

Макрос HANDLE_MSG

Макрос `HANDLE_MSG`

- Оконная функция должна представлять собой один длинный оператор **switch** со столькими блоками **case**, сколько сообщений Windows предполагается обрабатывать в программе.
- При обработке ряда сообщений, например `WM_COMMAND`, внутри блоков **case** приходится включать вложенные операторы **switch-case**, да еще не одного уровня вложенности.
- В результате функция `WndProc()` становится чрезвычайно длинной и запутанной.
- Весьма полезная идея структурированности программы исчезает почти полностью, так как все приложение оказывается состоящим из едва ли не единственной функции `WndProc()` со множеством разветвлений внутри.
- Заметного упрощения структуры программы можно добиться, используя группу макросов `HANDLE_MSG`.

Структура программы с макросом HANDLE_MSG

- В файле WINDOWSEX.H. определена группа макросов **HANDLE_MSG**, позволяющая упростить структуру программы.
- При использовании этих макросов:
 - все процедуры обработки сообщений выделяются в *отдельные функции*,
 - в оконной функции WndProc() остаются только строки переключения на эти функции при приходе того или иного сообщения.
- Оконная функция, даже при большом количестве обрабатываемых сообщений, становится
 - короткой
 - наглядной
- наличие же для обработки каждого сообщения отдельной функции также весьма упрощает разработку их алгоритмов, и особенно отладку.

Модификация программы

- Модифицируем программу, введя, в ее оконную функцию макрос `HANDLE_MSG`.
- Фактически изменению подвергнется только оконная функция.

• **/*Операторы препроцессора*/**

- `#include <windows.h>` //Два файла с определениями, макросами
- `#include <windowsx.h>` //и прототипами функций Windows

Модификация программы

- ***/*Прототип используемой в программе функции пользователя*/***

```
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);  
    //Оконная функция
```

```
void OnDestroy(HWND);           / Прототип функции OnDestroy
```

Модификация программы

/*Главная функция WinMain*/

```
int WINAPI WinMain(HINSTANCE hInst,HINSTANCE,LPSTR,int)
{
    char szClassName[]="MainWindow"; //Произвольное имя класса главного окна
    char szTitle[]="Программа MainWindow"; //Произвольный заголовок окна
    MSG Msg; //Структура Msg типа MSG для получения
              // сообщений Windows
    WNDCLASS wc; //Структура wc типа WNDCLASS
                //для задания характеристик окна
```

Модификация программы

/*Зарегистрируем класс главного окна*/

`memset(&wc,0,sizeof(wc)); //Обнуление всех членов структуры wc`

`wc.lpfnWndProc=WndProc; //Определим оконную процедуру для главного окна`

`wc.hInstance=hInst; //Дескриптор приложения`

`wc.hIcon=LoadIcon(NULL,IDI_APPLICATION);//Стандартная пиктограмма`

`wc.hCursor=LoadCursor(NULL,IDC_ARROW); //Стандартный курсор мыши`

`wc.hbrBackground=GetStockBrush(LTGRAY_BRUSH);//Светло-серый фона окна`

`wc.lpszClassName=szClassName; //Имя класса окна`

`RegisterClass(&wc); //Вызов функции Windows регистрации класса окна`

Модификация программы

/*Создадим главное окно и сделаем его видимым*/

```
HWND hwnd=CreateWindow(szClassName,szTitle, //Класс и заголовок окна
    WS_OVERLAPPEDWINDOW,10,10,300,100, //Стиль окна, координаты
    //размеры
    HWND_DESKTOP,NULL,hInst,NULL); //Родитель, меню, другие
    //параметры

ShowWindow(hwnd, SW_SHOWNORMAL); //Вызов функции Windows показа
    //окна
```

Модификация программы

/*Организуем цикл обработки сообщений*/

```
while(GetMessage(&Msg,NULL,0,0)) //Цикл обработки сообщений:  
    DispatchMessage(&Msg);    //получить сообщение, вызвать WndProc  
  
return 0;                        //После выхода из цикла вернуться в  
    //Windows  
}  
//Конец функции WinMain
```

Модификация программы

/*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 завершении приложения пользователем
```

```
        PostQuitMessage(0); //Вызов функции Windows завершения приложение
```

```
        return 0; //Возврат в Windows
```

```
✓ HANDLE_MSG(hwnd, WM_DESTROY, OnDestroy);
```

```
default:                //В случае всех остальных сообщений Windows обработка
```

```
    return(DefWindowProc(hwnd,msg,wParam,lParam)); //их по умолчанию
```

```
    }                    //Конец оператора switch
```

```
}                        //Конец функции WndProc
```

Модификация программы



```
/*Функция OnDestroy обработки сообщения WM_DESTROY*/  
void OnDestroy(HWND)  
{  
    PostQuitMessage(0); //Вызов функции Windows завершения  
    //приложения  
} //Конец функции OnDestroy
```

- В программе обрабатывается единственное сообщение WM_DESTROY. Соответственно в программу введена функция обработки этого сообщения OnDestroy.

Имена функций обработки сообщений

- В документации к Windows рекомендуется образовывать имена функций обработки сообщений из
 - ✓ – имени класса окна
 - ✓ – значка подчеркивания
 - слова On
 - имени соответствующего сообщения
- В нашей программе имя функций обработки сообщений должны выглядеть таким образом:
 - MainWindow_OnDestroy() ;
 - MainWindow_OnPaint() ;
 - MainWindow_OnCommand();
- Однако для функций обработки сообщений, поступающих в *главное* окно, будем ради краткости опускать префикс, характеризующий класс.
- Для функций, относящихся к внутренним окнам, префикс класса придется использовать, так как разные функции, должны разумеется, иметь разные имена.

Новая функция

- Введение в программу новой функции требует определения ее прототипа. Соответственно в раздел прототипов приложения включена строка

```
void OnDestroy(HWND);           //Прототип      функции      обработки  
                                //сообщения WM_DESTROY
```

- Функция **OnDestroy()** помещена в конце программы, после оконной функции `WndProc()`.
- Разумеется, порядок функций в исходном тексте программы не имеет никакого значения и может выбираться по усмотрению программиста, исходя из соображений максимальной наглядности текста программы.
- При описании прототипов функций обработки отдельных сообщений и при составлении текстов самих этих функций возникает вопрос об их параметрах и возвращаемых значениях.
- Наша функция `OnDestroy()`
 - ничего не возвращает и
 - требует один параметр типа `HWND` (очевидно, дескриптор главного окна).

- Однако в случае других функций это не так.
- Для каждой функции обработки того или иного сообщения характерен свой набор параметров и свой тип возвращаемого значения.
- Состав параметров определяется, характером сообщения
- Формально же состав и порядок параметров задаются макросами `HANDLE_MSG`
- Извлечь интересующую нас информацию о прототипе функции из текста макроса довольно затруднительно, даже если заниматься исследованием его структуры.
- Для облегчения программирования в файле `WINDOWSX.H` для каждого сообщения приведен прототип соответствующей функции с указанием
 - типа
 - порядка
 - смысла ее параметров.
- Более детальную информацию о данных, поступающих в приложение вместе с сообщением, можно получить с помощью интерактивного справочника среды программирования, вызвав справку по интересующему нас сообщению (например, `WM_DESTROY`). Таким образом, при написании функций обработки сообщений приходится постоянно обращаться к файлу `WINDOWSX.H` и справочной системе среды разработки.

Прототипы функций обработки сообщений

<i>Сообщение</i>	<i>Прототип функции обработки сообщения</i>
WM_COMMAND codeNotify);	void Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT
WM_CREATE lpCreateStruct);	BOOL Cls_OnCreate(HWND hwnd, CREATESTRUCT FAR*
WM_DESTROY	void Cls_OnDestroy(HWND hwnd);
WM_GETMINMAXINFO lpMinMaxInfo);	void Cls_OnGetMinMaxInfo(HWND hwnd, MINMAXINFO FAR*
WM_INITDIALOG	BOOL Cls_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam);
WM_MOUSEMOVE	void Cls_OnMouseMove(HWND hwnd, int x, int y, UINT keyFlags);

Сообщение**Прототип функции обработки сообщения**

WM_NOTIFY	BOOL	Cls_OnNotify(HWND hwnd, INT idCtrl, NMHDR* pnmh); ,
WM_PAINT	void	Cls_OnPaint(HWND hwnd);
WM_QUIT	void	Cls_OnQuit(HWND hwnd, int exitCode);
WM_RBUTTONDOWN	void	Cls_OnRButtonUp(HWND hwnd, int x, int y, UINT flags);
WM_SETCURSOR	BOOL	Cls_OnSetCursor(HWND hwnd, HWND hwndCursor, UINT codeHitTest, UINT msg);
WM_SETFOCUS	void	Cls_OnSetFocus(HWND hwnd, HWND hwndOldFocus);
WM_SHOWWINDOW	void	Cls_OnShowWindow(HWND hwnd, BOOL fShow, U?NT status);
WM_SIZE	void	Cls_OnSize(HWND hwnd, UINT state, int ex, int cy);
WM_SYSCHAR	void	Cls_OnSysChar(HWND hwnd, UINT ch, int cRepeat);
WM_SYSCOMMAND	void	Cls_OnSysCommand(HWND hwnd, UINT cmd, int x, int y);
WM_SYSKEY	void	Cls_OnSysKey(HWND hwnd, UINT vk, BOOL fDown, int cRepeat, UINT flags);
WM_TIMER	void	Cls_OnTimer(HWND hwnd, UINT id);

- Большинство функций обработки сообщений не имеет возвращаемых значений. Это создает дополнительные удобства; при обработке сообщений непосредственно в теле оконной функции не надо каждый раз выяснять с помощью справочной системы, какое значение следует возвращать после обработки данного сообщения.

Расширение макроса `HANDLE_MSG`

- Макрос `HANDLE_MSG` разворачивается в предложение языка C++ с ключевым словом **case**.
- Общий же для всех ключевых слов `case` оператор **switch** включается в текст оконной функции в явной форме:

```
LRESULT CALLBACK WndProc(HWND hwnd,UINT msg,WPARAM wParam,LPARAM  
    lParam)  
{  
switch(msg)  
{  
HANDLE_MSG(hwnd, WM_PAINT, OnPaint); // case WM_PAINT: OnPaint() ;  
HANDLE_MSG(hwnd, WM_DESTROY, OnDestroy);//case WM_DESTROY: OnDestroy();
```

Макрос HANDLE_MSG

- Для макроса HANDLE_MSG в составе файла WINDOWSX.H имеется следующее определение:

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

(знак обратной косой черты (\) обозначает переход на следующую строку).

Предложения

```
HANDLE_MSG(hwnd,WM_PAINT,OnPaint) ;
```

```
HANDLE_MSG(hwnd,WM_DESTROY,OnDestroy);
```

преобразуются в промежуточные макрорасширения

```
case (WM_PAINT):return
```

```
    HANDLE_WM_PAINT((hwnd),(wParam),(lParam),(OnPaint)) ;
```

```
case (WM_DESTROY):return
```

```
    HANDLE_WM_DESTROY((hwnd),(wParam),(lParam),(OnDestroy)) ;
```

- Знак ## в составе макроопределения обозначает сцепление (конкатенацию) и в данном случае служит для получения составных имен новых макросов HANDLE_WM_PAINT, HANDLE_WM_DESTROY и др.

Примеры

- Для каждого сообщения Windows в составе файла WINDOWSX.H имеется отдельный макрос такого вида, причем их макроопределения уже неодинаковы и зависят от характеристик конкретного сообщения.
- Для макроса `HANDLE_WM_DESTROY` дано следующее макроопределение:

```
#define HANDLE_WM_DESTROY(hwnd, wParam, lParam, fn) ((fn)(hwnd), 0L)
```

- Подставив это определение вместо `HANDLE_WM_DESTROY`
`case (WM_DESTROY):`
`return HANDLE_WM_DESTROY((hwnd),(wParam),(lParam),(OnDestroy))`
- и опустив ненужные скобки, получим окончательное макрорасширение:

```
case WM_DESTROY: return (OnDestroy (hwnd), 0L);
```

- По ходу расширения макроса убираются лишние (для данного сообщения) параметры `wParam` и `lParam` и образуется синтаксически правильное предложение `case`.

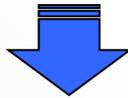
Как выполняется это предложение?

- Если пришло сообщение WM_DESTROY и аргумент msg функции WndProc() равен коду этого сообщения, то выполняется:
 - оператор return с двумя аргументами.
 - Прежде всего выполняется оператор, стоящий на месте первого аргумента, т. е. вызывается функция OnDestroy (hwnd). Эта функция не должна возвращать каких-либо значений.
 - После ее завершения срабатывает оператор return, возвращающий указанное значение - длинный 0.
- Любопытно, что завершающий знак ";", который обязательно должен быть в конце любого предложения языка C++, переходит в окончательный текст из *нашей* строки с макросом HANDLE_MSG.

Пример WM_PAINT:

- Схожим образом расширяется строка для сообщения WM_PAINT:

```
HANDLE_MSG(hwnd,WM_PAINT,OnPaint) ;
```



```
case WM_PAINT:return(OnPaint (hwnd),0L);
```

- Для других сообщений макросы вида `HANDLE_сообщение` имеют более сложные определения, в которых выполняются необходимые преобразования аргументов функции `WndProc()` в параметры функций обработки сообщений.

Пример HANDLE_WM_COMMAND:

Расширение макроса HANDLE_WM_COMMAND:

В результате наша строка

```
HANDLE_MSG(hwnd, WM_COMMAND, OnCommand) ;
```

```
#define HANDLE_MSG(hwnd, message, fn)\
```

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

сначала преобразуется в

```
case (WM_COMMAND):return HANDLE_WM_COMMAND((hwnd), (wParam), (lParam),  
    (OnCommand)) ;
```

а затем окончательно в

```
#define HANDLE_WM_COMMAND(hwnd,wParam,lParam,fn)\
```

```
((fn)((hwnd),(int)(wParam),(HWND)LOWORD(lParam),(UINT)(HIWORD(lParam)),0L)
```

```
case WM_COMMAND: return(OnCommand(hwnd,(int)wParam,(HWND)LOWORD(lParam),  
    (UINT)HIWORD(lParam)), 0L);
```

Функция обработки сообщения WM_COMMAND должна использоваться в соответствии со своим прототипом

```
void Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify);
```

Пример HANDLE_WM_COMMAND:

```
case WM_COMMAND: return(OnCommand(hwnd,(int)wParam,(HWND)LOWORD(lParam),  
    (UINT)HIWORD(lParam)), 0L);
```

Функция обработки сообщения `WM_COMMAND` должна использоваться в соответствии со своим прототипом

```
void Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify);
```

В результате в эту функцию, которая вызывается в случае выбора пользователем пункта меню или элемента управления диалогового окна, поступают следующие параметры:

- дескриптор окна `hwnd`;
- преобразованный в тип `int` аргумент `wParam` функции `WndProc()`, который определяет идентификатор выбранного пункта меню или элемента управления (например, кнопки);
- преобразованное в тип `HWND` младшее слово двухсловного аргумента `lParam`, которое определяет дескриптор окна, образующего выбранный элемент управления;
- преобразованное в тип `UINT` старшее слово того же двухсловного аргумента `lParam`, которое определяет код извещения, т. е. действие, выполненное над элементом управления.

Заключение

использование макроса `HANDLE_MSG`:

- существенно повышает структурированность программы, выделяя процедуры обработки сообщений в отдельные функции,
- позволяет распаковать аргументы функции `WndProc()`
- выделить из них данные, которые должны служить параметрами функций обработки конкретных сообщений
- в процессе выделения параметров выполняется необходимое преобразование их типов.

Макросы `HANDLE_MSG` часто называют "распаковщиками сообщений".