским меню в составе главного окна, совместное использование меню, диалоговых окон, элементов управления;

- изучить использование спискового элемента управления диалогового окна и программно создаваемого элемента управления на примере индикатора.

Содержание отчета:

- описание особенностей создания и управления меню, списков, индикаторов (описание классов, методов, событий и сообщений);

- описание приложений – результатов выполнения самостоятельных заданий.

Примечание:

- для всех заданий в отчете следует приводить диаграммы классов и диаграммы компонентов (в нотации UML) для описания структуры приложений.

Порядок выполнения.

1. Изучить свойства пользовательских меню (см. § 3.1).

2. **Создать** MFC-приложение на базе типового каркаса пользователя (ТКП), содержащее пользовательское меню с обработчиками, выводящими соответствующие комментирующие сообщения при выборе каждого конечного пункта меню (см. § 3.2 – задание № 3).

3. **Создать** MFC-приложение на базе ТКП, управляемое пользовательским меню, использующее диалоговые окна и позволяющее вводить и выводить значение. За основу взять демонстрационный пример (см. § 3.3 – задание № 4).

4*. **Модифицировать** предыдущий пример для вывода квадрата введенного значения в отдельном диалоговом окне.

5*. **Модифицировать** предыдущий пример: все действия по вводу-выводу значения выполнять в общем диалоговом окне ВВОД-ВЫВОД ДАННЫХ. Режим работы этого окна задавать кнопками: по кнопке ВВОД устанавливать режим ввода, а по кнопке ВЫВОД устанавливать режим вывода.

6. Изучить элемент управления «список» (см. § 2.1).

7. **Создать** MFC-приложение на базе ТКП для ведения списка строк – фамилий (см. § 2.2 – задание № 1). Интерфейс приложения реализовать в виде диалогового окна со списком фамилий и кнопками управления списком.

8. **Выполнить** самостоятельное задание (§ 2.3, п. 1 и 3) в виде одного MFC-приложения на базе ТКП.

9.* **Выполнить** самостоятельное задание (§ 2.3, п. 6).

10.* **Модифицировать** предыдущее (§ 2.3, п. 8) приложение, реализовав поддержку списковых, файловых и др. операций с помощью пользовательских классов.

11. Изучить элемент управления индикатор (см. § 2.4).

12. **Создать** MFC-приложение с индикатором (см. § 2.5 – задание № 2).

ЛИТЕРАТУРА

1. Паппас, К. Visual C++. Руководство для профессионалов; Пер. с англ. / К. Паппас, У. Мюррей. – СПб: BHV – Санкт-Петербург, 1996.

2. Орлов, С.А. Технологии разработки программного обеспечения: учебник для вузов. – СПб.: Питер, 2004. – 527 с.

3. Паппас, К. Эффективная работа: Visual C++.NET / К. Паппас, У. Мюррей. – СПб.: Питер, 2002. – 816 с.

Дополнительная литература

4. Страуструп, Б. Язык программирования СИ++. М.: Радио и связь, 1991. – 352 с.

5. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++, 2-е изд.; пер. с англ. – М.: Издательство БИНОМ, СПб: Невский диалект, 1998. – 560 с.

6. Франка, П. C++: учебный курс. – СПб.: Питер, 2005. – 522 с.

7. Шилдт, Г. Самоучитель C++, 3-е изд. – СПб.: БХВ-Петербург, 2003. – 688 с.

8. Поляков, А.Ю. Методы и алгоритмы компьютерной графики в примерах на Visual C++ / А.Ю. Поляков, В.А. Брусенцев. – СПб.: БХВ-Петербург, 2003. – 560 с.

9. Мюррей, У. Создание переносимых приложений для Windows / У. Мюррей, К. Паппас. – СПб.: BHV – Санкт-Петербург, 1997. – 816 с.

10. Финогенов, К.Г. Win32. Основы программирования. – М.: ДИАЛОГ-МИФИ, 2002. – 416 с.

11. Шилдт, Г. Справочник программиста по C/C++, 3-е изд. – М.: Изд. Дом Вильямс, 2003. – 432 с.

12. Шмуллер, Дж. Освой самостоятельно UML за 24 часа. – М.: Изд. Дом Вильямс, 2002. – 352 с.

13. Павловская? Т.А. C++. Объектно-ориентированное программирование: практикум / Т.А. Павловская, Ю.А. Щупак. – СПб.: Питер, 2004. – 265 с.

ПРИЛОЖЕНИЕ 1. Стили элемента управления «список»

Стиль	Описание
LBS_DISABLENOSCROLL	- вертикальная полоса прокрутки не отображается, если в списке недостаточно элементов, чтоб их "прокручивать"
LBS_EXTENDEDSEL	- позволяет пользователю выбирать сразу несколько элементов, строк, используя клавишу SHIFT, "мышь" или горячие клавиши
LBS_MULTICOLUMN	- многоколонковый список с возможностью горизонтальной прокрутки его содержимого (ширина колонки устанавливается с помощью сообщения LB_SETCOLUMNWIDTH)
LBS_MULTIPLESEL	- допускается выбор произвольного количества элементов, строк списка. При этом выбранная строка меняется каждый раз, когда пользователь выполняет "щелчок" на строке
LBS_NOSEL	- пользователь может просматривать список, но не может выбирать строки
LBS_NOTIFY	- при одинарном или двойном "щелчке" на строке посылается уведомляющее сообщение (LBN_DBLCLK, LBN_SELCANCEL, LBN_SELCHANGE)
LBS_SORT	- строки сортируются в алфавитном порядке, при одинарном или двойном "щелчке" на строке посылается уведомляющее сообщение
LBS_STANDARD	- строки сортируются в алфавитном порядке. Родительское окно получает сообщение ввода каждый раз при одинарном или двойном "щелчке" на строке. Сам список – с рамкой и полосой прокрутки
WS_TABSTOP	- используется в диалоговых окнах, позволяя пользователю перемещаться между элементами управления этого стиля с помощью клавиши tab

ПРИЛОЖЕНИЕ 2. Команды (методы) управления списком

CListBox	- конструктор, создает объект класса CListBox
Create	- инициализатор, создает элемент управления Windows типа list box и прикрепляет его к объекту класса CListBox
InitStorage	- распределяет память для элементов списка
GetCount	- общий метод, возвращает число строк в списке
GetTopIndex	- общий метод, возвращает номер первой видимой строки
GetItemData	- общий метод, возвращает 32-битовое значение элемента
SetItemData	- общий метод, устанавливает 32-битовое значение, ассоциируемое с элементом списка
GetSel	- общий метод, возвращает состояние выбора элемента
GetText	- общий метод, читает элемент списка в буфер
GetTextLen	- общий метод, возвращает длину в байтах элемента списка
GetCurSel	- возвращает номер выбранной строки
SetCurSel	- для списка с одиночным выбором выделяет или снимает выделение строки
SetSel	- для списка с множественным выбором выделяет или снимает выделение строк
GetSelCount	- для списка с множественным выбором возвращает число выделенных строк
AddString	- строковый метод, добавляет новую строку в список
DeleteString	- строковый метод, удаляет строку из списка
InsertString	- строковый метод, вставляет новую строку в указанное место списка
FindString	- строковый метод, обеспечивает поиск строки
FindStringExact	- строковый метод, обеспечивает поиск первой строки, которая совпадает с образцом
SelectString	- строковый метод, ищет и выделяет строку

ПРИЛОЖЕНИЕ 3. Приложение с элементом управления «список»

```
#include <afxwin.h>
#include <iostream>
#include "resource.h"
using namespace std;

class myDIALOG : public CDialog
{
public:
        myDIALOG(char *DialogName, CWnd *Owner): CDialog(DialogName, Owner)
{       };
BOOL OnInitDialog();
afx_msg void OnToDisplaySelectedFIO();
afx_msg void OnToAddFIO();
        afx_msg void OnToDeleteFIO();
DECLARE_MESSAGE_MAP()
};

BEGIN_MESSAGE_MAP(myDIALOG, CDialog)
        ON_LBN_DBLCLK(IDC_LIST1, OnToDisplaySelectedFIO)
        ON_COMMAND(IDC_BUTTON1, OnToAddFIO)
        ON_COMMAND(IDC_BUTTON2, OnToDeleteFIO)
END_MESSAGE_MAP()

afx_msg void myDIALOG::OnToDisplaySelectedFIO()
{
        char Str[80];
        MessageBox("Выбор двойным щелчком","РАБОТАЕТ ФУНКЦИЯ");
        CListBox *PListBox = (CListBox *) GetDlgItem(IDC_LIST1);
        int i = PListBox ->GetCurSel();
        PListBox ->GetText(i,Str);
        strcat(Str," - ваш выбор");
        MessageBox(Str,"РЕЗУЛЬТАТ ВЫБОРА");
};

afx_msg void myDIALOG::OnToAddFIO()
{
        static int j=0;
        char Str[80];char Str1[80];char Str2[80];
        j++;
        MessageBox("Добавить фамилию","РАБОТАЕТ ФУНКЦИЯ");
        CListBox *PListBox = (CListBox *) GetDlgItem(IDC_LIST1);
        int i = PListBox ->GetCurSel();
        if (i == LB_ERR )
        {
  MessageBox("Сделайте выбор","НАПОМИНАНИЕ",MB_OK | MB_ICONWARNING);
        }
        else
        {
            PListBox ->GetText(i,Str);
            strcpy(Str1,Str);
```

```
        strcat(Str1," - ваш выбор");
        MessageBox(Str1,"РЕЗУЛЬТАТ ВЫБОРА На ДОБАВЛЕНИЕ");
        wsprintf(Str2,"%d",j);
        strcat(Str,Str2);
        PListBox -> AddString(Str);
};
};

afx_msg void myDIALOG::OnToDeleteFIO()
{
char Str[80];
MessageBox("Удалить фамилию","РАБОТАЕТ ФУНКЦИЯ");
CListBox *PListBox = (CListBox *) GetDlgItem(IDC_LIST1);
        int i = PListBox ->GetCurSel();
if (i == LB_ERR )
{
        MessageBox("Выбирайте!","НАПОМИНАНИЕ", MB_OK|MB_ICONWARNING);
}
else
{
        i = PListBox ->GetCurSel();
        PListBox ->GetText(i,Str);
        strcat(Str," - удалять ?");
                if (MessageBox(Str,"РЕЗУЛЬТАТ ВЫБОРА ", MB_OKCANCEL) == IDOK)
        {
                PListBox ->DeleteString(i);
        };
};
};

BOOL myDIALOG::OnInitDialog()
{
CDialog::OnInitDialog();
CListBox *PListBox = (CListBox *) GetDlgItem(IDC_LIST1);
PListBox -> AddString("Иванов");
PListBox -> AddString("Петров");
        return TRUE;
};

class myAPPLICATION:public CWinApp
{
public:
BOOL InitInstance();
};

BOOL myAPPLICATION::InitInstance()
{
myDIALOG TheDialog((LPTSTR)IDD_DIALOG1,NULL);
TheDialog.DoModal();
return TRUE;
}

myAPPLICATION  ThisApplication;
```

ПРИЛОЖЕНИЕ 4. Приложение с программируемыми элементами управления

```
#include <afxwin.h>
#include <afxcmn.h>
#include <iostream>
#include "resource.h"
using namespace std;

class MY_DIALOG : public CDialog
{
        CProgressCtrl TheProgress;
        int m_ProgPos;
public:
     MY_DIALOG(char *DialogName, CWnd *Owner): CDialog(DialogName,  Owner)
        {
            InitCommonControls();
        };
        BOOL OnInitDialog();
        afx_msg void OnOK();
        DECLARE_MESSAGE_MAP()
};

BEGIN_MESSAGE_MAP(MY_DIALOG, CDialog)
   ON_COMMAND(IDOK, OnOK)
END_MESSAGE_MAP()

afx_msg void MY_DIALOG::OnOK()
{
        TheProgress.StepIt();
        m_ProgPos += 5;
        if (m_ProgPos >= 120)
              EndDialog(0);
};

BOOL MY_DIALOG::OnInitDialog()
{
        CDialog::OnInitDialog();
        InitCommonControls();
        RECT TheRect;
        TheRect.bottom= 60;
        TheRect.left = 10;
        TheRect.right = 150;
        TheRect.top = 50;
        TheProgress.Create(WS_VISIBLE | WS_CHILD | WS_BORDER ,
                                TheRect, this , 2000 );
        TheProgress.SetRange(0,120);
```

```
        TheProgress.SetStep(5);
        m_ProgPos = 0;
        return TRUE;
};

class APPLICATION:public CWinApp
{
public:
        BOOL InitInstance();
};

BOOL APPLICATION::InitInstance()
{
        InitCommonControls();
        MY_DIALOG TheDialog((LPTSTR)IDD_DIALOG1,NULL);
        TheDialog.DoModal();
        return TRUE;
}

APPLICATION App;
```

ПРИЛОЖЕНИЕ 5. Стили меню, пунктов меню

Стиль	Описание
CHECKED	- опция пункта, требует размещения рядом с выбранным пользователем пунктом меню отметки выбора (для пунктов меню верхнего уровня не используется)
GRAYED	- опция пункта, требует пометки пункта меню как неактивного (серым, "бледным" цветом). Такой пункт не может быть выбран пользователем
HELP	- опция пункта, определяет, что пункт меню может быть связан с командой вызова помощи (применяется только с пунктами меню типа MENUITEM)
INACTIVE	- опция пункта, требует, чтобы пункт меню выводился в списке меню, но не мог быть выбран в данных обстоятельствах
MENUBREAK	- опция пункта, определяет свойство, аналогичное MENUBARBREAK, но без использования разделительной черты
MENUBARBREAK	- опция пункта, используется для указания пункта меню, выполняющего роль разделителя для других пунктов. В меню верхнего уровня вызывает запись названия нового пункта с новой строки. В выпадающих меню название пункта будет размещено в новом столбце и отделено чертой
OWNERDRAW	- опция пункта, используется для указания, что состоянием и изображением пункта меню, включая выделенное, неактивное и отмеченное состояния, отвечает "владелец" меню
POPUP	- опция пункта, используется для указания типа пункта меню как POPUP. При выборе этого пункта выводится список пунктов подменю
DISCARDABLE	- опция меню, используется для указания, что меню может быть удалено из памяти, если оно больше не используется
FIXED	- опция меню, требует постоянного нахождения меню в памяти
LOADONCALL	- опция меню, используется для указания необходимости загрузки меню каждый раз при обращении к нему

ПРИЛОЖЕНИЕ 6. Приложение с пользовательским меню

```
#include <afxwin.h>
#include <iostream>
#include "resource.h"
using namespace std;

class WINDOW: public CFrameWnd
{
public:
        WINDOW();
        afx_msg void OnMenuItem_1 ();
        afx_msg void OnMenuItem_2_1 ();
        afx_msg void OnMenuItem_2_2_1 ();
        //afx_msg void OnDestroy ( );
        DECLARE_MESSAGE_MAP()
};

WINDOW::WINDOW()
{
        Create(NULL, "Приложение с пользовательским меню",
        WS_OVERLAPPEDWINDOW, rectDefault,
        NULL, (LPTSTR) 101);
        // или  NULL, (LPTSTR) IDR_MENU1);
}

afx_msg void WINDOW:: OnMenuItem_1 ( )
{
         MessageBox("пункт меню 1 "," Выбран ");
        SendMessage(WM_CLOSE);
};

afx_msg void WINDOW:: OnMenuItem_2_2_1 ( )
{
        MessageBox("пункт меню 2.2.1 "," Выбран ");
};

afx_msg void WINDOW:: OnMenuItem_2_1  ( )
{
        MessageBox("пункт меню 2.1 "," Выбран ");
};

BEGIN_MESSAGE_MAP(WINDOW,CFrameWnd)
        ON_COMMAND(IDM_MenuItem_1, OnMenuItem_1)
        ON_COMMAND(IDM_MenuItem_2_1, OnMenuItem_2_1)
        ON_COMMAND(IDM_MenuItem_2_2_1, OnMenuItem_2_2_1)
END_MESSAGE_MAP()
```

```
class APPLICATION : public CWinApp
{
public:
        BOOL InitInstance();
};

BOOL APPLICATION :: InitInstance()
{
        m_pMainWnd =  new WINDOW;
        m_pMainWnd -> ShowWindow(m_nCmdShow);
        m_pMainWnd -> UpdateWindow();
        return TRUE;
}

APPLICATION  TheApplication;
```

ПРИЛОЖЕНИЕ 7. Приложение с пользовательским меню и системой диалоговых окон

```
#include <afxwin.h>
#include <iostream>
#include "resource.h"
using namespace std;

char String[80];
class DIALOG_WINDOW : public CDialog
{
public:
        DIALOG_WINDOW(char *DialogName, CWnd *Owner):
        CDialog(DialogName, Owner){};
        afx_msg void OnEnter();
        DECLARE_MESSAGE_MAP()
};
BEGIN_MESSAGE_MAP(DIALOG_WINDOW, CDialog)
        ON_COMMAND(ID_ENTER, OnEnter)
END_MESSAGE_MAP()

afx_msg void DIALOG_WINDOW::OnEnter()
{
        CEdit *ptr = (CEdit *) GetDlgItem(IDC_EDIT);
        ptr->GetWindowText(String, sizeof String-1);
};
class WINDOW:public CFrameWnd
{
public:
        WINDOW();
        afx_msg void OnExit();
```

```
        afx_msg void OnEnter();
        afx_msg void OnDisplay();
        DECLARE_MESSAGE_MAP()
};

WINDOW::WINDOW()
{
        Create(NULL, "Обработка числа", WS_OVERLAPPEDWINDOW, rectDefault,
                NULL, (LPTSTR) IDR_MENU);
}

BEGIN_MESSAGE_MAP(WINDOW, CFrameWnd)
        ON_COMMAND(IDM_EXIT, OnExit)
        ON_COMMAND(IDM_ENTER, OnEnter)
        ON_COMMAND(IDM_DISPLAY, OnDisplay)
END_MESSAGE_MAP()

afx_msg void WINDOW::OnEnter()
{
        DIALOG_WINDOW MyDataEnterDialog((LPTSTR)IDD_DIALOG ,this);
        MyDataEnterDialog.DoModal();
};

afx_msg void WINDOW::OnDisplay()
{
        MessageBox(String,"Было введено: ");
};

afx_msg void WINDOW::OnExit()
{
        int Answer;
        Answer = MessageBox("Quit the Program?","Exit", MB_YESNO |
                                MB_ICONQUESTION);
        if (Answer == IDYES)
                SendMessage(WM_CLOSE);
};

class APPLICATION : public CWinApp
{
public:
        BOOL InitInstance();
};

BOOL APPLICATION :: InitInstance()
{
        m_pMainWnd =  new WINDOW;
        m_pMainWnd -> ShowWindow(m_nCmdShow);
        m_pMainWnd -> UpdateWindow();
        return TRUE;
}

APPLICATION  TheApplication;
```

ПРИЛОЖЕНИЕ 8. Приложение с пользовательским меню и системой диалоговых окон

```
#include <afxwin.h>
#include <iostream>
#include <string>
#include "resource.h"
using namespace std;

class aCDATA_DIALOG : public CDialog
{
public:
        aCDATA_DIALOG(char *DialogName, CWnd *Owner):
        CDialog(DialogName, Owner){};
        afx_msg void OnOK();
        afx_msg void OnCancel();
        DECLARE_MESSAGE_MAP()
};

BEGIN_MESSAGE_MAP(aCDATA_DIALOG, CDialog)
        ON_COMMAND(IDOK, OnOK)
        ON_COMMAND(IDCANCEL, OnCancel)
END_MESSAGE_MAP()

afx_msg void aCDATA_DIALOG::OnOK()
{
        MessageBox("OnOK","OnOK");
        CEdit *ebptr = (CEdit *) GetDlgItem(IDC_EDIT1);
        char Str[80];
        int l;
        l = ebptr->GetWindowText(Str, sizeof Str-1);
        MessageBox(Str,"Edit Box Contains");
        char Str1[80];
        int I1;
        I1 = GetDlgItemInt(IDC_EDIT1);
        wsprintf(Str1, "%d", I1);
        MessageBox(Str1,"Edit Box Contains");
        char Str2[80];
        int I2;
        float MyFloat;
        GetDlgItemText(IDC_EDIT1,Str2,strlen(Str2));
        MessageBox(Str2,"Edit Box Contains");
        MyFloat = atof(Str2);
};

afx_msg void aCDATA_DIALOG::OnCancel()
{
        MessageBox("OnCancel","OnCancel");
        int Answer;
        Answer = MessageBox("Уверены?","Cancel", MB_YESNO |
```

```
        if (Answer == IDYES)
                MB_ICONQUESTION);
        EndDialog(0);
};

class CSampleDialog : public CDialog
{
public:
        CSampleDialog(char *DialogName, CWnd *Owner):
        CDialog(DialogName, Owner){};
        afx_msg void OnFirstButton();
        afx_msg void OnSecondButton();
        DECLARE_MESSAGE_MAP()
};

BEGIN_MESSAGE_MAP(CSampleDialog, CDialog)
        ON_COMMAND(IDC_BUTTON1, OnFirstButton)
        ON_COMMAND(IDC_BUTTON2, OnSecondButton)
END_MESSAGE_MAP()

afx_msg void CSampleDialog::OnFirstButton()
{
        MessageBox("OnFirstButton","OnFirstButton");
};

afx_msg void CSampleDialog::OnSecondButton()
{
        MessageBox("OnSecondButton","OnSecondButton");
};

class CMainWin : public CFrameWnd
{
public:
        CMainWin();
        afx_msg void OnEdit();
        afx_msg void OnHelp();
        afx_msg void OnQuit();
        afx_msg void OnMessageBox();
        afx_msg void OnButtons();
        DECLARE_MESSAGE_MAP()
};

CMainWin::CMainWin()
{
        Create(NULL, "Окно с меню", WS_OVERLAPPEDWINDOW, rectDefault,
                NULL, (LPTSTR) IDR_MENU1);
}

afx_msg void CMainWin::OnEdit()
{
        MessageBox("OnEdit","OnEdit");
        aCDATA_DIALOG MyDataDialog((LPTSTR)IDD_DIALOG2 ,this);
```

```
        MyDataDialog.DoModal();
};
afx_msg void CMainWin::OnHelp()
{
        MessageBox("OnHelp","Help");
};
afx_msg void CMainWin::OnQuit()
{
        int Answer;
        Answer = MessageBox("Завершить?","Exit", MB_YESNO |
                                MB_ICONQUESTION);
        if (Answer == IDYES)
                SendMessage(WM_CLOSE);
};
afx_msg void CMainWin::OnMessageBox()
{
        MessageBox("OnMessageBox","OnMessageBox");
};
afx_msg void CMainWin::OnButtons()
{
        CSampleDialog MyDialog((LPTSTR)IDD_DIALOG1 /*"SampleDialog"*/,this);
        MyDialog.DoModal();
};
BEGIN_MESSAGE_MAP(CMainWin, CFrameWnd)
        ON_COMMAND(ID_EXIT_QUIT, OnQuit)
        ON_COMMAND(ID_DIALOGBOXES_MESSAGEBOX, OnMessageBox)
        ON_COMMAND(ID_DIALOGBOXES_BUTTONS, OnButtons)
        ON_COMMAND(ID_DIALOGBOXES_EDIT, OnEdit)
        ON_COMMAND(ID_HELP, OnHelp)
END_MESSAGE_MAP()

class CApp : public CWinApp
{
public:
        BOOL InitInstance();
};
BOOL CApp::InitInstance()
{
        m_pMainWnd = new CMainWin;
        m_pMainWnd->ShowWindow(m_nCmdShow);
        m_pMainWnd->UpdateWindow();
        return TRUE;
}
CApp App;
```

ОГЛАВЛЕНИЕ

Учебное издание

Составители:
Муравьев Геннадий Леонидович,
Мухов Сергей Владимирович,
Шуть Василий Николаевич

МЕТОДИЧЕСКОЕ ПОСОБИЕ

ОСНОВЫ СОЗДАНИЯ WINDOWS-ПРИЛОЖЕНИЙ В СИСТЕМЕ MICROSOFT VISUAL STUDIO C++ НА БАЗЕ БИБЛИОТЕКИ MFC. ИСПОЛЬЗОВАНИЕ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ. МЕНЮ

Часть 3

Ответственный за выпуск	***Муравьев Г.Л.***
Редактор	***Строкач Т.В.***
Компьютерный набор и верстка:	***Муравьев Г.Л., Кармаш Е.Л.***
Корректор	**Никитчик Е.В.**

Подписано к печати 20.09.2010 г. Формат 60*84 1/16. Гарнитура Arial Narrow. Бумага «Снегурочка». Усл. п. л. 2,79. Уч. изд. 3,0. Заказ № 923. Тираж 50 экз. Отпечатано на ризографе учреждения образования «Брестский государственный технический университет». 224017, Брест, ул. Московская. 267.