Модуль №3

Занятие №2

Версия 1.0.1

План занятия:

1. Повторение пройденного материала.
2. Элемент управления «текстовое поле ввода» (Edit Control).
3. Сообщения текстовых полей ввода.
4. Практическая часть.
5. Подведение итогов.
6. Домашнее задание.

1. Повторение пройденного материала

Данное занятие необходимо начать с краткого повторения материала предыдущего занятия. При общении со слушателями можно использовать следующие контрольные вопросы:

1) Для чего обычно используют кнопочные элементы управления, такие как **Button**, **Check Box**, **Radio Button**?
2) Какими способами можно создать кнопку на диалоге?
3) Какое сообщение приходит в диалоговую процедуру при воздействии на элемент управления диалога (например, при нажатии на кнопку)?
4) Какая дополнительная информация приходит с сообщением **WM_COMMAND**?
5) Какое уведомление приходит диалогу при нажатии на кнопку?

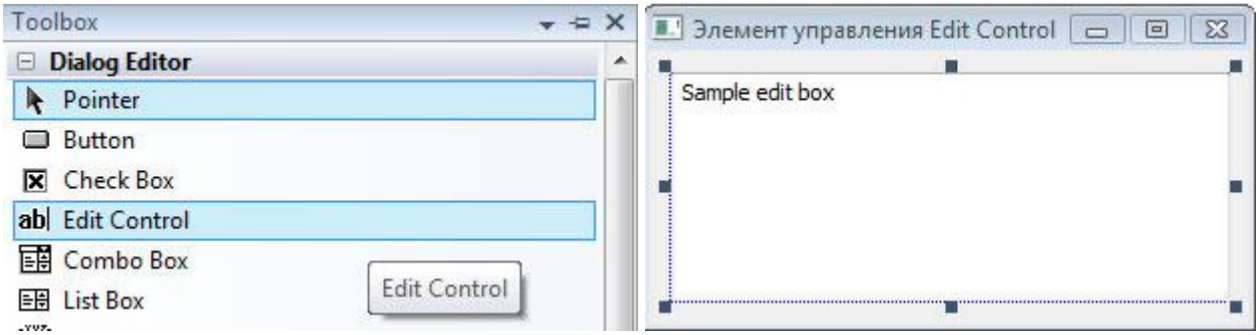
6) Какая функция позволяет перевести фокус ввода на элемент управления?
7) Какая функция позволяет получить дескриптор окна (элемента управления), обладающего фокусом ввода?
8) Как программным способом можно установить изображение на кнопке? Какой стиль в этом случае необходимо задать кнопке?
9) Какие существуют способы программной инициализации состояния элемента управления **Check Box**?
10) Каким образом можно получить состояние флажка?
11) Какое свойство элемента управления **Radio Button** позволяет объединить в группу несколько переключателей?
12) Какие существуют способы программной инициализации состояния переключателя?
13) Каким образом можно получить состояние переключателя?

2. Элемент управления «текстовое поле ввода» (Edit Control)

Повторив материал предыдущего занятия, ознакомить слушателей с ещё одним широко применяемым элементом управления **Edit Control**. Отметить, что на практике применяются различные текстовые поля ввода в самом широком спектре: от небольшого однострочного поля ввода до многострочного элемента управления с автоматическим переносом строк, как в программе **Microsoft Notepad**. Создать текстовое поле ввода на форме диалога можно двумя способами:

- с помощью средств интегрированной среды разработки **Microsoft Visual Studio**;
- посредством вызова функции **CreateWindowEx**.

При первом способе необходимо определить **Edit Control** в шаблоне диалогового окна на языке описания шаблона диалога. Это произойдёт автоматически, если активизировать окно **Toolbox** (<**Ctrl**><**Alt**><**X**>) и «перетащить» текстовое поле ввода на форму диалога.



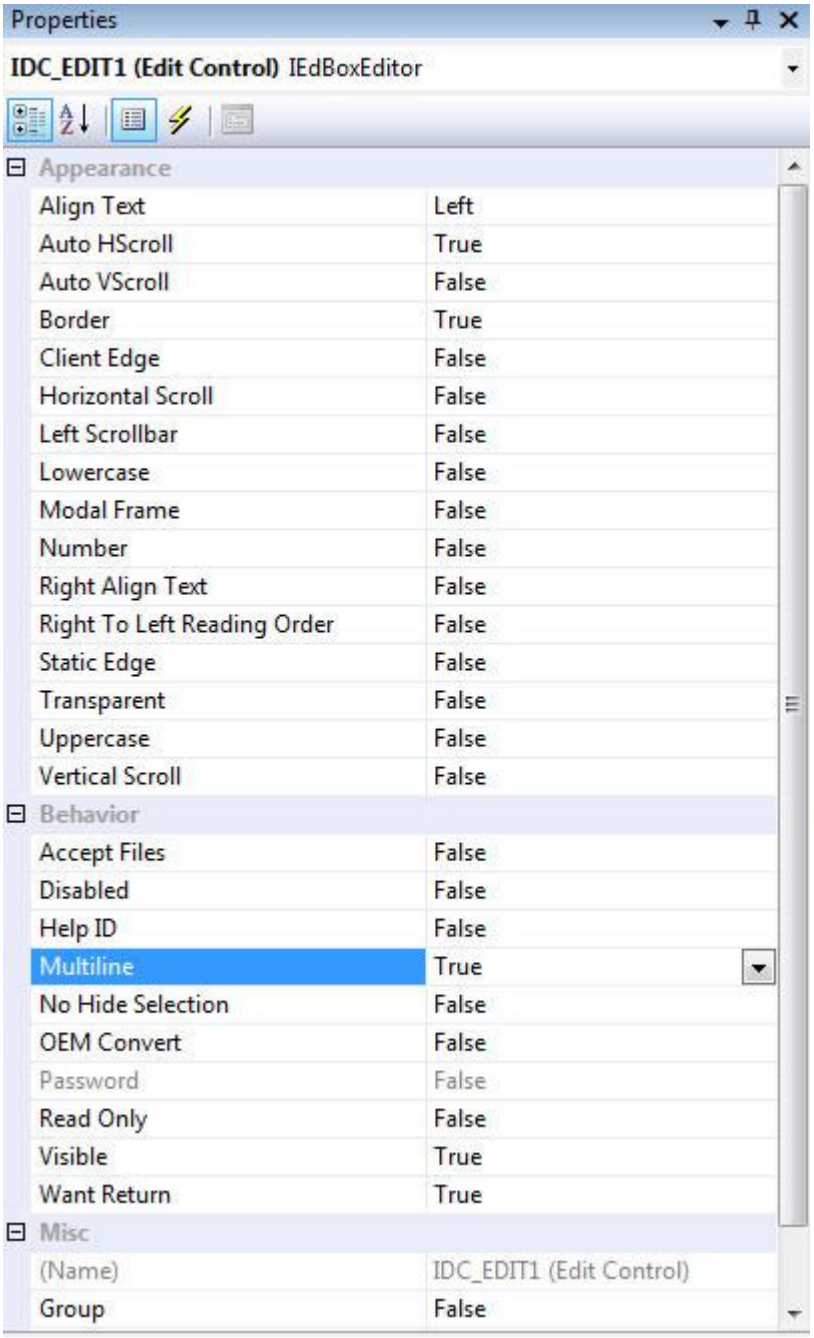
После размещения текстового поля ввода на форме диалога ему назначается идентификатор (например, **IDC_EDIT1**), который впоследствии можно изменить на идентификатор, отражающий семантику ресурса.

Необходимо ознакомить слушателей с некоторыми свойствами элемента управления **Edit Control.**

Отметить, что по умолчанию **Edit Control** является однострочным, с автоматической горизонтальной прокруткой (свойство **Auto HScroll**), с объемной рамкой (свойство **Border**) и выравниванием текста по левой границе окна (свойство **Align Text** со значением **Left**).

Если установить истинным значение свойства **Multiline**, то текстовое поле ввода будет работать в многострочном режиме. При этом станут доступными для использования свойства **Horizontal Scroll**, **Vertical Scroll**, **Auto VScroll**.

Значение **True** свойства **Number** разрешает вводить в **Edit Control** только цифры. Если же окно редактирования предназначено для ввода пароля, то следует установить истинным значение свойства **Password**, и тогда вместо вводимых символов в поле ввода будут отображаться символы звездочки.



Свойство **No Hide Selection** позволяет элементу управления всегда показывать выделенную подстроку, даже если **Edit Control** теряет фокус.

Свойство **Want Return** используется для многострочного окна редактирования. Если для этого свойства установлено значение **True**, то нажатие клавиши **<Enter>** приводит к вводу символа возврата каретки.

Взаимоисключающие свойства **Uppercase** и **Lowercase** преобразуют вводимые символы. В первом случае происходит конвертирование в символы верхнего регистра, а во втором — в символы нижнего регистра.

Истинное значение свойства **Read Only** устанавливает для **Edit Control** режим «только для чтения», не позволяющий пользователю вводить или редактировать текст.

В качестве примера, демонстрирующего использование текстовых полей ввода, привести следующий код:

```
#include <windows.h>
#include <tchar.h>
#include "resource.h"

HWND hLogin, hPassword;
BOOL CALLBACK DlgProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                  LPTSTR lpszCmdLine, int nCmdShow)
{
      return DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                        DlgProc);
}

BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
      switch(message)
      {
            case WM_CLOSE:
                  EndDialog(hWnd, 0);
                  return TRUE;

            case WM_INITDIALOG:
                  hLogin = GetDlgItem(hWnd, IDC_LOGIN);
                  hPassword = GetDlgItem(hWnd, IDC_PASSWORD);
                  return TRUE;

            case WM_COMMAND:
                  if(LOWORD(wParam) == IDC_ENTRY)
                  {
                        TCHAR text[100], login[20], password[20];
                        GetWindowText(hLogin, login, 20);
                        GetWindowText(hPassword, password, 20);
                        if(lstrlen(login) == 0 || lstrlen(password) == 0)
                              MessageBox(hWnd,
                              TEXT("Не введён логин или пароль!"),
                              TEXT("Авторизация"), MB_OK | MB_ICONSTOP);
                        else
                        {
                              wsprintf(text,
                              TEXT("Логин: %s\nПароль: %s"), login, password);
```

```
                              MessageBox(hWnd, text, TEXT("Авторизация"),
                                          MB_OK | MB_ICONINFORMATION);
                        }
                        SetWindowText(hLogin, NULL);
                        SetWindowText(hPassword, NULL);
                        SetFocus(hLogin);
                  }
                  return TRUE;
      }
      return FALSE;
}
```

Альтернативный способ создания текстового поля ввода - использование функции **CreateWindowEx**, рассмотренной на одном из предыдущих занятий. В этом случае во втором параметре функции передается имя предопределенного оконного класса – **EDIT**.

В качестве примера, демонстрирующего программный способ создания текстовых полей, привести следующий код:

```
#include <windows.h>
#include <tchar.h>
#include "resource.h"

HWND hEdit1, hEdit2, hButton;
HINSTANCE hInst;

BOOL CALLBACK DlgProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                     LPTSTR lpszCmdLine, int nCmdShow)
{
      hInst = hInstance;
      return DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                     DlgProc);
}

BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
      switch(message)
      {
            case WM_CLOSE:
                  EndDialog(hWnd, 0);
                  return TRUE;

            case WM_INITDIALOG:
                  hEdit1 = CreateWindowEx(WS_EX_CLIENTEDGE, TEXT("EDIT"), 0,
                        WS_CHILD | WS_VISIBLE | WS_VSCROLL | WS_TABSTOP |
                        ES_WANTRETURN | ES_MULTILINE | ES_AUTOVSCROLL,
                        10, 7, 150, 100, hWnd, 0, hInst, 0);
```

```
                hEdit2 = CreateWindowEx(WS_EX_CLIENTEDGE, TEXT("EDIT"), 0,
                      WS_CHILD | WS_VISIBLE | WS_VSCROLL | ES_WANTRETURN |
                      ES_MULTILINE | ES_AUTOVSCROLL | ES_READONLY,
                      170, 7, 150, 100, hWnd, 0, hInst, 0);

                hButton = CreateWindowEx(WS_EX_CLIENTEDGE |
                      WS_EX_DLGMODALFRAME, TEXT("BUTTON"),
                      TEXT("Click Me!"), WS_CHILD | WS_VISIBLE | WS_TABSTOP,
                      10, 110, 310, 40, hWnd, 0, hInst, 0);
                return TRUE;

            case WM_COMMAND:
                {
                      HWND h = (HWND) lParam;
                      if(h == hButton)
                      {
                            int length = SendMessage(hEdit1,
                                              WM_GETTEXTLENGTH, 0, 0);
                            TCHAR *buffer = new TCHAR[length + 1];
                            memset(buffer, 0, length + 1);
                            GetWindowText(hEdit1, buffer, length + 1);
                            SetWindowText(hEdit2, buffer);
                            delete[] buffer;
                      }
                }
                return TRUE;
      }
      return FALSE;
}
```

Анализируя вышеприведенный код, акцентировать внимание слушателей на сообщении **WM_GETTEXTLENGTH**, которое отправляется текстовому полю ввода с целью определения длины введённого текста.

Продемонстрировать слушателям возможность изменения стилей элементов управления. Подчеркнуть, что для этой цели используется функция API **SetWindowLong**:

```
LONG SetWindowLong(
    HWND hWnd, // дескриптор окна
    int nIndex, // индекс значения, которое нужно изменить
    LONG dwNewLong // новое значение
);
```

Функция API **GetWindowLong** позволяет получить текущие стили окна (элемента управления):

```
LONG GetWindowLong(
    HWND hWnd, // дескриптор окна
    int nIndex // индекс значения, которое нужно выбрать
);
```

В качестве примера, демонстрирующего использование вышеописанных функций, привести следующий код:

```
#include <windows.h>
#include <tchar.h>
#include "resource.h"

HWND hButton, hEdit, hStatic;

BOOL CALLBACK DlgProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                     LPTSTR lpszCmdLine, int nCmdShow)
{
      return DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                       DlgProc);
}

BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
      switch(message)
      {
            case WM_CLOSE:
                  EndDialog(hWnd, 0);
                  return TRUE;

            case WM_INITDIALOG:
                  hButton = GetDlgItem(hWnd, IDC_BUTTON1);
                  hEdit = GetDlgItem(hWnd, IDC_EDIT1);
                  hStatic = GetDlgItem(hWnd, IDC_STATIC1);
                  return TRUE;

            case WM_COMMAND:
                  if(LOWORD(wParam) == IDC_BUTTON2)
                  {
                        LONG styles = GetWindowLong(hButton, GWL_STYLE);
                        SetWindowLong(hButton, GWL_EXSTYLE, styles |
                                    WS_EX_CLIENTEDGE | WS_EX_DLGMODALFRAME);
                        InvalidateRect(hButton, 0, 1);
                  }
                  else if(LOWORD(wParam) == IDC_BUTTON3)
                  {
                        LONG styles = GetWindowLong(hEdit, GWL_STYLE);
                        SetWindowLong(hEdit, GWL_STYLE, styles | ES_RIGHT);
                        InvalidateRect(hEdit, 0, 1);
                  }
```

```
                else if(LOWORD(wParam) == IDC_BUTTON4)
                {
                      LONG styles = GetWindowLong(hStatic, GWL_STYLE);
                      SetWindowLong(hStatic, GWL_STYLE, styles |
                                    SS_BLACKRECT);
                      InvalidateRect(hStatic, 0, 1);
                }
                return TRUE;
      }
      return FALSE;
}
```

3. Сообщения текстовых полей ввода

Для выполнения различных операций по редактированию текста приложение может отправлять следующие сообщения элементу управления **Edit Control**:

Код сообщения	wParam	lParam	Описание
EM_SETSEL	iStart	iEnd	Выделить текст, начиная с позиции **iStart** и заканчивая позицией **iEnd**
EM_GETSEL	&iStart	&iEnd	Получить начальное и конечное положения текущего выделения
WM_CLEAR	0	0	Удалить выделенный текст
WM_CUT	0	0	Удалить выделенный текст и поместить его в буфер обмена
WM_COPY	0	0	Скопировать выделенный текст в буфер обмена **Windows**
WM_PASTE	0	0	Вставить текст из буфера обмена в месте, соответствующем позиции курсора
EM_CANUNDO	0	0	Определить возможность отмены последнего действия
WM_UNDO	0	0	Отменить последнее действие
WM_GETTEXT	nMax	szBuffer	Скопировать текст (не более **nMax** символов) из элемента управления в буфер **szBuffer**
EM_GETLINECOUNT	0	0	Получить число строк для многострочного окна редактирования

Код сообщения	wParam	lParam	Описание
EM_LINELENGTH	iLine	0	Получить длину строки **iLine**
EM_GETLINE	iLine	szBuffer	Скопировать строку **iLine** в буфер **szBuffer**
EM_LINEFROMCHAR	-1	0	Получить номер строки, в которой расположен курсор
EM_LINEINDEX	iLine	0	Получить номер первого символа строки **iLine**

Как известно, при воздействии на элемент управления диалога (например, при вводе текста в **Edit Control**), в диалоговую процедуру **DlgProc** поступает сообщение **WM_COMMAND,** в котором **LOWORD(wParam)** содержит идентификатор элемента управления, **HIWORD(wParam)** содержит код уведомления (например, **EN_CHANGE)**, а **lParam** – дескриптор элемента управления.

Некоторые из кодов уведомления от текстового поля ввода приведены в таблице.

Код уведомления	Интерпретация
EN_SETFOCUS	Окно получило фокус ввода
EN_KILLFOCUS	Окно потеряло фокус ввода
EN_UPDATE	Содержимое окна будет меняться
EN_CHANGE	Содержимое окна изменилось
EN ERRSPACE	Произошло переполнение буфера редактирования

Для демонстрации обработки сообщения **WM_COMMAND** с кодом уведомления **EN_CHANGE** модифицируем пример, рассмотренный ранее.

```
#include <windows.h>
#include <tchar.h>
#include "resource.h"

HWND hLogin, hPassword, hEntry;
TCHAR text[100], login[20], password[20];

BOOL CALLBACK DlgProc(HWND, UINT, WPARAM, LPARAM);
```

```
int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                  LPTSTR lpszCmdLine, int nCmdShow)
{
      return DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                        DlgProc);
}

BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
      switch(message)
      {
            case WM_CLOSE:
                  EndDialog(hWnd, 0);
                  return TRUE;

            case WM_INITDIALOG:
                  hLogin = GetDlgItem(hWnd, IDC_LOGIN);
                  hPassword = GetDlgItem(hWnd, IDC_PASSWORD);
                  hEntry = GetDlgItem(hWnd, IDC_ENTRY);
                  return TRUE;

            case WM_COMMAND:
                  if((LOWORD(wParam) == IDC_LOGIN ||
                  LOWORD(wParam) == IDC_PASSWORD) &&
                  HIWORD(wParam) == EN_CHANGE)
                  {
                        GetWindowText(hLogin, login, 20);
                        GetWindowText(hPassword, password, 20);
                        if(lstrlen(login) == 0 || lstrlen(password) < 6)
                              EnableWindow(hEntry, FALSE);
                        else
                              EnableWindow(hEntry, TRUE);
                  }
                  if(LOWORD(wParam) == IDC_ENTRY)
                  {
                        wsprintf(text, TEXT("Логин: %s\nПароль: %s"),
                                    login, password);
                        MessageBox(hWnd, text, TEXT("Авторизация"),
                                    MB_OK | MB_ICONINFORMATION);
                        SetWindowText(hLogin, NULL);
                        SetWindowText(hPassword, NULL);
                        SetFocus(hLogin);
                        EnableWindow(hEntry, FALSE);
                  }
                  return TRUE;
      }
      return FALSE;
}
```

Далее следует привести пример, в котором рассматриваются сообщения, отправляемые текстовому полю ввода, при выполнении различных операций по редактированию текста.

```
#include <windows.h>
#include <tchar.h>
#include "resource.h"

#define WM_POSITION WM_APP // Пользовательское сообщение

HWND hEdit, hCopy, hCut, hPaste, hDelete, hUndo, hSelectAll;
WNDPROC OriginalProc = NULL;

BOOL CALLBACK DlgProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                  LPTSTR lpszCmdLine, int nCmdShow)
{
      return DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                        DlgProc);
}

void GetPosition()
{
      // Получим номер строки, в которой расположен курсор
      int row = SendMessage(hEdit, EM_LINEFROMCHAR, -1, 0);
      DWORD Start, End;

      // Получим границы выделения текста
      SendMessage(hEdit, EM_GETSEL, (WPARAM)&Start, (LPARAM)&End);

      // Получим номер первого символа указанной строки
      int col = Start - SendMessage(hEdit, EM_LINEINDEX, row, 0);

      // Получим дескриптор родительского окна (диалога)
      HWND hParent = GetParent(hEdit);

      // Установим в заголовок главного окна текущие координаты курсора
      TCHAR buffer[50] = {0};
      wsprintf(buffer, TEXT("Строка %d, Столбец %d"), row+1, col+1);
      SetWindowText(hParent, buffer);
      SetFocus(hEdit); // Переведём фокус ввода на Edit Control
}

void EnableDisableButton()
{
      // Получим границы выделения текста
      DWORD dwPosition = SendMessage(hEdit, EM_GETSEL, 0, 0);
      WORD wBeginPosition = LOWORD(dwPosition);
      WORD wEndPosition = HIWORD(dwPosition);

      if(wEndPosition != wBeginPosition) // Выделен ли текст?
      {
            EnableWindow(hCopy, TRUE);
            EnableWindow(hCut, TRUE);
            EnableWindow(hDelete, TRUE);
      }
      else
      {
            EnableWindow(hCopy, FALSE);
            EnableWindow(hCut, FALSE);
            EnableWindow(hDelete, FALSE);
      }
```

```
	// Имеется ли текст в буфере обмена?
	if(IsClipboardFormatAvailable(CF_TEXT))
		EnableWindow(hPaste, TRUE);
	else
		EnableWindow(hPaste, FALSE);

	// Существует ли возможность отмены последнего действия?
	if(SendMessage(hEdit, EM_CANUNDO, 0, 0))
		EnableWindow(hUndo, TRUE);
	else
		EnableWindow(hUndo, FALSE);

	// Определим длину текста в Edit Control
	int length = SendMessage(hEdit, WM_GETTEXTLENGTH, 0, 0);

	// Выделен ли весь текст в Edit Control?
	if(length != wEndPosition - wBeginPosition)
		EnableWindow(hSelectAll, TRUE);
	else
		EnableWindow(hSelectAll, FALSE);
}

LRESULT CALLBACK EditProc(HWND hWnd, UINT message, WPARAM wParam,
                          LPARAM lParam)
{
	if(   message == WM_POSITION || message == WM_LBUTTONDOWN ||
		message == WM_LBUTTONUP || message == WM_KEYDOWN ||
		message == WM_KEYUP || message == WM_MOUSEMOVE &&
		(wParam & MK_LBUTTON))
	{
		EnableDisableButton();
		GetPosition();
	}
	// Вызов стандартного обработчика сообщений
	return CallWindowProc(OriginalProc, hWnd, message, wParam, lParam);
}


BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	switch(message)
	{
		case WM_CLOSE:
			EndDialog(hWnd, 0);
			return TRUE;

		case WM_INITDIALOG:
			hEdit = GetDlgItem(hWnd, IDC_EDIT1);
			hCopy = GetDlgItem(hWnd, ID_COPY);
			hCut = GetDlgItem(hWnd, ID_CUT);
			hPaste = GetDlgItem(hWnd, ID_PASTE);
			hDelete = GetDlgItem(hWnd, ID_DELETE);
			hSelectAll = GetDlgItem(hWnd, ID_SELECTALL);
			hUndo = GetDlgItem(hWnd, ID_UNDO);

			// переопределим оконную процедуру текстового поля
			OriginalProc = (WNDPROC) SetWindowLong(hEdit, GWL_WNDPROC,
                                LONG(EditProc));
```

```
                /* отправка пользовательского сообщения для определения
                      текущих координат курсора */
                SendMessage(hEdit, WM_POSITION, 0, 0);
                return TRUE;

          case WM_COMMAND:
                switch(LOWORD(wParam))
                {
                      case ID_UNDO:
                            // отменить последнее действие
                            SendMessage(hEdit, WM_UNDO, 0, 0);
                            // определить текущие координаты курсора
                            SendMessage(hEdit, WM_POSITION, 0, 0);
                            break;
                      case ID_COPY:
                            // скопировать выделенный фрагмент текста
                            SendMessage(hEdit, WM_COPY, 0, 0);
                            break;
                      case ID_PASTE:
                            // вставить текст в Edit из буфера обмена
                            SendMessage(hEdit, WM_PASTE, 0, 0);
                            // определить текущие координаты курсора
                            SendMessage(hEdit, WM_POSITION, 0, 0);
                            break;
                      case ID_CUT:
                            // вырезать выделенный фрагмент текста
                            SendMessage(hEdit, WM_CUT, 0, 0);
                            // определить текущие координаты курсора
                            SendMessage(hEdit, WM_POSITION, 0, 0);
                            break;
                      case ID_DELETE:
                            // удалить выделенный фрагмент текста
                            SendMessage(hEdit, WM_CLEAR, 0, 0);
                            // определить текущие координаты курсора
                            SendMessage(hEdit, WM_POSITION, 0, 0);
                            break;
                      case ID_SELECTALL:
                            // выделить весь текст в Edit Control
                            SendMessage(hEdit, EM_SETSEL, 0, -1);
                            // определить текущие координаты курсора
                            SendMessage(hEdit, WM_POSITION, 0, 0);
                            break;
                }
                return TRUE;
      }
      return FALSE;
}
```

Анализируя вышеприведенный код, акцентировать внимание слушателей на переопределении стандартной оконной процедуры текстового поля ввода посредством функции **SetWindowLong**, рассмотренной ранее. Отметить, что в контексте решаемой задачи это достаточно удобно, а в некоторых ситуациях без переопределения оригинальной окон-

ной процедуры элемента управления не обойтись. В конкретном примере переопределённая оконная процедура обрабатывает сообщения мыши, клавиатурные сообщения, а также пользовательское сообщение **WM_POSITION**, т.е. сообщение, определяемое программистом в приложении.

```
#define WM_POSITION WM_APP
```

Как известно, любое системное сообщение – это целочисленное значение. Акцентировать внимание слушателей на том, что допускается определять в приложении пользовательские сообщения. При этом важно, чтобы номера этих сообщений не совпадали с номерами системных сообщения. В этой связи неплохо бы знать, для каких целей используется тот или иной целочисленный диапазон.

Следует рассмотреть со слушателями следующие диапазоны целочисленных значений:

- диапазон 0 – 0x03FF зарезервирован для системных сообщений (например, #define WM_COMMAND 0x0111);
- диапазон 0x0400 – 0x7FFF обычно используется для идентификаторов окон, создаваемых программно (#define WM_USER 0x0400);
- диапазон 0x8000 – 0xBFFF применяется для определения пользовательских сообщений (#define WM_APP 0x8000);
- диапазон 0xC000 – 0xFFFF предназначен для строковых сообщений (такие сообщения используются при работе с немодальными диалогами и их необходимо регистрировать с помощью функции API **RegisterWindowMessage** – об этом речь пойдёт в одном из следующих занятий);
- целочисленные значения, превышающие 0xFFFF, зарезервированы системой.

Как следует из вышеприведенного кода, переопределённая оконная процедура текстового поля ввода должна вернуть значение, возвра-

щаемое функцией API **CallWindowProc**. Данная функция передаёт все сообщения на обработку стандартной оконной процедуре:

```
LRESULT CallWindowProc(
    WNDPROC lpPrevWndFunc, // указатель на предыдущую (оригинальную) оконную
      // процедуру. Это значение возвращается функцией GetWindowLong.
    HWND hWnd, // дескриптор окна, получающего сообщение
    UINT Msg, // идентификатор сообщения
    WPARAM wParam, // дополнительная информация о сообщении
    LPARAM lParam // дополнительная информация о сообщении
);
```

Кроме того, в приложении используется функция API **IsClipboardFormatAvailable**, которая определяет, имеются ли в буфере обмена данные в указанном формате (например, текстовые данные).

```
BOOL IsClipboardFormatAvailable(
          UINT format // формат данных
     );
```

4. Практическая часть

1. Написать приложение «Простейший калькулятор» на четыре действия (сложение, вычитание, умножение и деление). На форме диалога расположены три текстовых поля для ввода операндов и знака операции, кнопка, при нажатии на которую, подсчитывается результат, а также статик для вывода результата вычисления.
2. Написать приложение, которое по введенной дате определяет день недели. При этом день, месяц и год необходимо вводить в отдельные текстовые поля. Результат также следует выводить в текстовое

поле со стилем Read Only. Предусмотреть проверку корректности ввода даты.

5. Подведение итогов

Подвести общие итоги занятия. Выделить основные способы создания текстовых полей ввода – с помощью средств интегрированной среды разработки приложений, а также посредством функции CreateWindowEx. Отметить, какие сообщения текстовых полей ввода чаще всего используются. Напомнить слушателям о возможности переопределения стандартной оконной процедуры для любого элемента управления. Акцентировать внимание слушателей на наиболее тонких моментах изложенной темы.

6. Домашнее задание

Написать приложение, определяющее, сколько осталось времени до указанной даты (день, месяц и год вводятся в отдельные текстовые поля). Предусмотреть выдачу результата в годах, месяцах, днях, и при необходимости, в часах, минутах, секундах (если установлен флажок).