Тема 2.
Каркас приложения

Каркас приложения

Полноценная программа для Win32 должна содержать как минимум две функции:

- WinMain — главную функцию, в которой создается основное окно программы и запускается цикл обработки сообщений;
- WndProc — оконную процедуру, обеспечивающую обработку сообщений для основного окна программы.

Каркас приложения

На некотором псевдоязыке каркас Windows-программы можно представить следующим образом:

```
WinMain (список аргументов) {
    Подготовить и зарегистрировать класс окна с требуемыми
характеристиками;
    Создать экземпляр окна зарегистрированного класса;
    Пока не произошло необходимое для выхода событие
    {
       Извлечь очередное сообщение из очереди сообщений;
       Передать его через Windows оконной функции;
     }
    Возврат из программы;
}

WndProc (список аргументов) {
    Обработать полученное сообщение;
    Возврат;
}
```

Главная функция WinMain()

```
int WINAPI WinMain (HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpszCmdParam, int nCmdShow);
```

WINAPI — идентификатор типа, определяется как *stdcall*

Параметры:

hInstance — дескриптор приложения;

hPrevInstance — дескриптор предыдущего экземпляра приложения (в Win32 это всегда NULL);

lpszCmdParam — указатель на командную строку;

nCmdShow — состояние окна при начальном запуске.

Функция WinMain после своего завершения возвращает в Windows целочисленный результат.

Главная функция WinMain()

Последовательность действий:

1) *Зарегистрировать* в системе Windows класс главного окна.
2) *Создать* главное окно и *показать* его на экране.
3) Организовать *цикл обработки сообщений*, поступающих в приложение.

Регистрация класса окна

Оконный класс (window class), или *класс окна* — это структура, определяющая основные характеристики окна. Структура WNDCLASS имеет следующее описание:

```
typedef struct _WNDCLASS {
    UINT style;
    WNDPROC lpfnWndProc; // имя оконной функции
    int cbClsExtra; // кол-во доп. байт для размещения
данной структуры (все  =  0, устанавливаются системой)
    int cbWndExtra; // кол-во доп. байт для размещения
данного приложения (все  =  0, устанавливаются системой)
    HANDLE hInstance; // дескриптор приложения
    HICON hIcon; // дескриптор пиктограммы
    HCURSOR hCursor; // дескриптор курсора
    HBRUSH hbrBackground; // дескриптор цветной кисти
    LPCTSTR lpszMenuName; // идентификатор ресурса меню
    LPCTSTR lpszClassName; // имя для окон данного класса
} WNDCLASS;
```

Регистрация класса окна

```
        /*Зарегистрируем класс главного окна*/
WNDCLASS wc; //Структура для задания характеристик окна
memset( &wc, 0, sizeof(wc) ); //Обнуление всех членов
//или
ZeroMemory (&wc,sizeof(wc)); //Обнуление всех членов

wc.style  =  CS_HREDRAW | CS_VREDRAW; //Определяем стиль окна
wc.lpfnWndProc  =  WndProc; //Определяем оконную процедуру
wc.hInstance  =  hInstance; //Дескриптор приложения
wc.hIcon = LoadIcon( NULL, IDI_APPLICATION );//Пиктограмма
wc.hCursor  =  LoadCursor( NULL, IDC_ARROW ); //Курсор мыши

wc.hbrBackground  =  (HBRUSH)( COLOR_WINDOW + 1 ); //Белый фон
//или
wc.hbrBackground  =  CreateSolidBrush(RGB(0,0,255)); //Синий фон

wc.lpszClassName  =  L"HELLO"; //Имя класса главного окна

if (!RegisterClass(&wc)) //Собственно регистрация класса окна
   return FALSE;
```

Предопределенные идентификаторы пиктограмм

Значение	Описание
IDI_APPLICATION	Пиктограмма приложения по умолчанию
IDI_ASTERISK	То же, что и IDI_INFORMATION
IDI_ERROR	Пиктограмма в виде белого креста на фоне красного круга. Она используется в серьезных предупреждающих сообщениях
IDI_EXCLAMATION	То же, что и IDI_WARNING
IDI_HAND	То же, что и IDI_ERROR
IDI_INFORMATION	Пиктограмма «i», которая используется в информационных сообщениях
IDI_QUESTION	Пиктограмма «?»
IDI_WARNING	Пиктограмма «!», которая используется в предупреждающих сообщениях
IDI_WINLOGO	Логотип Windows

Предопределенные идентификаторы курсора

Значение	Описание
IDC_APPSTARTING	Стандартная стрелка и малые песочные часы
IDC_ARROW	Стандартная стрелка
IDC_CROSS	Перекрестье
IDC_HELP	Стрелка и вопросительный знак
IDC_IBEAM	Текстовый двутавр
IDC_NO	Перечеркнутый кружок
IDC_SIZEALL	Четырехконечная стрелка
IDC_SIZENESW	Двухконечная стрелка, указывающая на северо-восток и юго-запад
IDC_SIZENS	Двухконечная стрелка, указывающая на север и юг
IDC_SIZENWSE	Двухконечная стрелка, указывающая на северо-запад и юго-восток
IDC_SIZEWE	Двухконечная стрелка, указывающая на запад и восток
IDC_UPARROW	Вертикальная стрелка
IDC_WAIT	Песочные часы

Создание и показ окна

Для создания окна, в частности главного окна приложения, используется функция Windows *CreateWindow()*, требующая при вызове указания 11 параметров:

```
HWND CreateWindow(
    LPCTSTR lpClassName,   // имя зарегистрированного класса
    LPCTSTR lpWindowName,  // имя окна
    DWORD dwStyle,  // стиль окна
    int x, int y,   // горизонтальная и вертикальная позиции
    int nWidth, int nHeight, // ширина и высота окна
    HWND hWndParent, // дескриптор родительского окна
    HMENU hMenu,    // дескриптор меню окна или идентификатор
                    // элемента управления
    HINSTANCE hlnstance, // дескриптор экземпляра приложения
    LPVOID lParam);   // указатель на данные,
                    //передаваемые в сообщении WM_CREATE
```

Отображение на экране созданного окна

```
BOOL ShowWindow(HWND hWnd, int nCmdShow);
```

Создание и показ окна

```
/*Создадим главное окно и сделаем его видимым*/
    HWND hwnd;
    hwnd  =  CreateWindow(L"HELLO", //Класс окна
       L"HELLO",  WS_OVERLAPPEDWINDOW,//Заголовок, стиль окна
       CW_USEDEFAULT, 0, //Координаты,
       CW_USEDEFAULT, 0, //Размеры
       NULL, NULL, //Родитель, меню
       hInstance, NULL); //Дескриптор приложения, параметры

       ShowWindow(hwnd, nCmdShow);
       UpdateWindow(hwnd);
```

Обработка сообщений

Сообщения в Windows описываются с помощью структуры MSG:

```
typedef struct tagMSG {
  HWND hwnd;      //Дескриптор окна, которому
                  //адресовано сообщение
  UINT  message; //Код данного сообщения
  WPARAM  wParam; //Первая группа параметров сообщения
  LPARAM  lParam; //Вторая группа параметров сообщения
  DWORD  time;    //Время отправления сообщения
  POINT  pt;     //Позиция курсора мыши
} MSG;            //Новое имя для типа tagMSG
```

Часто используемые сообщения

Сообщение	Назначение
WM_CLOSE	Уведомляет окно о том, что оно должно быть закрыто
WM_COMMAND	Сообщение посылается, когда пользователь выбирает команду меню или посылает команду из элемента управления
WM_CREATE	Посылается, когда приложение создает окно вызовом функции CreateWindow или CreateWindowEx
WM_DESTROY	Посылается оконной процедуре уничтожаемого окна после того, как окно удалено с экрана
WM_INITDIALOG	Сообщение посылается оконной процедуре диалогового окна непосредственно перед тем, как оно будет отображено на экране
WM_MOVE	Посылается окну, которое переместилось на экране
WM_PAINT	Посылается окну, содержимое которого требует перерисовки
WM_SIZE	Посылается окну, размеры которого изменились
WM_TIMER	Уведомляет окно о том, что некоторый системный таймер, установленный функцией SetTimer, отсчитал заданный ему интервал

Функции обработки сообщений

Извлечение очередного сообщения осуществляется с помощью функции

```
BOOL GetMessage(LPMSG lpMsg, HWND hWnd,
                UINT wMsgFilterMin, UINT wMsgFilterMax);
```

Параметры:

- *lpMsg* - адрес структуры типа MSG, в которую помещается выбранное сообщение;
- *hWnd* - дескриптор окна, принимающего сообщение;
- *wMsgFilterMin* - минимальный номер принимаемого сообщения;
- *wMsgFilterMax* - максимальный номер принимаемого сообщения .

Если оба последних параметра равны нулю, то функция выбирает из очереди любое очередное сообщение.

Функции обработки сообщений

Функция GetMessage возвращает значение TRUE при извлечении любого сообщения, кроме одного — WM_QUIT. Получив сообщение WM_QUIT, функция возвращает значение FALSE.

Функция

```
void TranslateMessage (LPMSG lpMsg);
```

передает структуру , адресуемую `lpMsg`, в Windows для преобразования какого-либо сообщения с клавиатуры.

Функция

```
void DispatchMessage (LPMSG lpMsg);
```

передает структуру, адресуемую `lpMsg`, обратно в Windows.

Цикл обработки сообщений

```
        /*Организуем цикл обработки сообщений*/
MSG msg; //Структура msg для получения сообщений Windows

while( GetMessage(&msg, NULL, 0, 0)) //Получить сообщение,
{
       TranslateMessage(&msg); //преобразование сообщений от
                               //клавиатуры (может быть
                               //опущено)
       DispatchMessage(&msg); //вызвать WndProc
}

return msg.wParam;
```

Оконная функция

Оконная функция — это «функция обратного вызова», предназначенная для обработки сообщений, адресованных любому окну того «оконного класса», в котором содержится ссылка на данную процедуру.

Формат:

```
LRESULT CALLBACK имя_функции(HWND hWnd, UINT uMsg,
                 WPARAM wParam, LPARAM lParam);
```

Параметры:

hWnd - дескриптор окна, которому адресовано сообщение;

uMsg - код передаваемого сообщения;

wParam - первая группа параметров сообщения;

lParam - вторая группа параметров сообщения.

Оконная функция (пример)

```
    /*Оконная функция WndProc главного окна*/
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg,
            WPARAM wParam, LPARAM lParam )
{
 switch(uMsg) { //Переход по uMsg – номеру сообщения
    case WM_DESTROY: //При закрытии приложения пользователем
            PostQuitMessage(0); //Завершим приложение
            break;
    default: //Остальные сообщения обработать по умолчанию
            return DefWindowProc(hwnd, msg, wParam, lParam);
    }
    return 0;
 }
```

Обработка сообщений

Оконная функция должна обрабатывать *все* поступающие в нее сообщения, и ее текст должен быть приблизительно таким:

```
switch(msg) {
    case WM_CREATE:
      ...//Обработка сообщения WM_CREATE
    case WM_MOUSEMOVE:
      ...//Обработка сообщения WM_MOUSEMOVE
    case WM_TIMER:
      ...//Обработка сообщения WM_TIMER
    case WM_DESTROY:
      ...//Обработка сообщения WM_DESTROY
    //и т. д. для всех возможных сообщений
    }
```

Макрос HANDLE_MSG

Заметного упрощения структуры программы можно добиться, используя группу макросов **HANDLE_MSG**, определенных в файле <WINDOWSX.H>:

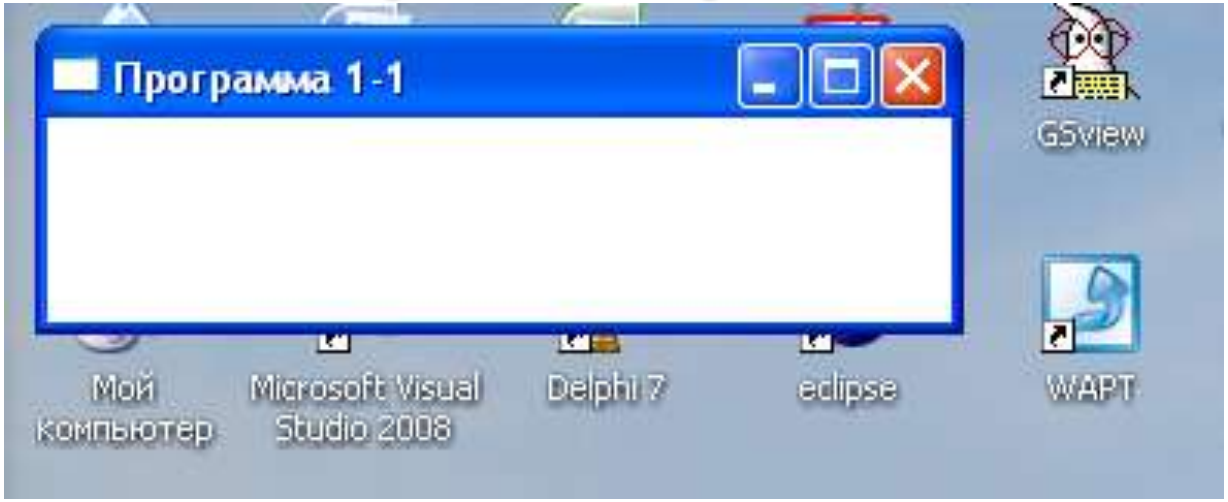
```
#define HANDLE_MSG(hwnd, message, fn)     \
    case (message):                       \
    return HANDLE_##message((hwnd), (wParam), (lParam), (fn))
```

При использовании этих макросов все процедуры обработки сообщений выделяются в *отдельные функции*, а в оконной функции **WndProc()** остаются только строки с макросами **HANDLE_MSG** (по числу обрабатываемых в программе сообщений), которые обеспечивают передачу управления на эти функции при приходе того или иного сообщения.

Оконная функция с макросами

```
void OnDestroy(HWND);
/*Оконная функция WndProc главного окна*/
LRESULT CALLBACK WndProc(HWND hwnd,UINT msg, WPARAM
wParam,LPARAM lParam)
{
        switch(msg)   {
            HANDLE_MSG(hwnd,WM_DESTROY,OnDestroy);
          default:
             return
              (DefWindowProc(hwnd,msg,wParam,lParam));
      }
}
/*Функция OnDestroy обработки сообщения WM_DESTROY*/
void OnDestroy(HWND)
{
      PostQuitMessage(0);//Завершим программу
}
```

Каркас приложения




```
/*Пример 1-1. Простейшая программа с главным окном*/
/*Операторы препроцессора*/
#include <windows.h>  // Два файла с определениями,
#include <windowsx.h> // макросами и прототипами
                      //функций Windows
/*Прототип используемой в программе функции
пользователя*/
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
```

Каркас приложения

```
/*Главная функция WinMain*/
int WINAPI WinMain(HINSTANCE hInst, HINSTANCE, LPSTR,
                   int nCmd)
{
      //Имя класса главного окна
      char szClassName[] = "MainWindow";
      char szTitle[] = "Программа 1-1";//Заголовок окна

      //Структура msg для получения сообщений Windows
      MSG msg;
      //Структура wc для задания характеристик окна
      WNDCLASS wc;

      /*Зарегистрируем класс главного окна*/
      //Обнуление всех членов wc
      ZeroMemory (&wc,sizeof(wc));
```

Каркас приложения

```
// Определяем оконную процедуру
wc.lpfnWndProc = WndProc;
wc.hInstance = hInst;//Дескриптор приложения
// Пиктограмма
wc.hIcon = LoadIcon(NULL,IDI_APPLICATION);
// Курсор мыши
wc.hCursor = LoadCursor(NULL,IDC_ARROW);
// Белый фон
wc.hbrBackground = GetStockBrush(WHITE_BRUSH);
wc.lpszClassName = szClassName;

//Собственно регистрация класса окна
RegisterClass(&wc);
```

Каркас приложения

```
  /*Создадим главное окно и сделаем его видимым*/
    HWND hwnd = CreateWindow(LPCSTR(szClassName),
                                  //Класс окна
       LPCSTR(szTitle), WS_OVERLAPPEDWINDOW,
                           //Заголовок, стиль окна
       10, 10, 300, 100, //Координаты, размеры
       NULL, NULL, //Родитель, меню
       hInst, NULL); //Дескриптор приложения, параметры

       ShowWindow(hwnd, nCmd); //Покажем окно

        /*Организуем цикл обработки сообщений*/
        while(GetMessage(&msg,NULL,0,0))
                                  //Получить сообщение,
            DispatchMessage(&msg); //вызвать WndProc
            return 0;    //После выхода из цикла
                         // вернуться в Windows
}   //Конец функции WinMain
```

Каркас приложения

```
/*Оконная функция WndProc главного окна*/
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
                         WPARAM wParam,LPARAM lParam)
{
     switch(msg)
     { //Переход по значению msg – номеру сообщения
       case WM_DESTROY: //При закрытии приложения
                        // пользователем
            PostQuitMessage(0); //Завершим приложение
            return 0; //Возврат в Windows
       default: //Остальные сообщения обработать
                //по умолчанию
       return(DefWindowProc(hwnd,msg,wParam,lParam));
     }//Конец оператора switch

}//Конец функции WndProc
```

Спасибо за внимание!