5. Использование в приложениях меню, кнопок, редакторов и таймеров

5.1. Работа с меню

Меню программы описывается в отдельном файле – файле ресурсов. Для того чтобы окно приложения имело меню, необходимо:

- описать меню в файле ресурсов (файл с расширением .rc);
- в заголовочном файле “resource.h” определить идентификаторы меню и его пунктов;
- добавить файл ресурсов к проекту;
- включить с помощью директивы *#include* заголовочный файл:

```
#include "resource.h"
```

- при определении класса окна в поле *lpszMenuName* указать идентификатор меню (для преобразования типа int в строковый необходимо использовать макрос MAKEINTRESOURCE) или указать идентификатор меню в качестве девятого параметра функции создания окна - *CreateWindow*;
- в функцию окна добавить обработку сообщения WM_COMMAND, для того, чтобы добавить отклики на выбор того или иного пункта меню.

5.1.1. Файл ресурсов

Файл ресурсов может содержать описание таких графических ресурсов программы, как меню, значки, курсоры, растровые изображения, диалоговые окна, панели инструментов. Обычно этот файл создается с помощью редакторов ресурсов, входящих в состав компилятора Visual C++. Редакторы ресурсов позволяют легко создавать меню и другие ресурсы.

Создать меню можно и без редактора ресурсов, используя текстовый редактор. Для этого можно ввести описание меню в текстовом виде, в файл с расширение .rc, например:

```
#include "resource.h"
IDR_MENU1  MENU  DISCARDABLE
BEGIN
    POPUP "&Фигуры"
    BEGIN
        MENUITEM "&Прямоугольники",ID_RECT
        MENUITEM "&Треугольники", ID_TREUG
    END
    POPUP "&Цвет"
    BEGIN
        MENUITEM "Красный",ID_RED
        MENUITEM "Зеленый",ID_GREEN
```

```
    END
END
```

Ключевое слово MENU используется для описания ресурса меню, перед ним указывается идентификатор меню, в данном примере IDR_MENU1. Вместо слов BEGIN и END можно использовать фигурные скобки {}.

Строка меню будет содержать два пункта с названиями “Фигуры” и “Цвет”, причем первые буквы слов будут подчеркнуты (на это указывает символ & перед буквой).

Ключевое слово POPUP определяет всплывающее подменю. Подчеркнутый символ указывает пользователю, что выбрать подменю можно нажатием данной клавиши вместе с клавишей Alt, например Alt-Ц для пункта “Цвет”.

Ключевое слово MENUITEM используется для конечных пунктов меню. С каждым конечным пунктом меню связывается идентификационный номер или идентификатор. Обычно используют идентификаторы, которые определяют в отдельном файле resource.h, например, ID_RED, ID_GREEN.

5.1.2. Файл “resource.h”

Заголовочный файл “resource.h” содержит директивы #define, которые определяют числовые значения идентификаторов пунктов меню, например:

```
#define     ID_RECT         100
#define     ID_TREUG        200
#define     ID_RED          300
#define     ID_GREEN        400
```

5.1.3. Обработка сообщений меню

Сообщение WM_COMMAND посылается, когда пользователь выбирает пункт меню. Идентификатор выбранного пункта меню передается в функцию окна через параметр *wParam*:

```
LOWORD(wParam)      // идентификатор пункта меню
```

В функции окна при обработке сообщения WM_COMMAND, для определения пункта меню, необходимо проверять нижнее слово параметра *wParam*.

Фрагмент программы:

```
case WM_COMMAND:
switch(LOWORD(wParam))
{case  ID_RED:
     color = RGB(255,0,0);
     break;
 case  ID_GREEN:
     color = RGB(0,255,0);
```

```
        break;
    }
    break;
```

5.1.4. Создание меню с помощью редактора ресурсов

Начиная с версии Microsoft Visual Studio 2005, в созданный мастером проект (Win32 Project) уже включено меню, содержащее пункты File и About. Для добавления к меню новых пунктов или изменения существующих, достаточно на панели Resource View выбрать тип ресурса Menu и внести все необходимые изменения в визуальном режиме с помощью редактора ресурсов.

При использовании редактора ресурсов описание структуры меню также сохраняется в файле ресурсов (файл с расширением .rc). Заголовочный файл resource.h создается редактором ресурсов автоматически. В этом файле определяются идентификационные номера соответствующие идентификаторам пунктов меню. Идентификаторы пунктов меню будут передаваться в функцию окна при выборе того или иного пункта меню.

5.2. Создание кнопок и редакторов

Для создания кнопки используется функция *CreateWindow* (или *CreateWindowEx*), где в качестве значения параметра, который отвечает за класс окна должно быть значение "BUTTON":

```
BtHWnd = CreateWindow (
          "BUTTON",           // имя класса окна
          "Reset",            // заголовок
          WS_CHILD|WS_VISIBLE|WS_BORDER,
          5, 10, 50, 40, hWnd, NULL, hInstance, NULL);
```

При обработке сообщения WM_COMMAND нужно сравнить значение параметра *lParam* с дескриптором кнопки.

Для создания редактора также используется функция *CreateWindow* (или *CreateWindowEx*), где в качестве значения параметра, который отвечает за класс окна должно быть значение "EDIT":

```
EdtHWnd = CreateWindow (
     "EDIT",        // имя класса окна
     "0",           // заголовок
WS_CHILD|WS_VISIBLE|WS_BORDER|WS_THICKFRAME,
     60, 10, 70, 70, hWnd,    NULL, hInstance, NULL);
```

Для кнопок и редакторов доступны дополнительные стили, которые позволяют изменить их вид или поведение.

Добавить кнопку или редактор к главному окну можно в функции окна при обработке сообщения WM_CREATE.

5.2.1. Помещение и получение текста из редактора (поля ввода)

Для того, чтобы поместить текст в редактор можно использовать функцию *SetWindowText* или послать редактору сообщение WM_SETTEXT с помощью функции *SendMessage*.

Функция *SendMessage* посылает указанное сообщение окну или нескольким окнам. Она вызывает функцию окна и не возвращает управление до тех пор, пока функция окна не обработает сообщение.

```
LRESULT SendMessage(
HWND hWnd,          //дескриптор окна - получателя
UINT Msg,           // посылаемое сообщение
WPARAM wParam,      // первый параметр сообщения
LPARAM lParam);     // второй параметр сообщения
```

Параметр *hWnd* - определяет окно, которое получит сообщение, *Msg* - определяет посылаемое сообщение, *wParam* и *lParam* - параметры сообщения.

Возвращаемое значение определяет результат обработки сообщения и зависит от посланного сообщения.

Функция *SetWindowText* меняет заголовок указанного окна, если окно имеет заголовок, если окно является элементом управления, тогда меняет текст управляющего элемента.

```
BOOL SetWindowText(
 HWND hWnd,    // дескриптор окна или управляющего элемента
 LPCTSTR lpString);      // указатель на строку
```

Параметр *hWnd* - определяет окно или управляющий элемент, чей текст будет изменен, *lpString* - указывает на строку, оканчивающуюся нулевым символом, которая будет новым названием окна или текстом управляющего элемента.

Функция возвращает ненулевое значение в случае удачного завершения.

При вызове функции *SetWindowText* указанному окну или управляющему элементу посылается сообщение WM_SETTEXT, которое можно также послать с помощью функции *SendMessage*.

В сообщении WM_SETTEXT параметр *wParam* не используется, должен быть = 0, параметр *lParam* = (LPARAM)(LPCTSTR) *lpsz*. Значение *lpsz* указывает на строку, заканчивающуюся нулевым символом, которое будет текстом окна.

Возвращаемое значение равно TRUE, если текст установлен, и FALSE - (для редактора), LB_ERRSPACE (для списка), CB_ERRSPACE (для комбо) если нет места для размещения текста.

Для считывания текста из редактора необходимо получить длину текста в редакторе с помощью функции *GetWindowTextLength* или, послав сообщение WM_TEXTLENGTH. Затем получить текст с помощью функции *GetWindowText* или, послав сообщение WM_GETTEXT.

Функция *GetWindowTextLength* считывает длину текста заголовка окна или текста управляющего элемента в символах.

```
int GetWindowTextLength( HWND hWnd  );
```

В случае успешного завершения функция возвращает длину текста в символах.

Сообщение WM_GETTEXTLENGTH посылается для того, чтобы определить длину текста, связанного с окном, не включая символ конца строки. Параметры *wParam* и *lParam* не используются, должны быть = 0. Возвращаемое значение - длина текста в символах.

Функция *GetWindowText* копирует текст заголовка указанного окна в буфер, если указанное окно - элемент управления, то из него копируется текст.

```
int GetWindowText(
  HWND hWnd,   // дескриптор окна или элемента управления
  LPTSTR lpString,  // указатель на буфер для текста
  int nMaxCount);   // количество символов для копирования
```

Параметр *hWnd* - определяет окно или управляющий элемент, содержащий текст, *lpString* - указатель на буфер, принимающий текст, параметр *nMaxCount* определяет максимальное количество символов, которое может быть скопировано в буфер, включая символ конца строки.

Сообщение WM_GETTEXT посылается для того, чтобы скопировать текст, относящийся к окну, в буфер выделенный инициатором сообщения. Параметр *wParam* = (WPARAM) cchTextMax определяет количество символов для копирования, параметр *lParam* = (LPARAM) lpszText - указатель на буфер для текста. Возвращаемое значение - количество фактически скопированных символов.

5.3. Работа с таймером

Таймеры необходимы, когда требуется выполнять какие-то действия через определенные промежутки времени. Для работы с таймером необходимо установить таймер с помощью функции *SetTimer*, в функцию окна добавить обработку сообщений от таймера - WM_TIMER.

Функция *SetTimer* создает таймер с определенным значением таймаута:

```
UINT SetTimer(
HWND hWnd,  // дескриптор окна для сообщений от таймера
 UINT nIDEvent,     // идентификатор таймера
 UINT uElapse,      // значение таймаута
 TIMERPROC lpTimerFunc); //адрес процедуры обработки
                         // сообщений от таймера
```

Параметр *hWnd* - определяет окно, ассоциированное с таймером, если задано значение NULL, то ни одно окно не ассоциируется с таймером и значение *nIDEvent* игнорируется. Параметр *nIDEvent*, не равный нулю,

идентифицирует таймер, *uElapse* - определяет значение таймаута в миллисекундах.

Параметр *lpTimerFunc* - указывает на функцию, которая вызывается по истечении таймаута, если задано значение NULL, то система посылает сообщение в очередь сообщений приложения.

Функция возвращает идентификатор таймера в случае успешного завершения и ноль в обратном случае.

После создания таймера, система, по окончании таймаута, посылает окну сообщение WM_TIMER. которое должно быть обработано в функции окна для выполнения необходимых действий. В сообщении WM_TIMER значение параметра *wParam* указывает на идентификатор таймера, возвращенный функцией *SetTimer*. В сообщении значение параметра *lParam* указывает на функцию обратного вызова, переданную в качестве параметра в функцию *SetTimer*.

По завершении работы с таймером его необходимо удалить с помощью функции KillTimer:

```
BOOL KillTimer(
 HWND hWnd,    //дескриптор окна, где установлен таймер
 UINT uIDEvent);  // идентификатор таймера
```

Параметр *uIDEvent* - определяет таймер, который должен быть удален, если таймер создавался функцией *SetTimer* с hWnd=NULL. Этот параметр должен быть равен значению, возвращенному функцией *SetTimer*.

Функция в случае успешного завершения возвращает ненулевой результат.

В приложении 7 приведен фрагмент программы, в котором по сообщению от таймера генерируется случайное число и суммируется с числом, которое находится в редакторе (поле ввода), результат помещается в редактор. Кнопка "Reset" обнуляет значение в редакторе. Генерация случайных чисел начинается по пункту меню "Start" и прекращается по пункту меню "Stop".

Приложение 2. Каркас стандартного Windows -приложения

```
#include “windows.h”
     /* объявление функции окна       */
LRESULT CALLBACK WindowFunc (HWND, UINT, WPARAM, LPARAM);

char szWinName[] = "MyWin";           // Имя "класса" окна
    /*       Главная функция          */
int WINAPI WinMain (HINSTANCE hThisInst, HINSTANCE hPrevInst,
                  LPSTR lpszArgs, int nWinMode)
{ HWND hWnd;                          // дескриптор окна
  MSG msg;                            // сообщение
  WNDCLASSEX wcl;                     // "класс" окна
      /*  Определение "класса" (стиля) окна */
  wcl.hInstance = hThisInst; /*дескриптор данного экземпляра*/
  wcl.lpszClassName = (LPCWSTR)szWinName;   // имя "класса" окна
  wcl.lpfnWndProc = WindowFunc;       // функция окна
  wcl.style = 0;                      // стиль по умолчанию
  wcl.cbSize = sizeof(WNDCLASSEX);    // размер структуры
  wcl.hIcon = LoadIcon(NULL,IDI_APPLICATION); //больш.иконка
  wcl.hIconSm = LoadIcon(NULL, IDI_WINLOGO); // малая иконка
  wcl.hCursor = LoadCursor(NULL, IDC_ARROW); //форма курсора
  wcl.lpszMenuName = NULL;      // меню не используется
  wcl.cbClsExtra = 0;           // дополнит. информации нет
  wcl.cbWndExtra = 0;
      /* Фон окна задается белым   */
  wcl.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
      /* Регистрация "класса" окна */
  if (!RegisterClassEx(&wcl))  return 0;
      /*     Создание окна          */
  hWnd = CreateWindowEx
  (  0, (LPCWSTR)szWinName,       // имя "класса" окна
     (LPCWSTR)"",                 // заголовок
     WS_OVERLAPPEDWINDOW,   // стандартное окно
     CW_USEDEFAULT,         // координата X - по умолчанию
     CW_USEDEFAULT,         // координата Y - по умолчанию
     CW_USEDEFAULT,         // ширина       - по умолчанию
     CW_USEDEFAULT,         // высота       - по умолчанию
     HWND_DESKTOP,          // родительского окна нет
     NULL,                  // меню нет
     hThisInst, // дескриптор данного экземпляра приложения
     NULL                  // дополнительных аргументов нет
  );
      /*     Отображение окна       */
  ShowWindow(hWnd, nWinMode);

      /*     Цикл сообщений         */
  while (GetMessage(&msg, NULL, 0, 0))
  { TranslateMessage(&msg);
    DispatchMessage (&msg);
  }
  return msg.wParam;
}
```

```
   /*-----------------------------------------------*/
   /*                Функция  окна                  */
   /*-----------------------------------------------*/
   /* Эту функцию вызывает Windows и передает ей  */
   /* на обработку сообщения из очереди сообщений */
   /*-----------------------------------------------*/
LRESULT CALLBACK WindowFunc (HWND hWnd, UINT message,
                         WPARAM wParam, LPARAM lParam)
{
  switch(message)
  { case WM_DESTROY:         /* "завершить программу" */
      PostQuitMessage(0);
      break;

    default:
      /* Остальные сообщения обрабатывать  */
      /* операционной системе              */
      return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}
```

Приложение 7. Программа, демонстрирующая вывод графики и работу с меню

```
#include "windows.h"
#include "resource.h"
     /*         объявление функции окна                */
LRESULT CALLBACK WindowFunc (HWND, UINT, WPARAM, LPARAM);
char szWinName[] = "MyWin";       // Имя "класса" окна
/*-------------------------------------------------------*/
/*                  Главная функция                      */
/*-------------------------------------------------------*/
int WINAPI WinMain (HINSTANCE hThisInst, HINSTANCE hPrevInst,
                    LPSTR lpszArgs, int nWinMode)
{ HWND hWnd;                         // дескриптор окна
  MSG msg;                           //  сообщение
  WNDCLASSEX wcl;                    // "класс" окна
      /*  Определение элементов "класса"  окна */
  wcl.hInstance = hThisInst;      //дескриптор данного экземпляра
  wcl.lpszClassName = szWinName;  // имя "класса" окна
             */
  wcl.lpfnWndProc = WindowFunc;   // функция окна
             */
  wcl.style = 0;
  wcl.cbSize = sizeof(WNDCLASSEX);
  wcl.hIcon = LoadIcon(NULL,IDI_APPLICATION);
  wcl.hIconSm = LoadIcon(NULL, IDI_WINLOGO);
  wcl.hCursor = LoadCursor(NULL, IDC_ARROW);
  wcl.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1); // меню
  wcl.cbClsExtra = 0; wcl.cbWndExtra = 0;
      /* Фон окна задается белым   */
  wcl.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
      /*         Регистрация "класса" окна        */
  if (!RegisterClassEx(&wcl))  return 0;
      /*         Создание окна                    */
  hWnd = CreateWindow
  (  szWinName,
     "Демонстрация вывода графики",
     WS_OVERLAPPEDWINDOW,
     CW_USEDEFAULT,  CW_USEDEFAULT,
     CW_USEDEFAULT,  CW_USEDEFAULT,
     HWND_DESKTOP,   NULL,
     hThisInst,  NULL);
      /*              Отображение окна            */
  ShowWindow(hWnd, nWinMode);
  UpdateWindow(hWnd);
      /*              Цикл сообщений              */
  while (GetMessage(&msg, NULL, 0, 0))
  { TranslateMessage(&msg);
    DispatchMessage (&msg);
  }
  return msg.wParam;
}
```

```
/*-------------------------------------------------*/
HDC memdc;          // контекст устройства памяти
HBITMAP hbit;       // растр изображения в окне
HBRUSH hbrush;      // дескриптор текущей кисти
HBRUSH hOldbrush;          // дескриптор прежней кисти
HPEN hRedpen, hBluepen;    // дескрипторы перьев
HPEN hOldpen;              // дескриптор прежнего пера
int maxX, maxY;            // размеры экрана
 /*   массивы точек  - координаты вершин треугольников     */
 POINT mp1[3]={{150,50},{50,150},{150,150}};// 1-й треуг-к
 POINT mp2[3]={{250,150},{350,50},{450,150}};// 2-й треуг-к
 COLORREF color= RGB(255,255,0);   //  цвет кисти
 /*---------------------------------------------------*/
 /*                      Функция  окна                */
 /*---------------------------------------------------*/
   LRESULT CALLBACK WindowFunc (HWND hWnd, UINT message,
                        WPARAM wParam, LPARAM lParam)
{HDC hdc;                  // контекст устройства окна
 PAINTSTRUCT paintstruct;  // характеристики области перерисовки
 switch(message)
 { case WM_CREATE:
       /* получение размеров окна */
       maxX = GetSystemMetrics (SM_CXSCREEN);
       maxY = GetSystemMetrics (SM_CYSCREEN);
       /* создание контекста устройства окна                */
       hdc = GetDC (hWnd);
       /* создание совместимого контекста устройства памяти */
       memdc = CreateCompatibleDC (hdc);
       /* создание совместимого растра                      */
       hbit = CreateCompatibleBitmap (hdc, maxX, maxY);
/* выбор растрового изображения в контекст устройства памяти */
       SelectObject (memdc, hbit);
       /* заполнение окна белой кистью                      */
       PatBlt (memdc, 0, 0, maxX, maxY, PATCOPY);
       /* создание красного и синего перьев                 */
       hRedpen = CreatePen (PS_SOLID, 2, RGB (200, 0, 0));
       hBluepen = CreatePen (PS_SOLID, 4, RGB (0,0,255));
       /* освобождение контекста устройства окна            */
       ReleaseDC (hWnd, hdc);
       break;
 case WM_COMMAND:                  // выбрана команда меню
  switch (wParam)
  {
    case ID_TREUG:
       PatBlt (memdc, 0, 0, maxX, maxY, PATCOPY);
       hbrush = CreateSolidBrush (color);
       hOldbrush = (HBRUSH)SelectObject (memdc,hbrush );
       hOldpen = (HPEN)SelectObject (memdc,hRedpen );
       Rectangle (memdc, 50, 50, 200, 200);
       SelectObject (memdc, hBluepen);
       RoundRect (memdc, 250,50, 400, 150, 30, 30);
       SelectObject (memdc, hOldpen);
       SelectObject (memdc, hOldbrush);
```

```
      DeleteObject (hbrush);
      Polygon(memdc,mp1,3);
      InvalidateRect (hWnd, NULL, 0);
      break;
    case ID_RED:
      color = RGB(250,0,0);
      break;
  }
  break;
 case WM_PAINT:                    //обновление окна
      hdc=BeginPaint (hWnd, &paintstruct);
      BitBlt (hdc, 0, 0, maxX, maxY, memdc, 0, 0, SRCCOPY);
      EndPaint (hWnd, &paintstruct);
      break;
  case  WM_DESTROY:                // "завершить программу"
      /* удаление перьев */
      DeleteObject (hRedpen);
      DeleteObject (hBluepen);
      DeleteDC (memdc);           // удаление контекста памяти
      PostQuitMessage(0);
      break;
  default:
      return DefWindowProc ( hWnd, message, wParam, lParam );
  }
  return 0;
 }
```

Файл lab_5.rc

```
#include "resource.h"
IDR_MENU1 MENU DISCARDABLE
BEGIN
    POPUP "&Фигуры"
    BEGIN
        MENUITEM "&Треугольники",       ID_TREUG
    END
    POPUP "&Цвет"
    BEGIN
        MENUITEM "Красный",             ID_RED
    END
END
```

Файл resource.h

```
#define IDR_MENU1          101
#define ID_TREUG           40003
#define ID_RED             40004
```

Приложение 8. Программа, демонстрирующая работу с меню, кнопками, редакторами и таймером

```
#include "windows.h"
#include "resource.h"
LRESULT CALLBACK WindowFunc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE hInstance;     // дескриптор приложения
HWND BtHWnd;             //    кнопка
HWND EdtHWnd;            //    редактор
char szWinName[] ="MyWin";    // имя класса окна
/*             Главная функция                      */
int WINAPI WinMain(HINSTANCE hThisInst, HINSTANCE hPrevInst,
LPSTR lpszArgs, int nWinMode )
{    HWND hWnd;     MSG msg;
     WNDCLASSEX wcl;
     wcl.hInstance=hThisInst;
     wcl.lpszClassName=szWinName;
     wcl.lpfnWndProc=WindowFunc;
     wcl.style=0;
     wcl.cbSize = sizeof(WNDCLASSEX);
     wcl.hIcon = LoadIcon(NULL,IDI_APPLICATION);
     wcl.hIconSm = LoadIcon(NULL,IDI_WINLOGO);
     wcl.hCursor = LoadCursor(NULL,IDC_ARROW);
     wcl.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
     wcl.cbClsExtra = 0; wcl.cbWndExtra = 0;
wcl.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
     if(!RegisterClassEx(&wcl)) return 0;
     hWnd = CreateWindow (
          szWinName,
          "Menu & Timer & Edit & Button", // заголовок
          WS_OVERLAPPEDWINDOW,
          CW_USEDEFAULT, CW_USEDEFAULT,
          CW_USEDEFAULT, CW_USEDEFAULT,
          HWND_DESKTOP,  NULL, hThisInst, NULL);
/* запоминаем дескриптор приложения
для создания дочерних элементов управления */
     hInstance = hThisInst;
     ShowWindow(hWnd,nWinMode);
     UpdateWindow(hWnd);
     while (GetMessage(&msg,NULL,0,0))
     {
          TranslateMessage(&msg);
          DispatchMessage(&msg);
     }
     return msg.wParam;
}
/*                 Функция окна                  */
LRESULT CALLBACK WindowFunc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
{    char msg[10];
     int r;
     switch (message){
```

```
case WM_DESTROY:              // завершение программы
     PostQuitMessage(0);
     break;
/* сообщение от пунктов меню и элементов управления */
case WM_COMMAND:
     if(LOWORD(wParam)==IDM_START) //пункт меню - Start
     { MessageBoxA(hWnd,"Timer started",
               "Generation started",MB_OK);
       SetTimer(hWnd,1,1000,NULL); // установка таймера
     }
     if(LOWORD(wParam)==IDM_STOP) // пункт меню - Stop
     { MessageBoxA(hWnd,"Timer stoped",
               "Generation stopped",MB_OK);
       KillTimer(hWnd,1); // завершение работы таймера
     }
     if(HWND(lParam)==BtHWnd) // сообщение от кнопки
     { lstrcpy(msg,"0");
     /* помещение текста в поле ввода            */
     SendMessage(EdtHWnd,WM_SETTEXT,0,(LPARAM)(LPCTSTR)msg);
     }
     break;

case WM_TIMER:                // сообщение от таймера
     int k;
     r=rand()%20;
     /* получение длины текста в поле ввода      */
     k=SendMessage(EdtHWnd,WM_GETTEXTLENGTH,0,0);
     /* получение текста из поля ввода           */
     SendMessage(EdtHWnd,WM_GETTEXT,k+1,(LPARAM)msg);
     /* перевод значения из строки в число и суммирование*/
     r+=atoi(msg);
     /* перевод из числа в строку                */
     wsprintfA(msg,"%d",r);
     /* помещение текста в поле ввода            */
     SendMessage(EdtHWnd,WM_SETTEXT,0,(LPARAM)(LPCTSTR)msg);
     break;

case WM_CREATE:               // сообщение о создании окна
/*                  создание кнопки              */
BtHWnd = CreateWindow (
     "BUTTON",                // имя класса окна
     "Reset",                 // заголовок
     WS_CHILD|WS_VISIBLE|WS_BORDER,    // стиль
     5, 10, 50, 40,           // координаты и размеры
     hWnd,               // дескриптор родительского окна
     NULL,               // меню нет
     hInstance,          // дескриптор приложения
     NULL);
/*       создание редактора(поля ввода)         */
EdtHWnd = CreateWindow(
     "EDIT",                  // имя класса окна
     "0",                     // заголовок
     WS_CHILD|WS_VISIBLE|WS_BORDER|WS_THICKFRAME,
```

```
            60, 10, 70, 70,          // координаты и размеры
            hWnd,                // дескриптор родительского окна
            NULL,                // меню нет
            hInstance,           // дескриптор приложения
            NULL);
        default:
            return DefWindowProc(hWnd, message,wParam,lParam);
        }
        return 0;
}
```

Файл lab_5_1.rc

```
#include "resource.h"
IDR_MENU1 MENU
BEGIN
    POPUP "Generator"
    BEGIN
        MENUITEM "Start", IDM_START
        MENUITEM "Stop",  IDM_STOP
    END
END
```

Файл resource.h

```
#define IDR_MENU1                   101
#define IDB_BT1                     103
#define IDM_START                   40001
#define IDM_STOP                    40002
```