Тема 2. **Операционная система Windows.**

Программирование с использованием средств Win32 API.

Функция окна

Программа на Си/C++ для Windows, как и для любой другой платформы, должна обязательно содержать некоторую "стартовую" функцию, которой передается управление при запуске программы. Вообще говоря, имя такой "стартовой" функции может различаться в различных компиляторах, но исторически сложилось так (кроме того, имеются еще и стандарты ANSI и ISO), что такой функцией является:

int **main**()

У этой функции может быть до трех параметров:

int **main**(int argc, char *argv[], char *env[])

argc - количество параметров в командной строке (включая имя программы),

argv - массив строк-параметров (argv[0] - имя программы),

env - массив строк-переменных окружения.

Многие компиляторы для Windows "понимают" такую стартовую функцию. Однако при этом они [illegible] приложение.

```
#include <stdio.h>
int main()
 {
    printf("Hello, world!"); getc(stdin);
   return 0;
 }
```

Цель – создание графического приложения под Windows

Тогда используется другая стартовая функция

int WINAPI WinMain (HINSTANCE hInst, HINSTANCE hpi, LPSTR cmdline, int ss)

- hInst - дескриптор для данного экземпляра программы,
- hpi - в Win32 не используется (всегда NULL),
- cmdline - командная строка,
- ss - код состояния главного окна.

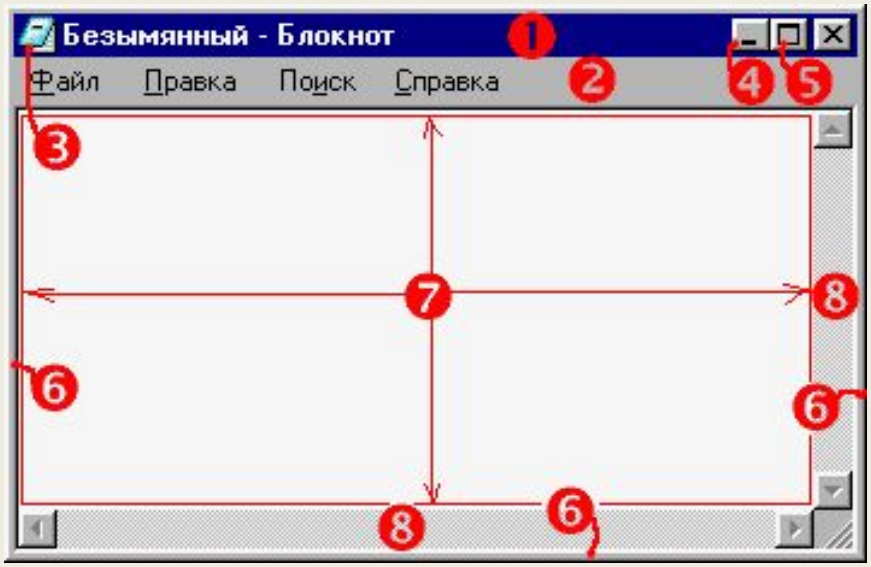
Кроме того, необходимо указать компилятору, что создается win32- GUI-приложение

```
#include <windows.h>
 int WINAPI WinMain(HINSTANCE,HINSTANCE,LPSTR,int)
{
    MessageBox(NULL,"Hello, World!","Test",MB_OK);
  return 0;
 }
```

Декскриптор - указатель на некоторую системную структуру или индекс в некоторой системной таблице.

```
typedef void *HANDLE; /* абстрактный дескриптор (например, файла) */
 typedef void *HMODULE; /* дескриптор модуля */
typedef void *HINSTANCE; /* дескриптор экземпляра программы */
typedef void *HKEY; /* дескриптор ключа в реестре */
 typedef void *HGDIOBJ; /* дескриптор графического примитива (перо,
шрифт, кисть, палитра,...) */
 typedef void *HWND; /* дескриптор окна */
typedef void *HMENU; /* дескриптор меню */
typedef void *HICON; /* дескриптор иконки */
typedef void *HBITMAP; /* дескриптор картинки */
typedef void *HFONT; /* дескриптор шрифта */
```

Структура окна приложения



- строка заголовка *title bar* (1),
- строка меню *menu bar* (2),
- системное меню *system menu* (3),
- кнопка сворачивания окна *minimize box* (4),
- кнопка разворачивания окна *maximize box* (5),
- рамка изменения размеров *sizing border* (6),
- клиентская область *client area* (7),
- горизонтальная и вертикальная полосы прокрутки *scroll bars* (8)

Меню, строка заголовка с системными кнопками, системное меню, рамка изменения размеров и полосы прокрутки относятся к области окна, называемой *неклиентской областью* (*non-client area*).

За содержимое и обслуживание клиентской области отвечает приложение.

Виды окон Windows

Кроме главного окна, приложение может использовать еще и другие типы окон: управляющие элементы (*controls*), диалоговые окна (*dialog boxes*), окна-сообщения (*message boxes*).

Управляющий элемент - окно, непосредственно обеспечивающее тот или иной способ ввода информации пользователем. К управляющим элементам относятся:
кнопки, поля ввода, списки, полосы прокрутки и т.п. Управляющие элементы обычно относятся к какому-либо диалоговому окну.

Диалоговое окно - это временное окно, напичканное управляющими элементами, обычно использующееся для получения дополнительной информации от пользователя. Диалоговые окна бывают модальные (*modal*) и немодальные (*modeless*). Модальное диалоговое окно требует, чтобы пользователь обязательно ввел обозначенную в окне информацию и закрыл окно прежде, чем приложение продолжит работу. Немодальное диалоговое окно позволяет пользователю, не закрывая диалогового окна, переключаться на другие окна этого приложения.

Окно-сообщение - это диалоговое окно предопределенного системой формата, предназначенное для вывода небольшого текстового сообщения с одной или несколькими кнопками.

В отличие от традиционного программирования на основе линейных алгоритмов, программы для Windows строятся по принципам событийно-управляемого программирования (*event-driven programming*) - стиля программирования, при котором поведение компонента системы определяется набором возможных внешних событий и ответных реакций компонента на них. Такими компонентами в Windows являются окна. С каждым окном в Windows связана определенная функция обработки событий. События для окон называются *сообщениями*. Сообщение относится к тому или иному типу, идентифицируемому определенным кодом (32-битным целым числом), и сопровождается парой 32-битных параметров (WPARAM и LPARAM), интерпретация которых зависит от типа сообщения.
В заголовочном файле windows.h для кодов сообщений определены константы с интуитивно понятными именами:

```
#define WM_CREATE 0x0001 /* сообщение о создании окна */
#define WM_DESTROY 0x0002 /* сообщение об уничтожении окна */
#define WM_COMMAND 0x0111 /*сообщение от команды меню или управляющего элемента */
```

Для стандартных управляющих элементов (библиотека Common Controls Library - COMCTL32.DLL) в Windows имеются предопределенные обработчики событий, которые при наступлении интересных событий сообщают полезную информацию окну, содержащему этот управляющий элемент. Стандартная библиотека Common Dialog Box Library (COMDLG32.DLL) содержит готовые полезные диалоговые окна с обработчиками: диалоги выбора файла, настроек печати, выбора шрифта, выбора цвета и др.

Структура программы Win32

```
#include <windows.h>
int WINAPI WinMain(HINSTANCE hInst,HINSTANCE,LPSTR cmdline,int ss)
{ /* Блок инициализации: создание класса главного окна, создание главного
окна, загрузка ресурсов и т.п. */

/* Цикл обработки событий: */
MSG msg; while (GetMessage(&msg,(HWND)NULL,0,0)) { TranslateMessage(&msg);
DispatchMessage(&msg); }
return msg.wParam;
}

LRESULT CALLBACK MainWinProc(HWND hw,UINT msg,WPARAM wp,LPARAM lp) {
 /* Обработка сообщений главного окна */
switch (msg) {
case WM_CREATE: /* ... */ return 0;
case WM_COMMAND: /* ... */ return 0;
 case WM_DESTROY: /* ... */ PostQuitMessage(0); return 0;
 /* ... */ }
return DefWindowProc(hw,msg,wp,lp) ;
}
```

Создание окон средствами Win32

Для создания окна вызывается функция:

```
HWND WINAPI CreateWindow(
LPCSTR lpClassName, /* имя класса */
LPCSTR lpWindowName, /* имя окна (заголовок) */
DWORD dwStyle, /* стиль (поведение) окна */
 int x, /* горизонтальная позиция окна на экране */
 int y, /* вертикальная позиция окна на экране */
 int nWidth, /* ширина окна */ int nHeight, /* высота окна */
HWND hWndParent, /* дескриптор родительского окна */
HMENU hMenu, /* дескриптор меню */
HANDLE hInstance, /* дескриптор экземпляра программы */
LPVOID lpParam /* указатель, обычно NULL */ )
```

Вместо параметров x, y, nWindth, nHeight допустимо передавать константу CW_USEDEFAULT, позволяющую операционной системе задать эти числа по ее усмотрению.

Интерпретация кода стиля определяется классом окна. Стиль определяет не только оформление окна, но и его поведение. Общие для всех классов константы стилей (при необходимости объединяются операцией побитовое ИЛИ):

Общие константы стилей

WS_DISABLED - при создании окно заблокировано (не может получать реакцию от пользователя);
WS_VISIBLE - при создании окно сразу же отображается (не надо вызывать **ShowWindow**);
WS_CAPTION - у окна есть строка заголовка;
WS_SYSMENU - у окна есть системное меню;
WS_MAXIMIZEBOX - у окна есть кнопка разворачивания;
WS_MINIMIZEBOX - у окна есть кнопка сворачивания;
WS_SIZEBOX или WS_THICKFRAME - у окна есть рамка изменения размеров;
WS_BORDER - у окна есть рамка (не подразумевает изменение размеров);
WS_HSCROLL или WS_VSCROLL - у окна есть горизонтальная или вертикальная прокрутка;
WS_OVERLAPPED или WS_TILED - "перекрываемое" окно - обычное окно с рамкой и строкой заголовка;
WS_POPUP - "всплывающее" окно;
WS_OVERLAPPEDWINDOW - "перекрываемое" окно с системным меню, кнопками сворачивания/разворачивания, рамкой изменения размеров, короче, типичный стиль для главного окна приложения.

Во время выполнения функции **CreateWindow** процедуре обработки событий окна посылается сообщение WM_CREATE. При успешном выполнении функции возвращается дескриптор созданного окна, при неудаче - NULL.

Работа с окнами

BOOL WINAPI **ShowWindow**(HWND hw, int ss)

Второй параметр этой функции - код состояния отображения окна. В качестве этого кода можно взять значение четвертого параметра, с которым была запущена функция **WinMain**.

Другие возможные значения этого параметра:

SW_SHOW - отобразить и активировать окно;
SW_HIDE - скрыть окно;
SW_MAXIMIZE - развернуть окно на весь экран;
SW_RESTORE - активировать окно и отобразить его в размерах по умолчанию;
SW_MINIMIZE - свернуть окно.

BOOL WINAPI **UpdateWindow**(HWND hw)

Windows использует два способа доставки сообщений процедуре обработки событий окна:

непосредственный вызов процедуры обработки событий (*внеочередные* или *неоткладываемые сообщения* - *nonqueued messages*);
помещение сообщения в связанный с данным приложением буфер типа FIFO, называемый *очередью сообщений* - *message queue*(*откладываемые сообщения* - *queued messages*).

К внеочередным сообщениям относятся те сообщения, которые непосредственно влияют на окно, например, сообщение активации окна WM_ACTIVATE и т.п. Кроме того, вне очереди сообщений обрабатываются сообщения, сгенерированные различными вызовами Win32 API, такими как **SetWindowPos**, **UpdateWindow**, **SendMessage**, **SendDlgItemMessage**...

Обработка сообщений окон

К откладываемым сообщениям относятся сообщения, связанные с реакцией пользователя: нажатие клавиш на клавиатуре, движение мышки и клики.

Чтобы извлечь сообщение из очереди, программа вызывает функцию

*BOOL WINAPI **GetMessage**(*
*MSG *lpmsg, /* сюда попадает сообщение со всякими параметрами */*
HWND hw, / извлекать только сообщения для указанного окна (NULL - все) */*
UINT wMsgFilterMin, / фильтр сообщений (нам не надо - ставим 0) */*
UINT wMsgFilterMax / фильтр сообщений (нам не надо - ставим 0) */*
)

Эта функция возвращает FALSE, если получено сообщение WM_QUIT, и TRUE в противном случае. Очевидно, что условием продолжения цикла обработки событий является результат этой функции. Если приложение хочет завершить свою работу, оно посылает само себе сообщение WM_QUIT при помощи функции

*void WINAPI **PostQuitMessage**(int nExitCode)*

Ее параметр - статус выхода приложения. Обычно эта функция вызывается в

Обработка сообщений окна

После извлечения сообщения из очереди следует вызвать функцию **TranslateMessage**, переводящую сообщения от нажатых клавиш в удобоваримый вид, а затем **DispatchMessage**, которая определяет предназначенное этому сообщению окно и вызывает соответствующую процедуру обработки событий.

*BOOL WINAPI **TranslateMessage**(const MSG *lpmsg)*

*LONG WINAPI **DispatchMessage**(const MSG *lpmsg)*

Результат возврата соответствует значению, которое вернула процедура обработки событий (обычно никому не нужен).

Процедура обработки сообщений окна должна быть объявлена по следующему прототипу:

LRESULT CALLBACK WindowProc(HWND hw,UINT msg,WPARAM wp,LPARAM lp)

Значения параметров: hw - дескриптор окна, которому предназначено сообщение, msg - код сообщения, wp и lp - 32-битные параметры сообщения, интерпретация которых зависит от кода сообщения.

Пример обработки сообщения окна

Например, сообщение WM_COMMAND посылается окну в трех случаях:

- пользователь выбрал какую-либо команду меню;
- пользователь нажал "горячую" клавишу (*accelerator*);
- в дочернем окне произошло определенное событие.

При этом параметры сообщения интерпретируются следующим образом.
Старшее слово параметра WPARAM содержит: 0 в первом случае, 1 во втором случае и код события в третьем случае.
Младшее слово WPARAM содержит целочисленный идентификатор пункта меню, "горячей" клавиши или дочернего управляющего элемента.
Параметр LPARAM в первых двух случаях содержит NULL, а в третьем случае - дескриптор окна управляющего элемента.

Процедура обработки событий должна вернуть определенное 32-битное значение, интерпретация которого также зависит от типа сообщения. В большинстве случаев, если сообщение успешно обработано, процедура возвращает значение 0.

Процедура обработки событий не должна игнорировать сообщения. Если процедура не обрабатывает какое-то сообщение, она должна вернуть его системе для обработки по умолчанию. Для этого вызывается функция:
LRESULT WINAPI **DefWindowProc**(HWND hw, UINT msg, WPARAM wp, LPARAM lp)

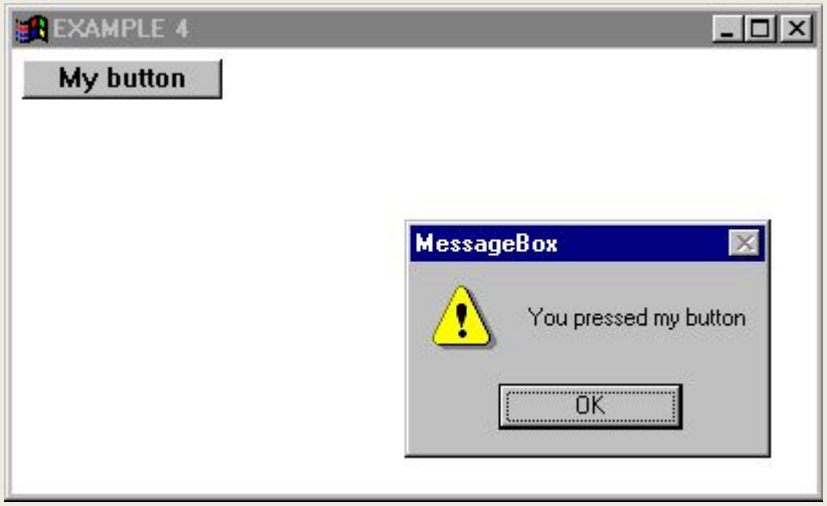
Пример приложения

```
 #include <windows.h>
LRESULT CALLBACK MainWinProc(HWND,UINT,WPARAM,LPARAM);
 #define ID_MYBUTTON 1 /* идентификатор для кнопки внутри главного окна */
 int WINAPI WinMain(HINSTANCE hInst,HINSTANCE,LPSTR,int ss) {
/* создаем и регистрируем класс главного окна */
 WNDCLASS wc;
 wc.style=0; wc.lpfnWndProc=MainWinProc; wc.cbClsExtra=wc.cbWndExtra=0;
 wc.hInstance=hInst; wc.hIcon=NULL; wc.hCursor=NULL;
 wc.hbrBackground=(HBRUSH)(COLOR_WINDOW+1); wc.lpszMenuName=NULL
; wc.lpszClassName="Example  MainWnd Class";
if (!RegisterClass(&wc)) return FALSE; /* создаем главное окно и отображаем его */
 HWND hMainWnd=CreateWindow("Example  MainWnd Class","EXAMPLE4 ",
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,
CW_USEDEFAULT,NULL,NULL,hInst,NULL);

if (!hMainWnd) return FALSE;

 ShowWindow(hMainWnd,ss);
 UpdateWindow(hMainWnd);
```

Пример приложения-2

```
MSG msg; /* цикл обработки событий */
 while (GetMessage(&msg,NULL,0,0)) { TranslateMessage(&msg);
 DispatchMessage(&msg); } return msg.wParam; }
/* процедура обработки сообщений для главного окна */
 LRESULT CALLBACK MainWinProc(HWND hw,UINT msg,WPARAM wp,LPARAM lp) {
 switch (msg) {
 case WM_CREATE: /* при создании окна внедряем в него кнопочку */
 CreateWindow("button","My button",WS_CHILD|BS_PUSHBUTTON|WS_VISIBLE, 5,5,100,20,
hw,(HMENU)ID_MYBUTTON,NULL,NULL);
/* стиль WS_CHILD означает, что это дочернее окно и для него вместо
дескриптора
 меню будет передан целочисленный идентификатор, который будет
использоваться
 дочерним окном для оповещения родительского окна через WM_COMMAND */
return 0;
 case WM_COMMAND: /* нажата наша кнопка? */
if ((HIWORD(wp)==0) && (LOWORD(wp)==ID_MYBUTTON))
   MessageBox(hw,"You pressed my button","MessageBox",MB_OK|MB_ICONWARNING);
    return 0;
 case WM_DESTROY: /* пользователь закрыл окно, программа может завершаться
*/
 PostQuitMessage(0);
 return 0; }
```

Результат запуска программы



Ресурсы Windows

Ресурсы - это бинарные данные, добавляемые в исполняемый файл при компоновке программы. В процессе разработки программы ресурсы описывают в отдельном текстовом файле - *файле описания ресурсов* , - а затем при помощи компилятора ресурсов переводят в бинарный вид и добавляют в исполняемый файл на этапе компоновки исполняемого файла. Использование ресурсов значительно облегчает работу программиста по визуализации графических примитивов интерфейса программы.

Виды ресурсов:

- иконки,
- курсоры, меню, диалоги,
- растровые изображения (BMP),
- векторные изображения (EMF),
- шрифты,
- таблицы горячих клавиш,
- таблицы строк, информация о версии программы или модуля

Файл описания ресурсов состоит из операторов, объединяемых в блоки. Один оператор занимает одну строку файла. Допускается использовать комментарии, определяемые так же, как в программе на языке Си.

Структура файла ресурсов .rc

Файл описания ресурсов перед компиляцией так же обрабатывается препроцессором, поэтому в нем можно использовать директивы препроцессора (#include, #define, ...) и макроопределения. В сложных "блочных" описаниях ресурсов вместо ключевых слов BEGIN и END можно использовать символы { и }, соответственно.

Пример 1

```
strings.rc
STRINGTABLE
BEGIN
      IDS_STRING1 “String 1”
      IDS_STRING2 “String 2”
END
Icon1 ICON icon1.ico
Text1 TEXT text1.txt
Cursor1 CURSOR cursor1.cur
Bitmap1 BITMAP bitmap1.bmp
```

Пример 2

```
FltBmp BITMAP {
 '42 4D A2 00 00 00 00 00 00 00 3E 00 00 00 28 00‘
 '00 00 19 00 00 00 19 00 00 00 01 00 01 00 00 00‘
 '00 00 64 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 FF FF FF 00 FF FF‘
 'FF 80 FF FF FF 80 FF FF FF 80 FF FF FF 80 FF FF‘
 'FF 80 FF FF FF 80 FF FF FF 80 C0 FF 81 80 FE FF'
'BF 80 FE FF BF 80 FE FF BF 80 FE FF BF 80 FE FF‘
 'BF 80 FE FF BF 80 FE FF BF 80 FE FF BF 80 FE FF
 'BF 80 FE FF BF 80 FE 00 3F 80 FF FF FF 80 FF FF'
'FF 80 FF FF FF 80 FF FF FF 80 FF FF FF 80 FF FF'
'FF 80'
}
```

Файл ресурсов. Меню.

Шаблон меню в ресурсах:

```
<id> MENU [load] [memory]
BEGIN
        MENUITEM <название пункта>, <код пункта> [,<параметры>]
        MENUITEM SEPARATOR
        POPUP <название>
        BEGIN
                MENUITEM …
        END
END
```

<название пункта> может содержать &, \t

<параметры> ::= CHECKED | GRAYED | INACTIVE|….

- CHECKED - рядом с пунктом меню отображается галочка,
- GRAYED - пункт меню неактивен (не может быть выбран) и отображается серым цветом и др.

WM_COMMAND, параметр wParam

wParam = <код пункта>

Работа с файлом ресурсов

Доступ к ресурсам, скомпонованным с исполняемым файлом, можно получить при помощи следующих функций:

*HICON WINAPI **LoadIcon**(HINSTANCE hInst, LPCSTR lpIconName)*

*HBITMAP WINAPI **LoadBitmap**(HINSTANCE hInst, LPCSTR lpBitmapName)*

*HCURSOR WINAPI **LoadCursor**(HINSTANCE hInst, LPCSTR lpCursorName)*

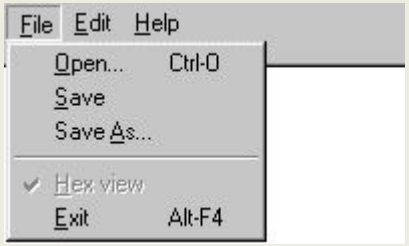
*HMENU WINAPI **LoadMenu**(HINSTANCE hInst, LPCSTR lpMenuName)*

Первый параметр этих функций - дескриптор экземпляра программы,
второй - идентификатор соответствующего ресурса.
Если ресурс идентифицируется не именем, а числом, то следует использовать макрос, объявленный в windows.h:

*#define **MAKEINTRESOURCE**(i) (LPSTR) ((DWORD) ((WORD) (i)))*
Например: HMENU hMainMenu=LoadMenu(hInst,MAKEINTRESOURCE(10));

Пример

```
Ex4_Icon ICON "myicon.ico"
 Ex4_Menu MENU
  {
  POPUP "&File"
   {
     MENUITEM "&Open...\tCtrl-O", 2
     MENUITEM "&Save", 3
     MENUITEM "Save &As...", 4
     MENUITEM SEPARATOR
     MENUITEM "&Hex view", 5,CHECKED GRAYED
     MENUITEM "&Exit\tAlt-F4", 6
   }
  POPUP "&Edit"
  { MENUITEM "&Copy", 7
    MENUITEM "&Paste", 8
    POPUP "Popup"
     {
      MENUITEM "1", 9
      MENUITEM "2", 10
      MENUITEM "3", 11 }
      MENUITEM SEPARATOR MENUITEM "Search", 12
      }
 POPUP "&Help" { MENUITEM "&About...\tF1", 13 } }
```



Подключение файла меню

В примере 1 *(слайд 15*) надо изменить строки
*wc.hIcon=NULL; wc.lpszMenuName=NULL;*на
wc.hIcon=LoadIcon(hInst,"Ex4_Icon");
wc.lpszMenuName="Ex4_Menu";

Кроме того, изменим обработчик событий WM_COMMAND так, чтобы при выборе того или иного пункта меню выводилось окно-сообщение с кодом выбранной команды:

```
case WM_COMMAND:
if (HIWORD(wp)==0) {
  char buf[256];
  switch (LOWORD(wp))
  { case 6: /* команда меню Exit */ PostQuitMessage(0);
   default: /* все остальные команды */
   wsprintf(buf,"Command code: %d",LOWORD(wp));
   MessageBox(hw,buf,"MessageBox",MB_OK|MB_ICONINFORMATION);
   }
  } return 0;
```

Использование акселераторов

Акселератор – комбинация горячих клавиш.

<id> ACCELERATORS […]
BEGIN
event, idvalue, [type] [options]
…
END

event ::= “char” | ASCII | virtkey
char
type ::= ASCII | VIRTKEY
options ::= NOINVERT | ALT |
SHIFT | CONTROL

wParam = idvalue

Сообщение: WM_COMMAND,
wParam

Пример

```
1  ACCELERATORS
{
  "^C",  IDDCLEAR        ; control C
  "K",   IDDCLEAR        ; shift K
  "k",   IDDELLIPSE, ALT  ; alt k
  98,    IDDRECT, ASCII   ; b
  66,    IDDSTAR, ASCII   ; B (shift b)
  "g",   IDDRECT         ; g
  "G",   IDDSTAR         ; G (shift G)
  VK_F1, IDDCLEAR, VIRTKEY              ; F1
  VK_F1, IDDSTAR, CONTROL, VIRTKEY      ; control F1
  VK_F1, IDDELLIPSE, SHIFT, VIRTKEY     ; shift F1
  VK_F1, IDDRECT, ALT, VIRTKEY          ; alt F1
  VK_F2, IDDCLEAR, ALT, SHIFT, VIRTKEY  ; alt shift F2
  VK_F2, IDDSTAR, CONTROL, SHIFT, VIRTKEY ; ctrl shift F2
  VK_F2, IDDRECT, ALT, CONTROL, VIRTKEY   ; alt control F2
}
```

Диалоги в Windows

- ✔ Чтобы создать диалоговое окно, приложение должно предоставить системе шаблон диалога, описывающий содержание и стиль диалога, и диалоговую процедуру. Диалоговая процедура выполняет примерно такие же задачи, что и процедура обработки событий окна. Диалоговые окна принадлежат к предопределенному классу окон.

- ✔ Windows использует этот класс и соответствующую процедуру обработки
  событий для модальных и немодальных диалогов. Эта процедура обрабатывает одни сообщения самостоятельно, а другие передает на обработку диалоговой процедуре приложения. У приложения нет непосредственного доступа к этому предопределенному классу и соответствующей ему процедуре обработки событий.

- ✔ Для изменения стиля и поведения диалога программа должна использовать шаблон диалогового окна и диалоговую процедуру.

- ✔ Для создания модального диалога используется функция **DialogBox**, а для создания немодального диалога - **CreateDialog**:

Функции создания диалогов

```
int WINAPI DialogBox(HANDLE hInst, LPCSTR template, HWND parent, DLGPROC DlgFunc)

HWND WINAPI CreateDialog(HANDLE hInst, LPCSTR template, HWND parent, DLGPROC
DlgFunc)
```

Параметры:

hInst - дескриптор экземпляра программы (модуля, в котором находится шаблон);
template - имя ресурса, описывающего диалог; parent - дескриптор родительского окна;
DlgFunc – диалоговая функция следующего формата:

BOOL CALLBACK *DlgFunc*(HWND hw, UINT msg, WPARAM wp, LPARAM lp)

Параметры диалоговой функции такие же, как у обычной функции обработки событий. Отличие этой функции - она вызывается из предопределенной функции обработки событий для диалоговых окон. Она должна вернуть значение TRUE, если обработала переданное ей сообщение, или FALSE в противном случае. Она ни в коем случае не должна сама вызывать **DefWindowProc**.
При создании диалогового окна диалоговая процедура получает сообщение WM_INITDIALOG. Если в ответ на это сообщение процедура возвращает FALSE, диалог не будет создан:
функция **DialogBox** вернет значение -1, а **CreateDialog** - NULL.

Диалоги-2

Модальное диалоговое окно блокирует указанное в качестве родительского окно и появляется поверх него (вне зависимости от стиля WS_VISIBLE).
Приложение закрывает модальное диалоговое окно при помощи функции

```
BOOL WINAPI EndDialog(HWND hw, int result)
```

Приложение должно вызвать эту функцию из диалоговой процедуры в ответ на сообщение от кнопок "OK", "Cancel" или команды "Close" из системного меню диалога.
Параметр result передается программе как результат возврата из функции **DialogBox**. Немодальное диалоговое окно появляется поверх указанного в качестве родительского окна, но не блокирует его. Диалоговое окно остается поверх родительского окна, даже если оно неактивно. Программа сама отвечает
за отображение/сокрытие окна (с помощью стиля WS_VISIBLE и функции **ShowWindow**). Сообщения для немодального диалогового окна оказываются в основной очереди сообщений программы. Чтобы эти сообщения были корректно
обработаны, следует включить в цикл обработки сообщений вызов функции:

```
BOOL WINAPI IsDialogMessage(HWND hwDlg, MSG *lpMsg)
```

Если эта функция вернула TRUE, то сообщение обработано и его не следует передавать функциям **TranslateMessage** и **DispatchMessage**.
Немодальное диалоговое окно уничтожается, если уничтожается его родительское
окно. Во всех остальных случаях программа должна сама заботиться об

Диалоги -3

Шаблон диалогового окна в файле ресурсов задается следующим образом:

<name> DIALOG x, y, w, h [options]
BEGIN
controls
END

options ::= CAPTION “text” | CLASS class | EXSTYLE = style | FONT ps, tf |
MENU name | STYLE styles

control [“текст”] id, x, y, w, h [, style] ...

Пример

```
Ex4_Dlg  DIALOG 50,50,90,40
 STYLE WS_POPUP|WS_CAPTION|DS_MODALFRAME
CAPTION "MyDlg"
FONT 10, "Arial"
{
 CONTROL "", 1, "STATIC", SS_LEFT, 5, 5, 80, 10
 CONTROL "OK", 2, "BUTTON", BS_DEFPUSHBUTTON, 5, 20, 80, 12
 }
```

Управляющие элементы

- Управляющие элементы, как и другие окна, принадлежат тому или иному классу окон.
- Windows предоставляет несколько предопределенных классов управляющих элементов.
- Программа может создавать управляющие элементы поштучно при помощи функции **CreateWindow** или все сразу, загружая их вместе с шаблоном диалога из своих ресурсов.
- Управляющие элементы - это всегда дочерние окна.
- Управляющие элементы при возникновении некоторых событий, связанных с реакцией пользователя, посылают своему родительскому окну *сообщения-оповещения* (*notification messages*) WM_COMMAND или WM_NOTIFY.

Как и любое другое окно, управляющий элемент может быть скрыт или отображен при помощи функции **ShowWindow**. Аналогично, управляющий элемент может быть блокирован или разблокирован при помощи функции:

```
BOOL WINAPI EnableWindow(HWND hw,BOOL bEnable)
```

В качестве второго параметра передается флаг TRUE (разблокировать) или FALSE (блокировать). Функция возвращает значение TRUE, если перед ее вызовом окно было заблокировано. Узнать текущий статус блокирования окна можно при помощи функции:

```
BOOL WINAPI IsWindowEnabled(HWND hw),
```

которая возвращает значение TRUE, если окно разблокировано.

Управляющие-2

Для многих управляющих элементов определены специальные сообщения, которые управляют видом или поведением таких элементов или позволяют получить параметры их состояния. Как и для любого другого окна эти сообщения
можно отправить с помощью функции:

```
LRESULT WINAPI SendMessage(HWND hw, UINT msg, WPARAM wp, LPARAM lp)
```

Все упомянутые функции работают с дескриптором окна, который для управляющих элементов в случае создания диалога по шаблону из ресурсов непосредственно неизвестен, но может быть получен по дескриптору диалога и идентификатору управляющего элемента вызовом:

```
HWND WINAPI GetDlgItem(HWND hDlg, int idDlgItem)
```

Для функции отсылки сообщений есть специальный вариант, предназначенный для более удобной работы с управляющими элементами:

LRESULT WINAPI **SendDlgItemMessage**(HWND hwndDlg, /* дескриптор родительского диалога */ int idControl,
/* идентификатор управляющего элемента */ UINT msg,
/* код сообщения */ WPARAM wp, /* параметр сообщения */ LPARAM lp /* параметр сообщения */)

Управляющие-3

Для управляющих элементов внутри диалогов специальный смысл имеют стили WS_TABSTOP и WS_GROUP.

- Если в диалоге имеются управляющие элементы со стилем WS_TABSTOP, то при нажатии пользователем на клавишу [Tab] (или [Shift]+[Tab]), текущий активный элемент диалога будет терять фокус и передавать его следующему за ним (или предыдущему) ближайшему элементу со стилем WS_TABSTOP.
- С помощью стиля WS_GROUP элементы диалога можно объединять в группы. Группа элементов начинается с элемента со стилем WS_GROUP и заканчивается элементом, после которого идет элемент со стилем WS_GROUP, или последним элементом в диалоге.
- Внутри группы только первый элемент должен иметь стиль WS_GROUP.

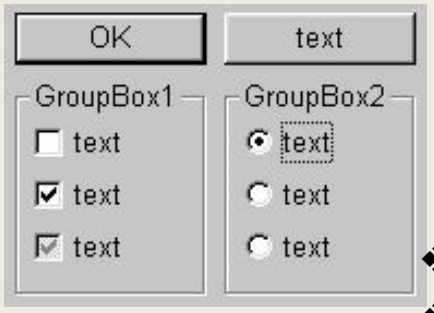
- Windows допускает перемещение внутри группы при помощи клавиш-стрелок.

Static

CONTROL "",-1, "STATIC", SS_BLACKFRAME, 5, 40, 20, 10
CONTROL "",-1, "STATIC", SS_GRAYFRAME, 30, 40, 20, 10
CONTROL "",-1, "STATIC", SS_WHITEFRAME, 55, 40, 20, 10
CONTROL "",-1, "STATIC", SS_BLACKRECT, 80, 40, 20, 10
CONTROL "",-1, "STATIC", SS_GRAYRECT, 105, 40, 20, 10
CONTROL "",-1, "STATIC", SS_WHITERECT, 130, 40, 20, 10
CONTROL "",-1, "STATIC", SS_ETCHEDFRAME,155, 40, 20, 10
/* Для статиков-иконок или картинок текстовое поле определяет имя ресурса */
CONTROL "Ex4_Bmp",-1, "STATIC", SS_BITMAP, 5, 55, -1, -1
CONTROL "Ex4_Icon",-1, "STATIC", SS_ICON, 65, 55, -1, -1
CONTROL "text",-1, "STATIC", SS_LEFT, 105, 55, 20, 10
CONTROL "text",-1, "STATIC", SS_CENTER, 130, 55, 20, 10
CONTROL "text",-1, "STATIC", SS_RIGHT, 155, 55, 20, 10
/* По умолчанию SS_LEFT, SS_RIGHT, SS_CENTER делают перенос по словам */
CONTROL "This is long text example",-1, "STATIC", SS_SIMPLE|SS_SUNKEN, 65, 70, 55, 15
CONTROL "This is long text example",-1, "STATIC", SS_LEFT|SS_SUNKEN, 125, 70, 50, 15
Для текстовых статиков со стилями SS_LEFT, SS_RIGHT или SS_CENTER существуют более простые операторы объявления ресурсов:
LTEXT "text",-1, 105, 55, 20, 10
CTEXT "text",-1, 130, 55, 20, 10
RTEXT "text",-1, 155, 55, 20, 10
LTEXT "This is long text example",-1, 65, 70, 55, 15, SS_LEFTNOWORDWRAP|SS_SUNKEN
LTEXT "This is long text example",-1,125, 70, 50, 15, SS_LEFT|SS_SUNKEN

```
BOOL WINAPI SetWindowText(HWND hw, LPCSTR lpsz)
BOOL WINAPI SetDlgItemText(HWND hDlg, int idControl, LPCTSTR
```

Button



Кнопка - это небольшое прямоугольное дочернее окно, обычно имеющее два состояния: нажато/отпущено или включено/выключено. Пользователь меняет состояние этого элемента щелчком мыши. К этому классу относятся:

- кнопки-"давилки" (*push buttons*),
- кнопки-"галочки" (*check boxes*),
- "радио"-кнопки (*radio buttons*)
- специальный тип групповых рамочек (*group boxes*).

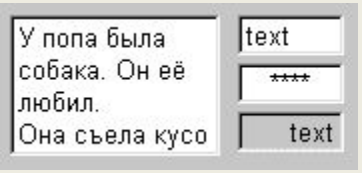
```
/* DEFPUSHBUTTON - кнопка по умолчанию (нажимается по [Enter]) */
 CONTROL "OK", 2, "BUTTON", BS_DEFPUSHBUTTON, 5, 20, 50, 12
 CONTROL "text", 3, "BUTTON", BS_PUSHBUTTON, 60, 20, 50, 12
 CONTROL "GroupBox1", -1, "BUTTON", BS_GROUPBOX, 5, 35, 50, 50
 CONTROL "text", 4, "BUTTON", BS_CHECKBOX, 10, 45, 30, 10
 CONTROL "text", 5, "BUTTON", BS_AUTOCHECKBOX, 10, 57, 30, 10
 CONTROL "text", 6, "BUTTON", BS_AUTO3STATE, 10, 69, 30, 10
 CONTROL "GroupBox2", -1, "BUTTON", BS_GROUPBOX, 60, 35, 50, 50
 CONTROL "text", 7, "BUTTON", BS_AUTORADIOBUTTON|WS_GROUP, 65, 45, 30, 10
 CONTROL "text", 8, "BUTTON", BS_AUTORADIOBUTTON, 65, 57, 30, 10
 CONTROL "text", 9, "BUTTON", BS_AUTORADIOBUTTON, 65, 69, 30, 10
```

Автоматические радио-кнопки должны быть объединены в группу при помощи стиля WS_GROUP, чтобы Windows корректно их обрабатывала.

Проверить состояние кнопки можно, послав ей сообщение BM_GETCHECK (wp=0; lp=0) или вызовом функции: `UINT WINAPI` **`IsDlgButtonChecked`**`(HWND hDlg, int idButton)`

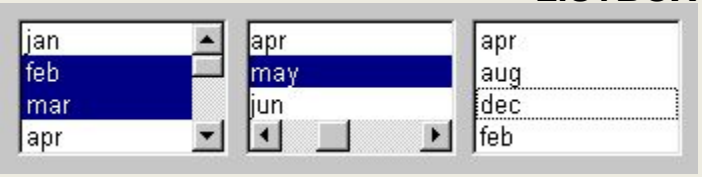
Остальные элементы управления

EDIT



CONTROL "" 4, "EDIT", ES_MULTILINE|ES_WANTRETURN|WS_BORDER 5, 45, 60, 35
CONTROL "text", 5, "EDIT", ES_LEFT|WS_BORDER, 70, 45, 30, 10
CONTROL "text", 6, "EDIT", ES_CENTER|ES_PASSWORD|WS_BORDER, 70, 57, 30, 10
CONTROL "text", 7, "EDIT", ES_RIGHT|ES_READONLY|WS_BORDER, 70, 69, 30, 10

LISTBOX



LISTBOX 3, 5, 45, 60, 35, LBS_MULTIPLESEL|WS_BORDER|WS_VSCROLL
LISTBOX 4, 70, 45, 60, 35, LBS_MULTICOLUMN|WS_BORDER|WS_HSCROLL
LISTBOX 5, 135, 45, 60, 35, LBS_SORT|LBS_NOSEL|WS_BORDER

COMBOBOX

COMBOBOX 3, 5, 45, 60, 70, CBS_DROPDOWN|CBS_AUTOHSCROLL|WS_VSCROLL
COMBOBOX 4, 70, 45, 60, 70, CBS_DROPDOWNLIST|WS_VSCROLL
COMBOBOX 5, 135, 45, 60, 70, CBS_SIMPLE|CBS_SORT

ListView, Up-Down, Progress Bars, Tooltips, TrackBars, RichEdit, ……

Стандартные диалоги

- BOOL WINAPI **ChooseColor**(CHOOSECOLOR* lpcc) - создает диалог, отображающий палитру цветов и позволяющий пользователю выбрать тот или иной цвет или создать свой.
- BOOL WINAPI **ChooseFont**(CHOOSEFONT* lpcf) - создает диалог, отображающий имена установленных в системе шрифтов, их кегль, стиль начертания и т.п.

- BOOL WINAPI **GetOpenFileName**(OPENFILENAME* lpofn) и
- BOOL WINAPI **GetSaveFileName**(OPENFILENAME* lpofn) - создают диалог, отображающий содержимое того или иного каталога, и позвояющий пользователю выбрать уникальное имя файла для открытия или сохранения.

- BOOL WINAPI **PrintDlg**(PRINTDLG* lppd) - создает диалог, позволяющий пользователю установить различные опции печати, например, диапазон страниц, количество копий и др.
- BOOL WINAPI **PageStupDlg**(PAGESETUPDLG* lppsd) - создает диалог, позволяющий пользователю выбрать различные параметры страницы: ориентацию, поля, размер бумаги и т.п.
- HWND WINAPI **FindText**(FINDREPLACE* lpfr) - создает диалог, позволяющий пользователю ввести строку для поиска и такие опции, как направление поиска.
- HWND WINAPI **ReplaceText**(FINDREPLACE* lpfr) - создает диалог, позволяющий пользователю ввести строку для поиска, строку для замены и опции замены (направление поиска, область поиска).