Р.М.Ганеев

ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ СРЕДСТВАМИ WIN32 API

Книга посвящена методическим основам проектирования пользовательского интерфейса средствами Win32 API. Основное внимание уделено динамическому проектированию и управлению базовыми элементами информационных систем - окнами, органами управления, меню и диалоговыми панелями.

Книга написана доступным языком, насыщена примерами программной реализации, все разделы сопровождаются вопросами контроля полученных знаний и вариантами упражнений. Она поможет читателю овладеть методикой проектирования эффективных приложений для Windows.

Для тех, кто знает основы языка Си и хочет проектировать компактные быстродействующие приложения.

Оглавление

Предисловие

В объектно-ориентированной операционной системе Windows интерфейс пользователя представляет собой целостную систему различных элементов. Элементами служат окна, органы управления, меню, диалоговые панели и другие объекты Windows. Каждый элемент задают множествами параметров состояния, входных и выходных сообщений. Для элементов одного класса описывают единый метод изменения параметров состояния и обработки входных и выходных сообщений. Основной задачей проектирования интерфейса пользователя является разработка целостной системы управления множеством состояний программного продукта.

Книга рассматривает проектирование приложений методами структурного программирования, что не исключает возможностей использования классов языка C++ и не противоречит им.

Книга состоит из введения и шести глав. В каждой главе подробно обсуждаются обязательные разделы изучаемого материала и кратко рассматриваются вспомогательные функции, относящиеся к этому разделу. В конце каждого раздела перечисляются вопросы контроля полученных знаний, а также приводятся варианты упражнений. Для первых четырех глав предусмотрены справочные материалы, которые оформлены в таблицах приложений в конце книги. В ссылках на таблицы N.M в книге N указывает на номер приложения, а M – на номер таблицы в приложении N.

Глава 1 посвящена созданию главного объекта Windows – окон. Рассматривается понятие **окно** с точки зрения языка программирования. Читатель знакомится со способами описания классов и функций окон, главной функции приложения. Отдельно рассмотрена функция для создания окон этих классов. Показан механизм получения и обработки сообщений. Рассматриваются базовые функции перемещения и изменения расположения окон, а также средства обмена сообщениями между приложением и пользователем.

Глава 2 рассматривает определенные в системе Windows классы окон. Это органы управления – статический текст, кнопки, списки, редактор текста, полосы прокрутки и комбинированные списки. Здесь читатель знакомится со способами создания и управления состоянием органов управления. В этой главе продолжается изучение функций обмена и обработки сообщений между окнами.

Глава 3 поясняет, как выводить тексты и графические примитивы в окнах. Здесь читатель знакомится с контекстом отображения и его ос-

новными атрибутами, основными понятиями метрики Windows и базовыми функциями отображения текста и графических примитивов. Примеры этой главы посвящены выводу таблиц, графиков, диаграмм и различным эффектам вывода текста.

Глава 4 рассматривает главное меню окна и его разделы, временные и плавающее меню. Подробно рассматриваются функции динамического создания, изменения меню и управления состоянием элементов меню. Здесь же изучают способы создания акселераторов и работы с ними.

Глава 5 посвящена панелям инструментов и строкам состояния. Здесь изучают базовые способы создания и использования панели инструментов и строки состояния. Также здесь показан механизм обработки уведомительных сообщений. Рассматриваемые в этой главе примеры и упражнения связаны с примерами и упражнениями из предыдущих глав.

Глава 6 рассматривает способы динамического создания диалоговых панелей. Здесь показаны модальные и немодальные панели, а также блокноты диалоговых панелей. Особое внимание уделено примерам обмена данными и обработке сообщений в диалоговых панелях и блокнотах панелей. Показаны способы создания стандартных диалоговых панелей операционной системы.

Глава 1
Создание окон

1.1. Определения

Программы для Windows называют **приложениями**. **Пользователь** с приложением взаимодействует через окна следующих видов:

1. **Окно приложения.** Оно организует работу, появляется первым при запуске и закрывается вместе с прекращением работы приложения.

2. **MDI-окна.** Они служат для одновременной обработки нескольких документов.

3. **Окно помощи.** Оно в большей степени работает под управлением операционной системы, чем под управлением приложения.

4. **Диалоговое окно.** Оно обеспечивает оперативный обмен данными между пользователем и приложением (рис. 1.1).

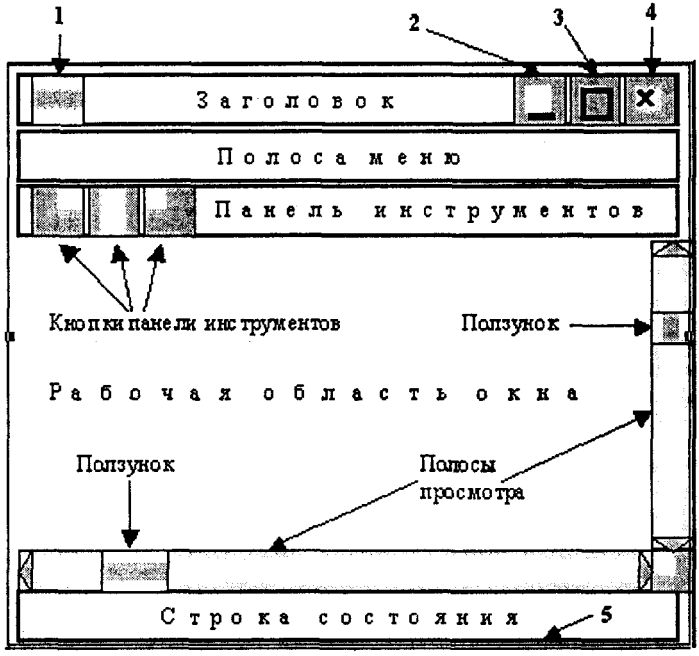

Рис. 1.1. Окно со множеством элементов:
1 – кнопка системного меню; 2 – кнопка сворачивания окна в пиктограмму;
3 – кнопка максимилизации или восстановления размеров окна;
4 – кнопка закрытия окна; 5 – рамки изменения размеров окна

Для **разработчика** окно является совокупностью большого количества элементов, функционирующих под управлением приложения и операционной системы. Пример такого окна изображен на рис. 1.1.

С точки зрения **языка программирования**, окна – это **объекты** (переменные), над которыми выполняют действия. Объект принадлежит к определенному **классу** (типу), который описывает множество данных (**параметров состояния окна**) и метод (**функцию**) изменения этих данных.

Главное окно на рис. 1.1 имеет обрамляющие двойные рамки, заголовок, различные кнопки, полосу меню, панель инструментов с кнопками, полосы просмотра, строку состояния и др. Эти элементы также являются окнами, имеют свои данные и метод их изменения, т. е. принадлежат к **классам окон**.

1.2. Класс окон

Для создания окна операционной системе указывают, к какому классу оно принадлежит. Если к моменту создания окна операционной системе известен класс создаваемого окна (например, это определенный в системе или зарегистрированный текущим или другим приложением класс), то можно воспользоваться именем этого класса. Иначе нужно создать новый класс (описать **функцию окна** и набор используемых ресурсов) и **зарегистрировать его**.

1.2.1. Описание используемых классом окон р есурсов

Набор используемых ресурсов класса задают в структуре типа **WNDCLASS**. Эта структура описана следующим образом:

```
typedef struct
{   UINT        style;
    WNDPROC     lpfnWndProc;
    int         cbClsExtra;
    int         cbWndExtra;
    HANDLE      hInstance;
    HICON       hIcon;
    HCURSOR     hCursor;
    HBRUSH      hbrBackground;
    LPCTSTR     lpszMenuName;
    LPCTSTR     lpszClassName;
}   WNDCLASS;
```

Например, если описана структура этого типа:

```
WNDCLASS   wc;
```

то присваивают определенные значения ее полям.

Назначение полей этой структуры:

1. **style** принимает значения из табл. 1.1. Например:

```
wc.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
```

2. **lpfnWndProc** указывает на функцию окна. Она будет рассмотрена ниже.
3. **cbClsExtra** равно количеству дополнительных байт класса.
4. **cbWndExtra** равно количеству дополнительных байт окна этого класса.
5. **hInstance** указывает на дескриптор текущего приложения.
6. **hIcon** указывает на имя пиктограммы (иконки), в которую превращается окно при минимизации. Ресурс иконки загружают функцией LoadIcon:

```
HICON LoadIcon( HINSTANCE hInst, LPCTSTR lpIconName);
```

Параметр lpIconName указывает на строку с именем загружаемой иконки, а hInst – дескриптор приложения, чей исполняемый код содержит загружаемый ресурс. Для загрузки стандартной иконки hInst приравнивают NULL, а в качестве lpIconName задают одно из значений табл. 1.2. Например, следующий оператор устанавливает стандартную пиктограмму окна приложения:

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

В случае аварийного завершения функция возвращает NULL и при сворачивании окон этого класса операционная система в качестве иконки использует логотип Windows.

7. **hCursor** задает вид курсора мыши при его прохождении над окном. Ресурс курсора загружают функцией LoadCursor:

```
HCURSOR LoadCursor( HINSTANCE hInst, LPCTSTR lpCursorName );
```

Параметр lpCursorName указывает на строку с именем загружаемого ресурса, а hInst – дескриптор приложения, чей исполняемый код содержит загружаемый ресурс. Для загрузки стандартного курсора hInst приравнивают NULL, а в качестве lpCursorName задают одно из значений табл. 1.3. Например, следующий оператор задает курсор в виде стандартной стрелки:

```
wc.hCursor = LoadCursor(NULL, IDC_ARROW);
```

В случае аварийного завершения функция возвращает NULL и окно будет заимствовать курсор предшествующего по движению мыши окна.

Функции LoadIcon и LoadCursor загружают ресурс иконки или курсора, только если ресурс не был загружен; иначе выбирают дескриптор загруженного ресурса.

8. **hbrBackground** задает дескриптор кисти закрашивания фона окна. В качестве кисти можно использовать "чистые" цвета или пиктограмму. Чаще используют значение системного цвета (табл. 1.4.) плюс 1. Цвета преобразуют в тип HBRUSH. Например, следующий оператор устанавливает системный цвет закрашивания фона:

```
wc.hbrBackground = (HBRUSH)( COLOR_WINDOW+1 );
```

Система сама удаляет кисть фона при освобождении класса. Если указать NULL, то приложение само должно красить фон окон. Для определения необходимости закрашивания приложение обрабатывает сообщение WM_ERASEBKGND или проверяет поле fErase структуры PAINTSTRUCT, заполненной функцией BeginPaint.

9. **lpszMenuName** указывает на имя ресурса главного меню окон этого класса. Если задать NULL, окна этого класса не имеют заданного по умолчанию меню. Например:

```
wc.lpszMenuName = (LPCTSTR)NULL;
```

10. **lpszClassName** указывает на текстовую строку, содержащую имя регистрируемого класса окон, например:

```
wc.lpszClassName = szName;
```

1.2.2. Пример регистрации класса окон

Для регистрации класса окон удобно использовать функцию следующего вида:

```
int RegClass( WNDPROC Proc, LPCTSTR szName )
{   WNDCLASS wc;
    wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.cbClsExtra = wc.cbWndExtra = 0;
    wc.lpfnWndProc = Proc; wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wc.lpszMenuName = (LPCTSTR)NULL;
    wc.lpszClassName = szName;
    return RegisterClass(&wc);
}
```

Формальными параметрами этой функции являются указатели на функцию окна и строку с именем класса. Имя, список формальных параметров и тип возврата функции могут быть любыми другими. Главное, чтобы приложению после регистрации были доступны имя зарегистрированного класса, а операционной системе – функция окна этого класса. Поясним некоторые операторы этой функции.

Не используются дополнительные данные:

```
wc.cbClsExtra = wc.cbWndExtra = 0;
```

Поле hInstance содержит дескриптор текущего приложения:

```
wc.hInstance = hInstance;
```

Функция RegisterClass регистрирует класс в операционной системе. Ее аргументом является адрес подготовленной структуры wc. При успешной регистрации класса она возвращает ненулевое значение, иначе – нуль. Оператор возврата функции RegClass:

```
return RegisterClass(&wc);
```

в случае успешной регистрации класса возвращает ненулевое значение, иначе – нуль.

1.2.3. Функция окна

Функция окна описывает реакцию окна на поступающие **сообщения**. Она от обычных функций отличается следующим:

- имеет стандартные тип возврата и список формальных параметров;
- вызывается только операционной системой при поступлении сообщения окну;
- сообщения, которые не обрабатываются функцией окна, возвращаются операционной системе.

Есть еще одно отличие. В объектно-ориентированном программировании методы изменения параметров состояния объекта (функции-члены) обычно описывают отдельно. Функция окна реализует единственный метод для изменения всех параметров состояния окна.

Имя функции окна – это обычное имя, определяемое разработчиком. При регистрации класса операционная система запоминает указатель на эту функцию.

Рассмотрим пример описания функции окна.

```
LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                          WPARAM wParam, LPARAM lParam )
{   static short cx, cy, left, top;   //Описание локальных переменных
    switch (msg)                      //Обработка сообщения
    {   case WM_CREATE:  {  ...;    return 0; }
        case WM_MOVE:
    {       left=LOWORD(lParam);  top=HIWORD(lParam);
            return 0;
    }
        case WM_SIZE:
    {       cx= LOWORD(lParam);  cy=HIWORD(lParam);
```

```
        return 0;
    }
        case WM_COMMAND:   //Обрабатываем команды
    {       switch ( LOWORD(wParam) )
            {   case CM_FILE_EXIT:
                {   DestroyWindow(hwnd); return 0; }
                //Далее могут быть ветви других команд
            }
            return 0;
        }
        case WM_DESTROY:
        {   ...; PostQuitMessage(0);  return 0; }
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Заголовок функции окна определен соглашениями Windows и имеет вид:

```
LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                          WPARAM wParam, LPARAM lParam )
```

Тип возврата LRESULT равноценен типу signed long. Модификатор CALLBACK указывает на соглашения о том, что эта функция вызывается операционной системой (такие функции называют **функциями обратного вызова**).

Имена типов UINT, WPARAM, LPARAM описаны следующим образом:

```
typedef   unsigned int   UINT;
typedef   UINT           WPARAM;
typedef   LONG           LPARAM;
```

Параметр hwnd – дескриптор окна-адресата, а msg, wParam и lParam описывают полученное сообщение. Например, параметр msg принимает код сообщения.

Рассмотрим, каким сообщениям соответствуют используемые в примере имена констант. В Windows описаны несколько сот **кодов сообщений** с префиксом **WM_**.

Код **WM_CREATE** поступает от функции CreateWindow (о ней речь пойдет ниже) перед созданием окна. Если после обработки этого сообщения функции CreateWindow возвращается значение -1, то окно не создается.

Код **WM_SIZE** функция окна получает после изменения размеров окна, а **WM_MOVE** – после перемещения левого верхнего угла рабочей области окна. Если при изменении размеров окна изменились координаты левого верхнего угла рабочей области, то функция окна сначала получает код WM_MOVE, а затем – WM_SIZE. После изменения режима

отображения окна функция окна получает код WM_SIZE, а затем – WM_MOVE.

Код **WM_COMMAND** функция окна получает при поступлении команды. Тогда младшее слово параметра wParam содержит код команды. Разработчик свои сообщения часто связывает с командами меню и описывает идентификаторы для этих команд. Например, для идентификатора CM_FILE_EXIT команды меню текст приложения должен содержать макроопределение вида

```
#define    CM_FILE_EXIT    3001
```

Код **WM_DESTROY** функции окна посылают перед разрушением окна. В примере функция окна, вызывая функцию DestroyWindow, сама себе отправляет сообщение WM_DESTROY.

Рассмотрим **сообщения от мыши**. Пусть курсор мыши находится над рабочей областью окна. После нажатия левой клавиши мыши функция окна получит код **WM_LBUTTONDOWN** а правой клавиши – **WM_RBUTTON DOWN**. После отжатия левой (правой) клавиши функция окна получит код **WM_LBUTTONUP (WM_RBUTTONUP)**. После перемещении курсора мыши функция окна получает код **WM_MOUSEMOVE**.

Можно получать сообщения от мыши, даже если курсор мыши не находится над рабочей областью окна. Для передачи мыши в монопольное использование окну hwnd вызывают функцию:

```
HWND SetCapture(HWND hwnd);
```

Эта функция возвращает дескриптор окна, у которого в монопольном пользовании была мышь, или NULL.

Вызов функции

```
void  ReleaseCapture(void);
```

отменяет режим монопольного использования.

Значение параметра **lParam** зависит от кода сообщения.

При поступлении кода WM_MOVE параметр lParam содержит экранные координаты левого верхнего угла рабочей области. Например:

```
left=LOWORD(lParam);        //Координата левого края рабочей области
top=HIWORD(lParam);         //Координата верхнего края рабочей области
```

Для кода WM_SIZE параметр lParam несет информацию о ширине cx и высоте cy рабочей области:

```
cx=LOWORD(lParam);          //Ширина рабочей области
cy=HIWORD(lParam);          //Высота рабочей области
```

При поступлении любых сообщений от мыши расстояние курсора мыши от левого края рабочей области равно LOWORD(lParam), а от верхнего края – HIWORD(lParam).

Обратите внимание на описание локальных переменных. Если хотят, чтобы какие-либо переменные сохраняли свои значения до следующего вызова функции окна, то их описывают с приставкой static. Например, если функция окна многократно использует ширину cx и высоту cy рабочей области окна, то их описывают как static.

Вызов функции DefWindowProc означает, что необработанное сообщение возвращают операционной системе. Например, в данном примере не обрабатывают сообщения от мыши (как и многие другие получаемые сообщения). Эти сообщения, тем не менее, передаются функции окна, но она их возвращает операционной системе.

В тексте примера отсутствуют какие-либо действия. При необходимости можно включить некоторые действия, например в местах, обозначенных многоточиями.

Внимание! Если функция окна обрабатывает код сообщения, то настоятельно **рекомендуется** вернуть определенное системой для этого кода значение.

1.3. Создание окон

Для создания окна вызывают функцию CreateWindow. Она создает **перекрывающееся, временное или дочернее окно** и устанавливает начальные значения некоторых его параметров. Эта функция объявлена следующим образом:

```
HWND CreateWindow(
    LPCTSTR     lpClassName,    //Указатель на имя класса
    LPCTSTR     lpWindowName,   //Указатель на имя окна
    DWORD       dwStyle,        //Стиль окна
    int         x,              //Координата левого края окна
    int         y,              //Координата верхнего края окна
    int         nWidth,         //Ширина окна
    int         nHeight,        //Высота окна
    HWND        hWndParent,     //Дескриптор окна-родителя или
                                //окна-владельца
    HMENU       hMenu,          //Дескриптор меню или идентифи-
                                //катор создаваемого дочернего окна
    HANDLE      hInstance,      //Дескриптор приложения
    LPVOID      lpParam         //Указатель на данные окна
);
```

Назначение аргументов вызова этой функции:

1. **lpClassName** указывает на имя зарегистрированного функцией RegisterClass или определенного операционной системой класса.

2. **dwStyle** задает **стиль окна**. Стиль окон будет отдельно рассмотрен ниже.
3. **x** – координата левого края окна в пикселях. Для перекрывающихся или временных окон x отсчитывают от левого края экрана, дочерних окон – от левого края рабочей области родительского окна. Если для перекрывающегося окна со стилем WS_VISIBLE в качестве **x** задать константу CW_USEDEFAULT, Windows устанавливает не обязательно нулевую **позицию по умолчанию для левого верхнего угла** окна и игнорирует координату y верхнего края окна. **Для временного и дочернего окна такая позиция по умолчанию равна (0, 0).**
4. **y** – координата верхнего края окна в пикселях. Для перекрывающихся и временных окон ее отсчитывают от верхнего края экрана, дочерних окон – от верхнего края рабочей области родительского окна.
5. **nWidth** – ширина окна в пикселях. Для перекрывающихся окон можно задать равной CW_USEDEFAULT. В этом случае система устанавливает ненулевые значения **ширины и высоты по умолчанию** и игнорирует заданную высоту **nHeight** окна. **Ширина и высота временного и дочернего окна по умолчанию равны нулю.**
6. **hWndParent** может указывать только на инициированное окно. Дескриптор **окна-родителя** обязательно указывают при создании дочернего окна. Дескриптор **окна-владельца** для временного окна указывать необязательно.
7. **hMenu** – дескриптор меню или идентификатор создаваемого дочернего окна.

Меню перекрывающегося или временного окна можно задать тремя способами:

1) в классе окон указывают имя меню, и все окна этого класса могут пользоваться этим меню;
2) имя меню указывают как аргумент функции CreateWindow, и создаваемое окно будет пользоваться этим меню, игнорируя меню класса;
3) меню создают в процессе или после создания окна.

В первом и третьем случаях аргумент hMenu равен NULL.

Если создается дочернее окно, то аргументом hMenu задают идентификатор этого окна (целочисленную константу).

Перед созданием окна функция CreateWindow посылает код сообщения WM_CREATE функции создаваемого окна. Она затем обрабатывает возвращаемое функцией окна значение и создает окно, если это значение равно 0, или не создает окна, если значение равно -1. Соответственно функция Create Window возвращает дескриптор созданного окна или NULL.

Стиль окна задает **внешнее поведение окна**.

Для описания стиля окна используют символические константы с префиксом WS_ (табл. 1.5).

По совокупности свойств различают перекрывающиеся (overlapped), временные (pop-up) и дочерние (child) окна.

Перекрывающиеся окна чаще используют в качестве окон приложения. Они **всегда имеют** заголовок (title bar), рамку и рабочую область окна (client region), **могут иметь** системное меню, кнопки восстановления размеров, закрытия и сворачивания окна в пиктограмму, горизонтальную и вертикальную полосы просмотра (scroll bar), меню, панель инструментов (tool bar) и строку состояния (status bar) (см. рис. 1.1). Базовый стиль таких окон описан константой WS_OVERLAPPED. Чаще используемый стиль окон WS_OVERLAPPED WINDOW вдобавок к базовому указывает, что окно имеет системное меню, кнопки восстановления размеров, закрытия и сворачивания окна.

Перекрывающееся окно может принадлежать другому окну (владельцу). Если **окно-владелец** сворачивается в пиктограмму, то **подчиненные** ему **окна** становятся невидимыми. При уничтожении окна автоматически уничтожаются подчиненные ему окна. Подчиненные окна всегда располагаются над поверхностью окна-владельца, загораживая его.

Временные окна обычно используют для вывода сообщений пользователю и остаются на экране непродолжительное время. Базовый стиль временного окна описан константой WS_POPUP. Такое окно по умолчанию не имеет заголовка. Чаще временное окно описывают константой WS_POPUPWINDOW. Для добавления к временному окну системного меню и заголовка стиль WS_POPUPWINDOW комбинируют со стилем WS_CAPTION. Во всем остальном временные окна – это специальный вид перекрывающихся окон.

Дочерние окна используют для создания органов управления. Определяемые системой классы органов управления (кнопки, полосы просмотра и т. п.) представляют собой дочерние окна. Базовый стиль дочерних окон описан константой WS_CHILD. Этот стиль полностью совпадает со стилем WS_ CHILDWINDOW. Дочерние окна не имеют кнопок минимизации и максимального увеличения размера, но всегда имеют окно-родителя. Они "прилипают" к поверхности родителя, перемещаются с ним и не могут выйти за пределы родительского окна.

1.4. Главная функция приложения

Приложения всегда описывают функцию с именем **WinMain**. Она получает управление при запуске приложения, выполняет присущие

обычным функциям действия, регистрирует классы окон, создает окна, опрашивает очередь сообщений и распределяет свои сообщения.

Пример описания функции WinMain:

```
int WINAPI WinMain( HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow )
{   MSG  msg;  HWND hwnd;
    if  ( !RegClass(WndProc, szClassName) ) return FALSE;
    hwnd = CreateWindow(szClassName, "Пример 1",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance,NULL);
    if (!hwnd) return FALSE;
    ShowWindow( hwnd, SW_SHOWMAXIMIZED );
    UpdateWindow(hwnd);
    while ( GetMessage(&msg, 0, 0, 0) )
    {   TranslateMessage(&msg);  DispatchMessage(&msg); }
    return msg.wParam;
}
```

Имя, тип возврата и список формальных параметров функции определены требованиями Windows API. Модификатор WINAPI указывает на то, что функция при получении аргументов сама должна откорректировать порядок расположения аргументов в стеке.

Параметр hInstance от операционной системы получает дескриптор текущего экземпляра приложения. Параметр hPrevInstance в 16-разрядных приложениях указывает на активный предыдущий экземпляр приложения или NULL. В приложениях Win32 он всегда равен NULL и по нему невозможно узнать о существовании других активных экземпляров этого приложения. Параметр lpszCmdLine указывает на строку с аргументами командной строки запуска приложения. При обычном запуске этот параметр равен NULL. Параметр nCmdShow передает приложению способ начального отображения окна.

В теле функции WinMain описаны переменные msg и hwnd. Переменная msg предназначена для временного хранения сообщений при их получении и распределении. Переменная hwnd хранит дескриптор созданного окна.

Сообщение msg представляет собой структуру типа MSG:

```
typedef struct
{   HWND      hwnd;       //Дескриптор окна-адресата
    UINT      message;    //Код сообщения
```

```
    WPARAM    wParam;     //Содержимое сообщения
    LPARAM    lParam;     //Содержимое сообщения
    DWORD     time;       //Время отправления
    POINT     pt;         //Координаты места отправления
}   MSG;                 //Имя типа
```

Для регистрации класса окон вызывают функцию RegClass:

```
if (!RegClass(WndProc, szClassName) ) return FALSE;
```

Для создания окна вызвана функция CreateWindow. Если окно не создано, приложение завершает свою работу.

Следующий после создания окна шаг выполняют не всегда. В данном примере окно не имеет стиля WS_VISIBLE и создано только в памяти. Для отображения окна вызывают функцию ShowWindow:

```
ShowWindow( hwnd, SW_SHOWMAXIMIZED );
```

Эта функция переводит окно hwnd в состояние, определяемое вторым параметром. Здесь окно hwnd отображают в максимально распахнутом виде.

Следующая функция сообщает функции окна hwnd о необходимости "перерисовки" рабочей области:

```
UpdateWindow(hwnd);
```

Далее следует цикл, который называется **циклом обработки сообщений**:

```
while ( GetMessage(&msg, 0, 0, 0) )
{   TranslateMessage(&msg);   DispatchMessage(&msg); }
```

Здесь функция GetMessage выбирает сообщение из очереди сообщений приложения, TranslateMessage транслирует клавиатурные сообщения, DispatchMessage распределяет сообщения по функциям окон. Именно **здесь происходит "вызов" функции окна**.

Рассмотрим прототипы вышеперечисленных функций.

Функция ShowWindow объявлена следующим образом:

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

При первом после запуска приложения ее вызове вторым аргументом можно передать значение формального параметра nCmdShow функции WinMain. В последующих вызовах вторым аргументом вызова нужно указывать одну из констант следующей таблицы:

Константа	*Действие функции ShowWindow*
SW_HIDE	Скрывает окно hwnd и активизирует другое окно
SW_MAXIMIZE	Максимизирует окно
SW_MINIMIZE	Сворачивает окно hwnd и активизирует расположенное под ним окно

SW_RESTORE	Восстанавливает свернутое окно hwnd
SW_SHOW	Активизирует окно hwnd в текущих размерах и позиции
SW_SHOWMAXIMIZED	Активизирует окно hwnd в максимально распахнутом виде
SW_SHOWMINIMIZED	Активизирует окно hwnd в свернутом виде
SW_SHOWMINNOACTIVE	Сворачивает окно hwnd
SW_SHOWNA	Показывает состояние окна hwnd
SW_SHOWNOACTIVATE	Показывает окно hwnd в текущих координатах
SW_SHOWNORMAL	Активизирует и отображает окно hwnd в первоначальных размерах и позиции

Функция GetMessage имеет следующий прототип:

```
BOOL GetMessage( LPMSG lpmsg, HWND hwnd,
        WORD uMsgFilterMin, WORD uMsgFilterMax );
```

Параметр lpmsg указывает на структуру типа MSG, в которую будет записано выбираемое сообщение. hwnd – дескриптор окна, для которого нужно выбрать сообщение. Если hwnd=NULL, то выбираются сообщения для всех окон приложения. Параметры uMsgFilterMin и uMsgFilterMax указывают диапазон выбираемых сообщений, задавая соответственно минимальное и максимальное значение кодов сообщений. Если для этих параметров указать нулевые значения, из очереди будут выбираться все сообщения.

Функция GetMessage выбранное сообщение удаляет из очереди и записывает его в структуру, адрес которой задан ее первым параметром. При выборке сообщения с кодом WM_QUIT она возвращает 0, и завершается цикл обработки сообщений. При выборке других сообщений функция GetMessage возвращает ненулевое значение.

После этого функция WinMain возвращает управление и значение поля wParam последнего сообщения операционной системе.

1.5. Структура текста приложения

Структура текста приложения определяется двумя требованиями:

1. Текст содержит описание функции с именем WinMain.

2. Если регистрируются новые классы, то текст содержит описание структуры типа WNDCLASS и функций окон этих классов.

Приложение листинга 1.1 регистрирует класс перекрывающихся окон, создает окно этого класса и передает управление циклу обработки сообщений.

Листинг 1.1. Иллюстрация структуры простейшего приложения.

```
#include <windows.h>
//Объявление функций
BOOL      RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM, LPARAM);

//Описание глобальных переменных
HINSTANCE   hInst;
char szClassName[ ]="WindowAppClass";
//Описание главной функции
int WINAPI  WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  hInst=hInstance;
    if (!RegClass( WndProc, szClassName,COLOR_WINDOW ))
        return FALSE;
    hwnd = CreateWindow(szClassName, "Мое приложение",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}

//Описание функции регистрации классов
BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc; wc.style = wc.cbClsExtra = wc.cbWndExtra = 0;
    wc.lpfnWndProc = Proc;     wc.hInstance = hInst;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = NULL;     wc.lpszClassName = szName;
    return (RegisterClass(&wc) != 0);.
}
//Описание функции окон
LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                          WPARAM wParam, LPARAM lParam )
```

```
{   switch (msg)
    {   case WM_DESTROY: { PostQuitMessage(0); return 0; }   }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Как видно из листинга, приложение является обычной программой языка Си. Текст содержит привычные директивы, объявления и описания. Отличие содержится в заголовке главной функции, в специфической функции окна и в способе ее вызова.

Рассмотрим алгоритм работы этого приложения.

1. Классы окон регистрируют в начале работы приложения.

2. Окно приложения создают до передачи управления циклу обработки сообщений.

3. Отсутствуют вызовы функций ShowWindow и UpdateWindow. Это обусловлено тем, что окно создано со стилем WS_VISIBLE. В таком случае функция CreateWindow посылает все сообщения, которые необходимы для отображения созданного окна и перерисовки его рабочей области.

4. Управление передают циклу обработки сообщений. В теле этого цикла выбираются и распределяются сообщения.

5. Функция окна обрабатывает сообщение WM_DESTROY, а остальные сообщения возвращает системе. **Функция окна приложения должна обрабатывать это сообщение**.

6. Приложение завершает работу.

Таким же является алгоритм работы большинства приложений. Отличие между ними в основном заключается в функциях окон.

Может показаться, что отображение и удаление окна – слишком простая задача для такого "сложного" текста.

Проанализируем, что же на самом деле может сделать это приложение.

Рис. 1.2 иллюстрирует, что увидит пользователь после запуска приложения.

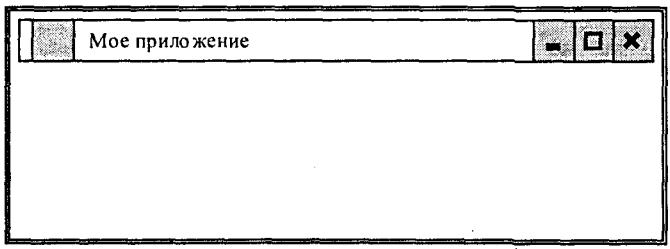

Рис. 1.2. Окно простейшего приложения

Как видим, стиль полностью определяет **внешнее поведение окна**. Благодаря стилю WS_OVERLAPPEDWINDOW окно содержит двойные

рамки изменения размеров и заголовок, в области заголовка расположены системное меню и кнопки для сворачивания, восстановления размеров и закрытия окна. С помощью этих элементов можно перемещать окно, изменять его размеры, сворачивать окно в пиктограмму, восстанавливать и закрыть окно.

Функциональные возможности окна могли бы быть необозримо шире. Это **окно принимает все сообщения** от пользователя, системы и других окон. То есть оно готово сотрудничать с пользователем и открыть доступ к ресурсам системы. Но описание функции окна в примере содержит обработку только одного сообщения. Остальные сообщения в примере не обрабатывают и передают операционной системе. По сути, они только удаляются из очереди сообщений.

Обработка сообщений в первую очередь подразумевает определение текущего состояния окон и управление этим состоянием, выдачу тех или иных сообщений пользователю. Приступим к изучению таких функций операционной системы. Сначала рассмотрим функции, которые связаны с процессом создания окон.

1.6. Вспомогательные функции создания окон

Windows API содержит множество функций, связанных с созданием окон. Это функции поиска, определения состояния, перемещения окон, а также обмена сообщениями с пользователем.

1.6.1. Функции поиска и определения состояния окон

Часто требуется определить, существует ли окно для некоторого дескриптора. На этот вопрос отвечает функция IsWindow:

```
BOOL IsWindow( HWND hwnd );
```

Если окно с дескриптором hwnd существует, функция возвращает ненулевое значение, иначе – нуль.

Если нужно определить, имеет ли заданное окно фокус ввода, вызывают функцию IsWindowEnabled:

```
BOOL IsWindowEnabled( HWND hwnd );
```

Если окно hwnd активно, функция возвращает ненулевое значение, иначе – нуль.

Для передачи или отнятия фокуса ввода у окна вызывают функцию EnableWindow:

```
BOOL EnableWindow( HWND hwnd, BOOL bEnable );
```

При bEnable=TRUE фокус ввода передают окну hwnd, иначе – блокируют это окно.

Если окно ранее было неактивно, возвращаемое значение отлично от нуля. Если окно было активно, возвращаемое значение – нуль.

Фокус ввода окну hwnd передают с помощью функции SetFocus:

```
SetFocus(hwnd);
```

Следующая функция возвращает ненулевое значение, если окно hwnd свернуто в пиктограмму:

```
BOOL IsIconic(HWND hwnd);
```

Функция FindWindow у операционной системы запрашивает дескриптор окна (не дочернего) класса lpClassName с заголовком lpWindowName. Эта функция объявлена следующим образом:

```
HWND FindWindow( LPCTSTR lpClassName,
                 LPCTSTR lpWindowName );
```

Если lpWindowName – NULL, то заголовок искомого окна может быть любым. Если lpClassName – NULL, искомое окно может принадлежать к любому классу. Рекомендуется указывать данные как можно более подробнее. Если такое окно найдено, функция возвращает его дескриптор, иначе – NULL.

1.6.2. Функции перемещения окон

Для перемещения и изменения размеров окна hwnd вызывают функцию MoveWindow. Ей передают новые координаты окна.

Функция MoveWindow объявлена следующим образом:

```
BOOL MoveWindow(  HWND hwnd,
    int     x,          //новая координата левого края окна
    int     y,          //новая координата верхнего края окна
    int     nWidth,     //новая ширина окна
    int     nHeight,    //новая высота окна
    BOOL    bRepaint    //флажок перекрашивания окна
);
```

Если bRepaint=TRUE, немедленно после перемещения перерисовываются те части экрана, на которых отразилось перемещение окна. Иначе окно перерисовывает себя только после обработки всех поступивших к моменту перемещения сообщений. При успешном выполнении функция возвращает ненулевое значение, иначе – нуль.

Функция SetWindowPos изменяет координаты окна hwnd на экране и его расположение по отношению к другим окнам:

```
BOOL SetWindowPos( HWND hwnd,
    HWND    hWndInsertAfter, //дескриптор порядка размещения
    int     x,               //новая координата левого края
```

```
    int     y,      //новая координата верхнего края
    int     cx,     //новая ширина
    int     cy,     //новая высота
    UINT    uFlags  //флажок позиционирования
);
```

В случае успешного выполнения она возвращает ненулевое значение.

Параметр hWndInsertAfter может быть дескриптором предшествующего окна или равным одному из следующих значений:

Значение	*Пояснение*
HWND_BOTTOM	Помещает окно ниже других окон
HWND_NOTOPMOST	Помещает временное или дочернее окно выше временных и дочерних, но ниже перекрывающихся окон
HWND_TOP	Помещает окно выше всех окон
HWND_TOPMOST	То же, что HWND_NOTOPMOST, но окно сохраняет позицию после потери активности

Параметр uFlags может быть комбинацией значений из нижеследующей таблицы:

Значение	*Пояснение*
SWP_DRAWFRAME	Вокруг окна рисовать заданную в классе окна рамку
SWP_FRAMECHANGED	Функции окна посылать сообщение об изменении размеров, даже если размер окна не изменяется
SWP_HIDEWINDOW	Скрыть окно
SWP_NOACTIVATE	Отменить активность окна
SWP_NOCOPYBITS	Стереть содержимое рабочей области
SWP_NOMOVE	Сохранить текущую позицию
SWP_NOOWNERZORDER или SWP_NOREPOSITION	Не изменять расположение окна относительно других окон
SWP_NOREDRAW	Не перерисовывать содержимое окна
SWP_NOSENDCHANGING	Не сообщать об изменении позиции
SWP_NOSIZE	Сохранить текущий размер
SWP_NOZORDER	Не менять расположение
SWP_SHOWWINDOW	Показать окно

При одновременной работе нескольких приложений одно из них можно вывести на передний план и передать его окну фокус ввода.

Для этих целей используют функцию SetForegroundWindow. Она помещает поток, который создал заданное окно, на передний план и активизирует это окно. Синтаксис этой функции:

```
BOOL SetForegroundWindow ( HWND hwnd );
```

В случае успешного перемещения функция возвращает ненулевое значение, иначе – нуль.

Для перемещения окна важно знать системные метрики экрана и окна. Их получают с помощью функции GetSystemMetrics. Она возвращает метрики и текущие установки конфигурации системы. Метрики системы – это габариты (ширина и высота) отображаемых элементов Windows. Все габариты возвращаются в пикселях.

Функция GetSystemMetrics объявлена следующим образом:

```
int GetSystemMetrics ( int nIndex );
```

Параметр nIndex указывает на возвращаемую метрику системы или установку конфигурации. Ниже приводится таблица некоторых имен для этого параметра:

Имя	*Возвращаемая характеристика*
SM_CXMIN	Минимальная ширина окна
SM_CYMIN	Минимальная высота окна
SM_CXSCREEN	Ширина экрана
SM_CYSCREEN	Высота экрана
SM_CYCAPTION	Высота заголовка
SM_CYMENU	Высота полосы меню

Например, следующий оператор определяет ширину экрана:

```
int w=GetSystemMetrics(SM_CXSCREEN);
```

Следующая функция позволяет получить координаты обрамляющего окно hwnd прямоугольника:

```
BOOL GetWindowRect( HWND hwnd, LPRECT lpRect );
```

Параметр lpRect указывает на структуру типа RECT:

```
typedef struct
{   LONG    left;       //левый край
    LONG    top;        //верхний край
    LONG    right;      //правый край
    LONG    bottom;     //нижний край
}   RECT;
```

Координаты прямоугольника отсчитываются от левого верхнего угла экрана в пикселях. В случае успешного выполнения функция возвращает ненулевое значение. Следующий фрагмент демонстрирует получение экранных координат прямоугольника окна:

```
RECT  rc;  GetWindowRect(hwnd, &rc);
```

Функция GetClientRect возвращает координаты прямоугольника, обрамляющего рабочую область окна:

```
BOOL GetClientRect( HWND hwnd, LPRECT lpRect );
```

При этом прямоугольник смещен в левый верхний угол экрана. То есть rc.left=rc.top=0, а значения полей right и bottom равны соответственно ширине и высоте рабочей области.

1.6.3. Сообщения приложения для пользователя

Для вывода текстовых сообщений и получения ответа пользователя применяют **окно сообщения**. Функция MessageBox создает, отображает, обеспечивает работу и закрывает окно сообщения:

```
int WINAPI MessageBox (
    HWND      hwnd,       //Дескриптор родительского окна
    LPCTSTR   lpText,     //Адрес текста сообщения
    LPCTSTR   lpCaption,  //Адрес заголовка окна сообщения
    UINT      uType       //Стиль окна сообщения
);
```

Если hwnd=NULL, окно сообщения не имеет родительского окна. При lpCaption= NULL в заголовке по умолчанию выводится строка "Ошибка". Параметр uType задают в виде комбинации флажков, определяющих содержание и поведение окна сообщения.

Следующие флаги определяют список кнопок окна сообщения:

Флаг	*Список имен кнопок*
MB_ABORTRETRYIGNORE	**"Стоп"**, "Повтор" и "Пропустить"
MB_OK	**OK**
MB_OKCANCEL	**OK** и "Отмена"
MB_RETRYCANCEL	**"Повтор"** и "Отмена"
MB_YESNO	**"Да"** и "Нет"
MB_YESNOCANCEL	**"Да"**, "Нет" и "Отмена"

Жирным шрифтом выделены кнопки по умолчанию. По умолчанию активна первая кнопка. Следующие флаги могут назначить одну из кнопок активной по умолчанию:

Флаг	*Номер кнопки*	*Флаг*	*Номер кнопки*
MB_DEFBUTTON1	Первая	MB_DEFBUTTON3	Третья
MB_DEFBUTTON2	Вторая	MB_DEFBUTTON4	Четвертая

Следующие флаги включают одну иконку в окно сообщения:

Флаги	*Вид иконки*
MB_ICONEXCLAMATION, MB_ICONWARNING	Восклицательный знак
MB_ICONINFORMATION, MB_ICONASTERISK	Символ i
MB_ICONQUESTION	Знак вопроса
MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND	Знак остановки

Следующие флаги определяют стиль окна сообщения:

Флаг	*Пояснение*
MB_APPLMODAL	Этот стиль по умолчанию. Окно hwnd переводится в неактивное состояние на время работы окна сообщения, но можно активизировать другие приложения или окна, не дочерние по отношению к hwnd
MB_SYSTEMMODAL	На время работы окна сообщения все другие приложения в неактивном состоянии
MB_TASKMODAL	Этот стиль подобен MB_APPLMODAL. Но если родительское окно не указано (hwnd= NULL), блокируются все перекрывающиеся окна этого приложения
MB_HELP	Добавляет кнопку "Справка" в окне сообщения
MB_RIGHT	Текст выравнивается по правому краю
MB_RTLREADING	Отображает символы сообщения и текста заголовка в направлении справа налево
MB_SETFOREGROUND	Окно сообщения выдвигается на передний план

Приложение может обработать ответ пользователя на сообщение, анализируя возвращаемое функцией MessageBox значение. В случае

ошибки возвращается 0. Иначе функция MessageBox возвращает константу, соответствующую нажатой кнопке. Ниже перечислены имена констант, соответствующие кнопкам окна сообщения:

Константа	*Нажата кнопка*	*Константа*	*Нажата кнопка*
IDABORT	"Стоп"	IDOK	OK
IDCANCEL	"Отмена"	IDRETRY	"Повтор"
IDIGNORE	"Пропустить"	IDYES	"Да"
IDNO	"Нет"		

Если окно сообщения содержит кнопку "Отмена", то значение IDCANCEL может быть возвращено и при нажатии клавиши Esc.

Windows предусматривает возможность выдачи и звуковых "сообщений". Например, при вызове функции MessageBeep компьютер проигрывает "звук" из множества зарегистрированных в системе "звуков". Синтаксис этой функции:

```
BOOL MessageBeep( UINT uType );
```

Параметр uType этой функции определяет звуковой тип и принимает одно из следующих значений:

Значение	*Обозначение в файле win.ini*
0xFFFFFFFF или -1	Стандартный звуковой сигнал
MB_ICONASTERISK	SystemAsterisk
MB_ICONEXCLAMATION	SystemExclamation
MB_ICONHAND	SystemHand
MB_ICONQUESTION	SystemQuestion
MB_OK или 0	SystemDefault

В случае успешного выполнения функция возвращает ненулевое значение, иначе – 0. После передачи сообщения о формировании звука, функция MessageBeep прекращает работу. Далее выполняются последующие операторы приложения, и одновременно проигрывается указанный звук.

Если требуется обратить внимание пользователя на определенное окно, то можно несколько раз изменить подсветку окна или его пиктограммы.

Для этой цели используют функцию FlashWindow, которая за один вызов один раз изменяет подсветку указанного окна:

```
BOOL FlashWindow(
   HWND hwnd,  //Дескриптор подсвечиваемого окна
```

```
    BOOL  bInvert //Параметр подсветки
);
```

Если параметр bInvert=TRUE, окно меняет подсветку. Если же bInvert=FALSE, окно только возвращается к исходному состоянию.

При подсветки неактивное окно может принять внешний вид активного окна, но не получит фокуса ввода.

Если окно было активно до вызова FlashWindow, возвращается ненулевое значение. Если окно не было активно, возвращается 0.

Пример. После нажатия левой клавиши мыши над окном hwnd 5 раз изменить подсветку неактивного окна OwnedHwnd.

Следующий фрагмент описывает версию решения этой задачи:

```
case WM_LBUTTONDOWN:
{   FlashWindow(OwnedHwnd, TRUE);
    for (int i=0; i<9; i++)
    {   for (long j=0; j<1e7; j++) ;
        FlashWindow(OwnedHwnd, TRUE);
    }
    return 0;
}
```

После нажатия левой клавиши мыши окно OwnedHwnd примет вид активного окна:

```
FlashWindow(OwnedHwnd, TRUE);
```

При i=0 выполняется некая "работа", обозначенная циклом

```
for (long j=0; j<1e7; j++) ;
```

Эта работа сводится к простой задержке времени. После этой паузы окно OwnedHwnd примет вид неактивного окна. При i=1 после паузы окно OwnedHwnd примет вид активного окна, а при i=2 после паузы окно OwnedHwnd примет вид неактивного окна... Таким образом, окно OwnedHwnd 5 раз изменяет свой внешний вид.

1.7. Примеры создания окон

Рассмотрим **два различных примера**.

В первом примере разработчик в тело главной функции добавляет фрагмент, который проверяет наличие активного экземпляра приложения и выполняет соответствующие действия. При этом тело функции окна, в которое обычно добавляется текст разработчика, содержит только минимально необходимый набор операторов.

Во втором примере главная функция содержит только присущий подавляющему большинству приложений набор описаний, а функция

окна содержит все те операторы, которые добавляет разработчик для решения поставленной задачи.

Обратите внимание, что тексты (листинги 1.2 и 1.3) приложений содержат одни и те же наборы типовых функций. Отличие лишь в их содержании.

1.7.1. Проверка наличия предыдущего экземпляра

Задача. Окно приложения стиля временного окна с кнопкой минимизации и с фоном цвета рабочего стола Windows занимает весь экран. При попытке запуска второго экземпляра на передний план переместить окно первого экземпляра и завершить работу.

Анализ задания. (Полный текст приложения приведен ниже, в листинге 1.2.)

Для поиска других экземпляров приложения воспользуемся функцией FindWindow:

```
HWND hwnd=FindWindow(szMainClass,szTitle);
```

Она возвращает дескриптор окна класса szMainClass с заголовком szTitle. Если такого окна нет, то это первый экземпляр приложения и дескриптору hwnd присваивается NULL. Иначе, возвращается дескриптор найденного окна.

Внимание! Функция FindWindow проверяет только совпадение имени класса и заголовка. То есть, если работает экземпляр другого приложения с окном класса с таким же именем szMainClass и заголовком szTitle, то функция вернет дескриптор окна другого приложения.

Если "предыдущий экземпляр приложения" обнаружен, выводим сообщение:

```
MessageBox(hwnd,
    "Можно запускать только одну копию приложения!!!\n\n"
    "Перемещаю на передний план первый экземпляр",
    szTitle, MB_OK | MB_ICONSTOP);
```

До перемещения окна на передний план проверяем состояние этого окна и, если оно свернуто, восстанавливаем его предыдущее состояние:

```
if ( IsIconic( hwnd ) )  ShowWindow( hwnd, SW_RESTORE );
```

Перемещаем окно первого экземпляра приложения на передний план:

```
SetForegroundWindow(hwnd);
```

Затем второй экземпляр приложения завершает работу.

Если предыдущий экземпляр приложения не найден, регистрируем класс окна:

```
if ( !RegClass(WndProc,szMainClass,COLOR_DESKTOP) )
    return FALSE;
```

Цвет фона окна приложения определяем по табл. 1.4 – COLOR_ DESKTOP.

Для создания максимально распахнутого окна приложения определяем ширину и высоту экрана в пикселях:

```
int w=GetSystemMetrics(SM_CXSCREEN);
int h=GetSystemMetrics(SM_CYSCREEN);
```

Создаем окно:

```
hwnd = CreateWindow(szMainClass, szTitle,
    WS_POPUPWINDOW | WS_CAPTION | WS_MINIMIZEBOX |
    WS_VISIBLE, 0, 0, w, h, 0, 0, hInstance, NULL);
```

Стиль временного окна с кнопкой минимизации и заголовком определяем по табл. 1.5 – WS_POPUPWINDOW | WS_CAPTION | WS_MINIMIZEBOX | WS_VISIBLE.

Далее активизируем цикл обработки и диспетчеризации сообщений:

```
while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
```

Это приложение не предусматривает обработки сообщений с клавиатуры. Поэтому в цикле отсутствует вызов функции трансляции клавиатурных сообщений.

Любые действия над окном передают функции окна:

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
                         WPARAM wParam, LPARAM lParam)
{   switch (msg)
    {   case WM_DESTROY: { PostQuitMessage(0); return 0; }   }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Такая функция окна ранее уже была рассмотрена.

Листинг 1.2. Запрет запуска второго экземпляра приложения.

```
#include <windows.h>

//Объявления функций
BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);

HINSTANCE   hInstance;
char    szMainClass[ ] = "MainClass";
char    szTitle[ ]= "Пример 1.2";

//Главная функция приложения
int WINAPI  WinMain(HINSTANCE hInst, HINSTANCE hPrevInst,
                    LPSTR lpszCmdLine, int nCmdShow)
```

```
{   MSG msg; hInstance=hInst;
    HWND hwnd=FindWindow(szMainClass,szTitle);
    if ( hwnd )
    {   MessageBox(hwnd,
            "Можно запускать только один экземпляр приложения!\n"
            "Перемещаю на передний план первый экземпляр",
            szTitle, MB_OK | MB_ICONSTOP);
        //Если окно свернуто, восстанавливаем
        if ( IsIconic( hwnd ) )
            ShowWindow(hwnd, SW_RESTORE);
        SetForegroundWindow(hwnd);
        return 0;
    }
    if ( !RegClass(WndProc,szMainClass,COLOR_DESKTOP) )
        return FALSE;
    //До создания окна узнаем габариты экрана в пикселях
    int w=GetSystemMetrics(SM_CXSCREEN)-1;  //Ширина
    int h=GetSystemMetrics(SM_CYSCREEN)-1;  //Высота
    hwnd = CreateWindow(szMainClass, szTitle,
        WS_POPUPWINDOW | WS_CAPTION |
        WS_MINIMIZEBOX | WS_VISIBLE,
        0, 0, w, h, 0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc;
    wc.style=wc.cbClsExtra=wc.cbWndExtra=0;
    wc.lpfnWndProc=Proc;
    wc.hInstance = hInstance;
    wc.hIcon=LoadIcon(NULL,IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName  = (LPCTSTR)NULL;
    wc.lpszClassName=szName;
    return (RegisterClass(&wc) != 0);
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
                         WPARAM wParam,  LPARAM lParam)
```

```
{   switch (msg)
    {   case WM_DESTROY:
        {   PostQuitMessage(0);   return 0;   }
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

1.7.2. Расположение окон черепицей

Задача. В правом нижнем углу рабочей области окна приложения при первом нажатии левой клавиши мыши отобразить окно с заголовком "1-е окно", при втором нажатии – окно с заголовком "2-е окно" и так до пяти окон. Начиная со второго, окна располагать под предыдущим, сдвигая вверх и влево так, чтобы был виден заголовок, а левый верхний угол 5-го окна должен совпадать с левым верхним углом рабочей области окна приложения. Если какие-либо из этих окон закрыты, то при нажатии левой клавиши мыши должно быть создано первое из закрытых окон.

Анализ задания. (Полный текст приложения приведен ниже, в листинге 1.3.)

Регистрируем класс окна приложения:

```
if ( !RegClass(WndProc, szMainClass, COLOR_WINDOW) )
    return FALSE;
```

Фон окна приложения в задаче не указан, поэтому по табл. 1.4 выбираем стандартный цвет COLOR_WINDOW.

Регистрируем класс временных окон:

```
if  ( !RegClass(WndPopup, szPopupClass,COLOR_BTNFACE) )
    return FALSE;
```

Для разнообразия цветом фона временных окон выбираем цвет трехмерных элементов.

Создаем окно приложения:

```
hwnd = CreateWindow(szMainClass, szTitle,
    WS_OVERLAPPEDWINDOW | WS_VISIBLE,
    CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, 0, 0, hInstance, NULL);
```

Для него указываем стиль WS_OVERLAPPEDWINDOW | WS_VISIBLE. Координаты окна задаем по умолчанию.

Далее активизируем цикл обработки сообщений:

```
while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
```

Сообщения из этого цикла попадают в две различные функции окон. Функция окна приложения WndProc теперь содержит описания, которые

решают поставленную задачу. А функция временных окон все сообщения возвращает операционной системе. Даже сообщение о закрытии временного окна обрабатывает операционная система.

Все действия по созданию, отображению и расположению временных окон выполняет функция окна приложения при обработке кодов сообщений WM_MOVE, WM_SIZE и WM_LBUTTONDOWN.

При обработке кода WM_MOVE нужно переместить существующие временные окна. При перемещении окна-владельца временные окна остаются на прежних позициях. Но по условию задачи они должны располагаться от правого нижнего до левого верхнего угла рабочей области окна-владельца. Поэтому запоминаем новые координаты левого верхнего угла рабочей области окна приложения:

```
left=LOWORD(lParam);    top=HIWORD(lParam);
```

Остается переместить существующие временные окна:

```
for ( short j=0; j<5; j++ )
    if ( IsWindow( hwnds[j] ) )
        MoveWindow( hwnds[j],    left+cxClient-Width-xStep*j,
            top+cyClient-Height-yStep*j, Width,Height,TRUE );
```

Координата левого края j-го временного окна отсчитывается от правого края рабочей области (left+cxClient) и равна величине:

```
left+cxClient-Width-xStep*j.
```

Аналогично отсчитывается координата верхнего края j-го временного окна.

При поступлении кода WM_SIZE требуется пересчитать размеры временных окон. Для этого запоминаем новую ширину и высоту рабочей области окна приложения:

```
cxClient=LOWORD(lParam);   cyClient=HIWORD(lParam);
```

Тогда ширину и высоту временных окон можно вычислить по выражениям Width=cxClient/2, Height=cyClient-4*yStep.

Временные окна должны быть смещены друг по отношению к другу. Шаг смещения окон по горизонтали зависит от ширины рабочей области. Поэтому величину шага также пересчитываем:

```
xStep=(cxClient-Width)/4;
```

Остается только переместить существующие временные окна.

Эта часть обработки кода WM_SIZE полностью совпадает с такой же частью обработки кода WM_MOVE:

```
for ( short j=0; j<5; j++ )
    if ( IsWindow( hwnds[j] ) )
        MoveWindow( hwnds[j],    left+cxClient-Width-xStep*j,
            top+cyClient-Height-yStep*j, Width,Height,TRUE );
```

Здесь перебирают все элементы массива дескрипторов hwnds, и, если для j-го элемента массива (дескриптора hwnds[j]) существует окно, это окно перемещается в новые координаты.

При обработке кода WM_LBUTTONDOWN (при нажатии левой клавиши мыши) перебирают элементы массива дескрипторов hwnds до тех пор, пока не обнаруживается, что для j-го элемента массива (дескриптора hwnds[j]) не существует окна:

```
for ( short j=0;  j<5,  IsWindow(hwnds[j]);  j++ ) ;
```

Если для всех дескрипторов массива существуют окна, обработка сообщения завершается. Иначе для этого дескриптора в определенном условиями задачи месте создается окно:

```
hwnds[j] = CreateWindow(szPopupClass, str,
    WS_POPUPWINDOW | WS_CAPTION | WS_VISIBLE,
    left+cxClient-Width-xStep*j, top+cyClient-Height-yStep*j,
    Width, Height, hwnd, 0, hInstance, NULL);
```

Следующее действие связано с изменением расположения только что созданного окна без перемещения:

```
if (j>0)
{   SetWindowPos(hwnds[j], hwnds[j-1], 0, 0, Width, Height, SWP_NOMOVE);
    SetForegroundWindow(hwnds[0]);
}
```

То есть функция SetWindowPos располагает окно hwnds[j] под предыдущим окном hwnds[j–1]. А функция SetForegroundWindow выдвигает на передний план первое окно из массива дескрипторов. Таким образом, при создании нового временного окна на переднем плане всегда окажется первое из них.

Этим завершается обработка кода WM_LBUTTONDOWN.

Листинг 1.3. Пример расположения окон черепицей в обратном порядке.

```
#include <windows.h>

//Объявление функций
BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM );
LRESULT CALLBACK WndPopup(HWND,UINT,WPARAM,LPARAM);

HINSTANCE  hInstance;
char szMainClass[ ] = "MainClass";
char szPopupClass[ ] = "PopupClass";
char szTitle[ ]= "Пример 1.3";
```

```
int WINAPI  WinMain(HINSTANCE hInst,   HINSTANCE hPrevInstance,
                    LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd; hInstance=hInst;
    if ( !RegClass(WndProc, szMainClass,COLOR_WINDOW) )
        return FALSE;
    if ( !RegClass(WndPopup,szPopupClass,COLOR_BTNFACE) )
        return FALSE;
    hwnd = CreateWindow(szMainClass, szTitle,
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc;    wc.style=wc.cbClsExtra=wc.cbWndExtra=0;
    wc.lpfnWndProc=Proc; wc.hInstance = hInstance;
    wc.hIcon=LoadIcon(NULL,IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground= (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = (LPCTSTR)NULL;
    wc.lpszClassName=szName;
    return (RegisterClass(&wc) != 0);
}
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
                         WPARAM wParam, LPARAM lParam)
{   static short cxClient, cyClient, yStep, xStep;
    static short left, top, Width, Height;
    //Описываем массив дескрипторов окон
    static HWND hwnds[5];
    switch (msg)
    {   case WM_CREATE:
        {   //Шаг смещения окон по вертикали
            yStep=GetSystemMetrics(SM_CYCAPTION);
            return 0;
        }
        case WM_MOVE:
        {   //Левый верхний угол рабочей области
```

```
    left=LOWORD(lParam); top=HIWORD(lParam);
    //Перемещаем существующие временные окна
    for (short j=0; j<5; j++)
        if (IsWindow(hwnds[j]))
            MoveWindow(hwnds[j],
                left+cxClient-Width-xStep*j,
                top+cyClient-Height-yStep*j,
                Width,Height,TRUE);   return 0;
}
case WM_SIZE:
{   //Ширина и высота рабочей области
    cxClient=LOWORD(lParam);
    cyClient=HIWORD(lParam);
    //Ширина и высота временных окон
    Width=cxClient/2;   Height=cyClient-4*yStep;
    //Шаг смещения окон по горизонтали
    xStep=(cxClient-Width)/4;
    //Перемещаем существующие временные окна
    for (short j=0; j<5; j++)
        if (IsWindow(hwnds[j]))
            MoveWindow(hwnds[j],
                left+cxClient-Width-xStep*j,
                top+cyClient-Height-yStep*j,
                Width,Height,TRUE);
    return 0;
}
case WM_LBUTTONDOWN:
{   //Ищем свободное место в массиве дескрипторов
    for (short j=0; j<5, IsWindow(hwnds[j]); j++) ;
    //Если свободного места нет, возвращаемся
    if (j>4) return 0;
    //Формируем заголовок окна
    char str[20]; _itoa(j+1, str, 10); strcat(str,"-е окно");
    //Создаем j-е временное окно
    hwnds[j] = CreateWindow(szPopupClass, str,
        WS_POPUPWINDOW | WS_CAPTION |
        WS_VISIBLE,
        left+cxClient-Width-xStep*j,
        top+cyClient-Height-yStep*j,
        Width, Height, hwnd, 0, hInstance, NULL);
    //Если создано не первое окно, перемещаем его вниз
```

```
            if (j>0)
            {   SetWindowPos( hwnds[j], hwnds[j-1], 0, 0,
                    Width, Height, SWP_NOMOVE );
                //Перемещаем на передний план первое окно
                SetForegroundWindow(hwnds[0]);
            }   return 0;
        }
        case WM_DESTROY: { PostQuitMessage(0); return 0; }
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}

LRESULT CALLBACK WndPopup(HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam)
{   return DefWindowProc(hwnd, msg, wParam, lParam); }
```

Контрольные вопросы

1. Чем отличается текст приложения, которое создает окна, от текста приложения, которое не создает окон?
2. Каков алгоритм работы главной функции приложения?
3. Какие действия должно выполнить приложение для создания окна?
4. Описания каких программных объектов нужно включить в текст приложения для регистрации класса окон?
5. В каком месте и как происходит вызов функции окна?
6. Каково назначение формальных параметров функции окна?
7. Какие сообщения получает функция окна на этапе создания окна?
8. Что происходит с теми сообщениями, которые функция окна не обрабатывает?
9. Каковы отличительные характеристики перекрывающихся, временных и дочерних окон?
10. Как единственный раз создать окно в теле функции окна-владельца?

Упражнения

1. После нажатия на левую (правую) клавишу мыши над рабочей областью окна в левом верхнем (правом нижнем) углу области отобразить временное окно размером в четверть области. Временное окно скрыть после отжатия клавиши в любом месте экрана.
2. В левом верхнем (в правом нижнем) углу рабочей области окна создать временное (дочернее) окно. После нажатия левой клавиши мыши

4 раза "мигает" временное окно, а после нажатия правой – 3 раза "мигает" дочернее окно. При этом окна выдают различные звуковые сигналы.

3. Создать окно приложения размером в одну шестнадцатую площади экрана с заголовком "Форматирование диска" без кнопок изменения размеров, закрытия и сворачивания в пиктограмму и без кнопки системного меню. При перемещении курсора мыши над рабочей областью окно должно "убегать" от курсора мыши в случайным образом выбранном направлении, оставаясь в пределах экрана.

4. В левом верхнем углу рабочей области окна создать временное окно площадью в одну шестнадцатую площади этой области. При нажатии на левую (правую) клавишу мыши временное окно переместить в соседний по ходу (против хода) часовой стрелки угол рабочей области.

5. При запуске i-го экземпляра (i>2) приложения спросить пользователя, нужно ли его запустить. Если пользователь ответит "Да", то запустить его. Иначе на передний план переместить 2-й экземпляр приложения и завершить работу i-го экземпляра.

6. Углы рабочей области окна приложения полностью занимают 4 временных окна одного класса. Если нажать левую клавишу мыши над временным окном, то это окно выдает сообщение о своем заголовке.

7. Окно приложения без заголовка занимает весь экран фоном рабочего стола.

8. В центре рабочей области окна располагается невидимое окно без заголовка размером в четверть площади рабочей области. После нажатия левой клавиши мыши над рабочей областью любого из окон окно без заголовка должно стать видимым, а после нажатия правой – невидимым.

9. В центре рабочей области окна отображено дочернее окно с фоном цвета трехмерных элементов с вертикальной и горизонтальной полосами просмотра размером в четверть этой области. Дочернее окно перемещается в тот угол рабочей области, где нажали левую клавишу мыши.

10. В центре рабочей области окна расположено окно без заголовка с вертикальной и горизонтальной полосами просмотра размером в четверть рабочей области. При нажатии разных клавиш мыши временное окно выдает разный звуковой сигнал.

11. Создать окно размером в четверть площади экрана. После двойного щелчка мыши окно перемещается так, что его центр совпадает с координатами курсора мыши в момент щелчка.

12. Дочернее окно размером 100*100 пикселей при перемещении курсора мыши над ним "убегает" от курсора мыши в произвольном направлении, оставаясь в пределах рабочей области родительского окна.

13. При запуске второго экземпляра приложения сообщить о запрете запуска нескольких экземпляров, на передний план переместить первый

экземпляр приложения, 3 раза изменить подсветку его окна, выдавая звуковое предупреждение, и завершить работу второго экземпляра.

14. В рабочей области окна приложения рядом друг с другом расположить 3 временных окна, каждое из которых по-своему реагирует на нажатие левой клавиши мыши.

15. Окно первого экземпляра приложения расположить в левом верхнем, второго – в правом верхнем, третьего – в левом нижнем, четвертого – в правом нижнем углу экрана. Причем все окна равных размеров и вместе занимают весь экран. В заголовке окна указать номер экземпляра. Запретить запуск пятого экземпляра.

16. Центр рабочей области окна занимает временное окно размером в четверть площади области в свернутом состоянии. После нажатия левой клавиши мыши над рабочей областью временное окно распахивается в центре области, а после нажатия правой – сворачивается в центре.

17. Центр рабочей области окна занимает временное окно размером в четверть площади области. Оно перемещается в тот угол рабочей области, где щелкнули левой клавишей мыши. А после щелчка правой клавишей мыши временное окно перемещается в угол, противоположный текущему углу.

18. Окно приложения занимает четверть экрана и расположено в левом верхнем углу. Создать временное окно такого же размера в правом нижнем углу экрана. Любое окно после нажатия левой клавиши мыши перемещается в свободный по ходу часовой стрелки угол.

19. Окно размером в четверть площади экрана расположено в центре экрана. После нажатия левой клавиши мыши окно несколько раз меняет подсветку и перемещается в угол экрана так, что курсор мыши оказывается за пределами окна.

20. При запуске не первого экземпляра приложения выдать предупреждающий звуковой сигнал и сообщить о количестве уже работающих копий этого приложения. Запустить экземпляр, только если согласен пользователь.

21. В рабочей области окна приложения рядом друг с другом расположить 4 временных окна, в заголовках которых указан номер окна. После нажатия левой клавиши мыши временное окно выдает сообщение, содержащее номер окна.

22. Правый верхний угол рабочей области окна приложения занимает временное окно размером в четверть этой области. После нажатия правой клавиши мыши над рабочей областью окна приложения временное окно сворачивается в пиктограмму в левом нижнем углу, а после нажатия левой – распахивается в правом верхнем углу рабочей области.

23. В углах рабочей области окна приложения созданы невидимые временные окна с заголовком. Каждое окно становится видимым после нажатия левой клавиши мыши над его частью рабочей области и становится невидимым после нажатия левой клавиши мыши над его рабочей областью.

24. Окно приложения размером в четверть площади экрана занимает один из углов экрана. После нажатия левой клавиши мыши окно сворачивается в пиктограмму. После щелчка по пиктограмме оно восстанавливается в другом углу экрана.

25. При запуске приложения показать окна уже существующих копий этого приложения и спросить пользователя, нужно ли запустить еще один экземпляр. Если пользователь ответит "Да", то запустить его. Иначе завершить работу приложения.

Глава 2
Органы управления

В гл. 1 рассматривалось понятие "окно". Окно может принадлежать только к зарегистрированному классу, и его необходимо создать и отобразить. После этого оно способно показывать свои возможности. Общение пользователя с приложением не ограничивается созерцанием отображаемых элементов и ответами в форме нажатия на одну из кнопок. Чаще всего обмен информацией между пользователем и приложением происходит в более сложной форме. Операционная система сама описывает несколько классов окон. Эти окна позволяют настраивать интерфейс в соответствии с потребностями решаемой задачи и возможностями пользователя. Такие классы называют **органами управления**. Это хорошо знакомые **кнопки, статические органы, списки, редакторы, комбинированные списки, полосы прокрутки**.

Наиболее часто и по особым правилам органы управления используют в составе диалоговых панелей. Диалоговым панелям посвящена 6-я глава. Целью этой главы является изучение особенностей описываемых самой операционной системой классов окон. Поэтому целесообразно их рассматривать вне зависимости от диалоговых панелей, как классы обычных окон.

Отличительные черты органов управления:

- для них уже описаны классы окон;
- все они дочерние окна (стиля WS_CHILD);
- для них описаны дополнительные стили и списки обрабатываемых и получаемых сообщений;
- для них почти всегда нужно описывать идентификаторы.

Родительское окно от дочерних окон получает сообщения. Полученное сообщение содержит идентификатор дочернего окна – отправителя сообщения. И функция родительского окна всегда сможет определить это дочернее окно.

Список имен изучаемых в данной главе классов и их сообщения перечислены в табл. 2.1.

В этой главе рассматриваются такие органы управления, как кнопки, статические органы, списки, редакторы, комбинированные списки и полосы прокрутки. Кроме них операционная система описывает классы **панель инструментов** и **строка состояния**. Панель инструментов предназначена для дублирования команд, а строка состояния выдает контекстные сообщения. Эти окна будут рассмотрены в 5-й главе.

2.1. Кнопки

2.1.1. Создание кнопок

Для создания кнопки можно вызвать хорошо знакомую функцию CreateWindow.

Вспомним синтаксис этой функции:

```
HWND CreateWindow(
    LPCTSTR   lpClassName,   //Указатель на имя класса
    LPCTSTR   lpWindowName,  //Указатель на имя окна
    DWORD     dwStyle,       //Стиль окна
    int       x,             //Координата левого края окна
    int       y,             //Координата верхнего края окна
    int       nWidth,        //Ширина окна
    int       nHeight,       //Высота окна
    HWND      hWndParent,    //Дескриптор окна-родителя или
                             //окна-владельца
    HMENU     hMenu,         //Дескриптор меню или идентифи-
                             //катор дочернего окна
    HANDLE    hInstance,     //Дескриптор экземпляра приложения
    LPVOID    lpParam        //Указатель на данные окна
);
```

Вызов функции CreateWindow для создания кнопки обладает следующими особенностями.

1. В качестве lpClassName указывают имя класса "button".

2. Аргумент lpWindowName определяет строку, которая будет написана на кнопке.

3. Аргумент dwStyle задает стиль кнопки. Его значение задают комбинацией константы WS_CHILD, константы с префиксом BS_ (табл. 2.2) и, почти всегда, константы WS_VISIBLE. Можно указывать **только одну константу** из табл. 2.2. **Исключением** являются константы BS_LEFTTEXT и BS_RIGHTBUTTON. Эти константы используют совместно с другими константами табл. 2.2 при создании флажков и переключателей с текстом, расположенным слева от квадратика флажка или кружка переключателя.

4. Значение hWndParent определяет родительское окно, в рабочей области которого создается кнопка. Дескриптор окна-родителя указывают обязательно.

5. Значение hMenu должно быть равно идентификатору кнопки.

6. Для органов управления в качестве lpParam указывают NULL.

Алгоритм создания кнопки:

1. **Описать идентификатор** кнопки. Например, это можно сделать так:

```
#define ID_BUTTON   3000
```

С помощью этого идентификатора функция окна-родителя будет "узнавать" кнопку. В описании идентификатора можно использовать произвольное целочисленное значение.

2. **Описать дескриптор** дочернего окна для кнопки. Например, в теле функции родительского окна это можно сделать следующим образом:

```
static HWND hButton;
```

3. **Создать кнопку**. Например, следующий оператор в левом верхнем углу рабочей области окна создает обычную кнопку с черной рамкой шириной в 200 и высотой в 20 пикселей с текстом "Пример кнопки":

```
hButton = CreateWindow("button", "Пример кнопки",
        WS_CHILD | WS_VISIBLE | BS_DEFPUSHBUTTON,
        0, 0, 200, 20, hwnd, (HMENU)ID_BUTTON,
        hInstance, NULL);
```

Листинг 2.1 приложения демонстрирует реализацию данного алгоритма.

Листинг 2.1. Пример создания кнопки.

```
#include <windows.h>

#define ID_BUTTON   3000

BOOL  RegClass(WNDPROC,LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char szClassName[ ] = "Example21";

int WINAPI WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  hInstance= hInst;
    if(!RegClass(WndProc, szClassName,COLOR_WINDOW))
        return FALSE;
    hwnd = CreateWindow(szClassName, "Приложение с кнопкой ",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT,  CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    ShowWindow( hwnd, SW_SHOWMAXIMIZED );   UpdateWindow( hwnd );
    while ( GetMessage(&msg,0,0,0) ) DispatchMessage(&msg);
```

```
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc;   wc.style = wc.cbClsExtra = wc.cbWndExtra = 0;
    wc.lpfnWndProc = Proc; wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = (LPCTSTR)NULL;
    wc.lpszClassName = szName;
    return (RegisterClass(&wc) != 0);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                          WPARAM wParam, LPARAM lParam )
{   static HWND hButton;
    switch (msg)
    {   case WM_CREATE:
        {   hButton = CreateWindow("button", "Пример кнопки",
                WS_CHILD | WS_VISIBLE |
                BS_DEFPUSHBUTTON, 0, 0, 200, 20, hwnd,
                (HMENU)ID_BUTTON, hInstance, NULL);
            return 0;
        }
        case WM_DESTROY: { PostQuitMessage(0); return 0; }
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Функция родительского окна будет получать от кнопки hButton сообщения с кодом WM_COMMAND. Этим сообщением кнопка информирует родительское окно о том, что с ней что-то сделали. Но описание функции окна-родителя пока не содержит обработку сообщения WM_COMMAND.

2.1.2. Кнопки и сообщения

До обсуждения сообщений, получаемых функцией окна, вспомним заголовок этой функции:

```
LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                          WPARAM wParam, LPARAM lParam )
```

Пусть функция родительского окна от кнопки получила сообщение WM_COMMAND. Параметр msg принимает код сообщения, т. е. msg= WM_COMMAND. При этом младшее слово параметра wParam (значение LOWORD(wParam)) содержит идентификатор кнопки. В предыдущем примере LOWORD(wParam)=ID_BUTTON. Старшее слово этого параметра равно коду извещения. По нему судят о совершенном над кнопкой действии. Например, при нажатии на кнопку код равен BN_CLICKED, т. е. HIWORD(wParam)=BN_CLICKED. Параметр lParam содержит дескриптор окна кнопки. В предыдущем примере lParam= (HWND)hButton.

Не все кнопки посылают сообщения. Например, кнопка стиля BS_GROUPBOX не обрабатывает сообщения от мыши или клавиатуры и не посылает сообщения родительскому окну. Ее используют в качестве объединяющей рамки с заголовком (рис. 2.1), внутри которой располагают другие органы управления. Пример такой группы рассматривается в нижеследующем примере (листинг 2.2).

Пользователь состоянием кнопок управляет с помощью мыши и клавиатуры. Приложения для изменения координат кнопок вызывают обычные функции перемещения и изменения расположения окон. А для управления состоянием кнопок передают сообщения.

Существует **два способа передачи сообщений** окнам.

1. **Запись сообщения в очередь** приложения. С этой целью вызывают функцию PostMessage:

```
BOOL PostMessage( HWND hwnd, UINT uMsg,
                  WPARAM wParam, LPARAM lParam );
```

Она помещает сообщение в очередь сообщений окна hwnd и возвращает управление. В случае успешной записи сообщения возвращаемое значение равно TRUE. Иначе – FALSE. Записанное сообщение будет выбрано в цикле обработки сообщений. Параметр uMsg содержит код, а wParam и lParam содержат параметры передаваемого сообщения.

2. **Непосредственная передача сообщения** функции окна. Для этого вызывают функцию SendMessage:

```
LRESULT WINAPI SendMessage(HWND hwnd, UINT uMsg,
                  WPARAM wParam, LPARAM lParam);
```

Параметры этой функции используются аналогично параметрам функции PostMessage. Но в отличие от нее функция SendMessage вызывает функцию окна и возвращает управление только после обработки сообщения функцией окна. Возвращаемое значение зависит от обработчика сообщения в функции окна.

Рассмотрим **особенности сообщений для кнопок**

Кнопки обычно находятся в двух состояниях (переключатели и флажки рассмотрим ниже) – нажатом или отжатом. Для изменения состояния кнопки передают сообщение BM_SETSTATE.

Например, для перевода кнопки hButton в нажатое состояние функции окна hButton передают сообщение BM_SETSTATE с параметром wParam, равным TRUE, и lParam, равным 0:

```
SendMessage(hButton, BM_SETSTATE, TRUE, 0L);
```

Кнопка стиля BS_PUSHBUTTON или BS_DEFPUSHBUTTON при нажатии автоматически "уходит вглубь", т. е. перерисовывается.

Для перевода кнопки в отжатое состояние передают сообщение BM_SETSTATE с параметром wParam, равным FALSE:

```
SendMessage(hButton, BM_SETSTATE, FALSE, 0L);
```

Чтобы узнать состояние кнопки, ее функции окна посылают сообщение BM_GETCHECK. В следующем фрагменте переменная nState примет значение кода состояния кнопки hButton:

```
WORD nState=(WORD)SendMessage(hButton, BM_GETCHECK, 0, 0L);
```

Параметры wParam и lParam сообщения BM_GETCHECK равны 0. Возвращаемое значение равно 0 для отжатой кнопки, выключенного переключателя или флажка, 1 – для нажатой кнопки, включенного переключателя или флажка и 2 – для переключателя или флажка в неактивном состоянии и отображается серым цветом.

Переключатели и флажки стилей BS_3STATE, BS_CHECKBOX и BS_RADIOBUTTON не перерисовываются при их переключении. Для их перерисовки нужно послать сообщение BM_SETCHECK.

Параметр wParam сообщения BM_SETCHECK задает новое состояние. В следующей таблице перечислены значения параметра wParam этого сообщения и устанавливаемые состояния переключателей и флажков:

wParam	*Состояние переключателя или флажка*
0	Выключенное (прямоугольник не перечеркнут, в кружке нет точки)
1	Включенное (прямоугольник перечеркнут, в кружке имеется точка)
2	Неактивное. Переключатель или флажок будет изображен серым цветом

Параметр lParam сообщения BM_SETCHECK равен 0.

Например, следующий оператор переводит переключатель или флажок с дескриптором hButton во включенное состояние:

```
SendMessage( hButton, BM_SETCHECK, 1, 0L );
```

Следующее приложение демонстрирует примеры обработки вышеперечисленных сообщений.

Листинг 2.2. Обработка сообщений от кнопок и управление кнопками.

```
#include <windows.h>
#define ID_BUTTON1  3001
#define ID_BUTTON2  3002

BOOL RegClass(WNDPROC,LPCTSTR, UINT);
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char    szClassName[ ] = "Example22";

int WINAPI WinMain(HINSTANCE hInst,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  hInstance= hInst;
    if ( !RegClass(WndProc, szClassName,COLOR_WINDOW) )
        return FALSE;
    hwnd = CreateWindow(szClassName, "Сообщения и кнопки",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg,0,0,0)) DispatchMessage(&msg);
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc;   wc.style=wc.cbClsExtra=wc.cbWndExtra=0;
    wc.lpfnWndProc = Proc;     wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = (LPCTSTR)NULL;
    wc.lpszClassName = szName;
    return (RegisterClass(&wc) != 0);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam )
{   static HWND hButton1, hButton2;
    switch (msg)
```

```
{   case WM_CREATE:
    {   CreateWindow("button", "Группа из двух кнопок",
            WS_CHILD | WS_VISIBLE | BS_GROUPBOX,
            130, 75, 190, 120, hwnd,
            (HMENU)0, hInstance, NULL);
        hButton1 = CreateWindow("button", "Верхняя кнопка",
            WS_CHILD | WS_VISIBLE |
            BS_DEFPUSHBUTTON,
            150, 100, 150, 30, hwnd,
            (HMENU)ID_BUTTON1, hInstance, NULL);
        hButton2 = CreateWindow("button", "Нижняя кнопка",
            WS_CHILD | WS_VISIBLE |
            BS_PUSHBUTTON,
            150, 150, 150, 30, hwnd,
            (HMENU)ID_BUTTON2, hInstance, NULL);
        return 0;
    }
    case WM_COMMAND:
    {   switch (LOWORD(wParam))
        {   case ID_BUTTON1:
            {   MessageBox(hwnd,
                    "Нажата верхняя кнопка",
                    "Сообщение от кнопки",MB_OK);
                SendMessage((HWND)lParam,
                    BM_SETSTATE,TRUE,0L);
                if ( !SendMessage(hButton2,
                    BM_GETCHECK,0,0L) )
                    SendMessage(hButton2,
                        BM_SETSTATE,FALSE,0L);
                return 0;
            }
            case ID_BUTTON2:
            {   MessageBox(hwnd,
                    "Нажата нижняя кнопка",
                    "Сообщение от кнопки",MB_OK);
                SendMessage((HWND)lParam,
                    BM_SETSTATE,TRUE,0L);
                if ( !SendMessage(hButton1,
                    BM_GETCHECK,0,0L) )
                    SendMessage(hButton1,
                        BM_SETSTATE,FALSE,0L);
```

```
                return 0;
            }
        }
        return 0;
    }
    case WM_DESTROY: { PostQuitMessage(0); return 0; }
}
return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

В этом примере создают 3 кнопки:

- прямоугольник, группирующий две обычные кнопки;
- кнопка с надписью "Верхняя кнопка";
- кнопка с надписью "Нижняя кнопка".

При запуске приложения будет создано окно приложения с заголовком "Сообщения и кнопки". В точке с координатами (130, 75) располагается левый верхний угол кнопки стиля BS_GROUPBOX размерами (190*120) пикселей и с заголовком "Группа из двух кнопок":

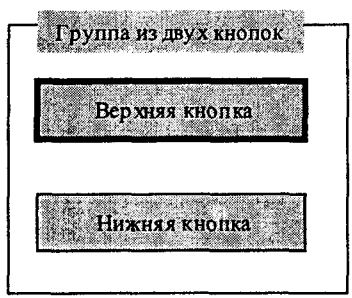

Рис. 2.1. Пример группы из двух кнопок

Если, например, расположить курсор мыши над нижней кнопкой и нажать левую клавишу мыши, то функция окна получит сообщение WM_COMMAND с параметром LOWORD(wParam)=ID_BUTTON2.

При обработке этого сообщения приложение выдает сообщение о том, что нажата нижняя кнопка:

```
MessageBox(hwnd,"Нажата нижняя кнопка", "Сообщение от кнопки", MB_OK);
```

Вспомним, что параметр lParam равен дескриптору окна, которое послало сообщение WM_COMMAND. Окну, на который указывает параметр lParam (в данном случае hButton2), посылаем сообщение:

```
SendMessage( (HWND)lParam, BM_SETSTATE, TRUE, 0L );
```

То есть, устанавливаем кнопку hButton2 в нажатое состояние.

Далее анализируем состояние кнопки hButton1:

```
SendMessage( hButton1, BM_GETCHECK, 0, 0L )
```

и если она находится в нажатом состоянии, то переводим ее в отжатое состояние:

```
SendMessage( hButton1, BM_SETSTATE, FALSE, 0L );
```

Приложение точно так же обрабатывает сообщение о нажатии верхней кнопки.

2.1.3. Флажки и переключатели

Важной разновидностью кнопок являются флажки и переключатели. Их используют для выбора режима работы приложения.

В приложениях переключатели стилей BS_RADIOBUTTON и BS_AUTO RADIOBUTTON используют аналогично кнопкам переключения диапазонов в радиоприемнике. В одной группе располагают несколько таких "радиопереключателей", причем включенным может быть только один из них. Такие переключатели называют **переключателями с зависимой фиксацией**. Включение одного переключателя в группе вызывает выключение остальных.

Флажки BS_CHECKBOX, BS_AUTOCHECKBOX, BS_3STATE, BS_AUTO3STATE используют как отдельные независимые флажки. В группе флажков одновременно могут быть включены несколько флажков.

С переключателем стиля BS_AUTORADIOBUTTON и флажком стиля BS_AUTOCHECKBOX работают так же, как и с кнопками стиля BS_PUSH BUTTON или BS_DEFPUSHBUTTON. При нажатии левой клавишей мыши такой переключатель или флажок автоматически изменяет свое состояние. При этом неперечеркнутый квадратик становится перечеркнутым и, наоборот, перечеркнутый квадратик становится неперечеркнутым. Состояние переключателя отмечается жирной точкой. При изменении состояния флажка или переключателя родительское окно получает сообщение WM_COMMAND с кодом извещения BN_CLICKED.

Флажок стиля BS_3STATE или BS_AUTO3STATE внешне похож на флажок стиля BS_CHECKBOX, но дополнительно имеет третье состояние. В этом третьем состоянии он изображается серым цветом и может использоваться, например, для индикации недоступного для установки параметра.

Слово AUTO в названии стиля используют для обозначения режима автоматической перерисовки переключателя при изменении его состояния.

2.2. Статический орган управления

Статический орган управления -- это окно класса "static". Он не посылает родительскому окну сообщение WM_COMMAND. Этот орган управления используют для оформления внешнего вида родительского окна. Статический орган управления имеет вид закрашенного или незакрашенного прямоугольника или строк текста в заданном прямоугольнике (табл. 2.3). Статические органы управления могут использоваться внутри диалоговых панелей для отображения пиктограмм.

Для создания статического органа управления вызывают функцию CreateWindow. В качестве первого параметра указывают "static".

Следующий оператор демонстрирует создание статического органа управления:

```
HWND hStatic = CreateWindow("static", NULL,
                    WS_CHILD | WS_VISIBLE | SS_BLACKFRAME,
                    20, 40, 100, 50, hwnd, (HMENU)0, hInstance, NULL);
```

Второй параметр определяет текст, который будет расположен внутри органа управления. Если текст не используется, этот параметр указывают как NULL.

В третьем параметре указывают один из стилей статического органа управления. В примере указан стиль SS_BLACKFRAME, т. е. статический орган представляет собой прямоугольную рамку системного цвета COLOR_ WINDOWFRAME ("черного" цвета рамок окон). Внутренняя область прямоугольника остается незакрашенной. Текст заголовка окна при этом стиле статического органа не используют. Соответствующий параметр функции CreateWindow указан как NULL.

Так как статический орган управления не посылает сообщения родительскому окну, в качестве девятого параметра (идентификатор органа управления) можно указать любое число, например 0.

Стили статического органа управления определяют внешний вид и применение органа (табл. 2.3). Статические органы управления удобно использовать для вывода текста.

2.3. Полоса прокрутки

2.3.1. Общие сведения

Полосы прокрутки широко используют для просмотра содержимого документа, которое не помещается в окне. Такие полосы бывают только горизонтальные или вертикальные. Они располагаются соответственно в нижней и правой части окна. Понятие **полоса прокрутки** не ограничивается полосой просмотра. Оно в общем случае представляет собой

множество окон класса **scrollbar** различного назначения (табл. 2.4). Наиболее сложную разновидность окон класса "scrollbar" представляют полосы просмотра.

Полоса просмотра состоит из нескольких объектов, имеющих различное назначение (рис. 2.2).

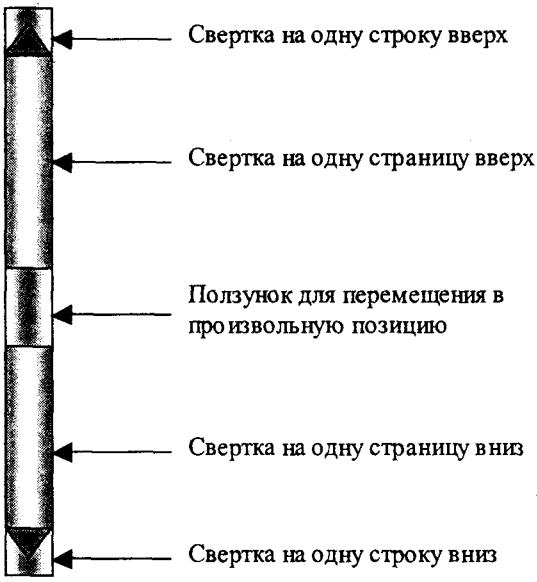

Рис. 2.2. Вертикальная полоса просмотра окна

Представим, что в окне отображен некий документ.

Если установить курсор мыши на верхнюю "кнопку с треугольником" полосы просмотра и нажать левую клавишу мыши, документ в окне сдвигается вниз на одну "строку". Если же аналогичным образом нажать на нижнюю "кнопку с треугольником", документ сдвигается на одну "строку" вверх. Ползунок при этом скачком сдвигается соответственно на одну "позицию" вверх или вниз.

Если установить курсор мыши в область полосы просмотра между ползунком и верхней кнопкой и нажать на левую клавишу мыши, документ сдвигается вниз на одну "страницу". Если щелкнуть левой клавишей мыши в области между ползунком и нижней кнопкой, документ сдвинется на одну "страницу" вверх. Ползунок при этом скачком сдвигается соответственно на одну "страницу" вверх или вниз.

Ползунок можно плавно перемещать мышью вдоль полосы просмотра. В процессе перемещения содержимое окна перемещается или отображается в позиции ползунка.

Таким образом, в зависимости от вида воздействия мыши полоса просмотра может посылать 7 различных сообщений родительскому окну.

Горизонтальная полоса просмотра состоит из тех же объектов, что и вертикальная. Она обеспечивает свертку документа в горизонтальном направлении.

Существуют и менее привычные виды полос прокрутки. Например, это полоса прокрутки стиля SBS_SIZEBOX. Пример такой полосы будет рассмотрен ниже.

2.3.2. Создание полосы прокрутки

Существует **3 способа создания полосы прокрутки**

1. Полосу прокрутки можно создать с помощью вызова функции CreateWindow. Этот способ аналогичен способу, используемому при создании кнопок или статических органов управления. Первым аргументом вызова функции CreateWindow указывают "scrollbar", а вторым – NULL (заголовок окна не используется). **Дескриптор полосы прокрутки используют функции управления полосами прокрутки.** Количество и разновидность полос, создаваемых таким способом, ничем не ограничено.

2. При создании любого окна с помощью функции CreateWindow можно указать, что окно имеет горизонтальную, вертикальную или обе полосы просмотра. Для того чтобы у окна появились вертикальная и/или горизонтальная полосы просмотра, при создании окна в третьем параметре функции CreateWindow указывают стиль WS_VSCROLL и/или WS_HSCROLL. Например:

```
hwnd = CreateWindow(szClassName, szWindowTitle,
       WS_OVERLAPPEDWINDOW | WS_VSCROLL | WS_HSCROLL,
       CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
       CW_USEDEFAULT, 0, 0, hInstance, NULL);
```

3. Некоторые органы управления также могут иметь полосы просмотра.

Внимание! Во втором и третьем способах дескрипторы для полос просмотра не создаются.

2.3.3. Простейшие полосы прокрутки

В качестве примера рассмотрим окно стиля SBS_SIZEBOX. Слово "простейшая" означает, что приложение лишь создает такую полосу, всю работу выполняет функция окна полосы.

Представим, что при создании окна получена полоса прокрутки стиля SBS_SIZEBOX. Например, следующий оператор создает такую полосу размером 16*16 пикселей в точке с координатами (100,100):

```
CreateWindow("scrollbar", NULL,
        WS_CHILD | WS_VISIBLE | SBS_SIZEBOX,
        100, 100, 16, 16, hwnd, (HMENU)0, hInstance, NULL);
```

В точке с координатами (100, 100) рабочей области окна hwnd постоянно будет находиться серый прямоугольник размером 16*16 пикселей. Если курсор мыши поместить над прямоугольником полосы прокрутки, нажать на левую клавишу и перемещать мышь, то нижняя и правая рамки окна приложения повторяют движение курсора. Если отпустить клавишу мыши, то эти рамки останутся в том положении, в котором находились в момент отжатия клавиши.

Полоса прокрутки стиля SBS_SIZEBOX работоспособна без помощи родительского окна. Она обходится без идентификатора и дескриптора окна. Но такая "простота" порождает неудобства. Например, в окне постоянно находится некий серый прямоугольник или его позиция не всегда удовлетворяет потребностям пользователя. Чтобы избежать таких неудобств, описывают дескриптор полосы и с его помощью управляют состоянием прямоугольника. То есть полоса прокрутки подчиняется обычным функциям управления окнами.

Приложение на следующем листинге свободно от этих недостатков.

Листинг 2.3. Полоса прокрутки стиля SBS_SIZEBOX.

```
#include <windows.h>
BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char    szClassName[ ]  = "AppClass";

int WINAPI  WinMain(HINSTANCE hInst,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  hInstance=hInst;
    if ( !RegClass(WndProc, szClassName, COLOR_WINDOW) )
        return FALSE;
    if (!(hwnd = CreateWindow(szClassName, "Прокрутка окна",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL)) )  return FALSE;
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}
```

```
BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
                                        UINT brBackground)
{   WNDCLASS wc;  wc.style=wc.cbClsExtra=wc.cbWndExtra=0;
    wc.lpfnWndProc = Proc;     wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = (LPCTSTR)NULL;
    wc.lpszClassName = szName; return (RegisterClass(&wc) != 0);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam )
{   static HWND hScroll;
    switch (msg)
    {   case WM_CREATE:
        {   hScroll= CreateWindow("scrollbar", NULL,
                WS_CHILD | SBS_SIZEBOX,
                0,0, 16, 16, hwnd, (HMENU)0, hInstance,NULL);
            return 0;
        }
        case WM_SIZE:
        {   ShowWindow(hScroll,SW_HIDE); return 0; }
        case WM_LBUTTONDOWN:
        {   MoveWindow(hScroll,
                LOWORD(lParam)-8, HIWORD(lParam)-8,16,16,
                TRUE);
            ShowWindow(hScroll,SW_NORMAL);
            return 0;
        }
        case WM_DESTROY: {     PostQuitMessage(0); return 0; }
    }   return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

В левом верхнем углу рабочей области окна создана невидимая полоса прокрутки стиля WS_CHILD | SBS_SIZEBOX.

После нажатия левой клавиши мыши на том месте рабочей области, где находится курсор мыши, появится серый прямоугольник размером 16*16 пикселей. С этой целью полоса прокрутки сначала перемещается в точку с координатами (LOWORD(lParam), HIWORD(lParam)):

```
MoveWindow(hScroll,
        LOWORD(lParam)-8, HIWORD(lParam)-8, 16,16,TRUE);
```

Добавка –8 "совмещает" центр прямоугольника с позицией курсора мыши. Затем полоса переводится в видимое состояние:

```
ShowWindow(hScroll,SW_NORMAL);
```

Если курсор мыши поместить над этим серым прямоугольником, нажать на левую клавишу и перемещать курсор мыши, то нижняя и правая рамки окна приложения будут повторять движение курсора. Если отпустить клавишу мыши, эти рамки останутся в том положении, в котором находились в момент отжатия клавиши.

После изменения размеров окна полоса прокрутки переводится в невидимое состояние:

```
ShowWindow(hScroll,SW_HIDE);
```

Так же создают полосу стиля SBS_SIZEGRIP. С ней работают так же, как и с полосой стиля SBS_SIZEBOX. Но она не изменяет размеры родительского окна, а выделяет прямоугольную область в рабочей области.

2.3.4. Сообщения от полосы прокрутки

Горизонтальные полосы прокрутки функции родительского окна посылают сообщение WM_HSCROLL, а вертикальные – сообщение WM_VSCROLL. При этом младшее слово параметра wParam содержит код совершенного действия, старшее слово – текущую позицию ползунка, а lParam указывает на дескриптор полосы прокрутки.

Следующая таблица перечисляет имена кодов действий и пояснения к ним:

Код действия	*Пояснение действия*
SB_ENDSCROLL	Отжата клавиша мыши (конец работы с полосой прокрутки)
SB_LINELEFT	Сдвиг окна вдоль документа влево (вверх) на единицу
SB_LINERIGHT	Сдвиг окна вдоль документа вправо (вниз) на единицу
SB_PAGELEFT	Сдвиг окна вдоль документа влево (вверх) на страницу
SB_PAGERIGHT	Сдвиг окна вдоль документа вправо (вниз) на страницу
SB_THUMBPOSITION	Ползунок остановился после плавного движения. Текущая позиция ползунка равна HIWORD(wParam)

SB_THUMBTRACK	Ползунок плавно движется. Текущая позиция ползунка равна HIWORD(wParam)
SB_TOP	Сдвиг влево (вверх) в начало документа
SB_BOTTOM	Сдвиг вправо (вниз) в конец документа

Сообщения с последними двумя кодами не поступают от полосы прокрутки. Их посылает само приложение при обработке сообщений от клавиатуры.

Внимание! Для сообщений WM_HSCROLL и WM_VSCROLL можно использовать одни и те же коды действий.

2.3.5. Управление полосой прокрутки

Позиция nPos ползунка может принимать значения из заданного диапазона [nMin, nMax]. Самая левая для горизонтальной полосы или самая верхняя для вертикальной полосы позиции ползунка отвечают минимальному значению nMin. Соответственно в самом правом или самом нижнем положении значение позиции равно nMax.

Функции управления состоянием полос прокрутки не вызывают для полос, созданных третьим способом. Если полоса прокрутки создана первым способом, то при вызове этих функций первым аргументом указывают дескриптор полосы. Иначе первым аргументом указывают дескриптор окна, которому принадлежит полоса просмотра. Вторым аргументом всегда указывают вид полосы. Допустимые значения для второго аргумента перечислены в следующей таблице:

Значение	*Вид полосы*
SB_CTL	Полоса прокрутки. Первым аргументом является дескриптор полосы прокрутки
SB_HORZ	Стандартная горизонтальная полоса просмотра окна
SB_VERT	Стандартная вертикальная полоса просмотра окна

Для **установки диапазона полосы прокрутки** вызывают функцию SetScrollRange:

```
BOOL SetScrollRange( HWND hwnd,  int nBar, int nMin, int nMax,  BOOL bRedraw );
```

Она устанавливает минимальное и максимальное значения позиции ползунка заданной полосы прокрутки. В случае успешной установки возвращаемое значение отлично от нуля.

При nMin=nMax полоса исчезает. Такой ситуации нужно избегать при управлении состоянием полос прокрутки. Аргумент bRedraw, если не равен нулю, указывает на необходимость перерисовки полосы. Этот аргумент чаще всего равен нулю. По умолчанию nMin=0 и nMax= 100

для стандартных полос просмотра окон и nMin=nMax=0 – для полос прокрутки, созданных первым способом.

Текущие установки диапазона узнают с помощью функции GetScrollRange:

```
BOOL GetScrollRange(HWND hwnd, int nBar, LPINT lpMinPos, LPINT lpMaxPos );
```

Параметры lpMinPos и lpMaxPos указывают на переменные, в которые будут записаны соответственно минимальное и максимальное значения диапазона полосы. Например, после выполнения следующего оператора переменные nMinPos и nMaxPos будут содержать значения границ диапазона полосы прокрутки hScroll:

```
GetScrollRange( hScroll, SB_CTL, &nMinPos, &nMaxPos );
```

Функция SetScrollPos устанавливает ползунок в заданную позицию:

```
int SetScrollPos(HWND hwnd, int nBar, int nPos, BOOL bRedraw);
```

Параметр nPos задает новое положение ползунка, и его значение должно находиться в пределах установленного диапазона. При bRedraw=TRUE полоса будет перерисована, иначе – нет. Эта функция возвращает значение предыдущей позиции или 0 в случае ошибки.

Для **определения текущей позиции ползунка** вызывают функцию GetScrollPos:

```
int GetScrollPos(HWND hwnd, int nBar);
```

Функция возвращает текущую позицию или 0 в случае ошибки.

С помощью функции ShowScrollBar можно **скрыть или показать полосу прокрутки**:

```
BOOL ShowScrollBar( HWND hwnd, int wBar, BOOL bShow );
```

Параметр wBar определяет вид полосы прокрутки. Кроме перечисленных ранее констант для скрытия обеих полос просмотра окна можно использовать константу SB_BOTH. Если параметр bShow равен FALSE, полоса прокрутки исчезнет. Нельзя вызывать эту функцию для скрытия полосы прокрутки при обработке сообщения от этой полосы.

Для **блокирования кнопок полосы прокрутки** вызывают функцию EnableScrollBar:

```
BOOL EnableScrollBar(HWND hwnd, UINT wSBflags, UINT wArrows);
```

Эта функция возвращает ненулевое значение при успешном завершении или 0 при ошибке (если, например, кнопки уже находятся в требуемом состоянии). Вызов другой функции управления для этой полосы активизирует блокированные кнопки.

Первый и второй аргументы подобны аргументам функции ShowScrollBar. Параметр wArrows определяет воздействие на полосу прокрутки и принимает одно из следующих значений:

Значение	*Пояснение*
ESB_DISABLE_BOTH	Отключить обе кнопки
ESB_DISABLE_DOWN	Отключить кнопку "Вниз" на вертикальной полосе
ESB_DISABLE_LEFT	Отключить кнопку "Влево" на горизонтальной полосе
ESB_DISABLE_LTUP	Отключить кнопку "Влево" на горизонтальной полосе или кнопку "вверх" на вертикальной полосе
ESB_DISABLE_RIGHT	Отключить кнопку "Вправо" на горизонтальной полосе
ESB_DISABLE_RTDN	Отключить кнопку "Вправо" на горизонтальной полосе или кнопку "вниз" на вертикальной полосе
ESB_DISABLE_UP	Отключить кнопку "Вверх" на вертикальной полосе
ESB_ENABLE_BOTH	Включить обе кнопки на полосе

2.3.6. Пример обработки сообщений от полос прокрутки

Задача. В окне приложения создать две полосы прокрутки (горизонтальную и вертикальную) для управления размерами двух прямоугольников (черного по горизонтали и темно-серого по вертикали).

Решение данной задачи полностью происходит в функции окна приложения, представленного листингом 2.4.

Листинг 2.4. Полоса прокрутки – орган управления.

```
#include <windows.h>

BOOL        RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT     CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char szClassName[ ]  = "ScrollClass";
char szTitle[ ] = "Полоса прокрутки – орган управления";

int WINAPI  WinMain(HINSTANCE hInst,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  hInstance=hInst;
    if ( !RegClass(WndProc, szClassName, COLOR_WINDOW) )
        return FALSE;
    if (!(hwnd = CreateWindow(szClassName, szTitle,
```

```
        WS_OVERLAPPEDWINDOW, 194, 50,
        246, 350, 0, 0, hInstance, NULL))) return FALSE;
    ShowWindow(hwnd, SW_SHOWNORMAL );
    UpdateWindow(hwnd);
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
                            UINT brBackground)
{   WNDCLASS wc;   wc.style=wc.cbClsExtra=wc.cbWndExtra=0;
    wc.lpfnWndProc = Proc;    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName=(LPCTSTR)NULL;
    wc.lpszClassName=szName; return (RegisterClass(&wc) != 0);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam,  LPARAM lParam )
{   static HWND hxScroll, hxStatic, hyScroll, hyStatic;
    static int nxPos, nxMin, nxMax, nyPos, nyMin, nyMax;
    switch (msg)
    {   case WM_CREATE:
        {   hxScroll = CreateWindow("scrollbar", NULL,
                WS_CHILD | WS_VISIBLE | SBS_HORZ,
                20, 60, 200, 25, hwnd, (HMENU)-1,
                hInstance, NULL);
            nxPos = 100; nxMin=1; nxMax=200;
            SetScrollRange(hxScroll, SB_CTL,
                            nxMin, nxMax, TRUE);
            SetScrollPos(hxScroll, SB_CTL, nxPos, TRUE);
            hxStatic = CreateWindow("static", NULL,
                WS_CHILD | WS_VISIBLE | SS_BLACKRECT,
                20, 40, nxPos, 15, hwnd, (HMENU)-1,
                hInstance, NULL);
            hyScroll = CreateWindow("scrollbar", NULL,
                WS_CHILD | WS_VISIBLE | SBS_VERT,
                20, 100, 90, 200, hwnd, (HMENU)-1,
                hInstance, NULL);
            nyPos = 100; nyMin=1; nyMax=200;
```

```
        SetScrollRange(hyScroll, SB_CTL, nyMin, nyMax, TRUE);
        SetScrollPos(hyScroll, SB_CTL, nyPos, TRUE);
        hyStatic = CreateWindow("static", NULL,
            WS_CHILD | WS_VISIBLE | SS_GRAYRECT,
            130, 100, 90, nyPos, hwnd, (HMENU)-1,
            hInstance, NULL);
        return 0;
    }
    case WM_HSCROLL:
    {   switch (LOWORD(wParam))
        {   case SB_PAGERIGHT:
            {   nxPos += 10;   break; }
            case SB_LINERIGHT:
            {   nxPos += 1;    break; }
            case SB_PAGELEFT:
            {   nxPos -= 10;   break; }
            case SB_LINELEFT:
            {   nxPos -= 1;          break; }
            case SB_TOP:
            {   nxPos = nxMin;break; }
            case SB_BOTTOM:
            {   nxPos = nxMax;    break; }
            case SB_THUMBPOSITION:
            case SB_THUMBTRACK:
            {   nxPos=HIWORD(wParam); break; }
            default: break;
        }
        if (nxPos > nxMax) nxPos = nxMax;
        if (nxPos < nxMin) nxPos = nxMin;
        SetScrollPos(hxScroll, SB_CTL, nxPos, TRUE);
        if (nxPos==nxMax)
            EnableScrollBar(hxScroll,SB_CTL,
                ESB_DISABLE_RIGHT);
        MoveWindow(hxStatic, 20, 40, nxPos, 15, TRUE);
        return 0;
    }
    case WM_VSCROLL:
    {   switch (LOWORD(wParam))
        {   case SB_PAGERIGHT:
            {   nyPos += 10;   break; }
            case SB_LINERIGHT:
```

```
                { nyPos += 1;    break; }
                case SB_PAGELEFT:
                { nyPos -= 10;   break; }
                case SB_LINELEFT:
                { nyPos -= 1;          break; }
                case SB_TOP:
                { nyPos = nyMin;break; }
                case SB_BOTTOM:
                { nyPos = nyMax;   break; }
                case SB_THUMBPOSITION:
                case SB_THUMBTRACK:
                { nyPos=HIWORD(wParam); break; }
                default: break;
            }
            if (nyPos > nyMax) nyPos = nyMax;
            if (nyPos < nyMin) nyPos = nyMin;
            SetScrollPos(hyScroll, SB_CTL, nyPos, TRUE);
            MoveWindow(hyStatic, 130, 100, 90, nyPos, TRUE);
            return 0;
        }
        case WM_DESTROY: { PostQuitMessage(0); return 0; }
    }   return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

На этапе создания окна приложения создают горизонтальную полосу прокрутки нестандартных размеров и без идентификатора:

```
hxScroll = CreateWindow("scrollbar", NULL,
        WS_CHILD | WS_VISIBLE | SBS_HORZ,
        20,60,200,25, hwnd, (HMENU)-1, hInstance, NULL);
```

Далее устанавливают диапазон перемещения и начальную позицию ползунка:

```
nxPos = 100; nxMin=1; nxMax=200;
SetScrollRange(hxScroll, SB_CTL, nxMin, nxMax, TRUE);
SetScrollPos(hxScroll, SB_CTL, nxPos, TRUE);
```

Затем создают статический орган – закрашенный прямоугольник черного цвета с шириной, равной позиции nxPos ползунка:

```
hxStatic = CreateWindow("static", NULL,
        WS_CHILD | WS_VISIBLE | SS_BLACKRECT,
        20,40,nxPos,15, hwnd, (HMENU)-1,hInstance,NULL);
```

Таким же образом создают вертикальную полосу прокрутки и прямоугольник темно-серого цвета.

После запуска приложения на экране появится окно, в рабочей области которого расположены элементы hxStatic, hxScroll, hyScroll и hyStatic (рис. 2.3).

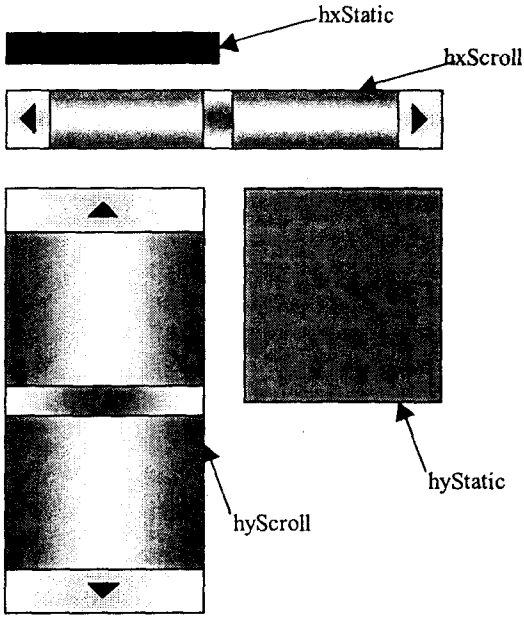

Рис. 2.3. Пример управления полосами прокрутки

Рассмотрим, как работает горизонтальная полоса прокрутки.

При нажатии на кнопку на левом конце полосы поступает сообщение WM_HSCROLL с кодом SB_LINELEFT. При обработке этого сообщения значение nxPos уменьшается на единицу: nxPos -= 1.

Далее ползунок устанавливают в новую позицию nxPos:

```
SetScrollPos(hxScroll, SB_CTL, nxPos, TRUE);
```

и перемещают черный прямоугольник – у окна hxScroll изменяют ширину:

```
MoveWindow(hxStatic, 20, 40, nxPos, 15, TRUE);
```

Так же обрабатывают и другие сообщения. Разница лишь в том, что при нажатии на область между кнопкой и ползунком происходит "постраничное" перелистывание – значение nxPos изменяется на 10. При плавном перемещении ползунка поступает сообщение с кодом SB_THUMBTRACK и текущая позиция определяется через старшее слово параметра wParam: nxPos=HIWORD(wParam). После плавной уста-

новки ползунка код действия равен SB_THUMBPOSITION и текущая позиция определяется через старшее слово параметра wParam:

```
nxPos=HIWORD(wParam)
```

В текст включен пример блокирования правой кнопки горизонтальной полосы прокрутки:

```
if (nxPos==nxMax)
        EnableScrollBar(hxScroll,SB_CTL,ESB_DISABLE_RIGHT);
```

Правая кнопка этой полосы блокируется при достижении крайней правой позиции, и нажатие на эту кнопку не вызывает никакой реакции. Вызов другой функции для этой полосы отменяет блокировку.

Параметры nxPos, nxMin и nxMax, а также шаг изменения значения nxPos на "позицию" или на "страницу" разработчик задает, исходя из потребностей решаемой задачи. В примере эти параметры выбраны с целью наглядного изменения размеров статических органов. В других задачах эти параметры могут быть другими. Например, значение "позиции" может быть равно средней ширине или высоте одного символа установленного шрифта при просмотре документов в окне.

Обработка сообщений от вертикальной полосы прокрутки происходит точно так же.

2.3.7. Новые функции управления полосами прокрутки

В Win32 описаны новые функции SetScrollInfo и GetScrollInfo, а также новая структура SCROLLINFO. Они позволяют более гибко управлять полосами прокрутки.

Структура SCROLLINFO содержит параметры полосы прокрутки. Параметры полосы прокрутки устанавливают с помощью функции SetScrollInfo. Для получения значений параметров вызывают функции GetScrollInfo. Структура SCROLLINFO описана следующим образом:

```
typedef struct
{   UINT   cbSize;
    UINT   fMask;
    Int    nMin;
    Int    nMax;
    UINT   nPage;
    Int    nPos;
    Int    nTrackPos;
}   SCROLLINFO;
```

Поле с именем cbSize определяет размер этой структуры.

Поле fMask определяет, значения каких параметров полосы нужно установить или получить. Значение fMask задают в виде комбинации следующих констант:

Константа	Пояснение
SIF_ALL	Эта константа равна комбинации SIF_PAGE, SIF_POS и SIF_RANGE
SIF_ DISABLENOSCROLL	Если устанавливаемые параметры делают полосу ненужной, отключить полосу, но не удалять
SIF_PAGE	Установить или получить nPage
SIF_POS	Установить или получить nPos
SIF_RANGE	Установить или получить nMin и nMax

Поля nMin и nMax задают диапазон, nPage – размер ползунка, nPos – текущую позицию ползунка. Поле nTrackPos содержит абсолютную позицию ползунка после плавного перемещения. Приложение не может устанавливать значение этого поля, но может выбрать его значение (например, при обработке сообщения SB_THUMBTRACK). Значение поля nPage должно находиться в пределах от 0 до nMax – nMin + 1, а значение nPos – между nMin и nMax.

Функция SetScrollInfo **устанавливает указанные параметры полосы прокрутки** и, при необходимости, перерисовывает полосу:

```
int SetScrollInfo( HWND hwnd, int fnBar, LPSCROLLINFO lpsi, BOOL fRedraw );
```

Параметры hwnd и fnBar несут тот же смысл, что и для других функций управления состоянием полос. Параметр lpsi указывает на структуру типа SCROLLINFO, а значение fRedraw определяет необходимость перерисовки полосы. Возвращаемое значение равно текущей позиции ползунка.

Функция GetScrollInfo **выбирает указанные параметры полосы прокрутки**:

```
BOOL GetScrollInfo (HWND hwnd, int fnBar, LPSCROLLINFO lpsi );
```

Смысл параметров тот же, что прежде. Эта функция записывает параметры полосы прокрутки, определяемые полем fMask, в соответствующие поля структуры, на которую указывает lpsi. В случае успешной выборки функция возвращает ненулевое значение.

2.3.8. Пример окна приложения с полосой пр осмотра

Задача. Создать окно приложения с вертикальной полосой просмотра. Обеспечить обработку команд перелистывания от клавиатуры. Длину ползунка увеличивать на единицу при построчном увеличении позиции ползунка и уменьшать на единицу при построчном уменьшении позиции ползунка.

Листинг 2.5 описывает решение этой задачи.

Листинг 2.5. Новые функции управления полосой просмотра.

```
#include <windows.h>

BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char szClassName[ ]  = "ScrollClass";
char szTitle[ ] = "Окно с полосой просмотра";
int WINAPI  WinMain(HINSTANCE hInst,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  hInstance=hInst;
    if ( !RegClass(WndProc, szClassName, COLOR_WINDOW) )
        return FALSE;
    if (!(hwnd = CreateWindow(szClassName, szTitle,
      WS_OVERLAPPEDWINDOW | WS_VSCROLL | WS_VISIBLE,
      194, 50, 246, 350, 0, 0, hInstance, NULL))) return FALSE;
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}
BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc; wc.style=wc.cbClsExtra=wc.cbWndExtra=0;
    wc.lpfnWndProc = Proc;    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName=(LPCTSTR)NULL;
    wc.lpszClassName=szName;   return (RegisterClass(&wc) != 0);
}
LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam )
{   static unsigned int nyPos, nyMin, nyMax, nyPage;
    static SCROLLINFO siy;
    switch (msg)
    {   case WM_CREATE:
        {   nyPos = nyMin=1; nyMax=200;  nyPage=10;
            siy.cbSize=sizeof(siy);  siy.fMask=SIF_ALL;
            siy.nMin=nyMin;          siy.nMax=nyMax;
            siy.nPage=nyPage;        siy.nPos=nyPos;
            SetScrollInfo(hwnd, SB_VERT, &siy, TRUE);
            return 0;
        }
```

```
case WM_VSCROLL:
{   switch (LOWORD(wParam))
    {   case SB_PAGERIGHT:
        {   nyPos += 10;   break; }
        case SB_LINERIGHT:
        {   nyPos += 1;  siy.nPage +=1; break; }
        case SB_PAGELEFT:
        {   nyPos -= 10;   break; }
        case SB_LINELEFT:
        {   nyPos -= 1;   siy.nPage -=1; break; }
        case SB_TOP:
        {   nyPos =nyMin; break; }
        case SB_BOTTOM:
        {   nyPos =nyMax;break; }
        case SB_THUMBPOSITION:
        case SB_THUMBTRACK:
        {   nyPos=HIWORD(wParam); break; }
    }
    if (nyPos > nyMax) nyPos = nyMax;
    if (nyPos < nyMin) nyPos = nyMin;
    siy.nPos=nyPos; siy.fMask=SIF_POS | SIF_PAGE;
    if (siy.nPage<nyPage || siy.nPage>nyMax)
        siy.nPage=nyPage;
    SetScrollInfo(hwnd, SB_VERT, &siy, TRUE);
    return 0;
}
case WM_KEYDOWN:
{   switch (wParam)
    {   case VK_HOME:
            if ( HIBYTE( GetKeyState(VK_CONTROL)) )
                SendMessage(hwnd,
                    WM_VSCROLL, SB_TOP, 0L);
            break;
        case VK_END:
            if ( HIBYTE( GetKeyState(VK_CONTROL)) )
                SendMessage(hwnd,
                    WM_VSCROLL, SB_BOTTOM,0L);
            break;
        case VK_UP:
            SendMessage(hwnd,
                    WM_VSCROLL, SB_LINELEFT, 0L);
```

```
                break;
            case VK_DOWN:
                SendMessage(hwnd,WM_VSCROLL, SB_LINERIGHT, 0L);
                break;
            case VK_PRIOR:
                SendMessage(hwnd, WM_VSCROLL, SB_PAGELEFT, 0L);
                break;
            case VK_NEXT:
                SendMessage(hwnd,WM_VSCROLL, SB_PAGERIGHT,0L);
                break;
        }
        return 0;
    }
    case WM_DESTROY: { PostQuitMessage(0); return 0; }
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Как видно из листинга, для создания полосы просмотра достаточно дополнить стиль окна стилем WS_VSCROLL. А дальнейшие действия по обработке сообщений от полосы такие же, как и при создании полосы прокрутки первым способом.

Появились следующие отличия.

1. Активно применяется функция SetScrollInfo. При инициализации полосы просмотра эта функция используется со значением поля fMask=SIF_ALL:

```
siy.fMask=SIF_ALL;
siy.nMin=nyMin;        siy.nMax=nyMax;  siy.nPage=nyPage; siy.nPos=nyPos;
SetScrollInfo(hwnd, SB_VERT, &siy, TRUE);
```

Это говорит о том, что устанавливают все параметры полосы. А значение последнего аргумента TRUE сообщает о необходимости перерисовки полосы. Сравнение с предыдущим примером показывает, что функция SetScrollInfo заменяет действия двух функций: установки диапазона SetScrollRange и текущей позиции SetScrollPos ползунка.

2. При обработке сообщения WM_VSCROLL та же функция, SetScrollInfo используется для установки только текущей позиции и длины ползунка:

```
siy.fMask=SIF_POS | SIF_PAGE;
siy.nPos=nyPos;   //Значение siy.nPage вычислено ранее
SetScrollInfo(hwnd, SB_VERT, &siy, TRUE);
```

Об этом сообщает значение поля fMask.

3. Подключена обработка сообщений от клавиатуры.

Это отличие рассмотрим более подробно.

При нажатии любой клавиши на клавиатуре функция окна получает сообщение WM_KEYDOWN с кодом wParam нажатой клавиши. В примере рассматриваются коды тех клавиш, которые связаны с перелистыванием в вертикальном направлении. Это Home (VK_HOME), End (VK_END), "Вверх" (VK_UP), "Вниз" (VK_DOWN), Page Up (VK_PRIOR) и Page Down (VK_NEXT).

Причем клавиши Home и End в вертикальном направлении перелистывают только в сочетании с правой или левой клавишей Ctrl. Поэтому при нажатии клавиши, например, Home проверяется состояние клавиши Ctrl:

```
if ( HIBYTE( GetKeyState(VK_CONTROL) ) ) ...
```

При вызове функции GetKeyState(VK_CONTROL) возвращается число типа int. Если значение старшего бита этого числа равно нулю, то указанная аргументом вызова клавиша не нажата. То есть если нажата клавиша Home или End, то проверяется состояние клавиш Ctrl. Если любая из них нажата, то считаем, что нужно установить ползунок в соответствующую позицию.

Например, при нажатии Ctrl+Home ползунок устанавливаем в верхнем положении. Для этого полосе просмотра окна hwnd посылаем сообщение SB_TOP:

```
SendMessage(hwnd, WM_VSCROLL, SB_TOP, 0L);
```

Так же обрабатывают сообщения от других клавиш клавиатуры. Только для них не проверяют состояния вспомогательных клавиш.

2.4. Редактор текста

2.4.1. Создание редактора

На базе класса с именем "edit" создают однострочный или многострочный текстовый редактор. Его используют для ввода значений строковых или числовых переменных, а также для создания и редактирования текстовых файлов. Этот редактор умеет выполнять функции выделения текста, работать с буфером обмена Clipboard и многое другое.

Для создания редактора вызывают функцию CreateWindow. Первым аргументом указывают имя класса "edit".

Кроме обычных стилей окна для текстового редактора указывают стили с символическими именами с префиксом ES_ (табл. 2.5). Они влияют на внешний вид редактора и выполняемые им функции.

Пример создания редактора текста:

```
#define ID_EDIT    2222
...
HWND hEdit = CreateWindow("edit", "Исходный текст",
    WS_CHILD | WS_VISIBLE | WS_BORDER | ES_LEFT,
    30, 30, 300, 30, hwnd, (HMENU) ID_EDIT, hInstance, NULL);
```

Второй аргумент здесь передает строковую константу "Исходный текст". Эта строка будет отображена в окне редактора сразу после его создания. Второй аргумент чаще указывает на NULL.

Стиль окна редактора содержит константу WS_BORDER. Поэтому вокруг окна редактора будет нарисована рамка. Другая константа, ES_LEFT устанавливает режим выравнивания содержимого редактора по левому краю. Эта же константа определяет, что нужно создать однострочный редактор. Остальные аргументы функции CreateWindow в этом примере такие же, как и для других органов управления.

2.4.2. Сообщения для редактора текста

Сообщения функции окна редактора посылают с помощью функции SendMessage. Коды сообщений для функции окна текстового редактора имеют имена с префиксом EM_. В нижеследующей таблице перечислены наиболее часто используемые коды сообщений и их параметры:

Код сообщения и его параметры	*Назначение*	*Возвращаемое значение*
EM_GETFIRSTVISIBLELINEw Param = lParam = 0;	Получение номера самой верхней видимой строки	Номер строки
EM_GETHANDLE wParam = lParam = 0;	Получение дескриптора блока памяти, хранящего редактируемый текст	Дескриптор блока памяти
EM_GETLINE wParam=(WPARAM)line – номер строки; lParam = (LPARAM)(LPCSTR)lpch – адрес буфера	Копирование строки из редактора текста в буфер	Количество копированных символов или 0
EM_GETLINECOUNT wParam = lParam = 0;	Получение количества строк в тексте	Количество строк или 1
EM_GETRECT, wParam=0; lParam= (LPARAM)(LPRECT) lprc;	Получение координат прямоугольной области редактора	Не используется

EM_LIMITTEXT wParam=(WPARAM)cchMax – длина текста в символах; lParam = 0;	Ограничение количества символов текста в окне. Выполняют сразу после создания редактора	Не используется
EM_SETHANDLE wParam=(WPARAM) (HLOCAL)hloc – дескриптор; lParam=0;	Установка дескриптора блока памяти для хранения редактируемого текста	"
EM_SETREADONLY wParam=(WPARAM) (BOOL)fR, lParam=0; fR=1 – установка, fR=0 – сброс.	Установка или сброс режима "только чтение"	TRUE или FALSE при ошибке
EM_SETRECT, wParam=0; lParam=(LPARAM) (LPRECT); lprc – адрес нового прямоугольника	Изменение прямоугольника редактирования текста	Не используется

2.4.3. Сообщения от редактора текста

Функция окна редактора текста посылает функции родительского окна сообщение WM_COMMAND. При этом младшее слово параметра wParam содержит идентификатор, а параметр lParam – дескриптор окна редактора текста. Старшее слово параметра wParam содержит код извещения о совершенном действии. В следующей таблице перечислены коды извещений:

Код	*Пояснение*
EN_CHANGE	Текст в окне редактирования изменился
EN_ERRSPACE	Недостаточно памяти для выполнения действия
EN_HSCROLL	Нажата горизонтальная полоса просмотра редактора, но свертка текста еще не произошла
EN_KILLFOCUS	Текстовый редактор потерял фокус ввода
EN_MAXTEXT	Превышен заданный для редактора размер текста
EN_SETFOCUS	Текстовый редактор получил фокус ввода
EN_UPDATE	Последняя операция редактирования выполнена, но еще не отразилась на содержимом редактора. За этим извещением после отображения изменений придет извещение с кодом EN_CHANGE
EN_VSCROLL	Нажата вертикальная полоса просмотра редактора, но свертка текста еще не произошла

Приложения обрабатывают, по крайней мере, извещение с кодом EN_ERRSPACE.

2.4.4. Пример работы с однострочным редактором

Задача. Создать однострочный редактор текста и кнопку. При нажатии на кнопку вывести окно сообщений с текстом, который содержится в окне редактора.

Ниже приводится листинг приложения, которое решает данную задачу.

Листинг 2.6. Получение содержимого однострочного редактора.

```
#include <windows.h>

#define    ID_EDIT      1001
#define    ID_BUTTON    1002
BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char    szClassName[ ] = "EditClass";
char    szTitle[ ] = "Однострочный редактор";

int WINAPI  WinMain(HINSTANCE hInst,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  hInstance=hInst;
    if ( !RegClass(WndProc, szClassName, COLOR_WINDOW) )
        return FALSE;
    if ( !(hwnd = CreateWindow(szClassName, szTitle,
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        100, 50, 364, 150, 0, 0, hInstance, NULL)) ) return FALSE;
    while(GetMessage(&msg, 0, 0, 0))
    {   TranslateMessage(&msg); DispatchMessage(&msg); }
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
                            UINT brBackground)
{   WNDCLASS wc; wc.style=wc.cbClsExtra=wc.cbWndExtra=0;
    wc.lpfnWndProc = Proc;    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName=(LPCTSTR)NULL;
    wc.lpszClassName=szName; return (RegisterClass(&wc) != 0);
}
```

```
LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                          wParam,  LPARAM lParam )
{   static HWND hEdit, hButton, hStatic;     char chBuff[80];
    switch (msg)
    {   case WM_CREATE:
        {   hStatic=CreateWindow("static",
                "Введите строку и нажмите кнопку 'OK'",
                WS_CHILD | WS_VISIBLE | SS_CENTER,
                30,10,300,20,hwnd,(HMENU)0,hInstance,NULL);
            hEdit = CreateWindow("edit", NULL,
                WS_CHILD | WS_VISIBLE | WS_BORDER |
                ES_LEFT, 30, 40, 300, 30, \
                hwnd,(HMENU)ID_EDIT,hInstance, NULL);
            hButton = CreateWindow("button", "OK",
                WS_CHILD | WS_VISIBLE |
                BS_PUSHBUTTON,130, 85, 100, 30,
                hwnd,(HMENU)ID_BUTTON,hInstance, NULL);
            SetFocus(hEdit);    return 0;
        }
        case WM_COMMAND:
        {   switch (LOWORD(wParam))
            {   case ID_EDIT:
                {   if ( (HIWORD(wParam)) ==
                        EN_ERRSPACE )
                        MessageBox(hwnd,
                          "Мало памяти",szTitle,MB_OK);
                    return 0;
                }
                case ID_BUTTON:
                {   SendMessage(hEdit,EM_GETLINE,
                        0,(LPARAM)chBuff);
                    MessageBox(hwnd, chBuff, szTitle,
                        MB_OK);
                    SetWindowText(hEdit,"\0");
                    SetFocus(hEdit);    return 0;
                }
            }
            return 0;
        }
        case WM_DESTROY: { PostQuitMessage(0); return 0; }
```

```
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Это приложение в своем главном окне создает статический орган (который представляет собой центрированный по горизонтали текст "Введите строку и нажмите кнопку 'OK'"), под ним – однострочный редактор текста и, еще ниже, кнопку с надписью OK (рис. 2.4).

Если ввести текст и нажать кнопку OK, на экране появится сообщение, состоящее из введенного текста. Например, в случае рис. 2.4 окно сообщения будет содержать строку "Я набираю любой текст и нажимаю к".

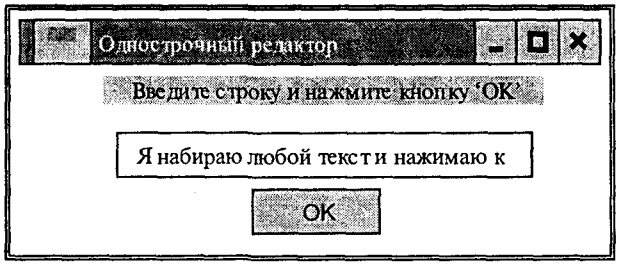

Рис. 2.4. Работа с однострочным редактором

После нажатия на кнопку OK функции окна редактора посылается сообщение с кодом EM_GETLINE:

```
SendMessage( hEdit,EM_GETLINE, 0, (LPARAM)chBuff );
```

Последний аргумент функции SendMessage указывает на буфер, куда следует записать содержимое редактора – chBuff.

Затем содержимое этого буфера отображается в окне сообщений:

```
MessageBox(hwnd, chBuff, szTitle, MB_OK);
```

В примере появилась новая функция:

```
BOOL SetWindowText(HWND hwnd, LPCTSTR lpString);
```

Она устанавливает новый текст заголовка окна или текст органа управления. Параметр lpzString указывает на устанавливаемую строку. В случае успешной установки возвращаемое значение отлично от нуля. В данном примере функция SetWindowText используется для очистки окна ввода.

Функция WinMain приложения имеет одну особенность. Так как текстовый редактор работает с символьными сообщениями, в цикле обработки сообщений вызывается функция TranslateMessage:

```
while(GetMessage(&msg, 0, 0, 0))
{   TranslateMessage(&msg);  DispatchMessage(&msg);  }
```

2.5. Списки строк

2.5.1. Создание списка

Списки строк создают на базе класса "listbox". Строки в списке нумерует операционная система, и номер первой строки равен 0. Списки могут быть одноколоночные и многоколоночные, с вертикальной (для одноколоночных списков) и горизонтальной (для многоколоночных списков) полосой просмотра. При создании списка первым аргументом функции CreateWindow является указатель на строку "listbox", а третий аргумент задают с помощью табл. 2.6. Например:

```
#define    ID_LIST    111
...
hListBox = CreateWindow("listbox", NULL,
    WS_CHILD | WS_VISIBLE | LBS_STANDARD |
    LBS_WANTKEYBOARDINPUT,
    30, 30, 200, 100, hwnd, (HMENU)ID_LIST, hInst, NULL);
```

Второй аргумент функции должен быть указан как NULL. Дополнительно к стилям окна WS_CHILD и WS_VISIBLE указывают стили списка, имена которых имеют префикс LBS_.

2.5.2. Сообщения от списка

Список со стилем LBS_NOTIFY посылает в функцию родительского окна сообщение WM_COMMAND. При этом младшее слово параметра wParam равно идентификатору списка. Параметр lParam содержит дескриптор списка. Старшее слово параметра wParam содержит один из следующих кодов извещения:

Код извещения	*Описание*
LBN_DBLCLK	Двойной щелчок левой клавишей мыши по строке списка
LBN_ERRSPACE	Ошибка при попытке заказать дополнительную память
LBN_KILLFOCUS	Список потерял фокус ввода
LBN_SELCANCEL	Пользователь отменил выбор в списке
LBN_SELCHANGE	Изменился номер выбранной строки
LBN_SETFOCUS	Список получил фокус ввода

2.5.3. Сообщения для списка

Приложение может посылать списку сообщения, вызывая функцию SendMessage. Символические имена этих сообщений имеют префикс LB_. Коды некоторых сообщений перечислены в следующей таблице:

Код сообщения и значения wParam и lParam	*Пояснение*
LB_ADDSTRING, 0, (LPARAM)lpszStr	Добавить в список строку lpszStr. Возвращает номер строки в списке
LB_DELETESTRING, (WPARAM)nIndex и 0L	Удалить строку с номером nIndex из списка. Возвращает количество оставшихся в списке строк
LB_DIR, (WPARAM)(UINT)uAttr и (LPARAM)lpszFileSpec	Заполнить список именами дисков, каталогов и файлов атрибутов uAttr по шаблону lpszFileSpec из текущего каталога. Возвращает номер последнего добавленного в список имени файла
LB_FINDSTRING, (WPARAM)nStart и (LPARAM)lpszStr	Начиная от строки с номером nStart, искать строку с префиксом lpszStr. Возвращает номер найденной строки или код ошибки
LB_FINDSTRINGEXACT, (WPARAM)nStart и (LPARAM)lpszStr	Начиная от строки с номером nStart, искать строку lpszStr. Возвращает номер найденной строки или код ошибки
LB_GETCARETINDEX, 0 и 0L	Определение номера строки, имеющей фокус ввода. Возвращает номер этой строки или код ошибки
LB_GETCOUNT, 0 и 0L	Определение количества строк в списке. Возвращает количество строк в списке или код ошибки
LB_GETCURSEL, 0 и 0L	Определение номера выделенной строки. Возвращает номер выделенной строки или код ошибки
LB_GETSEL, (WPARAM)nIndex и 0L	Определить, выбрана ли строка с номером nIndex. Возвращает положительное число, если строка выбрана, и 0, если не выбрана или код ошибки
LB_GETSELCOUNT, 0 и 0L.	Определить количество выбранных строк. Возвращает количество выбранных строк или код ошибки

Код сообщения и значения wParam и lParam	*Пояснение*
LB_GETSELITEMS, (WPARAM)cI и (LPARAM)(int FAR *)lpI	Заполнить буфер lpI размера cI номерами выбранных строк. Возвращает количество записанных в буфер номеров или код ошибки
LB_GETTEXT, (WPARAM)nIndex и (LPARAM)(int FAR *)lpB	Копировать текст строки с номером nIndex в буфер lpB. Возвращает длину строки или код ошибки
LB_GETTEXTLEN, (WPARAM)nIndex и 0L	Определить длину строки с номером nIndex. Возвращает длину строки или код ошибки
LB_GETTPOINDEX, 0 и 0L	Определить номер первой отображаемой строки. Возвращает номер строки или код ошибки
LB_INSERTSTRING, (WPARAM)nIndex и (LPARAM)(int FAR *)lpB	Вставить элемент lpB в список под номером nIndex. Возвращает номер, под которым вставлена строка, или код ошибки
LB_RESETCONTENT, 0 и 0L	Удалить все строки из списка. Возвращаемое значение: не используется
LB_SELECTSTRING, (WPARAM)nIndexStart и (LPARAM)(int FAR *)lpB	Начиная с позиции nIndexStart, найти и выделить строку с префиксом lpB. Возвращает номер найденной строки или код ошибки
LB_SELITEMRANGE (WPARAM)(BOOL)fSelect; MAKELPARAM(wF, wL)	Выделить (fSelect=TRUE) или отменить выделение (fSelect= FALSE) строк с номерами от wF до wL. Возвращает код ошибки
LB_SETCARETINDEX, (WPARAM)nIndex и MAKELPARAM(fS, 0)	Передать фокус ввода строке с номером nIndex. Если fS=TRUE, свертка выполняется до тех пор, пока указанная строка не будет видна хотя бы частично, если FALSE – до тех пор, пока строка не будет видна полностью. Возвращает код ошибки

Код сообщения и значения wParam и lParam	*Пояснение*
LB_SETCURSEL, (WPARAM)nIndex и 0L	Отменить предыдущий выбор и выбрать строку с номером nIndex. Если nIndex равно -1 отменяется выделение всех строк, возвращается значение LB_ERR, что в данном случае не говорит об ошибке. Иначе (nIndex не равно -1) возвращает код ошибки
LB_SETTOPINDEX, (WPARAM)nIndex и 0L	Свертка списка до тех пор, пока строка с номером nIndex не станет видимой. Возвращает код ошибки

2.5.4. Пример работы со списком

Задача. В окне приложения создать однострочный редактор, список и кнопку "Добавить". При нажатии на кнопку содержимое редактора записать в список. При двойном щелчке по строке списка сообщить о строке, которая была выбрана.

Решение этой задачи отображено в описании следующей функции окна (листинг 2.7).

При создании окна друг под другом создаются 4 органа управления с дескрипторами hStatic (статический орган), hEdit (однострочный редактор), hListBox (список) и hButton (кнопка).

Листинг 2.7. Работа со списком строк.

```
#include <windows.h>
#define    ID_EDIT      1000
#define    ID_LISTBOX   1001
#define    ID_BUTTON    1002

BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char    szClassName[ ] = "ListDir";
char    szTitle[ ] = "Работа со списком";

int WINAPI  WinMain(HINSTANCE hInst,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  hInstance=hInst;
    if ( !RegClass(WndProc, szClassName, COLOR_WINDOW) )
        return FALSE;
    if ( !(hwnd = CreateWindow(szClassName, szTitle,
```

```
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        100, 50, 364, 360, 0, 0, hInstance, NULL)) ) return FALSE;
    while(GetMessage(&msg, 0, 0, 0))
    {   TranslateMessage(&msg); DispatchMessage(&msg); }
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
                            UINT brBackground)
{   WNDCLASS wc; wc.style=wc.cbClsExtra=wc.cbWndExtra=0;
    wc.lpfnWndProc = Proc;    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName=(LPCTSTR)NULL;
    wc.lpszClassName=szName; return (RegisterClass(&wc) != 0);
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam)
{   static HWND hListBox, hEdit, hButton, hStatic;
    int uItem;   char Buf[80];
    switch (msg)
    {   case WM_CREATE:
        {   hStatic=CreateWindow("static",
                "Введите строку для ввода в список",
                WS_CHILD | WS_VISIBLE | SS_CENTER,
                30,10,300,20,hwnd,(HMENU)0,hInstance,NULL);
            hEdit = CreateWindow("edit", NULL,
                WS_CHILD | WS_VISIBLE | WS_BORDER |
                ES_LEFT | ES_AUTOHSCROLL,30, 40, 300, 30,
                hwnd,(HMENU)ID_EDIT,hInstance, NULL);
            hListBox = CreateWindow("listbox", NULL,
                WS_CHILD | WS_VISIBLE | LBS_STANDARD,
                30, 80, 300, 200, hwnd,
                (HMENU)ID_LISTBOX, hInstance, NULL);
            hButton = CreateWindow("button", "Добавить",
                WS_CHILD | WS_VISIBLE |
                BS_PUSHBUTTON, 130, 290, 100, 30, hwnd,
                (HMENU)ID_BUTTON,hInstance, NULL);
            SetFocus(hEdit);    return 0;
        }
        case WM_COMMAND:
```

```
    {   switch (LOWORD(wParam))
        {   case ID_LISTBOX:
            {   switch (HIWORD(wParam))
                {   case LBN_ERRSPACE:
                    {   MessageBox(hwnd, "Мало памяти", szTitle, MB_OK);
                        return 0;
                    }
                    case LBN_DBLCLK:
                    {   uItem=(int)SendMessage(hListBox,
                            LB_GETCURSEL, 0,0L);
                        if(uItem != LB_ERR)
                        {   SendMessage(hListBox,
                                LB_GETTEXT,
                                uItem,
                                (LPARAM)Buf);
                            MessageBox(hwnd, Buf,
                                szTitle,MB_OK);
                        }
                        return 0;
                    }
                    default: return 0;
                }
                return 0;
            }
            case ID_BUTTON:
            {   SendMessage(hEdit,EM_GETLINE,0,(LPARAM)Buf);
                SendMessage(hListBox,LB_ADDSTRING,0,(LPARAM)Buf);
                SetWindowText(hEdit,"\0");
                SetFocus(hEdit);
                return 0;
            }
            default:return 0;
        }
    }
    case WM_DESTROY: { PostQuitMessage(0); return 0; }
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Если нажать на кнопку "Добавить", то в буфер с именем Buf выбирается содержимое (даже пустое) однострочного редактора и содержимое буфера Buf записывается в список:

```
SendMessage( hEdit, EM_GETLINE, 0, (LPARAM)Buf );
SendMessage( hListBox, LB_ADDSTRING, 0, (LPARAM)Buf );
```

Обратите внимание, что редактор в этом примере может содержать и очень длинную строку. В любом случае она записывается в список. Ограничение накладывает лишь длина буфера Buf.

При двойном щелчке по строке списка определяется номер выделенного элемента:

```
uItem=(int)SendMessage( hListBox, LB_GETCURSEL, 0, 0L );
```

Если код возврата uItem не равен ошибке (LB_ERR), то в буфер с именем Buf выбирается строка с номером uItem:

```
SendMessage( hListBox, LB_GETTEXT, uItem, (LPARAM)Buf );
```

2.6. Комбинированный список

2.6.1. Создание комбинированного списка

Этот орган является комбинацией однострочного редактора текста и списка и создается на базе класса "Combobox". Комбинированный список создается так же, как и другие органы управления:

```
hComboBox = CreateWindow("ComboBox", NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL |
    CBS_AUTOHSCROLL | CBS_SIMPLE,
    30, 30, 200, 200,  hwnd, (HMENU) ID_COMBO, hInstance, NULL);
```

В стиле окон класса указывают символические имена с префиксом CBS_ (табл. 2.7).

2.6.2. Коды извещения

Комбинированный список посылает в функцию родительского окна сообщение WM_COMMAND. При этом младшее слово параметра wParam содержит идентификатор, а параметр lParam – дескриптор списка. Старшее слово параметра wParam содержит код извещения. Приведем список некоторых кодов извещений:

Код извещения	*Описание*
CBN_CLOSEUP	Список исчез (стал невидим)
CBN_DBLCLK	Двойной щелчок левой клавишей мыши по строке списка стиля CBS_SIMPLE
CBN_DROPDOWN	Список стал видимым
CBN_EDITCHANGE	Пользователь изменил содержимое окна редактирования и изменения отображены

CBN_EDITUPDATE	Пользователь изменил содержимое окна редактирования, изменения не отображены
CBN_ERRSPACE	Ошибка выделения дополнительной памяти
CBN_KILLFOCUS	Список теряет фокус ввода
CBN_SELENDCANCEL	Пользователь отменил выбор в списке
CBN_SELENDOK	Пользователь выбрал строку в списке
CBN_SELCHANGE	Изменился номер выбранной строки
CBN_SETFOCUS	Список получает фокус ввода

2.6.3. Сообщения для комбинированного списка

Для управления комбинированным списком используют набор сообщений, аналогичный сообщениям для списка и редактора текста. В следующей таблице перечислены коды некоторых сообщений:

Сообщение и параметры	*Пояснение*
CB_ADDSTRING, 0 и (LPARAM)lpszStr	Добавить строку lpszStr в список. Возвращает номер строки в списке или код ошибки
CB_DELETESTRING (WPARAM)nIndex и 0L	Удалить строку с номером nIndex из списка. Возвращает количество оставшихся в списке строк или код ошибки
CB_DIR, (WPARAM)(UINT)uAttr и (LPARAM) lpszFileSpec;	Заполнить список именами файлов с атрибутом uAttr – и шаблона lpszFileSpec, и каталогов в текущем каталоге, и именами дисков. Возвращает номер последнего имени файла, добавленного в список, или код ошибки
CB_FINDSTRING, (WPARAM)nIndexStart и (LPARAM) (LPCSTR)lpszStr	Начиная со строки с номером nIndexStart, искать строку с префиксом lpszStr. Возвращает номер найденной строки или код ошибки (если строки в списке нет)
CB_GETCOUNT, 0 и 0L	Определить количество строк в списке. Возвращает количество строк в списке или код ошибки
CB_GETCURSEL, 0 и 0L	Определить номер выделенной строки. Возвращает номер выделенной строки или код ошибки

Сообщение и параметры	*Пояснение*
CB_GETDROPPEDSTATE 0 и 0L	Определить, находится список в видимом или невидимом состоянии. Возвращает TRUE, если список виден, FALSE – если нет
CB_GETLBTEXT, (WPARAM)nIndex и (LPARAM)(int FAR *)lpB	Копировать текст строки с номером nIndex в буфер lpB. Возвращает длину строки в байтах или код ошибки
CB_INSERTSTRING, (WPARAM)nIndex и (LPARAM)(int FAR *)lpB	Вставка строки lpB в список под номером nIndex. Возвращает номер позиции, в которую вставлена строка, или код ошибки
CB_LIMITTEXT, (WPARAM)cCmax и 0L	Определить cCmax как максимально допустимое количество символов в окне редактирования. Возвращаемое значение не используется
CB_RESETCONTENT, 0 и 0L	Удалить все строки из списка. Возвращаемое значение не используется
CB_SELECTSTRING, (WPARAM)nIndexStart и (LPARAM)(int FAR *)lpB	Начиная со строки с номером nIndexStart, найти и выделить строку с префиксом lpB. Возвращает номер найденной строки или код ошибки
CB_SETCURSEL, (WPARAM)nIndex и 0L	Выделить строку с номером nIndex. Если nIndex равно -1, выделение всех строк будет отменено. При этом функция SendMessage вернет значение CB_ERR, что в данном случае не говорит об ошибке. Возвращает код ошибки (если значение nIndex не равно -1)
CB_SHOWDROPDOWN, (WPARAM)(BOOL)fExtended и 0L	Переключение списка в отображаемое или неотображаемое состояние. При fExtended=TRUE отображать, при FALSE – нет. Возвращаемое значение всегда не равно 0

2.6.4. Пример работы с комбинированным списком

Задача. В окне приложения создать однострочный редактор, комбинированный список и кнопку "Добавить". При нажатии на кнопку содержимое редактора записать в список. При двойном щелчке по строке списка сообщить о строке, которая была выбрана.

Листинг 2.8. Работа с комбинированным списком строк.

```
#include <windows.h>
#define ID_EDIT        1000
#define ID_COMBBOX 1001
#define ID_BUTTON    1002

BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE    hInstance;
char     szClassName[ ] = "ComboList";
char     szTitle[ ] = "Работа с комбинированным списком";

int WINAPI  WinMain(HINSTANCE hInst,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  hInstance=hInst;
    if ( !RegClass(WndProc, szClassName, COLOR_WINDOW) )
        return FALSE;
    if  ( !(hwnd = CreateWindow(szClassName, szTitle,
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        100, 50, 364, 360, 0, 0, hInstance, NULL)))     return FALSE;
    while ( GetMessage(&msg, 0, 0, 0) )
    {   TranslateMessage(&msg); DispatchMessage(&msg);  }
    return msg.wParam;
}
BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
                              UINT brBackground)
{   WNDCLASS wc; wc.style=wc.cbClsExtra=wc.cbWndExtra=0;
    wc.lpfnWndProc = Proc;    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName=(LPCTSTR)NULL;
    wc.lpszClassName=szName;   return (RegisterClass(&wc) != 0);
}
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam)
{   static HWND hCombBox, hEdit, hButton, hStatic;
    int uItem;   char Buf[80];
    switch (msg)
    {   case WM_CREATE:
        {   hStatic=CreateWindow("static",
```

```
        "Введите строку для ввода в список",
        WS_CHILD | WS_VISIBLE | SS_CENTER,
        30,10,300,20,hwnd,(HMENU)0,hInstance,NULL);
    hEdit = CreateWindow("edit", NULL,
        WS_CHILD | WS_VISIBLE | WS_BORDER |
        ES_LEFT | ES_AUTOHSCROLL,30, 40, 300, 30,
        hwnd,(HMENU)ID_EDIT,hInstance, NULL);
    hCombBox = CreateWindow("Combobox", NULL,
        WS_CHILD | WS_VISIBLE | WS_VSCROLL |
        CBS_SIMPLE | CBS_SORT,30, 80, 300, 200,
        hwnd, (HMENU)ID_COMBBOX, hInstance,
        NULL);
    hButton = CreateWindow("button", "Добавить",
        WS_CHILD | WS_VISIBLE |
        BS_PUSHBUTTON, 130, 290, 100, 30,
        hwnd,(HMENU)ID_BUTTON,hInstance, NULL);
    SetFocus(hEdit);    return 0;
}
case WM_COMMAND:
{   switch (LOWORD(wParam))
    {   case ID_COMBBOX:
        {   switch (HIWORD(wParam))
            {   case CBN_ERRSPACE:
                {   MessageBox(hwnd, "Мало памяти",szTitle, MB_OK);
                    return 0;
                }
                case CBN_DBLCLK:
                {   uItem=(int)SendMessage(hCombBox,
                        CB_GETCURSEL,0, 0L);
                    if (uItem != LB_ERR)
                    {   SendMessage(hCombBox, CB_GETLBTEXT,
                            uItem, (LPARAM)Buf);
                        MessageBox(hwnd, (LPSTR)Buf, szTitle,MB_OK);
                    }
                    return 0;
                }
                default: return 0;
            }
            return 0;
        }
        case ID_BUTTON:
```

```
            {   SendMessage(hEdit,EM_GETLINE,0,(LPARAM)Buf);
                SendMessage(hCombBox, CB_ADDSTRING,0,(LPARAM)Buf);
                SetWindowText(hEdit,"\0");
                SetFocus(hEdit);
                return 0;
            }
            default:return 0;
        }
    }
    case WM_DESTROY: {  PostQuitMessage(0);  return 0;  }
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Как видим, текст этого приложения от предыдущего отличается только видом используемого списка – вместо простого списка применяется комбинированный список.

Отличия в тексте приложения, как это нетрудно увидеть, связаны с именами класса "Combobox", кодов извещений и сообщений.

В процессе работы со списком тоже появляется одно важное отличие. В процессе выбора строки из списка простой просмотр списка может отнять много времени. Комбинированный список позволяет упростить поиск, если вы знаете хотя бы несколько первых букв префикса искомой строки. Наберите начальные буквы искомой строки в окне редактирования комбинированного списка – и нужная строка окажется перед вашими глазами.

Окна класса "Combobox" завершают изучение определенных в системе Windows классов окон.

Контрольные вопросы

1. Чем отличаются окна определенных в Windows классов от обычных окон?

2. Какие аргументы обязательны при создании органов управления с помощью функции CreateWindow?

3. Каким образом родительское окно идентифицирует сообщения от органов управления?

4. Чем отличается создание полосы прокрутки от создания других органов управления?

5. Какие способы передачи сообщений окнам существуют и чем они отличаются?

6. В чем преимущество использования комбинированного списка по сравнению с обычным списком?

7. Каковы основные этапы создания органов управления?

8. Что будет, если при создании комбинированного списка вторым аргументом указать на некоторую строку?

9. Какие органы управления не используют идентификаторы?

10. Какие органы управления и в каком случае не используют дескрипторы окон?

Упражнения

1. Рабочую область при любых изменениях размеров окна приложения полностью занимает многострочный редактор с полосами просмотра и фокусом ввода.

2. В левом верхнем углу рабочей области окна приложения создать список. Справа от списка создать комбинированный список. Списки должны отображаться в окне приложения после нажатия правой кнопки мыши, а исчезать – после нажатия левой. При отображении списков фокус ввода установить в окне выбора.

3. В рабочей области окна приложения отобразить кнопку по умолчанию с надписью "Да" и обычную кнопку с надписью "Отмена" с идентификаторами соответственно IDOK и IDCANCEL. Сообщить пользователю о действиях над этими кнопками.

4. В окне приложения отобразить однострочный редактор, под ним список и еще ниже – кнопку с надписью "Да". При нажатии на кнопку записать в окно редактора текст о нажатии кнопки.

5. Окно содержит пустую строку ввода с надписью "Введите фамилию, имя и отчество", группу переключателей "Пол" из двух пунктов ("Мужчина" и "Женщина") и кнопку "Да". При нажатии на кнопку "Да" непустое содержимое строки ввода записать в заголовок группы. Затем при выборе пола сообщить новое содержимое заголовка группы, выбранный пол и вернуться к исходному состоянию.

6. В центре экрана отобразить окно без заголовка с фоном цвета трехмерных элементов. Окно содержит текст о правах на программный продукт и кнопки "Выход" и "Да". При нажатии кнопки "Выход" завершить работу, а кнопки "Да" – текст в окне заменить текстом о правилах работы, а кнопки заменить кнопками "Назад" и "Далее". При нажатии кнопки "Назад" вернуться к исходному состоянию.

7. В окне приложения создать временное окно с фоном цвета трехмерных элементов, содержащее группу вариантов взаимно исключающих действий и 3 кнопки: "Да", "Нет" и "Отмена".

8. В окне приложения отобразить кнопку "Клавиши отжаты". При нажатии любой клавиши мыши над рабочей областью окна эта кнопка исчезает и под курсором мыши появляется кнопка "Клавиша нажата". При отжатии клавиши мыши вернуться к исходному состоянию.

9. В рабочей области отображена кнопка с надписью "Нажмите правую клавишу мыши над рабочей областью". После нажатия правой клавиши мыши кнопка скрывается и отображается многострочный редактор с текстом "Нажмите левую клавишу мыши над рабочей областью". После нажатия левой клавиши мыши вернуться к исходному состоянию.

10. Создать окно с фоном трехмерных элементов без рамок. При нажатии левой клавиши мыши в только вам известной точке рабочей области и последующем перемещении курсора мыши правая и нижняя границы окна должны повторять это перемещение.

11. При запуске первого экземпляра приложения в окне приложения отобразить поле ввода пароля, а при запуске следующих экземпляров того же приложения запросить, действительно ли нужно запускать еще один экземпляр. Если "Да", запустить еще один экземпляр, но пароль больше не запрашивать.

12. Верхнюю часть рабочей области окна занимают два статических органа шириной в половину ширины рабочей области. Под ними в остальной части рабочей области расположены две вертикальные полосы прокрутки. Каждый статический орган отображает текущее значение позиции ползунка полосы, находящейся под ним.

13. Создать однострочный редактор, список и кнопки "Добавить" и "Изменить". При нажатии на кнопку "Добавить" содержимое редактора (если оно не пустое) добавить в список, очистить редактор и передать ему фокус ввода. При нажатии на кнопку "Изменить" выбранную строку списка (если она выбрана) записать в редактор, удалить из списка и передать фокус ввода редактору.

14. Создать однострочный редактор, список и кнопку "Найти". Изначально заполнить список набором строк. При нажатии на кнопку "Найти" содержимое редактора (если оно не пустое) использовать в качестве начальных букв поиска строки в списке и выбрать найденную строку.

15. В окне расположить группу переключателей, группу флажков, кнопки "Да" и "Нет", причем кнопка "Да" выбирается по умолчанию. Если нажата кнопка "Да", то сообщить о состоянии переключателей и флажков. Если нажата кнопка "Нет", то завершить работу.

16. В рабочей области с фоном цвета кнопок расположить группу флажков, способных находиться в трех состояниях, с надписями слева и кнопки "Да" и "Выход". Кнопка "Да" выбирается по умолчанию. После нажатия кнопки "Выход" завершить работу.

17. Окно в центре экрана с фоном цвета трехмерных элементов содержит переключатели, имитирующие работу светофора. Названия цветов светофора расположены слева от переключателей. При нажатии на переключатель сообщить название его цвета.

18. В рабочей области окна перечислить названия изучаемых в семестре дисциплин и пометить названия изучаемых сегодня дисциплин. Выделить среди изучаемых сегодня дисциплин те, по которым нет экзамена.

19. В центре экрана отобразить окно без рамок, в верхней части которого отобразить текст "Анкетные данные служащего", ниже – поле ввода фамилии, имени и отчества, под которым группы переключателей о поле и образовании. Окно также содержит кнопки "Запись", "Очистка" и "Выход".

20. В окне приложения отобразить два статических органа с текстами "Группа 1" и "Группа 2". При нажатии левой клавиши мыши над первым органом должна появиться кнопка с надписью "Кнопка 1", а при нажатии над вторым органом – "Кнопка 2". Любая кнопка при нажатии должна исчезнуть.

21. Комбинированный список изначально заполнить произвольными различными строками. Рядом расположить кнопку "Вверх". Если выбрать строку и нажать кнопку "Вверх", то эта строка должна переместиться вверх на одну позицию в списке. Если эта строка уже первая в списке, то сообщить об этом и заблокировать кнопку.

22. Создать строку ввода с числом 0 и справа от нее – маленькую вертикальную полосу прокрутки. Под ними кнопки "Выход" и "Да". При нажатии на верхнюю кнопку полосы число в строке ввода увеличить на единицу, а на нижнюю кнопку – уменьшить на единицу. При нажатии кнопки "Да" отобразить содержимое строки ввода, а кнопки "Выход" – завершить работу.

23. Окно приложения без заголовка с рамкой стиля кнопки сообщает о форматировании гибкого диска и содержит кнопку "Прекратить". При попытке нажать на кнопку изменить текст сообщения и переместить кнопку в другое место.

24. Создать группу флажков с возможными вариантами данных о студенте и кнопку "Да". После нажатия на кнопку сообщить об установленных данных.

25. Создать два списка, между ними расположить кнопки "Переместить>>" и "<<Переместить". Левый список должен быть изначально заполнен списком строк. Если выбрать строку и нажать одну из кнопок, то выбранная строка должна переместиться слева направо или наоборот.

Глава 3
Вывод в окно

Вывод в окна обладает рядом особенностей.

1. Нельзя пользоваться функциями вывода библиотеки компилятора, поскольку они приспособлены для вывода в одно и единственное окно. В операционной системе Windows приложения выводят одновременно в различные окна. Система сама решает все проблемы, связанные с возможным перекрытием или перемещением этих окон. С этим обстоятельством связано то, что **в окно стремятся выводить в одном месте приложения – при обработке сообщения WM_PAINT**. Приложение описывают таким образом, чтобы при поступлении сообщения WM_PAINT функция окна могла перерисовать все окно или любую его заданную часть.

2. **Интерфейс графических устройств** (Graphics Device Interface – GDI) системы открывает доступ к большому количеству функций вывода. Приложения, обращаясь к функциям GDI, работают не с физическими устройствами вывода, а с логическими. То есть описание вызова функций не зависит от физического способа отображения. GDI передает указания о выводе драйверу устройства вывода. Драйвер работает непосредственно с физическим устройством и при управлении выводом учитывает его ограниченные возможности и аппаратные особенности. Благодаря этому **приложения способны работать с любым устройством вывода**, драйвер которого установлен в системе. В этой главе рассматриваются функции и параметры вывода в окно на экране видеомонитора. Но не все физические устройства вывода способны поддерживать те режимы, в которых работает видеомонитор. Поэтому, описывая последовательность операций вывода, например на принтер, следует учитывать ограниченные возможности установленного принтера.

3. **Параметры вывода устанавливают в контексте отображения** с помощью функций GDI. Контекст отображения – это структура данных, которая содержит характеристики устройства вывода и указатели на выбранные инструменты рисования. Функции GDI используют только выбранные в контекст отображения параметры и инструменты рисования. Например, для рисования линии некоторой толщины в контекст отображения приложение должно выбрать перо этой толщины.

4. **Дескриптор контекста отображения** служит первым аргументом вызова всех функций, связанных с выводом в окно.

3.1. Сообщение WM_PAINT

Стремление описывать вывод в окно при обработке сообщения WM_PAINT объясняется следующим. Windows следит за изменением расположения окон и при необходимости перерисовки окна посылает функции окна сообщение WM_PAINT. Функция окна при обработке этого сообщения перерисовывает все окно или указанные части.

Сообщение WM_PAINT обрабатывают следующим образом:

```
case WM_PAINT:
{   PAINTSTRUCT ps;
    HDC hdc=BeginPaint(hwnd, &ps);
    //Здесь настраивают контекст отображения и выводят в окно
    EndPaint(hwnd, &ps);
    return 0;
}
```

Структура PAINTSTRUCT описана следующим образом:

```
typedef struct
{   HDC   hdc;
    BOOL  fErase;
    RECT  rcPaint;
    BOOL  fRestore;
    BOOL  fIncUpdate;
    BYTE  rgbReserved[16];
}   PAINTSTRUCT;
```

Поля переменной ps заполняет функция BeginPaint. После ее вызова поле hdc структуры ps содержит дескриптор контекста отображения (тот же, который возвращает функция BeginPaint). Поле rcPaint содержит координаты прямоугольной области обновления. Поле fErase определяет необходимость стирания фона области обновления. Если fErase=TRUE, фон окна стирается, иначе фон окна не изменяется. Остальные поля использует операционная система, приложение не должно изменять их содержимое.

После выполнения операций вывода приложение освобождает используемые ресурсы – вызывает функцию EndPaint.

Задача. В центре рабочей области окна вывести текст.

Листинг 3.1 содержит решение данной задачи.

Листинг 3.1. Вывод текста в центре рабочей области.

```
#include <windows.h>
```

```
BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);

HINSTANCE    hInstance;
char     szClass[ ]="OutputClass";

int WINAPI  WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass,COLOR_WINDOW) )
        return FALSE;
    hwnd = CreateWindow(szClass, "Вывод текста",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    ShowWindow(hwnd, SW_SHOWMAXIMIZED );
    UpdateWindow(hwnd);
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
                                UINT brBackground)
{   WNDCLASS wc;  wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.cbClsExtra = wc.cbWndExtra = 0; wc.lpfnWndProc = Proc;
    wc.hInstance = hInstance;  wc.lpszClassName = szName;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = NULL;     return (RegisterClass(&wc) != 0);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam )
{   char     szText[ ]="Выводим какой-либо текст";
    static short cx, cy;
    switch (msg)
    {   case WM_SIZE:
        {   cx=LOWORD(lParam);      cy=HIWORD(lParam);
            return 0;
```

```
        }
        case WM_PAINT:
        {   PAINTSTRUCT     ps;
            HDC    hdc=BeginPaint(hwnd,&ps);
            //Настриваем атрибуты вывода текста
            SetTextColor(hdc,RGB(255, 0, 0));
            SetBkColor(hdc,RGB(0, 255, 255));
            SetTextAlign(hdc,TA_CENTER);
            //Выводим текст
            TextOut(hdc,cx/2,cy/2,szText,strlen(szText));
            EndPaint(hwnd,&ps);
            return 0;
        }
        case WM_DESTROY: { PostQuitMessage(0); return 0; }
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Внимание! В стиле класса окна, обрабатывающего сообщение WM_PAINT, нужно указывать, что оно должно перерисовывать содержимое при изменении размеров окна:

```
wc.style = CS_HREDRAW | CS_VREDRAW;
```

Окна класса с таким стилем получают сообщение WM_PAINT при изменении их размеров.

При обработке сообщения WM_PAINT добавились операторы настраивания атрибутов контекста отображения и вывода в окно:

```
//Настриваем атрибуты вывода текста
SetTextColor(hdc,RGB(255, 0, 0));
SetBkColor(hdc,RGB(0, 255, 255));
SetTextAlign(hdc,TA_CENTER);
//Выводим текст
TextOut(hdc,cx/2,cy/2,szText,strlen(szText));
```

Функции настраивания атрибутов и вывода в окно рассматриваются ниже.

Представим теперь, что **содержимое окна требуется обновить при обработке других сообщений**. Для этого функции обновляемого окна достаточно передать сообщение WM_PAINT.

Одна из функций, которые посылают сообщение WM_PAINT, а именно UpdateWindow, использовалась ранее.

Эта функция объявлена следующим образом:

```
BOOL UpdateWindow( HWND hwnd );
```

Она посылает сообщение WM_PAINT непосредственно функции окна hwnd. В случае успешного выполнения функция UpdateWindow возвращает ненулевое значение, иначе – 0. Обычно ее вызывают для немедленной перерисовки области обновления.

Функция InvalidateRect добавляет прямоугольник в область перерисовки окна hwnd. Она объявлена следующим образом:

```
BOOL InvalidateRect( HWND hwnd, CONST RECT *lpRect, BOOL bErase);
```

Параметр hwnd указывает на обновляемое окно. Если это параметр указывает на NULL, перерисовывают все окна. Параметр lpRect указывает на нуждающийся в обновлении прямоугольник в рабочей области. Если lpRect=NULL, то нужно обновить всю рабочую область окна hwnd. Параметр bErase определяет, нужно ли перекрасить фон указанной прямоугольной области. Если bErase=TRUE, фон перекрашивают, иначе фон остается неизменным.

В случае успешного выполнения функция возвращает ненулевое значение.

Добавляемые прямоугольники накапливаются до обработки сообщения WM_PAINT. В качестве единой области перерисовки Windows вычисляет один прямоугольник, который охватывает все добавленные прямоугольники.

Функция ValidateRect удаляет прямоугольную область из списка прямоугольников перерисовки. Прототип этой функции:

```
BOOL ValidateRect( HWND hwnd, CONST RECT *lpRect );
```

Параметр hwnd указывает на окно, из области перерисовки которого исключается прямоугольник. Если этот параметр NULL, Windows перерисовывает все окна – посылает сообщения WM_ERASEBKGND и WM_NCPAINT функциям всех окон. Параметр lpRect указывает на прямоугольник, который будет удален от области перерисовки. Если lpRect=NULL, из области обновления удаляют все прямоугольники.

В случае успешного выполнения функция возвращает ненулевое значение.

Функция BeginPaint из области обновления удаляет всю рабочую область окна.

Задача. В центре рабочей области окна вывести строку "Текст по умолчанию". После нажатия левой клавиши мыши содержимое этой строки сменить на текст "Нажата левая клавиша мыши".

Листинг 3.2. Перерисовка окна после нажатия левой клавиши мыши.

```
#include <windows.h>

BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
```

```
HINSTANCE   hInstance;
char    szClass[ ]="OutputClass2";
char    szText[50]="Текст по умолчанию";

int WINAPI  WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass,COLOR_WINDOW) )
        return FALSE;
    hwnd = CreateWindow(szClass, "Вывод текста",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
                            UINT brBackground)
{   WNDCLASS wc;  wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.cbClsExtra = wc.cbWndExtra = 0; wc.lpfnWndProc = Proc;
    wc.hInstance = hInstance;  wc.lpszClassName = szName;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = NULL;     return (RegisterClass(&wc) != 0);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam )
{   static short cx, cy;
    switch (msg)
    {   case WM_SIZE:
        {   cx=LOWORD(lParam);      cy=HIWORD(lParam);
            return 0;
        }
        case WM_PAINT:
        {   PAINTSTRUCT     ps;
            HDC    hdc=BeginPaint(hwnd,&ps);
            SetTextColor(hdc,RGB(255, 0, 0));
```

```
            SetBkColor(hdc,RGB(0, 255, 255));
            SetTextAlign(hdc,TA_CENTER);
            TextOut(hdc,cx/2,cy/2,szText,strlen(szText));
            EndPaint(hwnd,&ps);
            return 0;
        }
        case WM_LBUTTONDOWN:
        {   strcpy(szText,"Нажата левая клавиша мыши");
            InvalidateRect(hwnd,NULL,TRUE);
            return 0;
        }
        case WM_DESTROY: { PostQuitMessage(0); return 0; }
    }   return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Произошли следующие изменения:

1. Переменная szText описана глобально. При "вызове" функции окна эта переменная сохраняет последнее присвоенное значение.

2. При обработке сообщения WM_LBUTTONDOWN в переменную szText записывается текст "Нажата левая клавиша мыши" и в область перерисовки добавляется вся рабочая область:

```
strcpy(szText,"Нажата левая клавиша мыши");
InvalidateRect(hwnd,NULL,TRUE);
```

3.2. Виды контекста отображения

Существуют следующие виды контекста отображения:

- общий;
- для класса окон;
- личный;
- родительский;
- для окна.

Способы получения и освобождения контекста отображения зависят от вида контекста. Каждый контекст имеет свои особенности и назначение.

Общий контекст отображения

Этот контекст отличает высшая скорость доступа к нему. Для получения общего контекста вызывают функцию BeginPaint (при обработке сообщения WM_PAINT) или GetDC (для рисования при обработке других сообщений). Стиль класса окна с общим контекстом отображения не может содержать значения CS_OWNDC, CS_PARENTDC или CS_CLASSDC.

Функция GetDC возвращает контекст отображения для рабочей области окна hwnd:

```
HDC GetDC( HWND hwnd );
```

Параметры возвращаемого контекста зависят от стиля класса указанного окна. Для общего контекста функция GetDC устанавливает заданные по умолчанию атрибуты. А параметры контекста отображения класса окон или личного контекста отображения функция GetDC оставляет неизменными. Если hwnd=NULL, то возвращается **контекст видеомонитора** с началом координат в левом верхнем углу экрана. В последнем случае допускается вывод в любом месте экрана.

В случае ошибок функция GetDC возвращает NULL.

Эту функцию вызывают, если требуется вывод в окно во время обработки других сообщений, не посылая сообщения WM_PAINT.

Для освобождения общего контекста отображения, полученного функцией GetDC, вызывают функцию ReleaseDC. Контекст отображения для класса окон и личный контекст отображения освобождать не обязательно. Функция ReleaseDC освобождает только общий контекст отображения или контекст отображения для окна:

```
int ReleaseDC( HWND hwnd, HDC hdc );
```

Здесь hwnd – дескриптор окна, чей контекст должен быть освобожден, а hdc – освобождаемый контекст отображения.

Если контекст освобожден, возвращаемое значение 1, иначе – нуль.

Приложения вызывают функцию ReleaseDC для каждого вызова функции GetWindowDC и для каждого вызова функции GetDC.

Еще раз обращаем ваше внимание на необходимость своевременного освобождения общего контекста отображения.

Задача. При обработке сообщения WM_PAINT в центре рабочей области окна малиновым цветом шрифта на кремовом фоне вывести строку "Вывод при обработке сообщения WM_PAINT". После нажатия левой клавиши мыши на 60 пикселей выше синим цветом шрифта на желтом фоне вывести строку "Вывод при обработке сообщения WM_LBUTTONDOWN".

Листинг 3.3. Вывод текста при обработке различных сообщений.

```
#include <windows.h>
BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char szClass[ ]="OutputClass";
```

```
int WINAPI  WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass,COLOR_WINDOW) )
        return FALSE;
    hwnd = CreateWindow(szClass, "Вывод текста",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT,CW_USEDEFAULT,
        0, 0, hInstance, NULL );
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}
BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
                            UINT brBackground)
{   WNDCLASS wc;  wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.cbClsExtra = wc.cbWndExtra = 0; wc.lpfnWndProc = Proc;
    wc.hInstance = hInstance;  wc.lpszClassName = szName;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = NULL;     return (RegisterClass(&wc) != 0);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam )
{   static short cx, cy;
    switch (msg)
    {   case WM_SIZE:
        {   cx=LOWORD(lParam);      cy=HIWORD(lParam);
            return 0;
        }
        case WM_PAINT:
        {   char szText[ ]=
                "Вывод при обработке сообщения WM_PAINT";
            PAINTSTRUCT     ps;
            HDC    hdc=BeginPaint(hwnd,&ps);
            //Цветом вывода символов выбираем малиновый
            SetTextColor(hdc,RGB(255,0,255));
            //Цветом фона вывода символов выбираем кремовый
```

```
            SetBkColor(hdc,RGB(255,251,240));
            SetTextAlign(hdc,TA_CENTER);
            TextOut(hdc,cx/2,cy/2,szText,strlen(szText));
            EndPaint(hwnd,&ps);   return 0;
        }
        case WM_LBUTTONDOWN:
        {   char szText[ ]=
                "Вывод при обработке сообщения WM_LBUTTONDOWN";
            HDC hdc=GetDC(hwnd);
            //Цветом вывода символов выбираем синий
            SetTextColor(hdc,RGB(0, 0, 255));
            //Цветом фона вывода символов выбираем желтый
            SetBkColor(hdc,RGB(255, 255, 0));
            SetTextAlign(hdc,TA_CENTER);
            TextOut(hdc,cx/2,cy/2-60,szText,strlen(szText));
            ReleaseDC(hwnd, hdc);          return 0;
        }
        case WM_DESTROY: { PostQuitMessage(0); return 0;}
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

После запуска этого приложения на экране появляется окно приложения с заголовком "Вывод текста". В центре рабочей области окна малиновым цветом шрифта на кремовом фоне отображен текст "Вывод при обработке сообщения WM_PAINT".

Если нажать левую клавишу мыши, на 60 пикселей выше этой строки появится еще одна строка: "Вывод при обработке сообщения WM_LBUTTON DOWN" – синим цветом шрифта на желтом фоне.

Если переместить окно приложения или изменить его размеры, то на экране останется только та строка, которая выводится при обработке сообщения WM_PAINT. Вторая строка исчезнет. Это объясняется тем, что при необходимости перерисовки содержимого окна операционная система посылает сообщение WM_PAINT. После перерисовки фона рабочей области в окне приложения появится тот текст, который выводится при обработке сообщения WM_PAINT. Результаты вывода при обработке других сообщений стираются и не восстанавливаются.

Контекст отображения для класса окон

Можно создать единственный контекст отображения для всех окон некоторого класса. Для этого при регистрации класса указывают стиль CS_CLASSDC. Тогда все окна этого класса будут пользоваться одним общим контекстом отображения.

Приложения, однажды получив контекст отображения для класса окон, могут не освобождать его. Причем вызов функций EndPaint и ReleaseDC не освобождает такой контекст (хотя и не вредит работе приложения). Для сохранения единого стиля работы с контекстами рекомендуется всегда вызывать эти функции.

Контекст отображения для класса окон создают один раз. Если создано несколько окон этого класса, область ограничения и начало системы физических координат вывода автоматически настраиваются на то окно, которое использует контекст отображения. Но если на базе класса стиля CS_CLASSDC приложение создало только одно окно, оно может получить контекст отображения для класса окон один раз и не освобождать его.

Таким образом, контекст отображения для класса окон повышает производительность вывода в окна тем, что не требует настройку многочисленных атрибутов контекста отображения после каждого вызова функции BeginPaint или EndPaint.

Задача. Для окон класса стиля CS_CLASSDC создать единый контекст и установить его атрибуты по умолчанию. После нажатия левой клавиши мыши изменить атрибуты контекста, а после нажатия правой клавиши – восстановить исходные значения атрибутов.

Листинг 3.4. Установка атрибутов контекста для класса по умолчанию, изменение атрибутов и восстановление их исходных значений.

```
#include <windows.h>
BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
void        ContAttr(HWND hwnd);
HINSTANCE   hInstance;
char    szClass[ ]="ClassContext";
int WINAPI  WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd1, hwnd2; ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass,COLOR_WINDOW) )
        return FALSE;
    hwnd1 = CreateWindow(szClass, "Первое окно",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if  ( !hwnd1 ) return FALSE;
    hwnd2 = CreateWindow(szClass, "Второе окно",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
```

```
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if ( !hwnd2 ) return FALSE;
    //Устанавливаем исходные атрибуты контекста
    ContAttr(hwnd1);
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}
BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc;  wc.cbClsExtra = wc.cbWndExtra = 0;
    wc.style = CS_HREDRAW | CS_VREDRAW | CS_CLASSDC;
    wc.lpfnWndProc = Proc;          wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = NULL;     wc.lpszClassName = szName;
    return (RegisterClass(&wc) != 0);
}
LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam )
{   static short cx, cy;
    switch (msg)
    {   case WM_SIZE:
        {   cx=LOWORD(lParam);      cy=HIWORD(lParam);
            return 0;
        }
        case WM_PAINT:
        {   char szText[ ]= ·
                "Вывод с атрибутами контекста класса";
            PAINTSTRUCT     ps;
            HDC    hdc=BeginPaint(hwnd,&ps);
            TextOut(hdc,cx/2,cy/2,szText,strlen(szText));
            EndPaint(hwnd,&ps);   return 0;
        }
        case WM_LBUTTONDOWN:
        {   char szText[ ]=
              "Изменяем атрибуты контекста класса и выводим";
            HDC hdc=GetDC(hwnd);
            //Цветом вывода символов выбираем малиновый
            SetTextColor(hdc,RGB(255,0,255));
```

```
            //Цветом фона вывода символов выбираем кремовый
            SetBkColor(hdc,RGB(255,251,240));
            TextOut(hdc,cx/2,cy/2-60,szText,strlen(szText));
            ReleaseDC(hwnd, hdc);
            return 0;
        }
        case WM_RBUTTONDOWN:
        {   //Восстанавливаем атрибуты контекста
            ContAttr(hwnd);
            char szText[ ]=
                "Восстанавливаем атрибуты контекста класса и выводим";
            HDC hdc=GetDC(hwnd);
            TextOut(hdc,cx/2,cy/2+60,szText,strlen(szText));
            ReleaseDC(hwnd, hdc);
            return 0;
        }
        case WM_DESTROY: { PostQuitMessage(0); return 0; }
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
void ContAttr(HWND hwnd)
{   HDC hdc=GetDC(hwnd);
    //Цветом вывода символов устанавливаем синий
    SetTextColor(hdc,RGB(0, 0, 255));
    //Цветом фона вывода символов устанавливаем желтый
    SetBkColor(hdc,RGB(255, 255, 0));
    SetTextAlign(hdc,TA_CENTER);
    ReleaseDC(hwnd, hdc);
}
```

Приложение регистрирует класс со стилем контекста для класса:

```
wc.style = CS_HREDRAW | CS_VREDRAW | CS_CLASSDC;
```

Затем создает два различных окна этого класса с координатами по умолчанию и устанавливает исходные атрибуты контекста класса:

```
ContAttr(hwnd1);
```

При этом не имеет значения, дескриптор которого из окон данного класса служит аргументом этой функции. Для установки атрибутов контекста класса равноправны все окна данного класса.

Оба окна в центре рабочей области содержат один и тот же текст "Вывод с атрибутами контекста класса", выведенный синим шрифтом на желтом фоне.

Нажатие левой клавиши мыши над рабочей областью любого из двух окон приводит к изменению атрибутов контекста класса. В данном случае цвет шрифта изменяется на малиновый, а цвет фона – на кремовый. Выше текста "Вывод с атрибутами контекста класса" отображается новая строка, выведенная новыми атрибутами: "Изменяем атрибуты контекста класса и выводим". Обратите внимание, что ни в одном окне цвета шрифта и фона для строки "Вывод с атрибутами контекста класса" еще не изменились.

Если изменить размеры любого из окон, то в этом окне появится строка "Вывод с атрибутами контекста класса" с изменившимися цветами шрифта и фона. То есть при обработке сообщения WM_PAINT используются новые атрибуты контекста.

Нажатие правой клавиши мыши над рабочей областью любого из двух окон приводит к установке исходных атрибутов контекста класса. С этой целью опять вызываем функцию ContAttr. Ниже строки "Вывод с атрибутами контекста класса" появляется новая строка, выведенная с исходными атрибутами: "Восстанавливаем атрибуты контекста класса и выводим". Обратите внимание, что ни в одном окне цвета шрифта и фона для строки "Вывод с атрибутами контекста класса" еще не изменились.

Если изменить размеры любого из окон, то в этом окне появится строка "Вывод с атрибутами контекста класса" с исходными цветами шрифта и фона. То есть при обработке сообщения WM_PAINT используются исходные атрибуты контекста.

Личный контекст отображения

Этим контекстом обладают окна класса со стилем CS_OWNDC. Личный контекст получают один раз и настраивают его атрибуты, а освобождают только при завершении работы приложения. Функции BeginPaint, EndPaint, GetDC и ReleaseDC не влияют на личный контекст. Личный контекст экономит расход оперативной памяти.

Рассмотрим пример с таким контекстом. В предыдущем приложении стиль класса изменим на класс окон с личным контекстом:

```
wc.style = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
```

Тогда произойдут следующие изменения:

1. При запуске приложения устанавливаются атрибуты контекста только для окна hwnd1:

```
ContAttr(hwnd1);
```

Это можно заметить сразу после запуска. Окно hwnd1 при выводе использует атрибуты личного контекста, установленные функцией Con-

tAttr. Окно hwnd2 использует атрибуты личного контекста, установленные по умолчанию операционной системой.

2. Экспериментируя с окнами, нажимая на клавиши мыши, можно заметить, что изменения атрибутов контекста одного окна никак не изменяют атрибутов вывода в другом окне. То есть каждое окно обладает своим личным контекстом.

Родительский контекст отображения

Этот контекст используют дочерние окна. Он позволяет дочерним окнам "унаследовать" атрибуты контекста отображения у родительского окна. Например, дочернее окно может использовать для вывода текста тот же шрифт и цвета, что и родительское окно.

С этой целью в стиле класса дочерних окон указывают значение CS_PARENTDC.

Контекст отображения для окна

Описанные выше контексты позволяют рисовать только в рабочей области окна. Контекст отображения для окна позволяет рисовать в любом месте окна – в области заголовка, системного меню, рамок, кнопок изменения размера окна и т. п.

Контекст отображения для окна получают с помощью функции GetWindowDC. Далее он используется аналогично общему контексту отображения. При этом начало системы координат находится в левом верхнем углу окна, а не рабочей области. Прототип функции подобен прототипу функции GetDC:

```
HDC GetWindowDC( HWND hwnd );
```

Для освобождения контекста отображения для окна вызывают функцию ReleaseDC.

Для вывода в различные части окна нужно предварительно определить метрики окна с помощью функции GetSystemMetrics.

Задача. Заголовок окна вывести красным шрифтом.

Листинг 3.5. Вывод в область заголовка окна.

```
#include <windows.h>

BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char    szClass[ ] = "MyCaption";

int WINAPI  WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  ::hInstance=hInstance;
```

```
    if ( !RegClass(WndProc, szClass, COLOR_WINDOW) )
        return FALSE;
    hwnd = CreateWindow(szClass, NULL,
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}
BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc;  wc.style = wc.cbClsExtra=wc.cbWndExtra=0;
    wc.lpfnWndProc = Proc;     wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground+1);
    wc.lpszMenuName = NULL;     wc.lpszClassName = szName;
    return (RegisterClass(&wc) != 0);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam )
{   static short nCap, nFr; char szCap[ ]="Мой заголовок окна";
    switch (msg)
    {   case WM_CREATE:
        {   //Определяем высоту заголовка окна
            nCap= GetSystemMetrics(SM_CYCAPTION);
            //Определяем толщину рамки окна
            nFr = GetSystemMetrics(SM_CYFRAME);
            return 0;
        }
        case WM_MOVE:
        case WM_SIZE:
        {   HDC hdc = GetWindowDC(hwnd);
            //Установка атрибутов вывода и вывод текста
            SetTextColor(hdc,RGB(255,0,0));
            SetBkMode( hdc, TRANSPARENT);
            TextOut(hdc, nCap+nFr, 3*nFr/2, szCap, strlen(szCap));
            ReleaseDC(hwnd, hdc);
            return 0;
```

```
        }
        case WM_DESTROY: { PostQuitMessage(0);  return 0; }
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

При создании окна определены высота заголовка и толщина рамки окна:

```
nCap= GetSystemMetrics(SM_CYCAPTION);
nFr = GetSystemMetrics(SM_CYFRAME);
```

При любом изменении размеров и перемещении окна в область заголовка выводится строка "Мой заголовок окна". В соответствии с требованиями задачи установлены красный цвет шрифта и прозрачный режим вывода (текущий фон используется в качестве фона вывода текста):

```
SetTextColor(hdc,RGB(255,0,0));
SetBkMode( hdc, TRANSPARENT);
```

3.3. Установка атрибутов контекста отображения

Контекст отображения, кроме характеристик устройства вывода, содержит указатели на выбранные в контекст инструменты рисования. Инструменты настраивают с помощью атрибутов контекста отображения (их около 20). Например, атрибуты описывают систему координат, задают настройку цвета графических объектов и цвета фона. С их помощью можно выбрать перо для рисования линий и кисть для закрашивания внутренней области замкнутых фигур. При получении контекста отображения атрибуты рисования содержат значения по умолчанию.

Рассмотрим функции установки значений атрибутов.

По умолчанию для **фона операции вывода** в контекст отображения выбран белый цвет. Функция SetBkColor устанавливает новый цвет для фона вывода:

```
COLORREF  SetBkColor( HDC hdc, COLORREF crColor );
```

В случае успешной установки она возвращает значение предыдущего цвета фона. Параметр crColor задает новый цвет фона. Если установлен непрозрачный режим фона вывода, цвет фона crColor используют для заполнения промежутков между линиями изображения или символами текста. Этот цвет используют и при преобразовании разноцветных изображений в моноцветные.

Тип COLORREF равноценен типу DWORD. Для получения значения цвета типа COLORREF существует функция

```
COLORREF RGB(BYTE bRed, BYTE bGreen, BYTE bBlue);
```

Она возвращает значение **цвета в формате COLORREF** в виде интенсивности красной (bRed), зеленой (bGreen) и голубой (bBlue) составляющих цвета. Интенсивность каждого аргумента может принимать значения в диапазоне от 0 до 255. Если все три значения равны нулю, результат – черный. Если все три – 255, результат – белый. В качестве примера в табл. 3.1 перечислены красная, зеленая и голубая составляющие системных цветов.

В случае неудачи возвращаемое значение функции RGB равно CLR_INVALID.

Существует **два режима фона** – непрозрачный (OPAQUE – значение по умолчанию) и прозрачный (TRANSPARENT). В режиме непрозрачного вывода цвет фона текущей позиции заменяется цветом фона вывода. В режиме TRANSPARENT цвет фона не заменяется. Режим фона устанавливает функция SetBkMode:

```
int SetBkMode( HDC hdc, int iBkMode );
```

Параметр iBkMode принимает значение TRANSPARENT или OPAQUE. В случае успешного выполнения функция возвращает значение предыдущего режима, иначе – нуль.

Режим рисования позволяет выводить изображения инвертированием цвета фона, черным или белым цветом и т. д. По умолчанию для рисования используют цвет выбранного в контекст пера. Новый режим рисования устанавливает функция SetROP2:

```
int SetROP2( HDC hdc,     int fnDrawMode );
```

Параметр fnDrawMode определяет режим рисования и принимает значения из табл. 3.2. В случае успешного выполнения функция возвращает значение предыдущего режима рисования, иначе – нуль.

По умолчанию **цвет вывода текста** выбран черным. Другой цвет вывода символов текста устанавливает функция SetTextColor:

```
COLORREF SetTextColor(HDC hdc, COLORREF crColor );
```

Параметр crColor задает новый цвет. В случае успешной установки нового цвета возвращается значение предыдущего цвета, иначе – CLR_INVALID. Этот цвет также используют при преобразовании разноцветных изображений в моноцветные и наоборот.

По умолчанию **расстояние между буквами** равно 0. Расстояние между буквами устанавливает функция SetTextCharacterExtra:

```
int SetTextCharacterExtra( HDC hdc, int nCharExtra );
```

Параметр nCharExtra определяет расстояние между символами в логических единицах. Во всех режимах (кроме MM_TEXT) значение

nCharExtra кратно пикселю. В случае успешного выполнения функция возвращает предыдущий интервал между символами, иначе – 2^{31}.

По умолчанию для **закрашивания замкнутых фигур** выбрана кисть белого цвета. Если нужна другая кисть, то ее сначала создают, а затем выбирают в контекст. Есть две функции создания кисти. В случае успешного выполнения они возвращают дескриптор созданной кисти, иначе – NULL.

Функция CreateSolidBrush создает кисть цвета crColor:

```
HBRUSH CreateSolidBrush( COLORREF crColor );
```

Функция CreateHatchBrush создает штрихованную кисть:

```
HBRUSH CreateHatchBrush( int fnStyle, COLORREF clrref );
```

Параметр fnStyle определяет стиль штриховки и принимает одно из следующих значений:

Значение	*Стиль штриховки*
HS_BDIAGONAL	Линии под углом в 45 градусов
HS_CROSS	В клетку без наклона
HS_DIAGCROSS	В клетку с наклоном в 45 градусов
HS_FDIAGONAL	Линии под углом в 135 градусов
HS_HORIZONTAL	Горизонтальные линии
HS_VERTICAL	Вертикальные линии

Параметр clrref определяет цвет линий штриховки.

В контекст кисть выбирают, вызывая функцию SelectObject:

```
HGDIOBJ SelectObject( HDC hdc, HGDIOBJ hgdiobj );
```

Параметр hgdiobj описывает выбираемый объект (в этом случае – кисть). В случае успешного выполнения функция возвращает дескриптор предыдущего объекта.

Если созданная кисть более не нужна, в контекст выбирают предыдущую кисть, а созданную удаляют вызовом функции DeleteObject:

```
BOOL DeleteObject( HGDIOBJ hgdiobj);
```

Параметр hgdiobj может указывать на перо, кисть, шрифт, изображение, область или палитру. В случае успешного удаления возвращает ненулевое значение.

Пример. Создать кисть в клетку из горизонтальных и вертикальных линий малинового цвета, выбрать ее в контекст, закрасить фигуры и удалить созданную кисть.

Решение этой задачи может иметь следующий вид:

```
HBRUSH hOldBrush, hNewBrush;
hNewBrush =CreateHatchBrush(HS_CROSS, RGB(255,0, 255));
hOldBrush =(HBRUSH)SelectObject(hdc, hNewBrush);
//Здесь могут быть закрашены фигуры
SelectObject(hdc, hOldBrush);
DeleteObject(hNewBrush);
```

По умолчанию **линии рисуют** черным пером шириной в 1 пиксель. Функция CreatePen создает перо указанного стиля, ширины и цвета:

```
HPEN CreatePen( int fnPenStyle, int nWidth, COLORREF crColor );
```

Параметр fnPenStyle определяет стиль пера и принимает одно из следующих значений:

Значение	*Стиль линии пера*
PS_SOLID	Сплошная
PS_DASH	Штриховая
PS_DOT	Пунктирная
PS_DASHDOT	Штрихпунктирная, одна точка на одну черточку
PS_DASHDOTDOT	Штрихпунктирная, две точки на одну черточку
PS_NULL	Невидимая
PS_INSIDEFRAME	Линии для обводки замкнутых фигур

Параметр nWidth задает ширину пера. В зависимости от режима отображения ширина может быть указана в пикселях, долях дюйма или долях миллиметра. Ширина прерывистых линий может быть равна только единице. Параметр crColor задает цвет линий.

В случае успешного выполнения функция CreatePen возвращает дескриптор созданного пера, иначе – NULL.

Обратите внимание на особенности рисования толстых линий. Линия стиля PS_SOLID располагается по обе стороны базовой линии, указанной в функции рисования линии. Причем концы линии всегда закругляются. Линия стиля PS_INSIDEFRAME всегда расположена с внутренней стороны обводящей линии.

В контекст перо выбирают, вызывая функцию SelectObject. Если созданное перо более не нужно, в контекст выбирают предыдущее перо. Созданное перо удаляют вызовом функции DeleteObject.

Пример. Создать перо для рисования сплошных линий шириной в 3 пикселя голубого цвета, выбрать его в контекст, нарисовать линии и удалить созданное перо.

Решение этой задачи может иметь следующий вид:

```
HPEN hOldPen, hNewPen;
hNewPen =CreatePen(PS_SOLID, 3, RGB(0, 255, 255));
hOldPen =(HPEN)SelectObject(hdc, hNewPen);
//Здесь могут быть нарисованы линии
SelectObject(hdc, hOldPen);
DeleteObject(hNewPen);
```

По умолчанию **текущая позиция пера** равна (0, 0). Многие из функций рисования линий вывод начинают с текущей позиции. Для установки другой позиции пера вызывают функцию MoveToEx:

```
BOOL MoveToEx( HDC hdc, int X, int Y, LPPOINT lpPoint );
```

Параметры X и Y задают новую позицию в логических единицах, lpPoint указывает на структуру типа POINT:

```
typedef struct
{   LONG  x;
    LONG  y;
}   POINT;
```

В поля **x** и **y** этой структуры после вызова этой функции будут записаны координаты предыдущей позиции. При lpPoint=NULL предыдущая позиция не сохраняется.

В случае успешного перемещения функция возвращает ненулевое значение.

Режим отображения влияет на систему координат. Приложение может изменить направление и масштаб координатных осей.

По умолчанию установлен режим отображения MM_TEXT. В этом режиме начало системы координат находится в левом верхнем углу рабочей области, ось **x** направлена вправо, ось **y** – вниз, позиции отсчитывают в пикселях.

Режим отображения изменяют вызовом функции SetMapMode:

```
int SetMapMode( HDC hdc, int fnMapMode );
```

Параметр fnMapMode задает новый режим отображения и может принимать одно из следующих значений:

Значение	*Описание*
MM_ANISOTROPIC	Произвольные направления для осей координат, произвольные логические единицы
MM_HIENGLISH	Логическая единица равна 0.001 дюйма. Ось x направлена вправо, ось y – вверх
MM_HIMETRIC	Логическая единица равна 0.01 миллиметра. Ось x направлена вправо, ось y – вверх

MM_ISOTROPIC	Логические единицы произвольны и одинаковы по обеим осям
MM_LOENGLISH	Логическая единица равна 0.01 дюйма. Ось x направлена вправо, ось y – вверх
MM_LOMETRIC	Логическая единица равна 0.1 миллиметра. Ось x направлена вправо, ось y – вверх
MM_TEXT	Логические единицы равны размерам пикселя. Ось x направлена вправо, ось y – вниз
MM_TWIPS	Логическая единица равна двадцатой части точки принтера (1/1440 дюйма). Ось x направлена вправо, ось y – вверх

В режимах MM_ANISOTROPIC и MM_ISOTROPIC для установки ориентации осей и единиц вызывают функции SetWindowExtEx и SetViewport ExtEx.

В случае успешного выполнения функция возвращает значение предыдущего режима отображения, иначе – нуль.

По умолчанию **начало системы координат** установлено в точку (0,0). Для перемещения начала системы координат окна вызывают функцию SetWindowOrg:

```
BOOL SetWindowOrgEx( HDC hdc, int x, int y, LPPOINT lpPoint );
```

Параметры функции SetWindowOrgEx подобны параметрам функции MoveToEx.

Коэффициенты масштабирования по координатным осям равны 1:1 по умолчанию. Для обеспечения аппаратной независимости приложения Windows работают с логическими координатами, которые затем преобразуются в физические. При этом можно задавать размеры в миллиметрах или дюймах или устанавливать любые коэффициенты масштабирования.

Для изменения масштабов последовательно вызывают две функции: сначала функцию SetWindowExtEx, а потом SetViewportExtEx.

Синтаксис функции SetWindowExtEx:

```
BOOL SetWindowExtEx( HDC hdc, int nxWinExt, int nyWinExt, LPSIZE lpSize );
```

Параметры nxWinExt и nyWinExt задают горизонтальный и вертикальный размеры окна в логических единицах. Параметр lpSize указывает на структуру типа SIZE:

```
typedef struct
{  LONG  cx;
   LONG  cy;
}  SIZE;
```

В эту структуру будут записаны значения предыдущих логических размеров окна (если lpSize не равен NULL).

Синтаксис функции SetViewportExtEx:

```
BOOL SetViewportExtEx (HDC hdc, int nxViewExt, int nyViewExt, LPSIZE lpSize );
```

Параметры nxViewExt и nyViewExt определяют горизонтальный и вертикальный размеры окна в физических единицах. Параметр lpSize указывает на структуру типа SIZE. В эту структуру будут записаны предыдущие физические размеры окна (если lpSize не равен NULL).

Тогда логические координаты точки (xWin, yWin) в физические координаты (xView, yView) преобразуют по формулам:

$$xView = (xWin - xWin0) * \frac{nxViewExt}{nxWinExt} + xView0,$$

$$yView = (yWin - yWin0) * \frac{nyViewExt}{nyWinExt} + yView0,$$

где (xWin0, yWin0) и (xView0, yView0) указывают на смещение соответственно логической и физической системы координат.

Отсюда видно, что значения отношений

$$\frac{nxViewExt}{nxWinExt} \quad и \quad \frac{nyViewExt}{nyWinExt}$$

и есть устанавливаемые значения коэффициентов масштабирования.

При таком подходе обеспечивается большой выбор коэффициентов масштабирования и направлений осей координат. Действительно, значения nxWinExt и nyWinExt, а также nxViewExt и nyViewExt могут быть любыми допустимыми значениями типа int. Вдобавок, меняя знак коэффициента масштабирования, можно изменить направление оси координат.

Смещение физической и логической систем координат по умолчанию равно (0, 0). Для установки смещения используют функции SetViewportOrg Ex и SetWindowOrgEx.

Функция SetViewportOrgEx устанавливает новое начало (xView0, yView0) физической системы координат:

```
BOOL SetViewportOrgEx( HDC hdc, int xView0, int  yView0, LPPOINT lpPoint );
```

Значения xView0 и yView0 задают в физических единицах. В структуру, адрес которой передается через параметр lpPoint, записываются старые координаты начала системы координат. Функция возвращает TRUE в случае успеха и FALSE при возникновении ошибки.

Функция SetWindowOrgEx устанавливает начало (xWin0, yWin0) логической системы координат:

```
BOOL SetWindowOrgEx(HDC hdc, int xWin0, int yWin0, LPPOINT lpPoint);
```

Значения xWin0 и yWin0 задают в логических единицах. В структуру, адрес которой передается через параметр lpPoint, записываются старые координаты начала системы координат. Функция возвращает TRUE в случае успеха и FALSE при возникновении ошибки.

Рекомендуется изменять только одно начало отсчета – либо физических, *либо логических координат*.

Пример. Создать систему координат с началом отсчета в левом нижнем углу окна. Ось **x** направить слева направо, а ось **y** – снизу вверх. Логические значения высоты и ширины изменять от 0 до 1000. Установить одинаковый масштаб по осям **x** и **y**.

Следующий фрагмент кода решает эту задачу:

```
SetMapMode(hdc, MM_ISOTROPIC);
SetWindowExt(hdc, 1000, 1000);
SetViewportExt(hdc, cxClient, -cyClient);
SetViewportOrg(hdc, 0, cyClient);
```

3.4. Вывод текста

3.4.1. Настройка параметров шрифта

В контекст отображения по умолчанию выбирают системный шрифт. Для создания логического шрифта удобно вызвать функцию CreateFont Indirect:

```
HFONT WINAPI CreateFontIndirect(const LOGFONT FAR* lplf);
```

Она возвращает дескриптор созданного шрифта.

В качестве аргумента вызова функции CreateFontIndirect передают указатель на структуру типа LOGFONT:

```
typedef struct
{   LONG    lfHeight;
    LONG    lfWidth;
    LONG    lfEscapement;
    LONG    lfOrientation;
    LONG    lfWeight;
    BYTE    lfItalic;
    BYTE    lfUnderline;
    BYTE    lfStrikeOut;
    BYTE    lfCharSet;
    BYTE    lfOutPrecision;
    BYTE    lfClipPrecision;
    BYTE    lfQuality;
    BYTE    lfPitchAndFamily;
```

```
    TCHAR    lfFaceName[LF_FACESIZE];
}   LOGFONT;
```

Назначение полей этой структуры:

1. **lfHeight** – высота шрифта в логических единицах. Положительное значение поля lfHeight задает высоту ячейки вывода букв. Если lfHeight<0, то абсолютная величина значения lfHeight задает высоту символов. При lfHeight=0 создают шрифт с высотой символов по умолчанию: lfHeight=12.
2. **lfWidth** – ширина символов в логических единицах. Если указано нулевое значение, используют ширину по умолчанию.
3. **lfEscapement** – угол (в направлении против часовой стрелки) в десятых долях градуса между линией вывода строки и координатной осью X. Значение lfEscapement может отличаться от нуля только для масштабируемых и векторных шрифтов.
4. **lfOrientation** – угол (в направлении против часовой стрелки) в десятых долях градуса между линией основания символа и координатной осью X. Рекомендуется задавать равным значению lfEscapement.
5. **lfWeight** – вес шрифта. Определяет жирность символов шрифта и может находиться в пределах от 0 до 1000. Принимает значения констант из следующей таблицы:

Константа	*Вес*	*Константа*	*Вес*	*Константа*	*Вес*
FW_DONTCARE	0	FW_NORMAL	400	FW_BOLD	700
FW_THIN	100	FW_REGULAR	400	FW_EXTRABOLD	800
FW_EXTRALIGHT	200	FW_MEDIUM	500	FW_ULTRABOLD	800
FW_ULTRALIGHT	200	FW_SEMIBOLD	600	FW_BLACK	900
FW_LIGHT	300	FW_DEMIBOLD	600	FW_HEAVY	900

Многие шрифты содержат символы только весов FW_NORMAL, FW_REGULAR (нормальный) и FW_BOLD (полужирный). Если задать нулевое значение (lfWeight=0), используют вес по умолчанию.

6. Если **lfItalic** не равен 0, запрашивают шрифт с наклонными буквами.
7. Если **lfUnderline** не равен 0, запрашивают шрифт с подчеркиванием букв.
8. Если **lfStrikeOut** не равен 0, запрашивают шрифт с перечеркнутыми буквами.
9. **lfCharSet** – наиболее важное поле структуры. Оно задает набор требуемых символов. Может принимать значения констант из следующей таблицы:

Константа	*Описание*
ANSI_CHARSET	Набор символов в кодировке ANSI
DEFAULT_CHARSET	Задают для запроса логического шрифта
SYMBOL_CHARSET	Символьный шрифт (например, Wingdings)
SHIFTJIS_CHARSET	Нужен для работы с японской версией Windows
OEM_CHARSET	Набор символов в кодировке OEM

Если lfCharSet=0, будет выбран шрифт ANSI_CHARSET.

10. **lfOutPrecision** задает необходимую степень соответствия между параметрами запрашиваемого и предоставляемого шрифта. Можно указывать одну из следующих констант:

Константы	*Описание*
OUT_DEFAULT_PRECIS	Степень соответствия по умолчанию
OUT_STRING_PRECIS или OUT_CHARACTER_PRECIS	Выбрать шрифт с наибольшим соответствием в размерах символов
OUT_STROKE_PRECIS	Искать шрифт с полным соответствием
OUT_TT_PRECIS	Предпочтение шрифтам True Type
OUT_DEVICE_PRECIS	Выбрать шрифт устройства вывода
OUT_RASTER_PRECIS	Выбрать растровый шрифт
OUT_TT_ONLY_PRECIS	Выбирать только шрифты True Type

11. **lfClipPrecision** задает способ обрезания изображения символа на границе вывода. Можно использовать следующие константы: CLIP_STROKE_PRECIS, CLIP_MASK, CLIP_DEFAULT_PRECIS, CLIP_LH_ANGLES, CLIP_TT_ALWAYS, CLIP_EMBEDDED и CLIP_CHARACTER_PRECIS. Если указана CLIP_LH_ANGLES, направление вращения текста зависит от установленного режима отображения.

12. **lfQuality** задает качество отображения символов. Можно указывать одну из следующих констант:

Константа	*Описание*
DEFAULT_QUALITY	Качество не имеет значения
DRAFT_QUALITY	Низкое качество
PROOF_QUALITY	Высокое качество

13. **lfPitchAndFamily** задает ширину символов и определяет семейство шрифта. Фиксированная или переменная ширина символов задается при помощи следующих констант:

Константа	*Описание*
DEFAULT_PITCH	Не имеет значения, будет ли шрифт иметь фиксированную или переменную ширину символов
FIXED_PITCH	Нужен шрифт фиксированной ширины символов
VARIABLE_PITCH	Нужен шрифт с переменной шириной символов

Следующие константы задают семейство шрифта:

Константа	*Описание*
FF_DECORATIVE	Шрифт, содержащий маленькие рисунки (например, Wingdings)
FF_DONTCARE	Семейство шрифта не имеет значения
FF_MODERN	Семейство Modern. Фиксированная ширина символов, могут быть засечки (но могут и не быть)
FF_ROMAN	Семейство Roman. Переменная ширина букв, есть засечки
FF_SCRIPT	Семейство Script. Рукописный шрифт
FF_SWISS	Семейство Swiss. Переменная ширина букв, нет засечек

14. **lfFaceName** – строка с именем шрифта. Длина этой строки не должна превышать 32 символа. Если lfFaceName – пустая строка, используют шрифт, который соответствует по другим атрибутам.

Получив запрос, GDI начинает искать шрифт, сравнивая последовательно поля lfCharSet, lfPitchAndFamily и lfFaceName. После сравнения этих полей GDI сравнивает высоту букв шрифта, затем поля lfWidth, lfItalic, lfUnderline и lfStrikeOut.

3.4.2. Выбор шрифта в контекст отображения

Пусть в структуре типа LOGFONT заполнены нужные поля и ее адрес передан функции CreateFontIndirect. Тогда эта функция вернет дескриптор созданного шрифта hfont.

Для выбора шрифта hfont в контекст отображения hdc вызывают функцию SelectObject:

```
SelectObject(hdc, hfont);
```

Как только в шрифте hfont отпадет необходимость, его следует удалить при помощи макрокоманды DeleteObject, предварительно выбрав в контекст отображения предыдущий шрифт.

3.4.3. Функции вывода текста

Функция TextOut выводит символьную строку в указанной позиции, соблюдая выбранные атрибуты контекста отображения.

Функция TextOut объявлена следующим образом:

```
BOOL TextOut( HDC hdc,  int nXStart, int nYStart, LPCTSTR lpString, int cbString  );
```

Параметры этой функции:

1. **nXStart** задает логическую x-координату операции вывода.
2. **nYStart** задает логическую y-координату операции вывода.
3. **lpString** указывает на строку, которая будет выведена.
4. **cbString** равен количеству символов в выводимой строке.

Функция TextOut заданную строку выводит внутри воображаемого прямоугольника, сформированного ячейками символов выводимой строки. Координаты nXStart и nYStart, в зависимости от режима выравнивания, определяют различные точки этого прямоугольника.

Функция SetTextAlign выбирает **режим выравнивания** текста в *контекст отображения*:

```
UINT SetTextAlign( HDC hdc, UINT fMode );
```

Параметр fMode задает режим выравнивания и определяется при помощи трех групп флагов. Символические имена флагов начинаются с префикса TA_.

Первая группа флагов отвечает за выравнивание строки по горизонтали:

Флаг	*Описание*
TA_LEFT	Выравнивать по левой границе. Координата nXStart определяет левую границу воображаемого прямоугольника. **Этот режим выбран по умолчанию**
TA_CENTER	Выравнивать по центру. Координата nXStart определяет центр воображаемого прямоугольника
TA_RIGHT	Выравнивать по правой границе. Координата nXStart определяет правую границу воображаемого прямоугольника

Вторая группа флагов отвечает за выравнивание текста по вертикали:

Флаг	*Описание*
TA_TOP	Выравнивать по верхней границе. Координата nYStart определяет верхнюю границу воображаемого прямоугольника. **Этот режим выбран по умолчанию**

TA_BASELINE	Выравнивать по базовой линии выбранного шрифта
TA_BOTTOM	Выравнивать по нижней границе

Третья группа флагов относится к текущей позиции вывода:

Флаг	*Описание*
TA_NOUPDATECP	Не изменять значения текущей позиции вывода текста (**используется по умолчанию**)
TA_UPDATECP	После вывода вычислить новое значение текущей позиции

Из каждой группы флагов используют только один, например:

```
SetTextAlign(hdc, TA_CENTER | TA_BASELINE | TA_UPDATECP);
```

Функция SetTextAlign возвращает старое значение режима выравнивания или, в случае ошибки, – GDI_ERROR.

Если задан режим выравнивания TA_UPDATECP, функция TextOut начинает вывод текста с текущей позиции, игнорируя параметры, определяющие расположение текста в окне.

3.4.4. Пример вывода текста в окно

В следующем примере показано, как создать и выбрать шрифт с заданными характеристиками. Далее в левом нижнем углу окна приложения с различными наклоном, цветами фона и текста 10 раз выводится один и тот же текст.

Листинг 3.6. Пример вывода текста.

```
#include <windows.h>
BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char    szClass[ ]="TextOutClass";

int WINAPI  WinMain(HINSTANCE hInstance,
HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass,COLOR_WINDOW) )
return FALSE;
    hwnd = CreateWindow(szClass, "Вывод текста",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
```

```
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
UINT brBackground)
{   WNDCLASS wc;  wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.cbClsExtra = wc.cbWndExtra = 0; wc.lpfnWndProc = Proc;
    wc.hInstance = hInstance;  wc.lpszClassName = szName;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = NULL;     return (RegisterClass(&wc) != 0);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
WPARAM wParam, LPARAM lParam )
{   char    szFont[ ]="     Какой-либо текст";
    static  short cyClient;
    switch (msg)
    {   case WM_SIZE:    { cyClient=HIWORD(lParam); return 0; }
        case WM_PAINT:
        {   PAINTSTRUCT     ps;
            HDC    hdc=BeginPaint(hwnd,&ps);
            static   LOGFONT lf;
            lf.lfCharSet=DEFAULT_CHARSET;
            lf.lfPitchAndFamily=DEFAULT_PITCH;
            strcpy(lf.lfFaceName,"Times New Roman");
            lf.lfHeight=20;  lf.lfWeight=FW_BOLD;
            for (int i=0;i<10;i++)
            {   lf.lfOrientation=lf.lfEscapement=i*100;
                HFONT hNFont=CreateFontIndirect(&lf);
                HFONT hOFont=(HFONT)SelectObject(hdc,hNFont);
                SetTextColor(hdc,RGB(i*15, i*20, i*25));
                SetBkColor(hdc,RGB(255-i*15, 255-i*20,255-i*25));
                TextOut(hdc,0,cyClient-30,szFont,strlen(szFont));
                SelectObject(hdc,hOFont);
                DeleteObject(hNFont);
            }
            EndPaint(hwnd,&ps);
            return 0;
        }
```

```
        case WM_DESTROY: { PostQuitMessage(0); return 0; }
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

В ветви обработки сообщения WM_PAINT создают логический шрифт с именем lf:

```
static   LOGFONT lf;
```

и задают значения его полей. Слово static гарантирует, что изначально все поля инициированы нулем.

Далее определяют значения тех полей, которые описывают нужные характеристики шрифта. Причем часть полей определяют в теле цикла для разных значений i: lf.lfOrientation=lf.lfEscapement=i*100. При i=9 угол наклона базовой линии шрифта будет равен 90 градусам.

После задания нужных характеристик логического шрифта создают шрифт и выбирают его в контекст отображения:

```
hNFont=CreateFontIndirect(&lf);
hOFont=(HFONT)SelectObject(hdc,hNFont);
```

Затем устанавливают цвет вывода текста и цвет фона операции вывода:

```
SetTextColor(hdc,RGB(i*15, i*20, i*25));
SetBkColor(hdc, RGB(255-i*15, 255-i*20, 255-i*25));
```

Этим завершают процедуру установки атрибутов контекста отображения. Теперь можно вывести текст:

```
TextOut( hdc, 0, cyClient-30, szFont, strlen(szFont) );
```

Шрифт hNFont больше не нужен. Поэтому в контекст выбираем предыдущий шрифт и удаляем шрифт hNFont:

```
SelectObject(hdc,hOFont);
DeleteObject(hNFont);
```

Освобождают контекст отображения:

```
EndPaint(hwnd,&ps);
```

3.4.5. Определение метрик шрифта

Метрику шрифта, выбранного в контекст отображения hdc, определяют с помощью функции GetTextMetrics:

```
BOOL GetTextMetrics( HDC hdc, LPTEXTMETRIC lptm );
```

Параметр lptm указывает на структуру типа TEXTMETRIC, в которую нужно записать метрики шрифта. Все размеры записываются в логических единицах контекста отображения hdc. В случае успешного выполнения функция возвращает ненулевое значение.

Структура TEXTMETRIC предназначена для описания базовой информации о метриках шрифта и описана следующим образом:

```
typedef struct
{   LONG  tmHeight;
    LONG  tmAscent;
    LONG  tmDescent;
    LONG  tmInternalLeading;
    LONG  tmExternalLeading;
    LONG  tmAveCharWidth;
    LONG  tmMaxCharWidth;
    LONG  tmWeight;
    LONG  tmOverhang;
    LONG  tmDigitizedAspectX;
    LONG  tmDigitizedAspectY;
    BCHAR    tmFirstChar;
    BCHAR    tmLastChar;
    BCHAR    tmDefaultChar;
    BCHAR    tmBreakChar;
    BYTE  tmItalic;
    BYTE  tmUnderlined;
    BYTE  tmStruckOut;
    BYTE  tmPitchAndFamily;
    BYTE  tmCharSet;
}   TEXTMETRIC;
```

Назначение полей этой структуры:

1. **tmHeight** – общая высота букв, равна tmAscent+tmDescent.
2. **tmAscent** – часть высоты букв от базовой линии с учетом таких элементов, как тильда в букве "Й".
3. **tmDescent** – часть высоты букв ниже базовой линии.
4. **tmInternalLeading** – высота таких выступающих элементов, как тильда в букве "Й", и может быть равна нулю.
5. **tmExternalLeading** – высота межстрочного интервала, рекомендуемая разработчиком шрифта, может быть приравнена нулю.
6. **tmAveCharWidth** – средняя ширина строчных букв, равна ширине латинской буквы "x".
7. **tmMaxCharWidth** – ширина самой широкой буквы.
8. **tmWeight** – жирность шрифта, может находиться в пределах от 0 до 1000, как и для логического шрифта.

9. **tmOverhang** – величина изменения ширины символов при построении наклонного или полужирного шрифта из нормального шрифта.
10. **tmDigitizedAspectX** – разрешение устройства отображения по горизонтали.
11. **tmDigitizedAspectY** – разрешение устройства отображения по вертикали.
12. **tmFirstChar** – код первого символа в шрифте.
13. **tmLastChar** – код последнего символа в шрифте.
14. **tmDefaultChar** – код символа, заменяющего любой отсутствующий в шрифте символ.
15. **tmBreakChar** – код символа переноса слов с одной строки на другую при выравнивании текста.
16. **tmItalic**≠0 – означает наклонный шрифт.
17. **tmUnderlined**≠0 – означает подчеркнутый шрифт.
18. **tmStruckOut**≠0 – означает перечеркнутый шрифт.
19. **tmPitchAndFamily** – код семейства шрифта. Четыре младших бита этого кода комбинацией следующих констант определяют информацию о шаге и типе шрифта:

Константа	*Пояснение*
TMPF_FIXED_PITCH	Если этот бит установлен, то шрифт с переменным шагом, иначе – с постоянным
TMPF_VECTOR	Если этот бит установлен, то шрифт векторного типа
TMPF_TRUETYPE	Если этот бит установлен, то шрифт типа TrueType
TMPF_DEVICE	Если этот бит установлен, шрифт определен устройством

Четыре старших бита кода tmPitchAndFamily описывают семейство шрифта, так же как и для логического шрифта.

20. **tmCharSet** – код используемого набора символов, такой же, как и для логического шрифта.

Задача. В рабочей области окна вывести таблицу. В заголовке перечислить названия функций, а в строках таблицы отобразить значения этих функций. Причем вывод таблицы ограничить количеством столбцов и строк, которые полностью помещаются в рабочей области.

Следующее приложение решает эту задачу.

Листинг 3.7. Вывод таблицы в окно.

```
#include <windows.h>
#include <math.h>

BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char    szClass[ ] = "TableClass";

//Структура столбца данных
typedef struct
{   char      str[15];   //Поле имени столбца
    double    val[50];   //Массив данных столбца
}   COLUMN;              //Имя типа

int WINAPI  WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd; ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass, COLOR_WINDOW) )
        return FALSE;
    hwnd = CreateWindow(szClass, "ТАБЛИЦА",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
        UINT brBackground)
{   WNDCLASS wc; wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.cbClsExtra=wc.cbWndExtra=0; wc.lpfnWndProc = Proc;
    wc.hInstance = hInstance;  wc.lpszClassName = szName;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground+1);
    wc.lpszMenuName = NULL;     return (RegisterClass(&wc) != 0);
}

BOOL DrawLine( HDC hdc, int x0, int y0, int x, int y)
{   MoveToEx(hdc,x0,y0, NULL);
```

```
    return LineTo(hdc, x, y);
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
WPARAM wParam, LPARAM lParam)
{   //Описываем массив из 10 столбцов
    static COLUMN     cols[10];
    static int cx, cy, cxChar, cyChar;
    switch (msg)
    {   case WM_SIZE:
        {   cx=LOWORD(lParam); cy=HIWORD(lParam);
            return 0;
        }
        case WM_CREATE:
        {   //Заполняем заголовки столбцов
            strcpy(cols[0].str,"i");
            strcpy(cols[1].str,"10*sin(i/10.)");
            strcpy(cols[2].str,"20*sin(i/20.)");
            strcpy(cols[3].str,"30*sin(i/30.)");
            strcpy(cols[4].str,"40*sin(i/40.)");
            strcpy(cols[5].str,"50*sin(i/50.)");
            strcpy(cols[6].str,"60*sin(i/60.)");
            strcpy(cols[7].str,"70*sin(i/70.)");
            strcpy(cols[8].str,"80*sin(i/80.)");
            strcpy(cols[9].str,"90*sin(i/90.)");
            //Заполняем массивы данных столбцов
            for (int i=0; i<50; i++)
            {   //В первом столбце будут номера строк
                cols[0].val[i]=i;
                //В остальных столбцах будут данные
                for (int j=1; j<10; j++)
                    cols[j].val[i]=10*sin(i/10./j);
            }
            //Определяем средние высоту и ширину символов
            {   TEXTMETRIC tm;
                HDC    hdc=GetDC(hwnd);
                //Определяем метрики текста
                GetTextMetrics(hdc, &tm);
                ReleaseDC(hwnd,hdc);
                //Количество пикселей в высоте символа
                cyChar = tm.tmHeight + tm.tmExternalLeading;
```

```
            //Количество пикселей в средней ширине
            cxChar=tm.tmAveCharWidth+1;
        }
        return 0;
    }
    case WM_PAINT:
    {   PAINTSTRUCT ps;
        HDC hdc=BeginPaint(hwnd,&ps);
        char str[20];
        //Установим начальные параметры для вывода
        int  left=cxChar,          //Левый край
             top=cyChar/2          //Верх
             dx=cxChar,            //Пробел по оси X
             dy=cyChar/4,          //Пробел по оси Y
             hy=cyChar+dy+dy,      //Высота для строки
             right=cx-cxChar,      //Правый край
             bottom=cy-cyChar,     //Низ
             bounds[10],           //Массив ширин столбцов
             i=0, j=0;
        //Заполняем массив ширинами столбцов
        while (j<10)
        {   //Для j-го столбца выбираем ширину
            bounds[j]=strlen(cols[j].str);
            for (i=0; i<50; i++)
            {   _gcvt( cols[j].val[i], 7, str );
                int ss=strlen(str);
                if (bounds[j]<ss) bounds[j]=ss;
            }
            if (bounds[j]<=3) bounds[j]=4;
            if (bounds[j]>10) bounds[j]-=2;
            bounds[j] = cxChar*(bounds[j]);
            j++;
        }
        //Определяем максимальное количество столбцов,
        //которое помещается на ширине рабочей области
        int dd=left, maxcol=0;
        while ( maxcol<10 )
        {   if ( dd+bounds[maxcol]>right ) break;
            dd += bounds[maxcol];
            maxcol++;
        }
```

```
//right теперь указывает на правый край таблицы
right=dd;

//Подгоняем ширину окна к количеству столбцов
{   RECT rc; GetWindowRect(hwnd, &rc);
    MoveWindow(hwnd, rc.left, rc.top,
        //Изменяем только ширину окна
        rc.right-rc.left-(cx-right)+dx,
        rc.bottom-rc.top, TRUE);
}

//////НАЧАЛО ВЫВОДА ТАБЛИЦЫ//////
//Устанавливаем режим выравнивания
SetTextAlign(hdc, TA_RIGHT);
//////Начало вывода шапки таблицы//////
int  y=top;  //Текущая координата по оси Y
//Горизонтальная линия на всю ширину
DrawLine(hdc,left,y,right,y);
//Заголовки столбцов
int x=left;   //Текущая координата по оси X
//Вертикальная линия слева от столбца
DrawLine(hdc, x, y, x, y+hy);
for (j=0;j<maxcol; j++)
{   //Устанавливаем x на правой границе столбца
    x+=bounds[j];
    TextOut(hdc,x-dx,y+dy,  cols[j].str,strlen(cols[j].str));
    //Вертикальная линия справа от столбца
    DrawLine(hdc,x,y,x,y+hy);
}
//Горизонтальная линия на всю ширину
y+=hy;  DrawLine(hdc, left,y,right,y);
//////Конец вывода шапки таблицы//////
//////Начало вывода данных таблицы//////
i=0;//Счетчик номера строки данных
while ( i<50 && y<bottom )
{   //Вертикальная линия слева от столбца
    x=left;  DrawLine(hdc,x,y,x,y+hy);
    for (j=0;j<maxcol; j++)
    {   x+=bounds[j];
        //Преобразуем целое число в строку
        if (j==0)     _itoa((int)cols[j].val[i], str, 10);
        //Преобразуем вещественное число в строку
```

```
                else        _gcvt( cols[j].val[i], 7, str );
                TextOut(hdc,x-dx,y+dy,str,strlen(str));
                //Вертикальная линия справа от столбца
                DrawLine(hdc,x,y,x,y+hy);
            }
            //Горизонтальная линия на всю ширину
            y+=hy; DrawLine(hdc,left,y,right,y);
            i++;
        }
        //////Конец вывода данных таблицы//////
        //////КОНЕЦ ВЫВОДА ТАБЛИЦЫ//////
        EndPaint(hwnd,&ps);        return 0;
    }
    case WM_DESTROY: { PostQuitMessage(0); return 0; }
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Рассмотрим основные шаги решения данной задачи.

1. Таблицу сначала создают в памяти.

   1.1. Описывают тип структуры столбца данных:

```
typedef struct
{  char   str[15];     //Поле имени столбца
   double val[50];     //Массив данных столбца
}  COLUMN;             //Имя типа
```

   1.2. Описывают массив cols из 10 столбцов этого типа:

```
static  COLUMN  cols[10];
```

   1.3. Заполняют этот массив некоторыми значениями. Сначала заполняются заголовки столбцов:

```
strcpy(cols[0].str,"i");
strcpy(cols[1].str,"10*sin(i/10.)");
...
strcpy(cols[9].str,"90*sin(i/90.)");
```

Затем заполняют массивы данных столбцов:

```
for (int i=0; i<50; i++)
{  cols[0].val[i]=i;
   for (int j=1; j<10; j++) cols[j].val[i]=10*sin(i/10./j);
}
```

При этом в первый столбец записывают номера строк, а в другие столбцы записывают значения функций.

2. Определяют геометрические параметры вывода таблицы в рабочую область.

   2.1. Определяют среднюю высоту и ширину символов установленного шрифта:

```
TEXTMETRIC tm;  HDC    hdc=GetDC(hwnd);
//Определяем метрики текста
GetTextMetrics(hdc, &tm);
ReleaseDC(hwnd,hdc);
//Количество пикселей в высоте символа
cyChar = tm.tmHeight + tm.tmExternalLeading;
//Количество пикселей в средней ширине
cxChar=tm.tmAveCharWidth+1;
```

   Метрики шрифта определяют на этапе создания окна потому, что используют шрифт по умолчанию. Иначе метрики нужно определять после выбора шрифта в контекст отображения.

   2.2. Устанавливают границы вывода таблицы. Здесь left указывает на левую, top – на верхнюю, right – на правую, bottom – на нижнюю границу таблицы в рабочей области. Параметр dx равен величине пробела по оси X, а dy – по оси Y, hy задает высоту строки таблицы. Также описывают переменные bounds[10], i и j.

   2.3. Массив bounds заполняют значениями ширины столбцов:

```
while (j<10)
{   bounds[j]=strlen(cols[j].str);
    for (i=0; i<50; i++)
    {   _gcvt( cols[j].val[i], 7, str );
        int ss=strlen(str);
        if (bounds[j]<ss) bounds[j]=ss;
    }
    if (bounds[j]<=3) bounds[j]=4;
    if (bounds[j]>10) bounds[j]-=2;
    bounds[j] = cxChar*(bounds[j]);
    j++;
}
```

   Для j-го столбца вычисляют количество символов заголовка:

```
bounds[j]=strlen(cols[j].str);
```

   Далее определяют максимально необходимое количество символов для вывода в этот столбец:

```
_gcvt( cols[j].val[i], 7, str );
int ss=strlen(str);        if (bounds[j]<ss) bounds[j]=ss;
```

После определения необходимого количества символов вносят незначительные коррекции. Например, в столбце выделяют место для вывода не менее четырех символов:

```
if (bounds[j]<=3) bounds[j]=4;
```

А в случае большого (более 10) количества символов уменьшают количество выделяемых позиций:

```
if (bounds[j]>10) bounds[j]-=2;
```

Вычисляют ширину столбца в пикселях:

```
bounds[j] = cxChar*bounds[j];
```

2.4. Определяют максимальное количество maxcol столбцов, которое помещается в установленных границах:

```
int dd=left, maxcol=0;
while ( maxcol<10 )
{   if ( dd+bounds[maxcol]>right ) break;
    dd += bounds[maxcol];      maxcol++;
}
```

2.5. Значение right переносят на правую границу столбца maxcol:

```
right=dd;
```

2.6. Уменьшают ширину окна так, чтобы она была достаточна для вывода только maxcol столбцов:

```
RECT rc; GetWindowRect(hwnd, &rc);
MoveWindow(hwnd, rc.left, rc.top,
   rc.right-rc.left-(cx-right)+dx, rc.bottom-rc.top, TRUE);
```

3. *Выводят таблицу.*

3.1. *Устанавливают режим выравнивания по правой границе:*

```
SetTextAlign(hdc, TA_RIGHT);
```

Обратите внимание, что в этом режиме горизонтальная координата вывода текста указывает на точку, где завершается вывод правого конца выводимой строки.

3.2. Рисуют заголовки столбцов таблицы.

В первую очередь рисуют горизонтальную линию на всю ширину таблицы (от левой границы – left до правой границы – right):

```
DrawLine(hdc,left,y,right,y);
```

Для рисования заголовков столбцов вспомогательную координату по оси X устанавливают на левой границе (x=left). После этого рисуют вертикальную линию на левой границе таблицы высотой hy:

```
DrawLine(hdc, x, y, x, y+hy);
```

Вывод заголовка j-го столбца содержит следующие шаги:

3.2.1. Устанавливают x на правой границе столбца:

```
x+=bounds[j];
```

3.2.2. Выводят текст заголовка j-го столбца:

```
TextOut( hdc, x-dx, y+dy, cols[j].str, strlen(cols[j].str) );
```

При этом координата x-dx указывает на правую границу j-го столбца минус величина пробела по оси X.

3.2.3. Рисуют вертикальную линию справа от j-го столбца:

```
DrawLine( hdc, x, y, x, y+hy );
```

После вывода всех maxcol заголовков рисуют горизонтальную линию на всю ширину таблицы:

```
y+=hy; DrawLine(hdc, left,y,right,y);
```

Этим завершают вывод заголовков таблицы.

3.3. Выводят данные таблицы. Этот шаг практически ничем не отличается от шага 3.2. Здесь добавляют счетчик номера строки данных и операторы преобразования числовых данных в строковые. Кроме этого, вывод ограничивают 50 строками и значением bottom нижней границы вывода таблицы в рабочей области.

3.5. Рисование геометрических фигур

3.5.1. Функции рисования точки

Функция SetPixel устанавливает заданный цвет в точке с указанными координатами:

```
COLORREF SetPixel( HDC hdc, int x, int y, COLORREF crColor );
```

Параметры x и y определяют логические координаты точки, цвет которой будет изменен. Параметр crColor задает цвет окраски точки. В случае успешного выполнения функция возвращает установленное значение цвета. Это значение может отличаться от цвета crColor, если для цвета crColor не найден точно соответствующий цвет. В случае функциональных сбоев функция возвращает значение -1. Сбои могут быть связаны, например, с тем, что указанные координаты выходят за пределы области вывода.

Пример. Следующий оператор "окрашивает" красным цветом точку в начале системы координат (0, 0):

```
SetPixel( hdc, 0, 0, RGB( 255, 0, 0 ) );
```

Функция GetPixel определяет цвет точки с указанными координатами:

```
COLORREF GetPixel( HDC hdc, int x, int y );
```

Параметры x и y указывают логические координаты точки, цвет которой нужно определить. В случае успешного выполнения функция возвращает цвет точки, иначе – CLR_INVALID.

С помощью следующих трех макрокоманд можно определить значения цветовых компонент цвета rgb:

```
#define    GetRValue(rgb)    ((BYTE)(rgb))
#define    GetGValue(rgb)    ((BYTE)(((WORD)(rgb)) >> 8))
#define    GetBValue(rgb)    ((BYTE)((rgb)>>16))
```

3.5.2. Функции рисования линий

При рисовании линий активно используют значение текущей позиции пера. Чтобы узнать текущую позицию пера, вызывают функцию GetCurrentPositionEx:

```
BOOL GetCurrentPositionEx( HDC hdc, LPPOINT lpPoint );
```

Параметр lpPoint указывает на структуру типа POINT, в которую будут записаны логические координаты текущей позиции. В случае успешного выполнения функция возвращает ненулевое значение.

Чтобы нарисовать прямую линию, вызывают функцию LineTo:

```
BOOL LineTo( HDC hdc, int x, int y );
```

В случае успешного выполнения функция LineTo рисует прямую линию из текущей позиции до (но не включая ее) точки с координатами (x, y). При этом текущая позиция переносится в точку (x, y), а возвращаемое значение отлично от нуля. Для рисования прямой используются текущие установки пера и, если прерывистая линия, текущие установки кисти фона вывода.

При рисовании прямых линий часто приходится сначала перемещать текущую позицию в начало рисуемой прямой и, только затем, вызывать функцию LineTo. Во избежание такой "сложности" описывают функцию следующего вида:

```
BOOL DrawLine( HDC hdc, int x0, int y0, int x, int y)
{       POINT pt;
        MoveToEx(hdc,x0,y0,&pt);
        return LineTo(hdc, x, y);
}
```

Эта функция рисует прямую, начиная от точки (x0, y0), до точки (x, y) и переносит текущую позицию в точку (x, y).

Задача. Нарисовать график функции 25*cos(x) для x, изменяющегося от 0 до 31.4159.

Листинг 3.8. График функции.

```
#include <windows.h>
#include <math.h>

BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char    szClassName[ ] = "GraphClass";
typedef struct
{   char    name[10];
    float   x[100];
    float   y[100];
}   FUNC;

int WINAPI  WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClassName, COLOR_WINDOW) )
                    return FALSE;
    hwnd=CreateWindow(szClassName,"График косинуса",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc;  wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.cbClsExtra = wc.cbWndExtra = 0;  wc.lpfnWndProc = Proc;
    wc.hInstance = hInstance;  wc.lpszClassName = szName;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(HBRUSH)(brBackground+1);
    wc.lpszMenuName = NULL;    return (RegisterClass(&wc) != 0);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                                WPARAM wParam, LPARAM lParam )
{   static  short       cx, cy;
```

```
static  FUNC  myfunc;
switch (msg)
{   case WM_SIZE:
    {   cx=LOWORD(lParam);cy=HIWORD(lParam);
        return 0;
        }
    case WM_CREATE:
    {   strcpy(myfunc.name, "25*cos(x)");
        for (int i=0; i<100; i++)
        {   myfunc.x[i]=(float)(0.31415926535*i);
            myfunc.y[i]=(float)(25*cos(myfunc.x[i]));
        }
        return 0;
    }
    case WM_PAINT:
    {   PAINTSTRUCT ps;
        HDC hdc=BeginPaint(hwnd,&ps);
        //Задаем область для рисования графика
        int   x0=cx/10,      //Левый край графика
              xr=cx-x0,      //Правый край графика
              y0=cy/10,      //Верхний край графика
              yc=cy/2,       //Середина графика по оси y
              amp=yc-y0;     //Амплитуда графика на экране
        float ymax=0.,       //Максимум функции
              ymin=0.;       //Минимум функции
        //Выводим название функции
        TextOut(hdc, x0+4, y0/3,
                myfunc.name, strlen(myfunc.name));
        //Выводим название переменной
        TextOut(hdc, xr+x0/2, yc+y0/10, "t", 1);
        //Рисуем ось ординат
        MoveToEx(hdc,x0,cy-y0+y0/2,NULL);
        LineTo(hdc, x0, y0-y0/2);
        //Рисуем ось абсцисс
        MoveToEx(hdc,x0,yc,NULL);
        LineTo(hdc, xr+x0/2,yc);
        //Определяем максимум и минимум функции
        for (int i=0; i<100; i++)
        {   float ycurr=myfunc.y[i];
            if (ymax<ycurr) ymax=ycurr;
            else if (ymin>ycurr) ymin=ycurr;
        }
```

```
        if (ymax<fabs(ymin)) ymax=(float)fabs(ymin);
        //Устанавливаем масштаб по оси y
        float dy=(float)amp/ymax;
        //Устанавливаем масштаб по оси x
        float dx=(float)(xr-x0)/(myfunc.x[99]-myfunc.x[0]);
        //Для рисования создаем перо синего цвета
        HPEN hPen=CreatePen(PS_SOLID, 2, RGB(0,0,255));
        HPEN hOldPen=SelectObject( hdc, hPen );
        //Устанавливаем курсор в первую точку графика
        MoveToEx(hdc, x0, (int)(yc-dy*myfunc.y[0]),NULL);
        //Выводим линии от предыдущей точки к текущей
        for(i=1; i<100; i++)
        {   //Вычисляем координаты текущей точки
            int xcurr=(int)(dx*(myfunc.x[i]-myfunc.x[0])+x0),
            ycurr=(int)(yc-dy*myfunc.y[i]);
            //Рисуем прямую до текущей точки
            LineTo(hdc, xcurr, ycurr);
        }
        SelectObject(hdc,hOldPen);    DeleteObject(hPen);
        EndPaint(hwnd,&ps);   return 0;
    }
    case WM_DESTROY: {    PostQuitMessage(0); return 0; }
  }  return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

В приложении описывают тип структуры с именем FUNC:

```
typedef struct
{   char    name[10];
    float   x[100];
    float   y[100];
}   FUNC;
```

Структура такого типа будет содержать поле name из 10 символов, поле x из 100 вещественных чисел (для аргумента функции) и поле y из 100 вещественных чисел (для значений функции).

При создании окна структуру myfunc типа FUNC заполняют данными. В поле name записывают название функции:

```
strcpy(myfunc.name, "25*cos(x)");
```

А в поля x и y записывают значения соответственно аргумента и функции:

```
for (int i=0; i<100; i++)
{   myfunc.x[i]=(float)(0.31415926535*i);
    myfunc.y[i]=(float)(25*cos(myfunc.x[i]));
}
```

Обработка сообщения WM_PAINT состоит из следующих основных шагов:

1. Задают область для рисования графика в рабочей области:

```
int x0=cx/10,      //Левый край графика
    xr=cx-x0,      //Правый край графика
    y0=cy/10,      //Верхний край графика
    yc=cy/2,       //Середина графика по оси y
    ymax=0,        //Максимум функции
    ymin=0,        //Минимум функции
    amp=yc-y0;     //Амплитуда графика
```

Параметры области рисования графика здесь выбраны произвольным образом. Например, левый край графика с таким же успехом можно было выбрать вдвое меньше – x0=cx/20.

2. Далее выводят тексты названия функции:

```
TextOut(hdc, x0+4, y0/3, myfunc.name, strlen(myfunc.name));
```

и названия переменной:

```
TextOut(hdc, xr+x0/2, yc+y0/10, "t", 1);
```

3. Рисуют прямые линии осей координат:

```
MoveToEx( hdc, x0, cy-y0+y0/2, NULL );
LineTo( hdc, x0, y0-y0/2 );
MoveToEx( hdc, x0, yc, NULL );
LineTo( hdc, xr+x0/2, yc );
```

4. Размах графика по вертикали должен быть пропорционален максимальному и минимальному значениям функции.

   Поэтому сначала вычисляют максимум и минимум функции:

```
for (int i=0; i<100; i++)
{   int ycurr=(int)myfunc.y[i];
    if (ymax<ycurr) ymax=ycurr;
    else if (ymin>ycurr) ymin=ycurr;
}
```

Выбирают максимальный размах по вертикали:

```
if (ymax<abs(ymin))  ymax=abs(ymin);
```

Далее вычисляют значение коэффициента масштабирования по вертикали:

```
dy=(float)amp/ymax;
```

5. Размах графика по горизонтали должен быть пропорционален максимальному и минимальному значениям аргумента. Значение коэффициента масштабирования по горизонтали вычисляют по формуле

```
dx=(float)(xr-x0)/(myfunc.x[99]-myfunc.x[0]);
```

6. *Для рисования создают перо синего цвета:*

```
HPEN hPen=CreatePen( PS_SOLID, 2, RGB(0,0,255) );
```

Цвет и толщина пера выбраны произвольным образом. Далее в контекст выбирают созданное перо:

```
HPEN hOldPen=SelectObject( hdc, hPen );
```

Здесь дескриптор предыдущего пера запоминается для того, чтобы его выбрать в контекст перед уничтожением созданного пера.

7. Теперь начинают рисовать график функции.

Для этого первоначально устанавливают текущую позицию в точку начала графика:

```
MoveToEx( hdc, x0, (int)(yc-dy*myfunc.y[0]), NULL);
```

Очевидно, что она совпадает с левым краем x0 области рисования графика и отклонением по вертикали на целое количество пикселей dy*myfunc.y[0] от средней линии yc. Знак минус отклонения обусловлен тем, что координата y на экране по умолчанию отсчитывается сверху вниз.

Далее циклически рисуют прямые линии от предыдущей точки к текущей:

```
for(i=1; i<100; i++)
{   int xcurr=(int)(dx*(myfunc.x[i]-myfunc.x[0])+x0),
        ycurr=(int)(yc-dy*myfunc.y[i]);
    LineTo(hdc, xcurr, ycurr);
}
```

8. Завершив рисование созданным пером в контекст выбирают предыдущее перо и удаляют созданное:

```
SelectObject(hdc,hOldPen); DeleteObject(hPen);
```

Продолжим изучение функций рисования линий.

Функция Arc рисует дугу эллипса или окружности:

```
BOOL Arc( HDC hdc, int l, int t, int r, int b, int x0, int y0, int x, int y );
```

Следующий рисунок истолковывает геометрический смысл параметров функции Arc.

Представим, что эллипс вписан в прямоугольник, левый верхний угол которого определяется координатами (l, t), а правый нижний угол – (r, b). Здесь логические координаты l и r отсчитываются по горизонтали, t и b – по вертикали. Тогда воображаемый центр эллипса определяется равенствами xc=(l+r)/2 и yc=(t+b)/2. Начало дуги эллипса определяется пересечением эллипса с воображаемой прямой линией, проведенной из центра эллипса (xc, yc) в точку (x0, y0). Конец дуги определяется аналогично –

как пересечение эллипса с воображаемой прямой линией, проведенной из центра эллипса в точку (x, y). Дуга рисуется от начальной точки к конечной, в направлении против хода часовой стрелки. Если начальная и конечная точки дуги совпадают, то рисуется полный эллипс. Функция Arc игнорирует текущую позицию.

В случае успешного выполнения функция возвращает ненулевое значение.

При вызове функции Arc аргументы l, t, r и b задают в зависимости от места расположения эллипса, и их выбор обычно не вызывает затруднений. Для выбора значений аргументов x0, y0, x и y удобно эти точки расположить на дуге эллипса и использовать формулы расчета в полярных координатах. Например, координата любой точки (x, y) дуги эллипса на рис. 3.1 определяется по формуле

$x=xc+0.5*(r-l)*\cos(\varphi)$, $y=yc-0.5*(b-t)*\sin(\varphi)$.

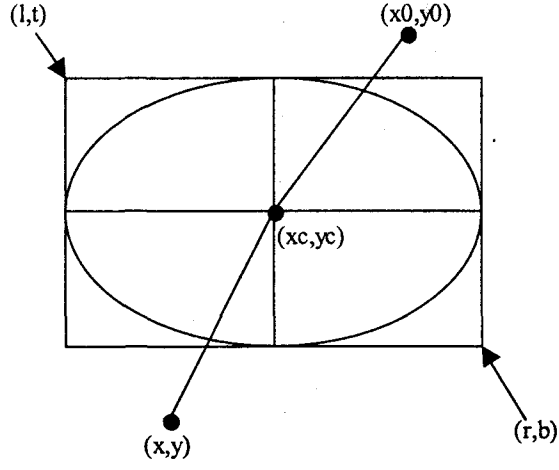

Рис. 3.1. Геометрическая интерпретация параметров функции Arc

Здесь φ – угол наклона прямой, проведенной от центра эллипса в точку (x, y). Угол отсчитывают против хода часовой стрелки.

Функция Polyline предназначена для рисования ломаных линий:

```
BOOL Polyline(HDC hdc, CONST POINT *lppt, int cPoints);
```

Она соединяет прямыми линиями точки, координаты которых указаны в массиве структур типа POINT. Параметр lppt указывает на массив структур POINT, cPoints равен количеству точек в массиве и должен быть больше 1. В случае успешного выполнения функция возвращает ненулевое значение. Функция Polyline игнорирует текущую позицию. Прямые рисуют от точки

первого элемента массива к точке второго элемента и т. д. Если ломаная линия не замкнута, ее последняя точка не рисуется.

Задача. Поверхность, обрамленную эллипсом, разделить на 3 сектора с условными долями 62.5, 12.5 и 25%.

Листинг 3.9. Эллиптическая диаграмма.

```
#include <windows.h>
#include <math.h>
#define PI  3.1415926535
BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char    szClass[ ] = "LinesClass";
int WINAPI  WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd; ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass, COLOR_WINDOW) ) return FALSE;
    hwnd = CreateWindow(szClass, "Эллиптическая диаграмма",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
                        UINT brBackground)
{   WNDCLASS wc; wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.cbClsExtra=wc.cbWndExtra=0; wc.lpfnWndProc = Proc;
    wc.hInstance = hInstance;  wc.lpszClassName = szName;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground+1);
    wc.lpszMenuName = NULL;    return (RegisterClass(&wc) != 0);
}

BOOL DrawLine( HDC hdc, int x0, int y0, int x, int y)
{   MoveToEx(hdc,x0,y0, NULL);
    return LineTo(hdc, x, y);
}
```

```
LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                          WPARAM wParam, LPARAM lParam )
{   static short cx, cy;
    switch (msg)
    {   case WM_SIZE:
        {   cx=LOWORD(lParam); cy=HIWORD(lParam);
            return 0;
        }
        case WM_PAINT:
        {   PAINTSTRUCT ps;
            HDC hdc=BeginPaint(hwnd,&ps);

            //Задаем область для рисования эллипса
            int  l=cx/10,     //Левый край эллипса
                 r=cx-l,      //Правый край эллипса
                 t=cy/10,     //Верхний край эллипса
                 b=cy-t,      //Нижний край эллипса
                 yc=cy/2,     //Центр эллипса по оси y
                 xc=cx/2;     //Центр эллипса по оси x
            //Задаем координаты точек дуги
            int  x0=(int)(xc+0.5*(r-l)),
                 y0=yc,
                 x=(int)(xc+0.5*(r-l)*cos(2.*PI*0.625)),
                 y=(int)(yc-0.5*(b-t)*sin(2.*PI*0.625));
            //Рисуем прямую линию
            DrawLine(hdc, xc, yc, x0, y0);
            //Рисуем дугу эллипса
            Arc(hdc, l, t, r, b, x0, y0, x, y);
            x0=x; y0=y;     //Запоминаем координаты точки
            //Выводим текст
            x=(int)(xc+0.4*cx*cos(2.*PI*0.3125));
            y=(int)(yc-0.4*cy*sin(2.*PI*0.3125));
            TextOut(hdc,x,y,"62,5 %",6);
            //Рисуем прямую линию
            DrawLine(hdc, xc, yc, x0, y0);
            x=(int)(xc+0.5*(r-l)*cos(2.*PI*0.75));
            y=(int)(yc-0.5*(b-t)*sin(2.*PI*0.75));
            //Рисуем дугу эллипса
            Arc(hdc, l, t, r, b, x0, y0, x, y);
            x0=x; y0=y;     //Запоминаем координаты точки
            //Выводим текст
```

```
            x=(int)(xc+0.4*cx*cos(2.*PI*0.6875));
            y=(int)(yc-0.4*cy*sin(2.*PI*0.6875));
            TextOut(hdc,x,y,"12,5 %",6);
            //Рисуем прямую линию
            DrawLine(hdc, xc, yc, x0, y0);
            x=(int)(xc+0.5*(r-l)*cos(2.*PI));
            y=(int)(yc-0.5*(b-t)*sin(2.*PI));
            //Рисуем дугу эллипса
            Arc(hdc, l, t, r, b, x0, y0, x, y);
            //Выводим текст
            x=(int)(xc+0.4*cx*cos(2.*PI*0.875));
            y=(int)(yc-0.4*cy*sin(2.*PI*0.875));
            TextOut(hdc,x,y,"25 %",4);
            EndPaint(hwnd,&ps);
            return 0;
        }
        case WM_DESTROY: { PostQuitMessage(0);  return 0; }
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

В этом приложении новым является только расчет значений координат точек дуги:

```
x=(int)(xc+0.5*(r-l)*cos(2.*PI*0.625)),
y=(int)(yc-0.5*(b-t)*sin(2.*PI*0.625));
```

Здесь значение 2.*PI*0.625 в радианах задает 62.5 % от 360 градусов.

Остальные операторы обсуждались ранее и не требуют пояснений.

3.5.3. Функции рисования замкнутых фигур

Функции рисования замкнутых фигур строят закрашенные или незакрашенные фигуры, такие, как прямоугольники, эллипсы, многоугольники с прямыми и скругленными углами и т. д. Для закрашивания внутренней области замкнутых фигур используют кисть контекста отображения. Внешний контур фигуры обводят пером контекста отображения. Учитывают и остальные атрибуты отображения. В зависимости от стиля пера граница фигуры может находиться полностью внутри заданного внешнего контура или выходить за его пределы. Если выбрать стиль пера PS_NULL, контур станет невидимым.

Для рисования прямоугольника вызывают функцию Rectangle:

```
BOOL Rectangle( HDC hdc, int l, int t, int r, int b );
```

Параметры l, t, r и b определяют логические координаты соответственно левого, верхнего, правого и нижнего краев прямоугольника. В случае успешного выполнения функция возвращает ненулевое значение. Функция игнорирует текущую позицию.

Функция RoundRect рисует прямоугольник с закругленными углами:

```
BOOL RoundRect( HDC hdc, int l, int t, int r, int b, int w, int h );
```

Первые 5 параметров этой функции совпадают с параметрами функции Rectangle. А параметры w и h задают соответственно ширину и высоту эллипса, дуги которого закругляют углы прямоугольника. В случае успешного выполнения функция возвращает ненулевое значение. Функция игнорирует текущую позицию.

Функция FillRect закрашивает прямоугольную область окна заданной кистью:

```
int FillRect( HDC hdc, CONST RECT *lprc, HBRUSH hbr );
```

Параметр lprc указывает на закрашиваемый прямоугольник. Параметр hbr идентифицирует кисть закрашивания, может быть дескриптором логической кисти или значением системного цвета. Например:

```
FillRect( hdc, &rect, (HBRUSH)(COLOR_WINDOW+1) );
```

Правая и нижняя границы прямоугольника не закрашиваются. В случае успешного выполнения функция возвращает ненулевое значение. Функция игнорирует текущую позицию.

Функция FrameRect рисует прямоугольную рамку:

```
int FrameRect( HDC hdc, CONST RECT *lprc, HBRUSH hbr );
```

Параметры этой функции подобны параметрам функции FillRect. Функция FrameRect кистью hbr рисует рамку вокруг заданного прямоугольника. Ширина пера, используемого для рисования рамки, всегда равна одной логической единице. В случае успешного выполнения функция возвращает ненулевое значение. Функция игнорирует текущую позицию.

Функция InvertRect инвертирует значения цветов точек заданной прямоугольной области lprc:

```
BOOL InvertRect (HDC hdc, CONST RECT *lprc );
```

В случае успешного выполнения функция возвращает ненулевое значение. Для восстановления цветов еще раз вызывают эту функцию.

Следующая функция пунктиром обозначает границы заданной прямоугольной области lprc:

```
BOOL DrawFocusRect( HDC hdc, CONST RECT *lprc );
```

Пунктирные линии строятся с использованием растровой операции ИСКЛЮЧАЮЩЕЕ ИЛИ цветов точек, лежащих на границе прямо-

угольника. Вызывая эту функцию второй раз, можно удалить выделение прямоугольника. При прокрутке содержимого окна выделенная граница остается на месте. Для этой функции не нужно выбирать перо, рисующее пунктирную линию. Перед вызовом функции DrawFocusRect устанавливают режим отображения MM_TEXT.

Для рисования эллипса используют функцию Ellipse:

```
BOOL Ellipse( HDC hdc, int l, int t, int r, int b );
```

Параметры функции Ellipse идентичны первым пяти параметрам функции Arc. В случае успешного выполнения функция возвращает ненулевое значение. Функция игнорирует текущую позицию.

Сегмент эллипса рисуют при помощи функции Chord:

```
BOOL Chord(HDC hdc, int l, int t, int r, int b, int x0, int y0, int x, int y );
```

Параметры этой функции аналогичны параметрам функции Arc.

В отличие от функции Arc, функция Chord соединяет хордой точки начала и конца дуги эллипса и закрашивает выделенный таким образом сегмент текущей кистью. В случае успешного выполнения функция возвращает ненулевое значение. Функция игнорирует текущую позицию.

Для рисования сектора эллипса используют функцию Pie:

```
BOOL Pie(HDC hdc, int l, int t, int r, int b, int x0, int y0, int x, int y );
```

Параметры этой функции аналогичны параметрам функций Arc и Chord. В отличие от функции Chord, функция Pie соединяет точки начала и конца дуги с центром эллипса и закрашивает выделенный таким образом сектор текущей кистью. В случае успешного выполнения функция возвращает ненулевое значение. Функция игнорирует текущую позицию.

Функция Polygon рисует многоугольник:

```
BOOL Polygon(HDC hdc, CONST POINT *lpPoints, int nCount );
```

Параметр lpPoints указывает на массив структур POINT, в котором находятся координаты вершин многоугольника. Параметр nCount равен размеру этого массива. В массиве каждая вершина должна быть указана только один раз. Функция Polygon автоматически соединяет первую и последнюю вершины многоугольника. В случае успешного выполнения функция возвращает ненулевое значение. Функция игнорирует текущую позицию.

Контрольные вопросы

1. Каковы особенности вывода в окно?
2. Из каких основных шагов состоит обработка сообщения WM_PAINT?
3. Какими способами выводят в окно при обработке любых сообщений?
4. Какие виды контекста отображения существуют и чем они отличаются?

5. Перечислите основные атрибуты рисования контекста отображения.

6. Из каких шагов состоит процедура выбора нового шрифта в контекст отображения и его удаление?

7. Какие функции используют при рисовании точек?

8. Какие функции рисования линий существуют?

9. Как закрашивают замкнутые фигуры?

Упражнения

1. От левого верхнего до нижнего правого угла рабочей области окна вывести текст таким образом, чтобы он полностью помещался в окне при любых изменениях размеров окна. Использовать шрифт Times New Roman Cyr.

2. В окне приложения многократно отобразить текст "Привет!!!" жирным шрифтом Arial Cyr. Вывод текста начинать от нижнего края окна и продолжать до верхнего края, плавно меняя цвет текста от желтого до черного и цвет фона от синего до желтого. При этом текст должен полностью помещаться по ширине при любых размерах окна.

3. В центре окна нарисовать эллиптическую диаграмму. Диаграмму разбить на секторы 25, 65 и 10 % красного, зеленого и голубого цветов и указать по центру дуги каждого сектора проценты. При всех изменениях размеров окна диаграмма должна быть отображена полностью.

4. Разработать окно перелистывания таблицы. Причем в окне всегда отображать заголовки столбцов. Первый столбец должен содержать номера строк, а другие столбцы – вещественные числа, усеченные до шести значащих цифр. Рисовать столько строк и столбцов, которые без обрезания помещаются в окне.

5. В окне приложения отобразить рейтинг участников какого-либо события: для каждого участника нарисовать геометрическое тело (например, цилиндр), высота которого пропорциональна рейтингу участника, ниже фигуры вывести фамилию участника, выше – рейтинг (в %). Причем текст должен быть наклонен на некоторый угол. Для всех участников использовать разные цвета.

6. В окне приложения вывести текст "Анкета" на русском языке и строго под ним "Anketa" на английском языке, первую букву "А" увеличить в размере и сделать общей для обоих слов. Букву "А" вывести красным цветом, остаток слова на русском языке – синим, на английском – зеленым. При этом выводимый текст должен занимать всю высоту или ширину окна при любых размерах окна.

7. Текст заголовка окна отобразить красным цветом на фоне заголовка, а правее этого текста нарисовать синий эллипс, вписанный в зеленый прямоугольник шириной в 3 полосы заголовка.

8. В рабочей области снизу вверх многократно отобразить строку "Пробный текст", плавно меняя фон и цвет и увеличивая высоту и толщину шрифта. Вывод центрировать по горизонтали. Начальную высоту шрифта взять по умолчанию. Вывод завершается, если очередная строка по высоте не помещается в рабочей области.

9. В окне приложения в три столбика вывести приветствие. В левом и правом столбцах размер шрифта должен возрастать сверху вниз, а в среднем – снизу вверх. Цвета вывода должны быть различны при каждом выводе текста. В случае перекрытия текстов различных столбцов текст не должен стираться, причем правый столбец может перекрывать левый, а средний – оба столбца.

10. В центре окна нарисовать мишень из 10 полей и в каждом поле вывести его значение (от 1 с краю до 10 в центре). При любых изменениях размеров окна мишень должна полностью отображаться в окне. Поля раскрасить случайно выбранными различными цветами.

11. В окне отобразить черным цветом оси координат (0xy) и синим цветом график функции c*cos(x), где x принимает значения от 0 до 10 π и c≠0. Размах вывода по осям ординат и абсцисс – 90 % размеров окна при любых изменениях размеров.

12. Центр рабочей области занимает малиновый прямоугольник с вписанным голубым эллипсом размером в половину области по осям. После нажатия левой клавиши мыши это место занимает синий прямоугольник, а правой клавиши – зеленый эллипс. При изменении размера окна вернуться к исходному состоянию.

13. Описать функцию, которая в указанном прямоугольнике строит эллиптическую диаграмму. Количество секторов, их доля в процентах и цвета закрашивания также передаются аргументами вызова функции.

14. Разрисовать рабочую область окна в разноцветные клеточки, очерченные черным цветом, плавно переходя от красного цвета в левом верхнем углу окна к синему в нижнем правом углу.

15. В центре экрана создать квадратное окно с текстом заголовка и стиля WS_POPUPWINDOW. В окне нарисовать шахматную доску и обозначить поля. При нажатии левой клавиши мыши над полем доски сообщить имя поля.

16. В рабочей части окна при движении мыши с нажатой левой клавишей рисовать траекторию движения курсора мыши. При нажатии правой клавиши мыши поменять цвет для рисования.

17. Рабочую область закрасить спектром цветов, начиная слева красным цветом и заканчивая, на правом краю, синим. Цвет изменять в 4 этапа: на первом этапе, при максимальном красном составляющем, увеличивать зеленую, на втором, при максимальном зеленом, уменьшать красную составляющую, затем, при максимальном зеленом, увеличивать синюю составляющую и, на последнем этапе, при максимальном синем уменьшать зеленую составляющую.

18. В окне отобразить клетки для игры в крестики-нолики. При нажатии левой клавиши мыши в клетке нарисовать крестик, а правой – нолик. Запретить заполнять клетку более одного раза. Рекомендация: если при перемещении курсора была нажата левая клавиша мыши, то параметр wParam содержит значение MK_LBUTTON.

19. Нарисовать светофор, в котором цвет "зажигается" при нажатии на эту лампу левой клавишей мыши. Одновременно может гореть только один цвет.

20. В окне с помощью мыши выделить прямоугольник и создать эффект мигания этого прямоугольника.

21. Нарисовать несколько геометрических фигур. При нажатии над любой из них показать, что выбрана эта фигура.

22. Нарисовать графики функций
2*a*Cos(kt)*exp(-nt) и 0.5*a*Sin(kt)*exp(-nt).

23. В рабочей области окна кистью по умолчанию в метрической системе координат нарисовать эллипс.

24. Создать окно с полосами просмотра и обеспечить просмотр всего содержимого таблицы.

25. Разработать функцию вывода текста в заданной позиции со специальным эффектом. Эффект заключается в том, что текст медленно "выплывает" слева направо. Используя эту функцию отобразить рейтинги участников некоторого события.

Глава 4
Меню

Меню используют для выбора команд и изменения режимов работы приложения. Строки меню отражают название раздела меню или смысл связанной со строкой команды. Если строка связана с командой, при ее выборе приложение получает сообщение WM_COMMAND и идентификатор этой команды.

Основным отображаемым элементом меню является **строка** или **графический объект**. Здесь рассматриваются только строки. Они в текстовом виде отображают названия разделов или команд меню, а также клавиш быстрого доступа. Строка может быть отмечена галочкой или иным образом. Такая строка используется как флажок или переключатель, изменяющий режим работы приложения. Если при выборе строки на экране должна появиться диалоговая панель, к слову справа добавляют многоточие. Заблокированные строки меню отображают серым цветом.

Любое перекрывающееся или временное окно может иметь меню. **Главное меню** находится ниже заголовка окна, и его строки расположены в одну или несколько линий. При выборе строки главного меню, как правило, активизируется раздел меню. **Раздел меню** представляет собой временное меню.

Строки **временного меню** расположены в один или несколько столбцов. Если временное меню может появляться в любом месте рабочей области, то оно называется плавающим. В некоторых случаях удобнее применять **плавающее меню**. Они "всплывают" после щелчка обычно правой клавиши мыши, и место "всплытия" нетрудно связать с координатами курсора мыши. Тогда легче выбрать нужную строку меню. Кроме того, по координатам курсора мыши можно определить объект, по изображению которого был сделан щелчок, и задать зависимый от этого объекта набор строк меню.

Если окно имеет **системное меню**, то оно расположено слева от текста заголовка окна.

По способу создания различают **статическое** и **динамическое меню**. Статическое меню создают до запуска и не изменяют в процессе работы приложения. Динамическое меню создают в процессе работы приложения. Динамическое меню после создания можно изменять или оставить неизменным. Работа со строками статических и динамических меню ничем не отличается. Здесь рассматриваются способы создания только динамических меню.

4.1. Элементы меню

Пользователь видит и работает со строками меню, а приложение работает с элементами меню. Любой элемент меню может быть описан с помощью структуры MENUITEMINFO:

```
typedef struct
{   UINT        cbSize;
    UINT        fMask;
    UINT        fType;
    UINT        fState;
    UINT        wID;
    HMENU       hSubMenu;
    HBITMAP     hbmpChecked;
    HBITMAP     hbmpUnchecked;
    DWORD       dwItemData;
    LPTSTR      dwTypeData;
    UINT        cch;
}   MENUITEMINFO;
```

Рассмотрим пример работы с переменной этого типа:

```
static  MENUITEMINFO mii;
```

В поле **cbsize** записывают размер структуры MENUITEMINFO в байтах:

```
mii.cbSize=sizeof(MENUITEMINFO);
```

Значение поля **fMask** указывает операционной системе, со значениями каких полей структуры mii нужно работать. Смысл работы зависит от действия, которое выполняют над элементом меню. Значение поля fMask представляет собой комбинацию констант из следующей таблицы:

Константа	*Со значениями каких полей работать*
MIIM_CHECKMARKS	hbmpChecked и hbmpUnchecked
MIIM_DATA	dwItemData
MIIM_ID	wID
MIIM_STATE	fState
MIIM_SUBMENU	hsubMenu
MIIM_TYPE	fType и dwTypeData

Например, при создании элемента меню значение поля fMask задают следующим образом:

```
mii.fMask = MIIM_STATE | MIIM_TYPE | MIIM_SUBMENU | MIIM_ID;
```

Тогда операционная система при создании элемента будет учитывать значения полей fState, fType, dwTypeData, hSubMenu и wID структуры mii.

Для установки состояния элемента меню значение поля fMask комбинируют по-другому:

```
mii.fMask = MIIM_STATE | MIIM_ID;
```

Тогда операционная система будет учитывать значения полей fState и wID.

Поле **fType** задает тип элемента. В следующей таблице перечислены некоторые допустимые значения:

Константа	*Пояснение*
MFT_BITMAP	Элемент меню отображать графическим объектом, дескриптор которого находится в младшем слове поля dwTypeData, а поле cch игнорировать
MFT_MENUBARBREAK	Элемент меню отображать в начале новой строки (при построчном отображении элементов) или нового столбца (при отображении элементов по столбцам). Между столбцами появится вертикальная линия
MFT_MENUBREAK	То же, что и MFT_MENUBARBREAK, но между столбцами нет вертикальной линии
MFT_RADIOCHECK	Если hbmpChecked=NULL, то элемент помечать кружочком, а не галочкой
MFT_RIGHTJUSTIFY	Выравнивать этот и последующие элементы меню по правому краю
MFT_SEPARATOR	В качестве элемента отобразить разделитель в виде горизонтальной линии. Поля dwTypeData и cch игнорировать
MFT_STRING	Элемент меню отображать строкой из поля dwTypeData, cch равно длине этой строки

Константы этой таблицы могут комбинироваться, но несовместимы MFT_BITMAP, MFT_SEPARATOR и MFT_STRING.

Пример. Для создания элемента меню, отображаемого в виде строки, записывают:

```
mii.fType=MFT_STRING;
```

Поле **fState** задает состояние элемента меню. Оно принимает следующие значения:

Значение	*Состояние элемента меню*
MFS_CHECKED	Отмечен
MFS_DEFAULT	Элемент по умолчанию
MFS_DISABLED	Заблокирован
MFS_ENABLED	В активном состоянии
MFS_GRAYED	Заблокирован
MFS_HILITE	Подсвечен
MFS_UNCHECKED	Не отмечен
MFS_UNHILITE	Не подсвечен

Поле **wID** содержит определяемый приложением идентификатор команды, связанный с элементом меню.

Пример. Пусть создается элемент меню, который отображается строкой "Создать" и связан с идентификатором CM_FILE_NEW.

Идентификатор создают обычным способом:

```
#define CM_FILE_NEW        3001
```

Тогда этот идентификатор в поле wID можно записать так:

```
mii.wID=CM_FILE_NEW;
```

Поле **hSubMenu** содержит дескриптор временного меню, которое должно появиться при выборе этого элемента. Если меню не должно появиться, это поле содержит NULL.

Поле **hbmpChecked** содержит дескриптор изображения метки для отмеченного элемента. Если это поле содержит NULL, используется символ переключателя или галочка.

Поле **hbmpUnchecked** содержит дескриптор изображения метки для неотмеченного элемента. Если это поле содержит NULL, данное состояние ничем не отображается.

Поле **dwItemData** – определяемое приложением значение, связанное с элементом меню.

Поле **dwTypeData** – содержание отображения элемента. В случае mii.fType=MFT_STRING в поле dwTypeData записывают строку.

Поле **cch** равно длине строки dwTypeData или 0.

4.2. Создание меню

Алгоритм создания меню в общем случае содержит следующие шаги:

1. Создание пустого меню hMenu.
2. Вставка элемента в меню hMenu.
3. Шаг 2 повторяем столько раз, сколько элементов содержит меню hMenu.

Этот алгоритм позволяет создавать многоуровневое меню. Например, вставляемый на 2-м шаге элемент может указывать на уже созданное временное меню. Причем любой элемент последнего также может указывать на временное меню.

Меню сначала создают только в памяти. Существуют функции, которые операционной системе указывают, какое меню создано, какому окну оно принадлежит и когда его отображать. Рассмотрим эти функции.

Функция CreateMenu создает пустое главное меню окна:

```
HMENU CreateMenu(VOID);
```

В случае успешного выполнения функция CreateMenu возвращает дескриптор созданного главного меню, иначе – NULL.

Функция CreatePopupMenu создает пустое временное меню:

```
HMENU CreatePopupMenu(VOID);
```

В случае успешного выполнения функция CreatePopupMenu возвращает дескриптор созданного временного меню, иначе – NULL.

Функция SetMenu подключает главное меню hMenu к окну hwnd:

```
BOOL SetMenu(HWND hwnd, HMENU hMenu );
```

Она удаляет предыдущее меню, но не разрушает его. Если же hMenu== NULL, то только удаляется главное меню. В случае успешного подключения функция возвращает ненулевое значение.

После изменений в меню, независимо от состояния окна hwnd, для перерисовки полосы меню нужно вызвать функцию DrawMenuBar:

```
void WINAPI DrawMenuBar(HWND hwnd);
```

Она перерисовывает полосу меню окна hwnd.

Таким образом, алгоритм подключения главного меню к окну и отображения *полосы меню содержит следующие шаги:*

1. Создание главного меню. Этот шаг полностью совпадает с алгоритмом создания меню.
2. Подключение главного меню к окну.
3. Перерисовка полосы меню окна.

После этого работу с меню обеспечивает операционная система.

Приложение в созданное меню может вставлять новые элементы или удалять старые, а также может изменять или запрашивать текущее состояние любого элемента.

4.2.1. Вставка элементов в меню

Функция InsertMenuItem вставляет элемент в меню hMenu:

```
BOOL InsertMenuItem(HMENU hMenu, UINT uItem,
                    BOOL fByPosition, LPMENUITEMINFO lpmii );
```

Значение аргумента uItem зависит от значения fByPosition. При fByPosition=FALSE значение uItem равно идентификатору команды вставляемого элемента, иначе – позиции того элемента в меню, перед которым нужно вставить новый элемент. Если меню содержит n элементов и нужно вставить новый элемент последним, то номер позиции приравнивают n (n=0, 1, 2, …).

Параметр lpmii указывает на структуру типа MENUITEMINFO, которая уже содержит информацию о вставляемом элементе.

В случае успешной вставки элемента функция возвращает ненулевое значение.

Задача. Описать функцию для вставки элемента в меню.

Нижеследующий фрагмент может служить примером решения данной задачи.

```
BOOL CreateMenuItem(HMENU hMenu, char *str, UINT uIns,
    UINT uCom, HMENU hSubMenu, BOOL flag, UINT fType)
{   MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE | MIIM_TYPE | MIIM_SUBMENU | MIIM_ID;
    mii.fType = fType;      mii.fState= MFS_ENABLED;
    mii.dwTypeData = str;  mii.cch=sizeof(str);
    mii.wID=uCom;          mii.hSubMenu=hSubMenu;
    return InsertMenuItem(hMenu, uIns, flag, &mii);
}
```

Рассмотрим список формальных параметров этой функции.

1. **hMenu** – дескриптор меню, в который вставляется элемент.
2. **str** указывает на текст отображения строки меню.
3. **uIns** равноценен параметру uItem функции InsertMenuItem.
4. **uCom**, когда он ненулевой, исполняет роль параметра wID функции InsertMenuItem.
5. **hSubMenu** – дескриптор временного меню, которое должно появиться при выборе строки str в меню hMenu.
6. **flag** совпадает с параметром fByPosition в InsertMenuItem.
7. **fType** принимает те же значения, что и поле fType структуры MENUITEMINFO.

Задача. Главное меню содержит разделы "Файлы" и "Правка". При выборе строки "Файлы" отображается временное меню со строками "Открыть", "Сохранить" и "Выход". При выборе строки "Правка" отображается временное меню со строками "Найти" и "Заменить".

Листинг 4.1. Пример приложения со статическим меню.

```
#include <windows.h>

#define CM_FILE_OPEN    1001
#define CM_FILE_SAVE    1002
#define CM_FILE_QUIT    1003
#define CM_EDIT_FIND    2001
#define CM_EDIT_REPLC   2002

BOOL        RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT     CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char        szClass[ ]="MenuWindow";

int WINAPI WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass,COLOR_WINDOW) )
        return FALSE;
    hwnd = CreateWindow(szClass,
        "Приложение со статическим меню",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0))
    {   TranslateMessage(&msg); DispatchMessage(&msg); }
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc; wc.style=wc.cbClsExtra=wc.cbWndExtra = 0;
    wc.lpfnWndProc = Proc;    wc.hInstance=hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = NULL;    wc.lpszClassName = szName;
    return (RegisterClass(&wc) != 0);
}

BOOL CreateMenuItem(HMENU hMenu, char *str, UINT uIns,
            UINT uCom, HMENU hSubMenu, BOOL flag, UINT fType)
```

```
{   MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE | MIIM_TYPE | MIIM_SUBMENU | MIIM_ID;
    mii.fType = fType;  mii.fState= MFS_ENABLED;
    mii.dwTypeData = str;  mii.cch=sizeof(str);
    mii.wID=uCom;          mii.hSubMenu=hSubMenu;
    return InsertMenuItem(hMenu, uIns, flag, &mii);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                  WPARAM wParam, LPARAM lParam )
{   static HMENU hMainMenu, hFileMenu, hEditMenu;
    switch (msg)
    {   case WM_CREATE:
        {   hMainMenu=CreateMenu();
            //Создаем временное меню для раздела "Файлы"
            hFileMenu=CreatePopupMenu();
            int i=0; //Инициализация позиции в меню hFileMenu
            CreateMenuItem( hFileMenu, "&Открыть", i++,
                    CM_FILE_OPEN, NULL, FALSE, MFT_STRING );
            CreateMenuItem( hFileMenu, "&Сохранить", i++,
                    CM_FILE_SAVE, NULL, FALSE, MFT_STRING );
            CreateMenuItem( hFileMenu, NULL, i++,
                    0, NULL, FALSE, MFT_SEPARATOR );
            CreateMenuItem( hFileMenu, "&Выход", i++,
                    CM_FILE_QUIT, NULL, FALSE, MFT_STRING );
            //Создаем временное меню для раздела "Правка"
            hEditMenu=CreatePopupMenu();
            i=0; //Инициализация позиции в меню hEditMenu
            CreateMenuItem( hEditMenu, "&Найти", i++,
                    CM_EDIT_FIND, NULL, FALSE, MFT_STRING );
            CreateMenuItem( hEditMenu, "&Заменить", i++,
                    CM_EDIT_REPLC,  NULL, FALSE, MFT_STRING );
            //Подключаем временные меню к главному меню
            i=0; //Инициализация позиции в меню hMainMenu
            CreateMenuItem( hMainMenu, "&Файл", i++,
                    0, hFileMenu,FALSE, MFT_STRING );
            CreateMenuItem( hMainMenu, "&Правка", i++,
                    0, hEditMenu,FALSE, MFT_STRING );
            SetMenu( hwnd, hMainMenu );  DrawMenuBar( hwnd );
return 0;
        }
```

```
        case WM_COMMAND:
        {   switch (LOWORD(wParam))
            {   case CM_FILE_OPEN:
                {   MessageBox(hwnd, "Команда CM_FILE_OPEN",
                        "Меню",MB_OK);  return 0;
                }
                case CM_FILE_SAVE:
                {   MessageBox(hwnd, "Команда CM_FILE_SAVE",
                        "Меню",MB_OK);  return 0;
                }
                case CM_EDIT_FIND:
                {   MessageBox(hwnd, "Команда CM_EDIT_FIND",
                        "Меню",MB_OK);  return 0;
                }
                case CM_EDIT_REPLC:
                {   MessageBox(hwnd, "Команда CM_EDIT_REPLC",
                        "Меню",MB_OK);  return 0;
                }
                case CM_FILE_QUIT:
                {   DestroyWindow(hwnd); return 0;   }
            }
            return 0;
        }
        case WM_DESTROY:  { PostQuitMessage(0); return 0; }
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Это приложение демонстрирует применение рассмотренной выше функции CreateMenuItem для создания статического меню. Меню названо статическим потому, что в процессе работы приложения оно не изменяется. Рассмотрим основные шаги создания меню.

1. Описывают дескрипторы для создаваемых меню:

```
static HMENU hMainMenu, hFileMenu, hEditMenu;
```

2. При обработке сообщения WM_CREATE создают всю систему меню (хотя не запрещается это делать и в любом другом месте).

   2.1. Создают пустое главное меню окна:

```
hMainMenu=CreateMenu();
```

   2.2. Создают временное меню для раздела "Файлы".

      2.2.1. Создают пустое временное меню:

```
hFileMenu=CreatePopupMenu();
```

2.2.2. Вставляют элементы в меню hFileMenu:

```
CreateMenuItem( hFileMenu, "&Открыть", i++, CM_FILE_OPEN,
        NULL, FALSE, MFT_STRING );
CreateMenuItem( hFileMenu,"&Сохранить",i++, CM_FILE_SAVE,
        NULL, FALSE, MFT_STRING );
CreateMenuItem( hFileMenu, NULL, i++,0,
        NULL, FALSE, MFT_SEPARATOR );
CreateMenuItem( hFileMenu, "&Выход", i++, CM_FILE_QUIT,
        NULL, FALSE, MFT_STRING );
```

Здесь аргумент "&Открыть" обозначает ту строку, которая будет отображена во временном меню. Ключ & обозначает, что символ "О" можно использовать для быстрого доступа к данной строке меню. Аргумент CM_FILEOPEN является идентификатором той команды, которую получит функция окна при выборе строки "&Открыть".

Так же вставляют второй и четвертый элементы меню. Третьим элементом является разделительная линия.

2.3. Создают временное меню для раздела "Правка".

2.3.1. Создают пустое временное меню:

```
hEditMenu=CreatePopupMenu();
```

2.3.2. Вставляют элементы в меню hEditMenu:

```
CreateMenuItem( hEditMenu, "&Найти", i++, CM_EDIT_FIND,
        NULL, FALSE, MFT_STRING );
CreateMenuItem(hEditMenu,"&Заменить", i++, CM_EDIT_REPLC,
        NULL, FALSE, MFT_STRING );
```

2.4. Подключают временные меню hFileMenu и hEditMenu к главному меню hMainMenu:

```
CreateMenuItem( hMainMenu, "&Файл", i++, 0, hFileMenu, FALSE, MFT_STRING );
CreateMenuItem( hMainMenu, "&Правка", i++,
        0, hEditMenu, FALSE, MFT_STRING );
```

3. Подключают главное меню к окну и перерисовывают полосу меню:

```
SetMenu(hwnd,hMainMenu); DrawMenuBar(hwnd);
```

Приложение обрабатывает команды меню. Как можно увидеть из листинга, обработка команд от меню ничем не отличается от обработки команд от других органов.

4.2.2. Удаление элементов из меню

Функция DeleteMenu удаляет элемент из меню и освобождает все связанные с ним ресурсы. Если удаляемый элемент указывает на временное меню, то функция DeleteMenu удаляет и это временное меню. Прототип функции:

```
BOOL DeleteMenu( HMENU hMenu, UINT uPosition, UINT uFlags );
```

Параметры:

1. **hMenu** – дескриптор изменяемого меню.
2. **uPosition** определяет удаляемый элемент. Его значение связано со значением параметра uFlags.
3. **uFlags** задает способ истолкования параметра uPosition. Если uFlags=MF_BYCOMMAND (значение по умолчанию), то значение uPosition равно идентификатору команды удаляемого элемента. Если же uFlags=MF_BYPOSITION, то значение uPosition равно позиции удаляемого элемента в меню.

В случае успешного удаления функция возвращает ненулевое значение.

Функция RemoveMenu удаляет элемент из меню, не разрушая связанные с ним ресурсы. Эти ресурсы можно использовать в дальнейшем. Прототип функции:

```
BOOL RemoveMenu( HMENU hMenu, UINT uPosition, UINT uFlags );
```

Параметры функции RemoveMenu подобны параметрам функции DeleteMenu. В случае успешного удаления элемента RemoveMenu возвращает ненулевое значение.

Для разрушения меню вызывают функцию DestroyMenu:

```
BOOL DestroyMenu( HMENU hMenu );
```

Эта функция разрушает заданное меню и освобождает ресурсы, которые меню занимает. Эту функцию окна вызывают для разрушения тех меню, которые они создали, но не подключили к себе. Подключенные меню автоматически разрушаются при разрушении окна.

В случае успешного разрушения функция возвращает ненулевое значение.

Задача. Главное меню изначально содержит раздел "Файлы". При выборе строки "Файлы" отображается временное меню со строками "Открыть", "Сохранить", разделительной линией и строкой "Выход". При выборе строки "Открыть" в главное меню добавить элемент "Правка". При выборе строки "Правка" отображается временное меню со строками "Найти" и "Заменить". Если же затем выбрать строку "Сохранить", то из главного меню удалить элемент "Правка".

Листинг 4.2. Пример удаления элементов меню.

```
#include <windows.h>

#define CM_FILE_OPEN    1001
#define CM_FILE_SAVE    1002
#define CM_FILE_QUIT    1003
#define CM_EDIT_FIND    2001
#define CM_EDIT_REPLC   2002

BOOL        RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT     CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char        szClass[ ]="RemoveMenu";

int WINAPI WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass,COLOR_WINDOW) )
        return FALSE;
    hwnd = CreateWindow(szClass,
        "Пример удаления элементов меню",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0))
    {   TranslateMessage(&msg); DispatchMessage(&msg);  }
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc; wc.style=wc.cbClsExtra=wc.cbWndExtra = 0;
    wc.lpfnWndProc = Proc;    wc.hInstance=hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = NULL;    wc.lpszClassName = szName;
    return (RegisterClass(&wc) != 0);
}

BOOL CreateMenuItem(HMENU hMenu, char *str, UINT uIns,
        UINT uCom, HMENU hSubMenu, BOOL flag, UINT fType)
```

```
{   MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE | MIIM_TYPE | MIIM_SUBMENU | MIIM_ID;
    mii.fType = fType;  mii.fState= MFS_ENABLED;
    mii.dwTypeData = str;  mii.cch=sizeof(str);
    mii.wID=uCom;      mii.hSubMenu=hSubMenu;
    return InsertMenuItem(hMenu, uIns, flag, &mii);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                          WPARAM wParam, LPARAM lParam )
{   static   HMENU    hMainMenu, hFileMenu, hEditMenu;
    static   BOOL  fFileOpened;
    switch (msg)
    {   case WM_CREATE:
        {   hMainMenu=CreateMenu();
            //Создаем временное меню для раздела "Файлы"
            hFileMenu=CreatePopupMenu();
            int i=0; //Инициализация позиции в меню hFileMenu
            CreateMenuItem( hFileMenu, "&Открыть", i++,
                CM_FILE_OPEN, NULL, FALSE, MFT_STRING );
            CreateMenuItem( hFileMenu, "&Сохранить", i++,
                CM_FILE_SAVE, NULL, FALSE, MFT_STRING );
            CreateMenuItem( hFileMenu, NULL, i++,
                0, NULL, FALSE, MFT_SEPARATOR );
            CreateMenuItem( hFileMenu, "&Выход", i++,
                CM_FILE_QUIT, NULL, FALSE, MFT_STRING );
            //Создаем временное меню для раздела "Правка"
            hEditMenu=CreatePopupMenu();
            i=0; //Инициализация позиции в меню hEditMenu
            CreateMenuItem( hEditMenu, "&Найти", i++,
                CM_EDIT_FIND, NULL, FALSE, MFT_STRING );
            CreateMenuItem( hEditMenu, "&Заменить", i++,
                CM_EDIT_REPLC,
                NULL, FALSE, MFT_STRING );
            //Подключаем временные меню к главному меню
            i=0; //Инициализация позиции в меню hMainMenu
            CreateMenuItem( hMainMenu, "&Файл", i++,
                0, hFileMenu,FALSE, MFT_STRING );
            SetMenu(hwnd,hMainMenu);  DrawMenuBar(hwnd);
            fFileOpened=FALSE;   return 0;
        }
```

```
    case WM_COMMAND:
    {   switch (LOWORD(wParam))
        {   case CM_FILE_OPEN:
            {   if (!fFileOpened)
                {   CreateMenuItem( hMainMenu, "&Правка", 1, 0,
                        hEditMenu, FALSE, MFT_STRING );
                    DrawMenuBar( hwnd );
                    fFileOpened=TRUE;
                }
                return 0;
            }
            case CM_FILE_SAVE:
            {   if (fFileOpened)
                {   RemoveMenu(hMainMenu,1, MF_BYPOSITION);
                    DrawMenuBar( hwnd );
                    fFileOpened=FALSE;
                }
                return 0;
            }
            case CM_EDIT_FIND:
            {   MessageBox(hwnd, "Команда CM_EDIT_FIND",
                    "Меню",MB_OK);
                return 0;
            }
            case CM_EDIT_REPLC:
            {   MessageBox(hwnd, "Команда CM_EDIT_REPLC",
                    "Меню",MB_OK);
                return 0;
            }
            case CM_FILE_QUIT:
            {   DestroyWindow(hwnd); return 0;   }
        }
        return 0;
    }
    case WM_DESTROY:
    {   if (!fFileOpened) DestroyMenu( hEditMenu );
        PostQuitMessage(0);
        return 0;
    }
}
```

```
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Это приложение от предыдущего отличается следующим:

1. Временное меню hEditMenu в главное меню вставляют лишь после выбора строки "Открыть". При этом проверяют флаг состояния fFileOpened – если файл открыт, то меню hEditMenu уже вставлено в главное меню и еще раз его вставлять не нужно:

```
if ( !fFileOpened )
{   CreateMenuItem( hMainMenu, "&Правка", 1, 0,  hEditMenu,
            FALSE, MFT_STRING );
    DrawMenuBar( hwnd );     fFileOpened=TRUE;
}
```

Как видим, операция вставки буквально скопирована из предыдущего приложения.

2. Временное меню hEditMenu удаляют из главного меню после выбора строки "Сохранить". Здесь закрытие файла подразумевается. При этом проверяют флаг состояния fFileOpened – если файл закрыт, то меню hEditMenu уже удалено и еще раз его удалять не нужно:

```
if (fFileOpened)
{   RemoveMenu( hMainMenu, 1, MF_BYPOSITION );
    DrawMenuBar( hwnd );       fFileOpened=FALSE;
}
```

3. До разрушения окна проверяют, вставлено ли меню hEditMenu в главное меню. Если нет, то его нужно разрушать отдельно:

```
if ( !fFileOpened ) DestroyMenu( hEditMenu );
```

4.2.3. Управление состоянием элементов меню

Следующая функция изменяет состояние элемента меню hMenu:

```
BOOL SetMenuItemInfo( MENU hMenu, UINT uItem,
            BOOL fByPosition, LPMENUITEMINFO lpmii );
```

При fByPosition=FALSE значение uItem равно идентификатору команды элемента, иначе – позиции элемента в меню. Значение lpmii указывает на структуру типа MENUITEMINFO, которая уже содержит новые данные для элемента. В случае успешного выполнения функция возвращает ненулевое значение.

Проанализируем вызов функции SetMenuItemInfo. Первые 3 аргумента указывают на элемент, параметры которого изменяют. Последний аргумент содержит параметры нового состояния элемента. При этом изменяют те параметры, на которые указывает поле fMask.

Задача. Описать функцию для изменения состояния элемента.

Следующая функция является примером решения данной задачи:

```
BOOL SetMenuItem(HMENU hMenu, UINT uIns, UINT fState, BOOL flag)
{   MENUITEMINFO mii;
    mii.cbSize  = sizeof(MENUITEMINFO);
    mii.fMask   = MIIM_STATE | MIIM_ID;
    mii.fState  = fState;
    mii.wID = uIns;
    return SetMenuItemInfo( hMenu, uIns, flag, &mii );
}
```

Список формальных параметров этой функции:

1. **hMenu** – дескриптор меню, состояние элемента которого нужно изменить.
2. **uIns** и **flag** совпадают соответственно с параметрами uItem и fByPosition функции SetMenuItemInfo.
3. **fState** задает новое состояние элемента.

Поле fMask указывает, что у элемента изменяется только значение поля fState. А поле wID структуры mii операционная система использует для получения указания на изменяемый элемент.

Следующая функция выбирает данные элемента меню hMenu:

```
BOOL GetMenuItemInfo (HMENU hMenu, UINT uItem,
                      BOOL fByPosition, LPMENUITEMINFO lpmii );
```

При fByPosition=FALSE значение uItem равно идентификатору команды элемента, иначе – позиции элемента в меню. Значение lpmii указывает на структуру типа MENUITEMINFO, поле fMask которой содержит указания на запрашиваемые параметры. В эту же структуру будут записаны значения запрошенных параметров элемента. В случае успешного выполнения функция возвращает ненулевое значение.

Задача. Описать функцию для запроса состояния элемента.

Следующая функция является примером решения данной задачи:

```
UINT GetMenuItem(HMENU hMenu, UINT uIns, BOOL flag)
{   MENUITEMINFO mii;
    mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE;
    GetMenuItemInfo(hMenu, uIns, flag, &mii);
    return mii.fState;
}
```

Список формальных параметров этой функции:

1. **hMenu** – дескриптор меню, состояние элемента которого нужно запросить.
2. **uIns** и **flag** совпадают соответственно с параметрами uItem и fByPosition функции GetMenuItemInfo.

Функция возвращает значение состояния элемента.

Поле fMask указывает, что запрашивается только значение поля fState элемента.

Задача. Главное меню изначально содержит раздел "Файлы" со строками "Открыть", "Сохранить" и "Выход". Причем строка "Сохранить" заблокирована. При выборе строки "Открыть" заблокировать ее, разблокировать строку "Сохранить" и вставить разделы "Правка" и "Вид" в главное меню.

При выборе строки "Правка" отображается временное меню со строками "Найти" и "Заменить". Причем эти строки при их выборе должны быть отмечены как независимые флажки.

При выборе строки "Вид" отображается временное меню со строками "Обычный" и "Структура". Причем эти строки при их выборе должны быть отмечены как зависимые переключатели.

При выборе строки "Сохранить" заблокировать ее. Затем разблокировать строку "Открыть" и из главного меню удалить разделы "Правка" и "Вид".

Листинг 4.3. Управление состоянием строк меню.

```
#include <windows.h>

#define CM_FILE_OPEN    1001
#define CM_FILE_SAVE    1002
#define CM_FILE_QUIT    1003
#define CM_EDIT_FIND    2001
#define CM_EDIT_REPLC   2002
#define CM_VIEW_NORM    3001
#define CM_VIEW_STRC    3002

BOOL        RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT     CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char        szClass[ ]="SetMenuInfo";

int WINAPI WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass,COLOR_WINDOW) )
```

```
        return FALSE;
    hwnd = CreateWindow(szClass,
        "Управление состоянием элементов меню",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0))
    {   TranslateMessage(&msg); DispatchMessage(&msg); }
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc; wc.style=wc.cbClsExtra=wc.cbWndExtra = 0;
    wc.lpfnWndProc = Proc;    wc.hInstance=hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = NULL;    wc.lpszClassName = szName;
    return (RegisterClass(&wc) != 0);
}

BOOL SetMenuItem(HMENU hMenu, UINT uIns, UINT fState, BOOL flag)
{   MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask   = MIIM_STATE | MIIM_ID;
    mii.fState   = fState;    mii.wID = uIns;
    return SetMenuItemInfo( hMenu, uIns, flag, &mii );
}

UINT GetMenuItem(HMENU hMenu, UINT uIns, BOOL flag)
{   MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE;
    GetMenuItemInfo(hMenu, uIns, flag, &mii);
    return mii.fState;
}

BOOL CreateMenuItem(HMENU hMenu, char *str, UINT uIns,
            UINT uCom, HMENU hSubMenu, BOOL flag, UINT fType)
{   MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE | MIIM_TYPE | MIIM_SUBMENU | MIIM_ID;
    mii.fType = fType;          mii.fState= MFS_ENABLED;
```

```
    mii.dwTypeData = str;     mii.cch=sizeof(str);
    mii.wID=uCom;             mii.hSubMenu=hSubMenu;
    return InsertMenuItem(hMenu, uIns, flag, &mii);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                          WPARAM wParam, LPARAM lParam )
{   static  HMENU hMainMenu, hFileMenu, hEditMenu, hViewMenu;
    switch (msg)
    {   case WM_CREATE:
        {   hMainMenu=CreateMenu();
            //Создаем временное меню для раздела "Файлы"
            hFileMenu=CreatePopupMenu();
            int i=0; //Инициализация позиции в меню hFileMenu
            CreateMenuItem( hFileMenu, "&Открыть", i++,
                CM_FILE_OPEN,
                NULL, FALSE, MFT_STRING );
            CreateMenuItem( hFileMenu, "&Сохранить", i++,
                CM_FILE_SAVE,
                NULL, FALSE, MFT_STRING );
            CreateMenuItem( hFileMenu, NULL, i++,  0,
                NULL, FALSE, MFT_SEPARATOR );
            CreateMenuItem(hFileMenu,"&Выход",i++,
                CM_FILE_QUIT,
                NULL, FALSE, MFT_STRING );
            //Создаем временное меню для раздела "Правка"
            hEditMenu=CreatePopupMenu();
            i=0; //Инициализация позиции в меню hEditMenu
            CreateMenuItem( hEditMenu, "&Найти", i++,
                CM_EDIT_FIND,
                NULL, FALSE, MFT_STRING );
            CreateMenuItem( hEditMenu, "&Заменить", i++,
                CM_EDIT_REPLC,
                NULL, FALSE, MFT_STRING );
            //Создаем временное меню для раздела "Вид"
            hViewMenu=CreatePopupMenu();
            i=0; //Инициализация позиции в меню hViewMenu
            CreateMenuItem( hViewMenu, "&Обычный", i++,
                CM_VIEW_NORM,
                NULL, FALSE, MFT_RADIOCHECK );
            CreateMenuItem( hViewMenu, "&Структура", i++,
```

```
            CM_VIEW_STRC,
            NULL, FALSE, MFT_RADIOCHECK );
        //Подключаем временные меню к главному меню
        i=0; //Инициализация позиции в меню hMainMenu
        CreateMenuItem( hMainMenu, "&Файл", i++, 0,
            hFileMenu,FALSE, MFT_STRING );
        SetMenu(hwnd,hMainMenu);
        SetMenuItem(hFileMenu, CM_FILE_SAVE,
                        MFS_GRAYED, FALSE);
        DrawMenuBar(hwnd);  return 0;
    }
    case WM_COMMAND:
    {   switch (LOWORD(wParam))
        {   case CM_FILE_OPEN:
            {   SetMenuItem( hFileMenu, CM_FILE_OPEN,
                    MFS_GRAYED, FALSE);
                SetMenuItem( hFileMenu, CM_FILE_SAVE,
                    MFS_ENABLED, FALSE);
                CreateMenuItem( hMainMenu, "&Правка", 1, 0, hEditMenu,
                    FALSE, MFT_STRING);
                CreateMenuItem(hMainMenu,  "&Вид", 2, 0, hViewMenu,
                    FALSE, MFT_STRING);
                DrawMenuBar(hwnd);
                return 0;
            }
            case CM_FILE_SAVE:
            {   SetMenuItem( hFileMenu, CM_FILE_SAVE,
                    MFS_GRAYED, FALSE);
                SetMenuItem( hFileMenu, CM_FILE_OPEN,
                    MFS_ENABLED, FALSE);
                RemoveMenu(hMainMenu,1, MF_BYPOSITION);
                RemoveMenu(hMainMenu,1, MF_BYPOSITION);
                DrawMenuBar(hwnd);
                return 0;
            }
            case CM_EDIT_FIND:
            {   if (GetMenuItem(hEditMenu,
                    CM_EDIT_FIND, FALSE)==MFS_CHECKED)
                    SetMenuItem( hEditMenu, CM_EDIT_FIND,
                        MFS_UNCHECKED,FALSE);
                else    SetMenuItem( hEditMenu, CM_EDIT_FIND,
```

```
                    MFS_CHECKED, FALSE);
                return 0;
            }
            case CM_EDIT_REPLC:
            {   if (GetMenuItem(hEditMenu,
                    CM_EDIT_REPLC, FALSE)==MFS_CHECKED)
                    SetMenuItem( hEditMenu,
                        CM_EDIT_REPLC, MFS_UNCHECKED,FALSE);
                else    SetMenuItem(hEditMenu,
                        CM_EDIT_REPLC, MFS_CHECKED, FALSE);
                return 0;
            }
            case CM_VIEW_NORM:
            {   SetMenuItem( hViewMenu,
                    CM_VIEW_NORM, MFS_CHECKED, FALSE);
                SetMenuItem( hViewMenu,
                    CM_VIEW_STRC, MFS_UNCHECKED, FALSE);
                return 0;
            }
            case CM_VIEW_STRC:
            {   SetMenuItem( hViewMenu,
                    CM_VIEW_NORM, MFS_UNCHECKED, FALSE);
                SetMenuItem( hViewMenu,
                    CM_VIEW_STRC, MFS_CHECKED, FALSE);
                return 0;
            }
            case CM_FILE_QUIT:
            {   DestroyWindow(hwnd); return 0;   }
        }
        return 0;
    }
    case WM_DESTROY:
    {   DestroyMenu(hEditMenu); DestroyMenu(hViewMenu);
        PostQuitMessage(0);
        return 0;
    }
}
return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Это приложение от предыдущего отличается следующим:

1. Временные меню hEditMenu и hViewMenu в главное меню вставляют после выбора строки "Открыть". Отсутствует проверка состояния fFileOpened. Взамен введено управление состоянием элементов меню. Например, пока файл не открыт, заблокирован элемент "Сохранить":

```
SetMenuItem(hFileMenu, CM_FILE_SAVE, MFS_GRAYED, FALSE);
```

А после открытия файла этот элемент разблокируется:

```
SetMenuItem(hFileMenu, CM_FILE_SAVE, MFS_ENABLED, FALSE);
```

и блокируется элемент "Открыть":

```
SetMenuItem( hFileMenu, CM_FILE_OPEN, MFS_GRAYED, FALSE);
```

Блокировка элемента гарантирует, что его строку нельзя выбрать.

2. Временные меню hEditMenu и hViewMenu удаляют из главного меню после выбора строки "Сохранить".

3. В разделе "Правка" строки "Найти" и "Заменить" могут быть отмечены галочкой. Обратите внимание, что при вставке этих элементов ничего не изменилось. Для отметки галочкой достаточно установить состояние MFS_CHECKED. Например:

```
SetMenuItem(hEditMenu, CM_EDIT_FIND, MFS_CHECKED, FALSE);
```

В приложении эта часть задачи решается более широко. Например, проверяется текущее состояние элемента. Если элемент перед выбором был уже отмечен галочкой, то снимается отметка, и если он перед выбором не был отмечен, то отмечается галочкой:

```
if ( GetMenuItem(hEditMenu,CM_EDIT_FIND, FALSE)==MFS_CHECKED )
        SetMenuItem(hEditMenu, CM_EDIT_FIND, MFS_UNCHECKED,FALSE);
else    SetMenuItem(hEditMenu, CM_EDIT_FIND, MFS_CHECKED,FALSE);
```

4. Элементы временного меню hViewMenu используются как зависимые переключатели. Поэтому задан тип MFT_RADIOCHECK:

```
CreateMenuItem(hViewMenu,"&Обычный",i++,
        CM_VIEW_NORM,NULL,FALSE, MFT_RADIOCHECK);
```

Это гарантирует, что метка элемента будет отображаться черным кружочком (а не галочкой).

5. При выборе любой строки меню hViewMenu устанавливают этот элемент в отмеченное состояние, а другой элемент этого меню – в неотмеченное. Например, при выборе элемента CM_VIEW_NORM устанавливают этот элемент в отмеченное состояние:

```
SetMenuItem(hViewMenu,CM_VIEW_NORM,MFS_CHECKED,FALSE);
```

а элемент CM_ VIEW_STRC устанавливают в неотмеченное состояние:

```
SetMenuItem(hViewMenu,CM_ VIEW_STRC,
   MFS_UNCHECKED,FALSE);
```

4.2.4. Получение информации о меню

Функция GetMenu возвращает дескриптор главного меню окна hwnd:

```
HMENU GetMenu(HWND hwnd);
```

Если окно не имеет меню, функция возвращает NULL.

Функция GetSubMenu возвращает дескриптор временного меню, которое расположено в позиции nPos указанного меню hmenu:

```
HMENU GetSubMenu(HMENU hmenu, int nPos);
```

Причем первому временному меню в hmenu соответствует нулевое значение параметра nPos. Если функция GetSubMenu вернула значение NULL, то hmenu в позиции nPos не содержит временное меню.

Функция IsMenu возвращает ненулевое значение, если hmenu является дескриптором меню, иначе – возвращает нуль:

```
BOOL IsMenu(HMENU hmenu);
```

Функция GetMenuItemCount возвращает количество элементов в меню hmenu:

```
int GetMenuItemCount(HMENU hmenu);
```

В случае ошибки возвращаемое значение равно -1.

Задача. Окно приложения при создании должно узнать информацию о разделе меню "Файлы" приложения из листинга 4.3, сообщить о количестве элементов в этом разделе. После нажатия левой клавиши мыши окну этого приложения послать команду элемента в позиции 0 ("Открыть").

Листинг 4.4. Управление состоянием строк меню другого окна.

```
#include <windows.h>

BOOL        RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT     CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char        szClass[ ]="GetMenuInfo";

int WINAPI WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass,COLOR_WINDOW) )
        return FALSE;
    hwnd = CreateWindow(szClass,
        "Управление состоянием строк меню другого окна",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
```

```
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc; wc.style=wc.cbClsExtra=wc.cbWndExtra = 0;
    wc.lpfnWndProc = Proc;    wc.hInstance=hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = NULL;    wc.lpszClassName = szName;
    return (RegisterClass(&wc) != 0);
}

UINT GetMenuItem(HMENU hMenu, UINT uIns)
{   MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_ID;   GetMenuItemInfo(hMenu, uIns, 1, &mii);
    return mii.wID;
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam )
{   static  HMENU  hMainMenu, hFiles; static HWND hOther;
    switch (msg)
    {   case WM_CREATE:
        {   hOther=FindWindow("SetMenuInfo",
                    "Управление состоянием строк меню");
            if (!hOther)
            {   MessageBox(hwnd,"Такого окна нет",
                    "Сообщение",MB_OK);    return 0;
            }
            hMainMenu=GetMenu(hOther);
            if (!hMainMenu)
            {   MessageBox(hwnd,"Окно не имеет меню",
                    "Сообщение",MB_OK);    return 0;
            }
            hFiles=GetSubMenu(hMainMenu, 0);
            if (!hFiles)
            {   MessageBox(hwnd,"Нет такого элемента меню",
```

```
                "Сообщение",MB_OK);     return 0;
        }
        int items=GetMenuItemCount(hFiles);
        char str[50]; _itoa(items,str,10);
        strcat(str," – элементов в меню "Файлы" другого окна");
        MessageBox(hwnd,str,"Сообщение",MB_OK);
        return 0;
    }
    case WM_LBUTTONDOWN:
    {   UINT uCom=GetMenuItem(hFiles, 0);
        if (!uCom)
            MessageBox(hwnd, "Команда равна нулю",
                "Сообщение",MB_OK);
        SendMessage(hOther, WM_COMMAND, uCom, 0L);
        return 0;
    }
    case WM_DESTROY: {    PostQuitMessage(0);  return 0; }
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

До запуска этого приложения должно быть запущено приложение из листинга 4.3.

Рассмотрим, как работает это приложение.

Приложение основную работу выполняет при обработке сообщения WM_CREATE.

Перечислим эти действия:

1. Определяет, существует ли окно класса "SetMenuInfo" с заголовком "Управление состоянием строк меню":

```
hOther=FindWindow("SetMenuInfo",
                "Управление состоянием строк меню");
```

2. Запрашивает дескриптор главного меню этого окна:

```
hMainMenu=GetMenu(hOther);
```

3. Запрашивает дескриптор временного меню в позиции 0:

```
hFiles=GetSubMenu(hMainMenu, 0);
```

4. Определяет количество элементов в меню hFiles:

```
int items=GetMenuItemCount(hFiles);
```

5. Формирует и выдает сообщение об этом:

```
char str[50]; _itoa(items,str,10);
strcat(str," – элементов в меню "Файлы" другого окна");
MessageBox(hwnd,str,"Сообщение",MB_OK);
```

После нажатия левой клавиши мыши активизируется первый элемент меню:

1. Определяется значение команды элемента меню hFiles:

```
UINT uCom=GetMenuItem(hFiles, 0);
```

Обратите внимание, что функция с именем GetMenuItem в этом приложении приведена к виду:

```
UINT GetMenuItem(HMENU hMenu, UINT uIns)
{   MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_ID;   GetMenuItemInfo(hMenu, uIns, 1, &mii);
    return mii.wID;
}
```

Теперь эта функция возвращает идентификатор команды элемента в указанной позиции. Например, вызов GetMenuItem(hFiles,0) возвращает идентификатор команды первого элемента меню hFiles.

2. Посылается сообщение WM_COMMAND с идентификатором команды uCom:

```
SendMessage(hOther, WM_COMMAND, uCom, 0L);
```

По сути, функции окна приложения из листинга 4.3 посылается сообщение WM_COMMAND с кодом wParam, младшее слово которого равно идентификатору команды CM_FILE_OPEN.

Приложение листинга 4.3 немедленно выполняет эту команду. Эта команда поступает независимо от состояния элемента "Открыть" (он может быть и заблокирован).

Сообщение WM_COMMAND с идентификатором uCom можно посылать сколь угодно раз. Каждый раз в главное меню вставляются элементы "Правка" и "Вид". Это говорит о том, что нельзя полагаться на то, что заблокированный элемент меню не может послать команду. Перед обработкой сообщения нужно проверять состояние элемента, которое имеет право посылать это сообщение.

4.3. Сообщения от меню

Меню посылает сообщения в функцию того окна, к которому оно подключено функцией SetMenu.

4.3.1. Сообщение WM_INITMENU

Сообщение WM_INITMENU поступает **перед отображением главного меню**. Параметр wParam равен дескриптору меню. Если сообщение WM_INITMENU обрабатывают, то возвращают 0. Обработка обычно сводится к изменению состояния элементов меню.

4.3.2. Сообщение WM_INITMENUPOPUP

Это сообщение поступает **перед отображением временного меню**. Параметр wParam равен дескриптору меню. Младшее слово параметра lParam равно позиции этого меню в меню верхнего уровня, старшее слово lParam равно 1 для системного меню и 0 – для обычного. Если это сообщение обрабатывают, то возвращают 0. Обработка обычно сводится к изменению состояния элементов меню.

4.3.3. Сообщение WM_COMMAND

Это сообщение поступает после выбора строки меню. Младшее слово параметра wParam равно идентификатору выбранной команды, а старшее слово равно 0. После обработки сообщения возвращают 0.

4.3.4. Сообщение WM_MENUSELECT

Сообщение WM_MENUSELECT поступает в процессе перемещения курсора меню по строкам меню. Младшее слово параметра wParam равно идентификатору команды или позиции строки (если при выборе строки отображается временное меню), а старшее слово содержит флажки состояния элементов меню из следующей таблицы:

Значение	*Состояние элемента*
MF_CHECKED	Отмечен
MF_DISABLED	Заблокирован
MF_GRAYED	Недоступен
MF_HILITE	Высвечен
MF_MOUSESELECT	Выбран мышью
MF_POPUP	Открывает временное меню
MF_SYSMENU	Принадлежит системному меню окна. Параметр lParam содержит дескриптор этого меню

Если старшее слово wParam содержит 0xFFFF и lParam=NULL, то Windows закрыл меню. Параметр lParam содержит дескриптор меню, по которому перемещается курсор.

Если это сообщение обрабатывают, то возвращают 0.

4.4. Плавающее меню

Плавающее меню создают обычным способом, но не вставляют в другое меню. Для отображения и выбора строк этого меню вызывают функцию TrackPopupMenu:

```
BOOL TrackPopupMenu( HMENU hMenu, UINT uFlags,  int x, int y,   int nReserved,
HWND hwnd,  CONST RECT *prcRect );
```

Эта функция выводит на экран плавающее меню и создает свой собственный цикл обработки сообщений, завершающий работу после выбора строки. Она не возвращает управление до тех пор, пока работа с меню не будет завершена выбором строки или отказом от выбора.

Параметры:

1. **hMenu** – дескриптор отображаемого плавающего меню. Он может быть создан функцией CreatePopupMenu или получен с помощью функции GetSubMenu.
2. **uFlags** – комбинация флажков, которые задают функциональные параметры плавающего меню.
   2.1. Следующие константы задают способ размещения меню по горизонтали относительно параметра **x**:

Константа	*Пояснение*
TPM_CENTERALIGN	Центр меню по горизонтали совпадает с x
TPM_LEFTALIGN	Левый край меню совпадает с x
TPM_RIGHTALIGN	Правый край меню совпадает с x

   2.2. Следующие константы задают способ размещения меню по вертикали относительно параметра **y**:

Константа	*Пояснение*
TPM_BOTTOMALIGN	Нижний край меню совпадает с y
TPM_TOPALIGN	Верхний край меню совпадает с y
TPM_VCENTERALIGN	Центр меню по вертикали совпадает с y

   2.3. Следующие константы задают способ выбора строк меню без указания окна-владельца для меню:

Константа	*Пояснение*
TPM_NONOTIFY	Не посылать сообщения о выборе строки
TPM_RETURNCMD	Возвращать идентификатор выбранной команды

2.4. Следующие константы задают кнопку мыши, которую прослеживает плавающее меню:

Константа	*Пояснение*
TPM_LEFTBUTTON	Прослеживает левую кнопку мыши
TPM_RIGHTBUTTON	Прослеживает правую кнопку мыши

3. **x** – координата по горизонтали от левого края экрана.
4. **y** – координата по вертикали от верхнего края экрана.
5. **nReserved** – зарезервированный параметр, должен быть всегда равен нулю.
6. **hwnd** – дескриптор уже существующего окна-владельца, которое получит сообщения от меню. Сообщение WM_COMMAND окно получит только после завершения работы функции TrackPopupMenu.
7. **prcRect** – указатель на прямоугольную область, находясь в пределах которой можно работать с меню. Если сделать щелчок мышью за пределами этого прямоугольника, плавающее меню исчезнет. Если prcRect=NULL, то эта область ограничена прямоугольной рамкой плавающего меню.

В случае успешного выполнения функция возвращает ненулевое значение. Если в параметре uFlags задано TPM_RETURNCMD, то возвращаемое значение равно идентификатору команды выбранной строки. Если элемент не выбран, возвращаемое значение – нуль.

Задача. После нажатия правой клавиши мыши отобразить плавающее меню.

Листинг 4.5. Плавающее меню.

```
#include <windows.h>

#define  CM_EDIT_CUT     2003
#define  CM_EDIT_COPY    2004
#define  CM_EDIT_PASTE   2005

BOOL        RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT     CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char        szClass[ ]="FloatMenu";

int WINAPI WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass,COLOR_WINDOW) )
        return FALSE;
```

```
    hwnd = CreateWindow(szClass, "Окно с плавающим меню",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc; wc.style=wc.cbClsExtra=wc.cbWndExtra = 0;
    wc.lpfnWndProc = Proc;    wc.hInstance=hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = NULL;    wc.lpszClassName = szName;
    return (RegisterClass(&wc) != 0);
}
BOOL CreateMenuItem(HMENU hMenu, char *str, UINT uIns,
        UINT uCom, HMENU hSubMenu, BOOL flag, UINT fType)
{   MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE | MIIM_TYPE | MIIM_SUBMENU | MIIM_ID;
    mii.fType = fType;  mii.fState= MFS_ENABLED;
    mii.dwTypeData = str;  mii.cch=sizeof(str);
    mii.wID=uCom;     mii.hSubMenu=hSubMenu;
    return InsertMenuItem(hMenu, uIns, flag, &mii);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam )
{   switch (msg)
    {   case WM_COMMAND:
        {   switch (LOWORD(wParam))
            {   case CM_EDIT_CUT:
                {   MessageBox(hwnd,"Вырезать", "Сообщение", MB_OK );
                    return 0;
                }
                case CM_EDIT_COPY:
                {   MessageBox(hwnd,"Копировать", "Сообщение",MB_OK);
                    return 0;
                }
                case CM_EDIT_PASTE:
```

```
            {   MessageBox(hwnd,"Вклеить", "Сообщение", MB_OK );
                return 0;
            }
        }   return 0;
    }
    case WM_RBUTTONDOWN:
    {   //Берем экранные координаты курсора мыши
        DWORD   xyPos=GetMessagePos();
        WORD    xPos=LOWORD(xyPos),
                yPos=HIWORD(xyPos);
        HMENU  hFloatMenu=CreatePopupMenu();
        int i=0;
        CreateMenuItem( hFloatMenu, "Вырезать", i++,
            CM_EDIT_CUT, NULL, FALSE, MFT_STRING );
        CreateMenuItem( hFloatMenu, "Копировать", i++,
            CM_EDIT_COPY, NULL, FALSE, MFT_STRING );
        CreateMenuItem( hFloatMenu, "Вклеить", i++,
            CM_EDIT_PASTE, NULL, FALSE, MFT_STRING );
        //Выводим меню в позиции курсора мыши
        TrackPopupMenu(hFloatMenu,
            TPM_CENTERALIGN | TPM_LEFTBUTTON | TPM_VCENTERALIGN,
            xPos, yPos, 0, hwnd, NULL );
        DestroyMenu( hFloatMenu );
        return 0;
    }
    case WM_DESTROY: { PostQuitMessage(0); return 0; }
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

При нажатии правой клавиши мыши в функцию окна поступает сообщение WM_RBUTTONDOWN. При этом параметр lParam содержит координаты курсора мыши относительно левого верхнего угла рабочей области окна. Но для функции TrackPopupMenu требуются экранные координаты. Чтобы меню всплыло вблизи курсора мыши, определяют экранные координаты курсора. С этой целью вызывают функцию GetMessagePos и выделяют горизонтальную xPos и вертикальную yPos составляющие координат:

```
xyPos=GetMessagePos();
xPos=LOWORD(xyPos), yPos=HIWORD(xyPos);
```

Далее обычным способом создают временное меню.

На последнем этапе выводят всплывающее меню:

```
TrackPopupMenu(hFloatMenu,
    TPM_CENTERALIGN | TPM_LEFTBUTTON |
    TPM_VCENTERALIGN, xPos, yPos, 0, hwnd, NULL);
```

Второй аргумент вызова и координаты xPos, yPos подобраны так, что после "всплытия" меню курсор мыши оказывается точно посредине меню.

После выбора строки меню автоматически исчезает, но перед этим посылает сообщение WM_COMMAND с кодом выбранной команды. При обработке этого сообщения на экран выдается сообщение о выбранной строке. Затем уничтожают созданное временное меню.

4.5. Акселераторы

Для быстрого доступа к командам используют акселераторы. Их иногда называют "клавишами быстрого вызова" команд меню. В действительности же акселераторы могут быть связаны с любыми командами.

При поступлении сообщения WM_COMMAND от акселератора младшее слово параметра wParam содержит идентификатор связанной с акселератором команды, а старшее слово wParam равно 1.

Операционная система широко использует акселераторы. Например, стандартное системное меню практически любого окна содержит строку "Закрыть Alt+F4" и команда этой строки связана с акселератором Alt+F4. То есть одновременное нажатие клавиш Alt и F4 равноценно выбору строки "Закрыть Alt+F4" системного меню активного окна.

Приложение все используемые акселераторы должно записать в одну таблицу и работать с дескриптором этой таблицы. Для работы с таблицей существует несколько функций.

Функция CreateAcceleratorTable создает таблицу акселераторов:

```
HACCEL CreateAcceleratorTable( LPACCEL lpaccl, int cEntries );
```

Здесь lpaccl – указатель на массив структур типа ACCEL, который содержит описания акселераторов, а cEntries – количество структур в массиве lpaccl. В случае успешного создания функция возвращает дескриптор созданной таблицы, иначе – NULL.

Каждую созданную функцией CreateAcceleratorTable таблицу нужно разрушать до завершения работы приложения. Для этого вызывают функцию DestroyAcceleratorTable.

Структура ACCEL задает данные одного акселератора и описана следующим образом:

```
typedef struct
{   BYTE  fVirt;
    WORD key;
```

```
    WORD cmd;
}   ACCEL;
```

Назначение полей структуры ACCEL:

1. **fVirt** задает флажки акселератора и может быть комбинацией следующих значений:

Значение	*Пояснение*
FALT	Клавишу key нажимают при нажатой клавише Alt
FCONTROL	Клавишу key нажимают при нажатой клавише Ctrl
FNOINVERT	Определяет, что никакой верхнего уровня пункт меню не высвечен, когда акселератор используется. Если этот флажок не определен, верхнего уровня пункт меню будет высвечен, если возможно, когда акселератор используется
FSHIFT	Клавиша key нажимают при нажатой клавише Shift
FVIRTKEY	Поле key содержит код виртуальной клавиши. Если флажок FVIRTKEY не установлен, то предполагается, что поле key содержит код символа ASCII

2. **key** задает акселератор и может быть кодом виртуальной клавиши или символа ASCII. Список допустимых значений кода приведен в табл. 4.1.
3. **cmd** – идентификатор команды. Если нажать определяемую акселератором комбинацию клавиш, то функция окна получит сообщение WM_COMMAND или WM_SYSCOMMAND. При этом младшее слово параметра wParam будет содержать значение cmd.

Задача. Создать таблицу акселераторов для генерирования восьми наиболее часто используемых команд.

Следующий фрагмент содержит образец решения данной задачи.

```
#define CM_FILE_NEW    1000
#define CM_FILE_OPEN   1001
#define CM_FILE_SAVE   1002
#define CM_FILE_QUIT   1003
#define CM_EDIT_CUT    2000
#define CM_EDIT_PASTE  2001
#define CM_EDIT_COPY   2002
#define CM_EDIT_DEL2003
...
HACCEL   CreateAccelTable( void )
{   //Массив акселераторов
    ACCEL Accel[8];
```

```
    //Создать
    Accel[0].fVirt   = FVIRTKEY | FCONTROL;
    Accel[0].key     = 0x4e;
    Accel[0].cmd     = CM_FILE_NEW;
    //Открыть
    Accel[1].fVirt   = FVIRTKEY | FCONTROL;
    Accel[1].key     = 0x4f;
    Accel[1].cmd     = CM_FILE_OPEN;
    //Сохранить
    Accel[2].fVirt   = FVIRTKEY | FCONTROL;
    Accel[2].key     = 0x53;
    Accel[2].cmd     = CM_FILE_SAVE;
    //Выход
    Accel[3].fVirt   = FVIRTKEY | FALT;
    Accel[3].key     = 0x73;
    Accel[3].cmd     = CM_FILE_QUIT;
    //Вырезать
    Accel[4].fVirt   = FVIRTKEY | FCONTROL;
    Accel[4].key     = 0x58;
    Accel[4].cmd     = CM_EDIT_CUT;
    //Вклеить
    Accel[5].fVirt   = FVIRTKEY | FCONTROL;
    Accel[5].key     = 0x56;
    Accel[5].cmd     = CM_EDIT_PASTE;
    //Копировать
    Accel[6].fVirt   = FVIRTKEY | FCONTROL;
    Accel[6].key     = 0x43;
    Accel[6].cmd     = CM_EDIT_COPY;
    //Удалить
    Accel[7].fVirt   = FVIRTKEY;
    Accel[7].key     = 0x2e;
    Accel[7].cmd     = CM_EDIT_DEL;
    return CreateAcceleratorTable( (LPACCEL)Accel, 8 );
}
```

Проанализируем этот фрагмент.

1. Задают идентификаторы команд, с которыми связаны акселераторы:

```
#define CM_FILE_NEW        1000
...
#define CM_EDIT_DEL        2003
```

2. Описывают функцию, которая заполняет массив акселераторами, создает таблицу и возвращает дескриптор созданной таблицы. Заголовок функции имеет вид:

```
HACCEL   CreateAccelTable( void )
```

2.1. Описывают массив акселераторов:

```
ACCEL Accel[8];
```

2.2. Для каждого элемента массива задают данные. Например, первую структуру в массиве заполняют следующим образом:

```
//Создать
Accel[0].fVirt   = FVIRTKEY | FCONTROL;
Accel[0].key     = 0x4e;
Accel[0].cmd     = CM_FILE_NEW;
```

Рассмотрим, для чего и каким образом выбираются эти данные.

Пусть раздел "Файлы" главного меню окна содержит элемент "Создать" с идентификатором команды CM_FILE_NEW и при выборе этой строки функция окна получает сообщение WM_COMMAND с параметром wParam, младшее слово которого содержит идентификатор CM_FILE_NEW.

В приложениях выбор строки создания нового файла (документа, проекта и т. д.) обычно дублируют сочетанием клавиш Ctrl+N. Для создания такого же акселератора для строки "Создать", в поле fVirt записано значение FCONTROL. Это означает, что клавиша, заданная полем key, должна быть нажата при нажатой клавише Ctrl. В поле key записан код выбранной клавиши – 0x4e. Как видно из табл. 4.1, этому коду соответствует клавиша <N>.

Для того чтобы акселератор, состоящий из комбинации клавиши <N> и клавиши Ctrl, работал вне зависимости от состояния клавиши <Caps Lock>, в поле key записан виртуальный код. Об этом отдельно сообщается в поле fVirt – оно содержит значение FVIRTKEY. Виртуальный код имеет свои преимущества. Например, при использовании кодов ASCII акселератор активизировался бы только при установке режима заглавных букв с помощью клавиши <Caps Lock>.

Так же задают значения остальных элементов массива.

2.3. Указатель на массив и количество элементов массива передают функции CreateAcceleratorTable:

```
CreateAcceleratorTable( (LPACCEL)Accel, 8 );
```

Эта функция в случае успешного выполнения возвращает дескриптор созданной таблицы акселераторов, иначе – NULL.

2.4. Возвращенное функцией CreateAcceleratorTable возвращают из функции CreateAccelTable:

```
return CreateAcceleratorTable( (LPACCEL)Accel, 8 );
```

Для того чтобы при нажатии акселераторов операционная система могла формировать соответствующие сообщения, вызывают функцию TranslateAccelerator:

```
int TranslateAccelerator( HWND hwnd, HACCEL hAccTable, LPMSG lpMsg );
```

Эта функция транслирует сообщение WM_KEYDOWN или WM_SYSKEYDOWN в сообщение вида к WM_COMMAND или WM_SYSCOMMAND и затем посылает сообщение WM_COMMAND или WM_SYSCOMMAND непосредственно функции окна hwnd. Она завершает работу только после обработки сообщения. В случае успешного выполнения функция возвращает ненулевое значение.

Параметры:

1. **hwnd** – дескриптор окна.
2. **hAccTable** – дескриптор таблицы акселераторов. Эта таблица должна быть или загружена вызовом функции LoadAccelerators или создана функцией CreateAcceleratorTable.
3. **lpMsg** – указатель на структуру типа MSG, выбранный из очереди сообщений функцией GetMessage или PeekMessage.

Для отличения сообщения функции TranslateAccelerator от сообщений других источников (например, от меню) операционная система в старшее слово параметра wParam сообщения WM_COMMAND или WM_SYSCOMMAND записывает 1.

Сообщения акселераторов, дублирующих строки системного меню, транслируют в сообщение WM_SYSCOMMAND. Сообщения других акселераторов транслируют в сообщение WM_COMMAND.

Если функция TranslateAccelerator вернула ненулевое значение, это означает, что сообщение обработано функцией указанного окна. И это сообщение не нужно передавать функции TranslateMessage или DispatchMessage.

Перечислим **сходства и отличия использования акселератора от процесса выбора строки меню**:

1. Если команда акселератора соответствует команде элемента меню, то при нажатии акселератора функция окна получает сообщения WM_INITMENU и WM_INITMENUPOPUP так же, как и при выборе строки меню с помощью мыши.

2. При нажатии акселератора функция окна не получит сообщений WM_INITMENU и WM_INITMENUPOPUP в следующих случаях:

- окно заблокировано;
- элемент меню заблокирован;
- акселератор не соответствует элементу системного меню и это окно свернуто;
- поступают сообщения от мыши.

3. Если указанное в вызове функции TranslateAccelerator окно активно и другие окна не имеют фокуса ввода (это может быть, если указанное окно свернуто), то TranslateAccelerator вместо WM_KEYUP или WM_KEYDOWN транслирует сообщение WM_SYSKEYUP или WM_SYSKEYDOWN.

4. Если указанное функции TranslateAccelerator окно свернуто, то она посылает сообщение WM_COMMAND, только если нажатый акселератор не связан с командой ни одного из элементов меню указанного окна.

Для использования акселераторов цикл обработки сообщений нужно изменить следующим образом:

```
HACCEL hAccel=CreateAccelTable();
...
while ( GetMessage(&msg, 0, 0, 0) )
{
    if ( !hAccel || !TranslateAccelerator(hwnd, hAccel, &msg) )
    {
        TranslateMessage(&msg); DispatchMessage(&msg);
    }
}
```

В этом фрагменте переменная hAccel содержит дескриптор созданной таблицы акселераторов. Если hAccel не равен NULL, вызывается функция TranslateAccelerator. Эта функция ищет в очереди сообщений сообщения от клавиатуры, соответствующие определенным в таблице акселераторам, преобразует такие сообщения в сообщения WM_COMMAND и WM_SYSCOMMAND и посылает их в функцию окна, минуя очередь сообщений приложения.

Младшее слово параметра wParam в последних двух сообщениях равно идентификатору, указанному в таблице акселераторов для данной комбинации клавиш. Старшее слово параметра wParam содержит 1 для сообщений, которые пришли от акселераторов, и 0 для сообщений, которые пришли от меню.

Задача. В окне приложения разработать меню, содержащее типичный набор строк для работы с файлами. Наиболее часто применяемые команды продублировать акселераторами.

Листинг 4.6. Акселераторы в меню.

```
#include <windows.h>

#define  CM_FILE_NEW     1000
#define  CM_FILE_OPEN    1001
#define  CM_FILE_SAVE    1002
#define  CM_FILE_QUIT    1003
#define  CM_EDIT_CUT     2000
#define  CM_EDIT_PASTE   2001
#define  CM_EDIT_COPY    2002
#define  CM_EDIT_DEL     2003
#define  CM_HELP_HELP    3000
#define  CM_HELP_ABOUT   3001

BOOL        RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT     CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HACCEL      CreateAccelTable( void );
HINSTANCE   hInstance;
char        szClass[ ]="AccelClass";

int WINAPI  WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd; ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass, COLOR_WINDOW) )
        return FALSE;
    hwnd = CreateWindow(szClass,"Акселераторы",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
        0,0,hInstance,NULL);
    if (!hwnd) return FALSE;
    HACCEL hAccel=CreateAccelTable();
    while(GetMessage(&msg, 0, 0, 0))
    {   if (!hAccel || !TranslateAccelerator(hwnd, hAccel, &msg))
        {   TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
    DestroyAcceleratorTable(hAccel);
    return msg.wParam;
}
```

```
BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc;  wc.style=wc.cbClsExtra=wc.cbWndExtra=0;
    wc.lpfnWndProc = Proc;    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground+1);
    wc.lpszMenuName=NULL; wc.lpszClassName=szName;
    return (RegisterClass(&wc) != 0);
}

BOOL CreateMenuItem(HMENU hMenu, char *str, UINT uIns,
                    UINT uCom, HMENU hSubMenu, BOOL flag, UINT fType)
{   MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE | MIIM_TYPE | MIIM_SUBMENU | MIIM_ID;
    mii.fType = fType;          mii.fState= MFS_ENABLED;
    mii.dwTypeData = str;       mii.cch=sizeof(str);
    mii.wID=uCom;               mii.hSubMenu=hSubMenu;
    return InsertMenuItem(hMenu, uIns, flag, &mii);
}

HACCEL   CreateAccelTable( void )
{   //Массив акселераторов
    ACCEL Accel[8];
    //Создать
    Accel[0].fVirt= FVIRTKEY | FCONTROL;
    Accel[0].key  = 0x4e;   Accel[0].cmd  = CM_FILE_NEW;
    //Открыть
    Accel[1].fVirt= FVIRTKEY | FCONTROL;
    Accel[1].key  = 0x4f;   Accel[1].cmd  = CM_FILE_OPEN;
    //Сохранить
    Accel[2].fVirt= FVIRTKEY | FCONTROL;
    Accel[2].key  = 0x53;   Accel[2].cmd  = CM_FILE_SAVE;
    //Выход
    Accel[3].fVirt= FVIRTKEY | FALT;
    Accel[3].key  = 0x73;   Accel[3].cmd  = CM_FILE_QUIT;
    //Вырезать
    Accel[4].fVirt= FVIRTKEY | FCONTROL;
    Accel[4].key  = 0x58;   Accel[4].cmd  = CM_EDIT_CUT;
    //Вклеить
    Accel[5].fVirt= FVIRTKEY | FCONTROL;
    Accel[5].key  = 0x56;   Accel[5].cmd  = CM_EDIT_PASTE;
    //Копировать
```

```
    Accel[6].fVirt= FVIRTKEY | FCONTROL;
    Accel[6].key = 0x43;   Accel[6].cmd = CM_EDIT_COPY;
    //Удалить
    Accel[7].fVirt= FVIRTKEY;
    Accel[7].key = 0x2e;   Accel[7].cmd = CM_EDIT_DEL;
    return CreateAcceleratorTable((LPACCEL)Accel,8);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                          WPARAM wParam, LPARAM lParam )
{   static HMENU  hMainMenu, hFileMenu, hEditMenu, hHelpMenu;
    switch (msg)
    {   case WM_CREATE:
        {   hMainMenu=CreateMenu();
            SetMenu(hwnd,hMainMenu);
            //Создаем временное меню для раздела "Файлы"
            hFileMenu=CreatePopupMenu();
            int i=0; //Инициализация позиции в меню
            CreateMenuItem(hFileMenu,"&Новый\t Ctrl+N",
                i++, CM_FILE_NEW, NULL, FALSE, MFT_STRING);
            CreateMenuItem(hFileMenu,"&Открыть\t Ctrl+O",
                i++,CM_FILE_OPEN, NULL, FALSE, MFT_STRING);
            CreateMenuItem(hFileMenu,"&Сохранить\t Ctrl+S",
                i++,CM_FILE_SAVE, NULL, FALSE, MFT_STRING);
            CreateMenuItem(hFileMenu,NULL,i++,0,
                NULL, FALSE, MFT_SEPARATOR);
            CreateMenuItem(hFileMenu,"Выход\t Alt+F4",
                i++,CM_FILE_QUIT, NULL, FALSE, MFT_STRING);
            hEditMenu=CreatePopupMenu();
            i=0; //Инициализация позиции в меню
            CreateMenuItem(hEditMenu,"&Вырезать\t Ctrl+X",
                i++,CM_EDIT_CUT, NULL, FALSE, MFT_STRING);
            CreateMenuItem(hEditMenu,"B&ставить\t Ctrl+V",
                i++,CM_EDIT_PASTE, NULL, FALSE, MFT_STRING);
            CreateMenuItem(hEditMenu,"&Копировать\t Ctrl+C",
                i++, CM_EDIT_COPY, NULL, FALSE, MFT_STRING);
            CreateMenuItem(hEditMenu,"&Удалить\t Delete",
                i++, CM_EDIT_DEL, NULL, FALSE, MFT_STRING);
            hHelpMenu=CreatePopupMenu();
            i=0; //Инициализация позиции в меню
            CreateMenuItem(hHelpMenu,"&Помощь",
```

```
        i++,CM_HELP_HELP, NULL, FALSE, MFT_STRING);
    CreateMenuItem(hHelpMenu,NULL,i++,0,
        NULL, FALSE, MFT_SEPARATOR);
    CreateMenuItem(hHelpMenu,"&О программе",
        i++,CM_HELP_ABOUT, NULL, FALSE, MFT_STRING);
    //Подключаем временные меню к главному меню
    i=0; //Инициализация позиции в меню
    CreateMenuItem(hMainMenu,"&Файл", i++, 0,
        hFileMenu,FALSE, MFT_STRING);
    CreateMenuItem(hMainMenu,"&Правка", i++, 0,
        hEditMenu,FALSE, MFT_STRING);
    CreateMenuItem(hMainMenu,"&Помощь", i++, 0,
        hHelpMenu,FALSE, MFT_STRING);
    DrawMenuBar(hwnd);  return 0;
}
case WM_COMMAND:
{   char str[30];
    if (HIWORD(wParam)==1)
        strcpy(str,"Сообщение от акселератора");
    else    strcpy(str,"Сообщение от меню");
    switch (LOWORD(wParam))
    {   case CM_FILE_NEW:
        {   MessageBox(hwnd,"Создать", str, MB_OK);
            return 0;
        }
        case CM_FILE_OPEN:
        {   MessageBox(hwnd,"Открыть", str, MB_OK);
            return 0;
        }
        case CM_FILE_SAVE:
        {   MessageBox(hwnd,"Сохранить", str, MB_OK);
            return 0;
        }
        case CM_FILE_QUIT: {    DestroyWindow(hwnd); return 0; }
        case CM_EDIT_CUT:
        {   MessageBox(hwnd,"Вырезать", str, MB_OK);
            return 0;
        }
        case CM_EDIT_PASTE:
        {   MessageBox(hwnd,"Вклеить", str, MB_OK);
            return 0;
```

```
            }
        case CM_EDIT_COPY:
        {   MessageBox(hwnd,"Копировать", str, MB_OK);
            return 0;
            }
        case CM_EDIT_DEL:
        {   MessageBox(hwnd,"Удалить", str, MB_OK);
            return 0;
            }
        case CM_HELP_HELP:
        {   MessageBox(hwnd,"Помощь","Помощь", MB_OK);
            return 0;
            }
        case CM_HELP_ABOUT:
        {   MessageBox(hwnd,
            "Демонстрация подключения акселераторов",
            "О программе", MB_OK);
            return 0;
            }
    }
  }
  case WM_DESTROY: {   PostQuitMessage(0); return 0; }
 } return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Проанализируем те изменения в этом приложении, которые обусловлены включением работы с акселераторами.

1. Главная функция содержит описание дескриптора таблицы акселераторов, трансляцию сообщений акселераторов в цикле обработки сообщений и разрушение таблицы перед завершением работы приложения:

```
HACCEL hAccel=CreateAccelTable();
while(GetMessage(&msg, 0, 0, 0))
{  if (!hAccel || !TranslateAccelerator(hwnd, hAccel, &msg))
   {  TranslateMessage(&msg); DispatchMessage(&msg);
      }
}
DestroyAcceleratorTable(hAccel);
```

Дескриптор таблицы акселераторов должен быть описан до цикла обработки сообщений. Здесь таблица создается единственный раз при запуске приложения. Если таблица создается в другом месте или модифицируется при обработке тех или иных сообщений, то этот дескриптор

нужно описывать глобально. Например, до описания главной функции нужно записать оператор вида HACCEL hAccel;.

Тогда в любом месте приложения можно построить новую таблицу акселераторов и запомнить их дескриптор обычным способом. Например, если есть функция с именем CreateAccelTable1, то можно записать так:

```
hAccel=CreateAccelTable1();
```

2. Функция CreateAccelTable была подробно рассмотрена выше. Таких функций может быть несколько, если требуется модифицировать таблицу акселераторов в процессе работы приложения.

3. Для различения сообщения от акселераторов используют значение старшего слова параметра wParam. При поступлении сообщения WM_COMMAND от акселератора это значение равно 1.

Нижеследующий фрагмент анализирует это значение и составляет заголовок окна сообщения в зависимости от того, каким образом было послано сообщение:

```
char str[30];
if ( HIWORD(wParam)==1 ) strcpy(str,"Сообщение от акселератора");
else     strcpy(str,"Сообщение от меню");
```

Контрольные вопросы

1. Перечислите основные виды меню и укажите различия между ними.

2. Запишите алгоритм создания главного меню, состоящего из двух разделов, если в каждом разделе строки связаны с командами.

3. Для чего в цикле обработки сообщений в листинге 4.1 вызывают функцию TranslateMessage?

4. Какие меню посылают сообщения функции окна и какие меню нужно разрушать при обработке сообщения WM_DESTROY?

5. Каким образом можно послать сообщение WM_COMMAND с идентификатором команды, если связанная с этой командой строка заблокирована?

6. Чем отличаются плавающие меню от остальных видов меню?

7. Чем отличается генерация команды с помощью акселератора от генерации команды путем выбора строки меню?

8. В каком порядке нужно описать основные объекты приложения, если таблица акселераторов изменяется при обработке сообщений?

9. В чем преимущество использования виртуального кода в акселераторах?

Упражнения

1. Главное меню содержит строки "Невидимый курсор", "Обычный курсор" и "Выход". Создать плавающее меню с такими же строками. Чтобы курсор стал невидимым, вызвать функцию ShowCursor(0); видимым – ShowCursor(1).

2. Раздел "Пользователи" главного меню содержит список пользователей. При выборе пользователя в главном меню появляются дополнительные разделы. При смене пользователя меняются и эти разделы.

3. При открытии или создании документа появляется раздел "Правка" с единственной командой "Выделить". После выбора этой команды в этом разделе добавляются строки "Удалить" и "Копировать", а команда "Выделить" отмечается галочкой (которая убирается при повторном выборе). Если выбрать команду "Копировать", то элемент "Удалить" заменяется элементом "Вставить".

4. Главное меню содержит раздел "Файл", в котором перечислены строки с именами команд "Создать", "Открыть" и "Выход". После выбора строк "Создать" или "Открыть" добавить строки "Сохранить" и "Печать", а также раздел "Правка" со строками "Вырезать", "Вклеить" и "Копировать". Команды связать с акселераторами.

5. Главное меню содержит раздел "Файл" с именами команд "Создать", "Открыть", "Сохранить", "Закрыть", "Печать" и "Выход", а также раздел "Правка" со строками "Вырезать", "Вклеить" и "Копировать". После выбора команды "Закрыть" удалить раздел "Правка". Команды связать с акселераторами.

6. Главное меню содержит раздел "Рисунок" с именами четырех геометрических фигур. После выбора фигуры отобразить фигуру в определенной части окна. При нажатии правой клавиши мыши над любой отображенной фигурой на месте нажатия отобразить плавающее меню с соответствующими выбранной фигуре командами.

7. Главное меню содержит раздел "Файл" со строками "Создать", "Открыть" и "Выход". При выборе строки "Создать" или "Открыть" создать окно, которое содержит меню с разделами "Правка" и "Эффекты". Команды связать с акселераторами.

8. Главное меню содержит раздел "Файл" со строками "Создать", "Открыть" и "Выход". При выборе строк "Создать" или "Открыть" создать окно, которое содержит раздел меню "Фигуры" со списком имен геометрических фигур; при выборе имени отобразить фигуру с таким именем и пометить имя галочкой. При повторном выборе имени убрать фигуру и удалить галочку.

9. Главное меню содержит раздел "Фигуры" с именами геометрических фигур. При выборе названия фигуры в главное меню добавить раздел с названием фигуры и перечислением основных ее параметров в этом разделе. При повторном нажатии должны исчезнуть этот раздел и галочка.

10. Строки главного меню расположить в нескольких линиях, а строки плавающего меню – в одну линию.

11. Создать плавающее меню для выбора и установки вида курсора мыши. При выборе имени вида курсора курсор мыши должен принять соответствующий вид. Для загрузки и показа курсора использовать операторы вида

```
HCURSOR hCursor=LoadCursor(NULL, IDC_CROSS);
SetCursor(hCursor);
```

12. Главное меню содержит раздел "Файл" со строками "Новый", "Открыть" и "Выход" и раздел "Помощь" со строками "Содержание" и "О программе". При выборе строки "Содержание" появляются строки "Введение", "Часть 1", "Часть 2" ... а при выборе строки "Часть ..." появляются строки "Раздел 1", "Раздел 2" ...

13. Рабочую область окна полностью занимают два временных окна. Главное меню первого окна содержит раздел "Файл" со строками "Открыть" и "Выход". Если выбрать строку "Открыть", то во втором окне появляется главное меню с разделом "Правка".

14. Плавающее меню содержит строки "Спрятать", "Показать", "Масштаб", "Свойства". Строка "Масштаб" указывает на временное меню из четырех зависимых строк: "50%", "100%", "150%" и "200%", при выборе одна из которых отмечается кружочком.

15. Рабочую область окна приложения занимают два временных окна. Главное меню первого временного окна содержит раздел "Файл" с командами "Создать", "Открыть", "Демо-версия" и "Выход". Причем состоянием строки "Демо-версия" управляет второе окно.

16. Главное меню содержит раздел "Файл" со строками "Создать", "Открыть" и "Выход". При выборе команды "Создать" или "Открыть" добавить раздел "Правка" со строками "Вырезать", "Вклеить" и "Копировать". Команды только отображаемых строк связать с акселераторами.

17. Главное меню содержит раздел "Файл", в котором перечислены строки с именами команд "Создать", "Открыть" и "Выход", которые могут быть отмечены как зависимые переключатели. После выбора строк "Создать" или "Открыть" добавить раздел "Правка" с командами "Вырезать", "Вклеить" и "Копировать", которые могут быть отмечены как независимые флажки. Команды отмеченных строк связать с акселераторами.

18. Главное меню содержит раздел "Фигуры" с зависимым списком имен геометрических фигур. При выборе имени должна быть отображена только эта фигура и отмечено кружочком только это имя.

19. Раздел "Файл" содержит строки "Создать", "Открыть", "Демо-версия" и "Выход". При выборе строки "Создать" или "Открыть" создать перекрывающееся окно с разделом меню "Эффекты". Список строк раздела "Эффекты" зависит от состояния строки "Демо-версия". Команды отображенных строк меню связать с акселераторами.

20. Главное меню содержит раздел "Файл" со строкой "Открыть". При выборе этой строки в главное меню добавить раздел "Правка" со строками "Вырезать", "Копировать" и "Удалить", удалить строку "Открыть" и добавить строку "Закрыть". При выборе строки "Закрыть" вернуться к исходному состоянию. Команды отображенных строк меню связать с акселераторами.

21. Рабочую область окна приложения полностью занимает временное окно с пустым главным меню. Главное меню окна приложения содержит раздел "Файл" со строками "Открыть" и "Закрыть" (заблокирована). При выборе команды "Открыть" создать главное меню временного окна с разделом "Правка" со строками "Вырезать", "Копировать" и "Удалить". После этого заблокировать строку "Открыть" и разблокировать команду "Закрыть". При выборе строки "Закрыть" вернуться к исходному состоянию.

22. На месте нажатия правой клавиши мыши всплывает меню. Если курсор мыши ближе к верхнему или нижнему краю рабочей области, то строки меню выстроить в линию, иначе – в столбик.

23. Главное меню содержит раздел "Пользователи", в котором перечислены строки с именами типов пользователей. После выбора типа пользователя этот раздел исчезает и появляется раздел "Данные", в котором перечислены общие для всех типов пользователей и типичные только для выбранного типа строки данных. Команды отображенных строк связать с акселераторами.

24. Главное меню содержит раздел "Цвета" с пятью именами стандартных цветов Windows и раздел "Фигуры" с именами трех плоских фигур. После выбора цвета и фигуры отобразить фигуру выбранным цветом, а соответствующие строки меню отметить галочкой. При повторном нажатии должны исчезнуть эта фигура и галочки.

25. На месте нажатия правой клавиши мыши отобразить плавающее меню, отмеченные галочкой элементы которого указывают на временные меню, строки которых служат зависимыми переключателями.

Глава 5
Панель инструментов и строка состояния

Панель инструментов и строка состояния являются окнами дополнительных классов Win32 API.

Функции для работы с окнами таких классов объявлены в заголовочном файле commctrl.h. Поэтому в текст приложения необходимо включить файл commctrl.h. Эти функции описаны в файле comctl32.dll. До сих пор мы использовали объекты статических библиотек системы: kernel32.lib, user32.lib, gdi32.lib, winspool.lib, comdlg32.lib, advapi32.lib, shell32.lib, ole32.lib, oleaut32.lib, uuid.lib, odbc32.lib и odbccp32.lib. Эти библиотеки автоматически подключаются к приложению Win32 API. В этом можно убедиться, заглянув в установки проекта интегрированной среды (Project\Settings\Link→ "Object/library modules:"). Для использования дополнительных классов окон сюда же нужно приписать текст "comctl32.lib".

Теперь можно пользоваться дополнительными классами.

5.1. Панель инструментов

Создание и использование панели инструментов очень похоже на работу с акселераторами. Кнопки панели, как и акселераторы, связаны с командами. Подобно дескриптору таблицы акселераторов создается и дескриптор таблицы кнопок. Отличие заключается в структуре кнопок, функции создания и форме отображения панели.

5.1.1. Создание панели инструментов

Для создания панели инструментов вызывают функцию

```
HWND CreateToolbarEx(HWND hwnd, DWORD ws, UINT wID,
    int nBitmaps, HINSTANCE hBMInst, UINT wBMID,
    LPCTBBUTTON lpButtons, int iNumButtons,
    int dxButton, int dyButton, dxBitmap, dyBitmap,
    UINT uStructSize );
```

Она создает окно панели инструментов и добавляет в него заданные кнопки.

Параметры функции:

1. **hwnd** – дескриптор родительского окна панели.

2. **ws** – стиль панели. Должен содержать константу WS_CHILD, часто дополняют константой TBSTYLE_TOOLTIPS.
3. **wID** – идентификатор панели инструментов.
4. **nBitmaps** – количество изображений кнопок в контейнере, хранящемся в файле hBMInst под номером wBMID. Изображения кнопок (обычно формата ICO, размером 32x32, 16 цветов) хранятся в файлах EXE и DLL, упакованные по несколько штук в контейнерах. В одном файле может быть несколько контейнеров. Функция находит контейнеры по порядковому номеру, извлекает из них нужные изображения и выводит на кнопках.
5. **hBMInst** – дескриптор экземпляра приложения, содержащего контейнер с кнопками.
6. **wBMID** – идентификатор запрашиваемого формата кнопки.
7. **lpButtons** – указатель на массив структур типа TBBUTTON, который содержит информацию о кнопках создаваемой панели инструментов.
8. **iNumButtons** – количество кнопок панели инструментов.
9. **dxButton** – ширина кнопок в пикселях.
10. **dyButton** – высота кнопок в пикселях.
11. **dxBitmap** – ширина изображения кнопки в пикселях.
12. **dyBitmap** – высота изображения кнопки в пикселях.
13. **uStructSize** – размер структуры TBBUTTON.

Если параметры dxButton, dyButton, dxBitmap и dyBitmap задать равными 0, то размеры кнопок берутся по умолчанию.

В случае успешного создания панели функция возвращает дескриптор окна созданной панели инструментов, иначе – NULL.

Структура TBBUTTON содержит информацию о кнопке панели инструментов:

```
typedef struct
{   int     iBitmap;
    int     idCommand;
    BYTE    fsState;
    BYTE    fsStyle;
    DWORD   dwData;
    int     iString;
}   TBBUTTON;
```

Назначение полей этой структуры:

1. **iBitmap** – номер изображения кнопки в контейнере.

2. **idCommand** – идентификатор связанный с кнопкой команды. Если fsStyle=TBSTYLE_SEP, то idCommand должен быть равен 0.
3. **fsState** – флажки состояния кнопки. В общем случае кнопки панели могут иметь комбинацию следующих состояний:

Константа	*Состояние кнопки*
TBSTATE_CHECKED	Кнопка стиля TBSTYLE_CHECK нажата
TBSTATE_ENABLED	Кнопка доступна
TBSTATE_HIDDEN	Кнопка невидима и недоступна
TBSTATE_INDETERMINATE	Кнопка недоступна
TBSTATE_PRESSED	Кнопка нажата
TBSTATE_WRAP	Используют в комбинации с константой TBSTATE_ENABLED. Тогда следующая кнопка панели будет изображена в начале очередной строки

Приложение может посылать панели инструментов сообщения TB_GETSTATE и TB_SETSTATE.

Сообщение TB_GETSTATE возвращает значение состояния кнопки панели (например, кнопка доступна, нажата или отмечена). При этом wParam приравнивают идентификатору команды кнопки, а lParam=0. В случае аварии возвращаемое значение равно -1.

Сообщение TB_SETSTATE устанавливает состояние кнопки панели. При этом wParam равен идентификатору команды кнопки, а lParam= (LPARAM)MAKELONG(fsState, 0) задает устанавливаемое состояние fsState. В случае аварии возвращаемое значение равно 0.

4. **fsStyle** – стиль кнопки. Может быть комбинацией следующих констант:

Константа	*Стиль кнопки*
TBSTYLE_TOOLTIPS	Кнопка посылает уведомительное сообщение WM_NOTIFY при остановке курсора мыши над ней на полсекунды
TBSTYLE_WRAPABLE	Кнопки разместить в несколько линий
TBSTYLE_BUTTON	Стандартная кнопка
TBSTYLE_CHECK	Кнопка, которая может находиться в нажатом и ненажатом состояниях. В зависимости от состояния кнопка отображается различным цветовым фоном

Константа	*Стиль кнопки*
TBSTYLE_CHECKGROUP	Группа кнопок стиля TBSTYLE_CHECK, только одна из которых может находиться в нажатом состоянии
TBSTYLE_GROUP	Группа стандартных кнопок
TBSTYLE_SEP	Кнопка в виде малого промежутка

Кнопка стиля TBSTYLE_BUTTON ведет себя подобно стандартной кнопке. Кнопка стиля TBSTYLE_CHECK также подобна стандартной кнопке, но после каждого нажатия переключается между нажатым и ненажатым состояниями. Можно создавать группу обычных кнопок (стиль TBSTYLE_GROUP) и группу переключаемых кнопок (стиль TBSTYLE_CHECKGROUP). В последней группе только одна кнопка может находиться в нажатом состоянии.

5. **dwData** – задаваемое приложением значение.
6. **iString** – номер строки кнопки.

Параметры dwData и iString чаще всего задают нулевыми.

Задача. Главное меню окна содержит разделы "Файлы" и "Правка". При выборе строки "Файлы" отображается временное меню со строками "Открыть", "Сохранить" и "Выход". При выборе строки "Правка" отображается временное меню со строками "Найти" и "Заменить". Команды строк "Открыть", "Сохранить", "Найти" и "Заменить" продублировать кнопками панели инструментов.

Листинг 5.1. Пример меню и панели инструментов.

```
#include <windows.h>
#include <commctrl.h>
#pragma comment (lib,"comctl32.lib")
#define ID_TOOLBAR      100

#define CM_FILE_OPEN      1001
#define CM_FILE_SAVE      1002
#define CM_FILE_QUIT      1003
#define CM_EDIT_FIND      2001
#define CM_EDIT_REPLC     2002

BOOL        RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT     CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char        szClass[ ]="MenuTool";
```

```
int WINAPI WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG msg;  HWND hwnd;  :: hInstance=hInstance;
    if ( !RegClass(WndProc, szClass,COLOR_WINDOW) )
            return FALSE;
    hwnd = CreateWindow(szClass, " Панель инструментов",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0))
    {   TranslateMessage(&msg);  DispatchMessage(&msg);  }
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
                        UINT brBackground)
{   WNDCLASS wc; wc.style=wc.cbClsExtra=wc.cbWndExtra = 0;
    wc.lpfnWndProc = Proc;     wc.hInstance=hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = NULL;     wc.lpszClassName = szName;
    return (RegisterClass(&wc) != 0);
}

BOOL CreateMenuItem(HMENU hMenu, char *str, UINT uIns,
            UINT uCom, HMENU hSubMenu, BOOL flag, UINT fType)
{   MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE | MIIM_TYPE | MIIM_SUBMENU | MIIM_ID;
    mii.fType = fType;          mii.fState= MFS_ENABLED;
    mii.dwTypeData = str;       mii.cch=sizeof(str);
    mii.wID=uCom;               mii.hSubMenu=hSubMenu;
    return InsertMenuItem(hMenu, uIns, flag, &mii);
}

HWND CreateToolBar(HWND hwnd, DWORD dwStyle, UINT uCom)
{   static TBBUTTON but[6];
    but[0].fsStyle=TBSTYLE_SEP;
    but[1].iBitmap=STD_FILEOPEN;
    but[1].idCommand=CM_FILE_OPEN;
```

```
    but[1].fsState=TBSTATE_ENABLED;
    but[1].fsStyle=TBSTYLE_GROUP;

    but[2].iBitmap=STD_FILESAVE;
    but[2].idCommand=CM_FILE_SAVE;
    but[2].fsState=TBSTATE_ENABLED;
    but[2].fsStyle=TBSTYLE_GROUP;
    but[3].fsStyle=TBSTYLE_SEP;

    but[4].iBitmap=STD_FIND;
    but[4].idCommand=CM_EDIT_FIND;
    but[4].fsState=TBSTATE_ENABLED;
    but[4].fsStyle=TBSTYLE_CHECKGROUP;

    but[5].iBitmap=STD_REPLACE;
    but[5].idCommand=CM_EDIT_REPLC;
    but[5].fsState=TBSTATE_ENABLED;
    but[5].fsStyle=TBSTYLE_CHECKGROUP;

    return CreateToolbarEx( hwnd, dwStyle, uCom, 0,
        HINST_COMMCTRL, IDB_STD_LARGE_COLOR,
        but, 6, 0, 0, 0, 0, sizeof(TBBUTTON) );
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                          WPARAM wParam, LPARAM lParam )
{   static  HMENU   hMainMenu, hFileMenu, hEditMenu;
    static  HWND hToolbar;
    switch (msg)
    {   case WM_SIZE:
        {   MoveWindow(hToolbar,0,0,0,0,TRUE);  return 0; }
        case WM_CREATE:
        {   hMainMenu=CreateMenu();
            //Создаем временное меню для раздела "Файлы"
            hFileMenu=CreatePopupMenu();
            int i=0; //Инициализация позиции в меню hFileMenu
            CreateMenuItem(hFileMenu,"&Открыть",i++,
                    CM_FILE_OPEN, NULL, FALSE, MFT_STRING);
            CreateMenuItem(hFileMenu,"&Сохранить",i++,
                    CM_FILE_SAVE, NULL, FALSE, MFT_STRING);
            CreateMenuItem(hFileMenu,"&Выход",i++,
                    CM_FILE_QUIT, NULL, FALSE, MFT_STRING);
            //Создаем временное меню для раздела "Правка"
```

```
    hEditMenu=CreatePopupMenu();
    i=0; //Инициализация позиции в меню hEditMenu
    CreateMenuItem(hEditMenu,"&Найти",i++,
            CM_EDIT_FIND, NULL, FALSE, MFT_STRING);
    CreateMenuItem(hEditMenu,"&Заменить",i++,
            CM_EDIT_REPLC, NULL, FALSE, MFT_STRING);
    //Подключаем временные меню к главному меню
    i=0; //Инициализация позиции в меню hMainMenu
    CreateMenuItem(hMainMenu,"&Файл",i++, 0,
       hFileMenu,FALSE, MFT_STRING);
    CreateMenuItem(hMainMenu,"&Правка",i++,0,
       hEditMenu,FALSE, MFT_STRING);
    SetMenu(hwnd,hMainMenu);
    DrawMenuBar(hwnd);

    //Создаем панель инструментов
    DWORD dwStyle = WS_CHILD | WS_VISIBLE | WS_DLGFRAME;
    hToolbar=CreateToolBar(hwnd, dwStyle, ID_TOOLBAR);
    return 0;
}
case WM_COMMAND:
{   switch (LOWORD(wParam))
    {   case CM_FILE_OPEN:
        {   MessageBox(hwnd,
                    "Команда CM_FILE_OPEN", "Меню",MB_OK);
            return 0;
        }
        case CM_FILE_SAVE:
        {   MessageBox(hwnd,
                    "Команда CM_FILE_SAVE", "Меню",MB_OK);
            return 0;
        }
        case CM_FILE_QUIT:
        {   DestroyWindow(hwnd); return 0;   }
        case CM_EDIT_FIND:
        {   MessageBox(hwnd,
                    "Команда CM_EDIT_FIND", "Меню",MB_OK);
            return 0;
        }
        case CM_EDIT_REPLC:
        {   MessageBox(hwnd,
```

```
                    "Команда CM_EDIT_REPLC", "Меню",MB_OK);
                return 0;
            }
        }
        return 0;
    }
    case WM_DESTROY: {    PostQuitMessage(0); return 0; }
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Нетрудно увидеть, что это приложение является развитием приложения листинга 4.1. Рассмотрим, что оно делает.

При запуске приложение создает меню и панель инструментов из шести кнопок. Первая и четвертая кнопки представляют собой "малый промежуток" на левом краю и между кнопками "Сохранить" и "Найти" панели инструментов. Остальные 4 кнопки соответствуют легко узнаваемым стандартным кнопкам Windows: "Открыть", "Сохранить", "Найти" и "Заменить". При нажатии на любую из этих кнопок функция окна приложения получает такое же сообщение, как если бы была выбрана соответствующая строка меню. Например, при нажатии на кнопку "Открыть" функция окна получит сообщение WM_COMMAND с идентификатором CM_FILE_OPEN в младшем слове параметра wParam.

Рассмотрим изменения в тексте приложения:

1. Включены еще один заголовочный файл, библиотека "comctl32.lib" и создан идентификатор окна панели инструментов:

```
#include <commctrl.h>
#pragma comment (lib,"comctl32.lib")
#define ID_TOOLBAR              100
```

Это связано с тем, что панель инструментов, по сути, есть орган управления и, как все органы управления, имеет свой идентификатор.

2. Описана функция CreateToolBar, которая в родительском окне hwnd создает панель инструментов с идентификатором uCom (в нашем случае это ID_TOOLBAR) стиля dwStyle и возвращает дескриптор окна созданной панели.

Рассмотрим операторы функции CreateToolBar:

2.1. Сначала описан массив из пяти элементов типа TBBUTTON:

```
static TBBUTTON but[5];
```

2.2. Первая и четвертая кнопки является "малым промежутком", о чем сообщает значение поля fsStyle:

```
but[0].fsStyle=TBSTYLE_SEP; but[3].fsStyle=TBSTYLE_SEP;
```

Остальные поля кнопки этого стиля могут быть нулевыми.

2.3. Вторая кнопка связана с командой CM_FILE_OPEN. Для его отображения используется изображение с номером STD_FILEOPEN:

```
but[1].iBitmap=STD_FILEOPEN;
```

Здесь используются номера стандартных изображений иконок из файла commctrl.h:

```
#define  STD_CUT          0
#define  STD_COPY         1
#define  STD_PASTE        2
#define  STD_UNDO         3
#define  STD_REDOW        4
#define  STD_DELETE       5
#define  STD_FILENEW      6
#define  STD_FILEOPEN     7
#define  STD_FILESAVE     8
#define  STD_PRINTPRE     9
#define  STD_PROPERTIES   10
#define  STD_HELP         11
#define  STD_FIND         12
#define  STD_REPLACE      13
#define  STD_PRINT        14
```

Для кнопок можно использовать и другие изображения. В частности, можно самим построить изображения иконок. Но эти задачи здесь не рассматриваются.

В поле idCommand второго элемента массива записывают значение команды, с которой нужно связать кнопку:

```
but[1].idCommand=CM_FILE_OPEN;
```

Далее записывают значение состояния и стиль кнопки:

```
but[1].fsState=TBSTATE_ENABLED;
but[1].fsStyle= TBSTYLE_GROUP;
```

В стиле этой кнопки с таким же успехом можно было пользоваться константой TBSTYLE_BUTTON.

Остальные кнопки создаются таким же образом.

Для разнообразия последние две кнопки имеют стиль зависимых переключателей одной группы:

```
but[4].fsStyle=TBSTYLE_CHECKGROUP;
but[5].fsStyle=TBSTYLE_CHECKGROUP;
```

Это означает, что только одна из них может находиться в нажатом состоянии. Эти кнопки будут вести себя подобно зависимым переключателям стиля BS_AUTORADIOBUTTON.

2.4. После задания массива кнопок создается панель инструментов:

```
CreateToolbarEx( hwnd, dwStyle, uCom, 0,
        HINST_COMMCTRL, IDB_STD_LARGE_COLOR,
        but, 16, 0, 0, 0, 0, sizeof(TBBUTTON));
```

Четвертым аргументом здесь указан 0. Это допускается только для системных контейнеров изображений.

В качестве дескриптора экземпляра приложения, содержащего контейнер, используется константа HINST_COMMCTRL. Она определена в файле commctrl.h:

```
#define HINST_COMMCTRL   ((HINSTANCE)-1)
```

Идентификатор IDB_STD_LARGE_COLOR формата кнопки для панели инструментов также описан в файле commctrl.h:

```
#define  IDB_STD_SMALL_COLOR   0
#define  IDB_STD_LARGE_COLOR   1
```

Во многих случаях, например в диалоговых панелях, могут пригодиться другие форматы:

```
#define  IDB_VIEW_SMALL_COLOR  4
#define  IDB_VIEW_LARGE_COLOR  5
#define  IDB_HIST_SMALL_COLOR  8
#define  IDB_HIST_LARGE_COLOR  9
```

2.5. Функция CreateToolBar возвращает дескриптор окна созданной панели инструментов.

3. В функции родительского окна описывают дескриптор окна панели инструментов:

```
static  HWND hToolbar;
```

4. При изменении размеров родительского окна "перемещают" панель инструментов:

```
case WM_SIZE:
{   MoveWindow( hToolbar, 0, 0, 0, 0, TRUE ); return 0;  }
```

Обратите внимание, что все координаты и размеры "перемещения" могут быть равны нулю. Это объясняется тем, что при перемещении функция окна панели автоматически устанавливает ширину панели по ширине рабочей области родительского окна и высоту панели по высоте кнопок.

5. После создания и перерисовки полосы меню родительского окна создают панель инструментов.

5.1. Указывают стиль панели инструментов:

```
dwStyle = WS_CHILD | WS_VISIBLE | WS_DLGFRAME;
```

5.2. Вызывают функцию CreateToolBar:

```
hToolbar=CreateToolBar(hwnd, dwStyle, ID_TOOLBAR);
```

Другие изменения, обусловленные созданием панели инструментов, в тексте приложения не обязательны.

5.1.2. Управление состоянием кнопок панели

Панель инструментов во многом ведет себя как обычное окно. Например, в нем, кроме кнопок, можно расположить другие органы управления и управлять ими обычным способом. Но управление кнопками панели имеет свои отличия.

Задача. Главное меню окна изначально содержит раздел "Файлы". При выборе строки "Файлы" отображается временное меню со строками "Открыть", "Сохранить" и "Выход". Причем строка "Сохранить" заблокирована.

При выборе строки "Открыть" заблокировать ее, разблокировать строку "Сохранить" и в главное меню добавить раздел "Правка".

При выборе строки "Правка" отображается временное меню со строками "Найти" и "Заменить". Причем эти строки при их выборе должны быть отмечены как независимые флажки.

При выборе строки "Сохранить" заблокировать ее, разблокировать строку "Открыть" и из главного меню удалить элемент "Правка".

Команды для строк "Открыть", "Сохранить", "Найти" и "Заменить" продублировать кнопками панели инструментов. Кнопки должны изменять свои состояния подобно соответствующим строкам.

Листинг 5.2. Управление кнопками панели инструментов.

```
#include <windows.h>
#include <commctrl.h>
#pragma comment (lib,"comctl32.lib")
#define ID_TOOLBAR      100
#define CM_FILE_OPEN    1001
#define CM_FILE_SAVE    1002
#define CM_FILE_QUIT    1003
#define CM_EDIT_FIND    2001
#define CM_EDIT_REPLC   2002
```

```
BOOL        RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT     CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char        szClass[ ]="CtrlToolbar";

int WINAPI WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass,COLOR_WINDOW) )
            return FALSE;
    hwnd = CreateWindow(szClass, "Панель инструментов",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0))
    {   TranslateMessage(&msg); DispatchMessage(&msg); }
    return msg.wParam;
}
BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc; wc.style=wc.cbClsExtra=wc.cbWndExtra = 0;
    wc.lpfnWndProc = Proc;    wc.hInstance=hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = NULL;    wc.lpszClassName = szName;
    return (RegisterClass(&wc) != 0);
}

BOOL SetMenuItem(HMENU hMenu, UINT uIns, UINT fState, BOOL flag)
{   MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask   = MIIM_STATE | MIIM_ID;
    mii.fState = fState;  mii.wID = uIns;
    return SetMenuItemInfo( hMenu, uIns, flag, &mii );
}

UINT GetMenuItem(HMENU hMenu, UINT uIns, BOOL flag)
{   MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE;
    GetMenuItemInfo(hMenu, uIns, flag, &mii);
    return mii.fState;
}
```

```
BOOL CreateMenuItem(HMENU hMenu, char *str, UINT uIns,
          UINT uCom, HMENU hSubMenu, BOOL flag, UINT fType)
{   MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE | MIIM_TYPE | MIIM_SUBMENU | MIIM_ID;
    mii.fType = fType;  mii.fState= MFS_ENABLED;
    mii.dwTypeData = str;  mii.cch=sizeof(str);
    mii.wID=uCom;      mii.hSubMenu=hSubMenu;
    return InsertMenuItem(hMenu, uIns, flag, &mii);
}

HWND CreateToolBar(HWND hwnd, DWORD dwStyle, UINT uCom)
{   static TBBUTTON but[6];
    but[0].fsStyle=TBSTYLE_SEP;

    but[1].iBitmap=STD_FILEOPEN;
    but[1].idCommand=CM_FILE_OPEN;
    but[1].fsState=TBSTATE_ENABLED;
    but[1].fsStyle=TBSTYLE_GROUP;

    but[2].iBitmap=STD_FILESAVE;
    but[2].idCommand=CM_FILE_SAVE;
    but[2].fsState=TBSTATE_ENABLED;
    but[2].fsStyle=TBSTYLE_GROUP;

    but[3].fsStyle=TBSTYLE_SEP;

    but[4].iBitmap=STD_FIND;
    but[4].idCommand=CM_EDIT_FIND;
    but[4].fsState=TBSTATE_HIDDEN;
    but[4].fsStyle=TBSTYLE_GROUP;

    but[5].iBitmap=STD_REPLACE;
    but[5].idCommand=CM_EDIT_REPLC;
    but[5].fsState=TBSTATE_HIDDEN;
    but[5].fsStyle=TBSTYLE_GROUP;

    return CreateToolbarEx(hwnd,dwStyle,uCom,0,
        HINST_COMMCTRL, IDB_STD_LARGE_COLOR,
        but,6,0,0,0,0,sizeof(TBBUTTON));
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                          WPARAM wParam, LPARAM lParam )
```

```
{   static   HMENU    hMainMenu, hFileMenu, hEditMenu;
    static   HWND hToolbar;
    switch (msg)
    {   case WM_SIZE:
        {   MoveWindow(hToolbar,0, 0, 0, 0,TRUE);   return 0; }
        case WM_CREATE:
        {   hMainMenu=CreateMenu();
            //Создаем временное меню для раздела "Файлы"
            hFileMenu=CreatePopupMenu();
            int i=0; //Инициализация позиции в меню hFileMenu
            CreateMenuItem( hFileMenu, "&Открыть", i++,
                CM_FILE_OPEN, NULL, FALSE, MFT_STRING );
            CreateMenuItem( hFileMenu, "&Сохранить", i++,
                CM_FILE_SAVE, NULL, FALSE, MFT_STRING );
            CreateMenuItem( hFileMenu, NULL, i++, 0,
                NULL, FALSE, MFT_SEPARATOR );
            CreateMenuItem( hFileMenu, "&Выход", i++,
                CM_FILE_QUIT, NULL, FALSE, MFT_STRING );
            //Создаем временное меню для раздела "Правка"
            hEditMenu=CreatePopupMenu();
            i=0; //Инициализация позиции в меню hEditMenu
            CreateMenuItem( hEditMenu, "&Найти", i++,
                CM_EDIT_FIND, NULL, FALSE, MFT_STRING );
            CreateMenuItem( hEditMenu, "&Заменить", i++,
                CM_EDIT_REPLC, NULL, FALSE, MFT_STRING );
            //Подключаем временные меню к главному меню
            i=0; //Инициализация позиции в меню hMainMenu
            CreateMenuItem( hMainMenu, "&Файл", i++, 0,
                hFileMenu, FALSE, MFT_STRING );
            SetMenu(hwnd,hMainMenu);
            SetMenuItem(hFileMenu,
                CM_FILE_SAVE, MFS_GRAYED, FALSE);
            DrawMenuBar(hwnd);
            //Создаем панель инструментов
            DWORD dwStyle = WS_CHILD | WS_VISIBLE | WS_DLGFRAME;
            hToolbar=CreateToolBar(hwnd, dwStyle, ID_TOOLBAR);
            return 0;
        }
        case WM_COMMAND:
        {   switch (LOWORD(wParam))
            {   case CM_FILE_OPEN:
```

```
{   if (GetMenuItem(hFileMenu,
        CM_FILE_OPEN, FALSE)==MFS_GRAYED)  return 0;
    CreateMenuItem(hMainMenu, "&Правка",1,0,
        hEditMenu,FALSE, MFT_STRING);
    SetMenuItem(hFileMenu, CM_FILE_OPEN,
        MFS_GRAYED, FALSE);
    SetMenuItem(hFileMenu, CM_FILE_SAVE,
        MFS_ENABLED, FALSE);
    DrawMenuBar(hwnd);
    SendMessage(hToolbar,TB_SETSTATE,
        (WPARAM)CM_FILE_OPEN, (LPARAM)MAKELONG
        (TBSTATE_INDETERMINATE,0));
    SendMessage(hToolbar,TB_SETSTATE,
        (WPARAM)CM_FILE_SAVE, (LPARAM)MAKELONG
        (TBSTATE_ENABLED,0));
    if (GetMenuItem(hEditMenu,
        CM_EDIT_FIND, FALSE)==MFS_CHECKED)
        SendMessage(hToolbar, TB_SETSTATE,
            (WPARAM)CM_EDIT_FIND, (LPARAM)MAKELONG
            (TBSTATE_CHECKED | TBSTATE_ENABLED,0));
    else    SendMessage(hToolbar, TB_SETSTATE,
            (WPARAM)CM_EDIT_FIND, (LPARAM)MAKELONG
            (TBSTATE_ENABLED,0));
    if (GetMenuItem(hEditMenu,
        CM_EDIT_REPLC,FALSE)==MFS_CHECKED)
        SendMessage(hToolbar, TB_SETSTATE,
            (WPARAM)CM_EDIT_REPLC, (LPARAM)MAKELONG
            (TBSTATE_CHECKED | TBSTATE_ENABLED,0));
    else    SendMessage(hToolbar, TB_SETSTATE,
            (WPARAM)CM_EDIT_REPLC, (LPARAM)MAKELONG
            (TBSTATE_ENABLED,0));
    return 0;
}
case CM_FILE_SAVE:
{   if (GetMenuItem(hFileMenu,
            CM_FILE_SAVE, FALSE)==MFS_GRAYED)
            return 0;
    RemoveMenu(hMainMenu,1, MF_BYPOSITION);
    SetMenuItem(hFileMenu,
            CM_FILE_SAVE, MFS_GRAYED, FALSE);
    SetMenuItem(hFileMenu,
```

```
                CM_FILE_OPEN, MFS_ENABLED, FALSE);
        DrawMenuBar(hwnd);
        SendMessage(hToolbar,TB_SETSTATE,
                (WPARAM)CM_FILE_OPEN, (LPARAM)MAKELONG
                (TBSTATE_ENABLED,0));
        SendMessage(hToolbar,TB_SETSTATE,
                (WPARAM)CM_FILE_SAVE, (LPARAM)MAKELONG
                (TBSTATE_INDETERMINATE,0));
        SendMessage(hToolbar,TB_SETSTATE,
                (WPARAM)CM_EDIT_FIND, (LPARAM)MAKELONG
                (TBSTATE_HIDDEN,0));
        SendMessage(hToolbar,TB_SETSTATE,
                (WPARAM)CM_EDIT_REPLC, (LPARAM)MAKELONG
                (TBSTATE_HIDDEN,0));
        return 0;
    }
    case CM_EDIT_FIND:
    {   if (GetMenuItem(hEditMenu,
                CM_EDIT_FIND, FALSE)==MFS_CHECKED)
        {   SetMenuItem(hEditMenu,
                CM_EDIT_FIND, MFS_UNCHECKED, FALSE);
            SendMessage(hToolbar,
                TB_SETSTATE, (WPARAM)CM_EDIT_FIND,
                (LPARAM)MAKELONG (TBSTATE_ENABLED,0));
        }
        else
        {   SetMenuItem(hEditMenu,
                CM_EDIT_FIND, MFS_CHECKED, FALSE);
            SendMessage(hToolbar, TB_SETSTATE,
                (WPARAM)CM_EDIT_FIND, (LPARAM)MAKELONG
                (TBSTATE_CHECKED | TBSTATE_ENABLED,0));
        }   return 0;
    }
    case CM_EDIT_REPLC:
    {   if (GetMenuItem(hEditMenu,
                CM_EDIT_REPLC, FALSE)==MFS_CHECKED)
        {   SetMenuItem(hEditMenu,
                CM_EDIT_REPLC, MFS_UNCHECKED, FALSE);
            SendMessage(hToolbar, TB_SETSTATE,
                (WPARAM)CM_EDIT_REPLC, (LPARAM)MAKELONG
                (TBSTATE_ENABLED,0));
```

```
                }
                else
                {   SetMenuItem(hEditMenu,
                            CM_EDIT_REPLC, MFS_CHECKED, FALSE);
                    SendMessage(hToolbar, TB_SETSTATE,
                            (WPARAM)CM_EDIT_REPLC, (LPARAM)MAKELONG
                            (TBSTATE_CHECKED | TBSTATE_ENABLED,0));
                }   return 0;
            }
            case CM_FILE_QUIT:
            {   DestroyWindow(hwnd); return 0;   }
        }   return 0;
    }
    case WM_DESTROY: { PostQuitMessage(0); return 0; }
}
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Работа этого приложения похожа на работу приложения из листинга 4.3. В целях сокращения объема текста в этом примере отсутствуют раздел меню "Вид" и соответствующие кнопки панели инструментов.

При запуске приложения полоса меню отображает только раздел "Файл" главного меню. А на панели инструментов отображены две кнопки: "Открыть" и "Сохранить". Причем последняя заблокирована.

Рассмотрим, как это достигнуто.

Функция CreateToolBar изначально создает панель из шести кнопок: "малый промежуток", "Открыть", "Сохранить", "Малый промежуток", "Найти" и "Заменить". При этом изначально доступна только кнопка "Открыть", о чем говорит ее состояние:

```
but[1].fsState=TBSTATE_ENABLED;
```

Кнопка "Сохранить" видима, но недоступна:

```
but[2].fsState=TBSTATE_INDETERMINATE;
```

Кнопки "Найти" и "Заменить" изначально создают невидимыми:

```
but[4].fsState=TBSTATE_HIDDEN;    but[5].fsState=TBSTATE_HIDDEN;
```

Если выбрать строку или нажать на кнопку "Открыть", в главном меню появляется раздел "Правка" со строками "Найти" и "Заменить", строка "Открыть" блокируется, "Сохранить" – разблокируется. На панели инструментов эти действия дублируют: появляются кнопки "Найти" и "Заменить", кнопка "Открыть" блокируется, "Сохранить" – разблокируется.

Рассмотрим, как это достигается.

Очевидно, что эти действия выполняют при обработке сообщения WM_COMMAND со значением LOWORD(wParam), равным CM_FILE_OPEN. Обсудим шаги обработки этого сообщения.

1. Первая часть хорошо знакома по примеру листинга 4.3.
   1.1. Если строка "Открыть" заблокирована, то обработку завершить.
   1.2. Иначе подключить раздел "Правка" к главному меню.
   1.3. Заблокировать строку "Открыть" в разделе "Файл" и разблокировать строку "Сохранить".
   1.4. Перерисовать полосу меню.
2. Вторая часть посвящена управлению состоянием кнопок панели инструментов. Здесь вызывают функцию SendMessage со специфическими для кнопок панели инструментов параметрами.
   2.1. Заблокировать кнопку "Открыть":

```
SendMessage(hToolbar,TB_SETSTATE, (WPARAM)CM_FILE_OPEN,
    (LPARAM)MAKELONG(TBSTATE_INDETERMINATE,0));
```

   2.2. Разблокировать кнопку "Сохранить":

```
SendMessage(hToolbar,TB_SETSTATE,  (WPARAM)CM_FILE_SAVE,
    (LPARAM)MAKELONG(TBSTATE_ENABLED,0));
```

   2.3. Кнопка "Найти" дублирует одноименную строку меню и отображает состояние этого элемента меню. Поэтому перед отображением кнопки "Найти" нужно анализировать состояние элемента меню "Найти". Если этот элемент меню отмечен, то и кнопку "Найти" нужно отметить и оставить в доступном состоянии:

```
SendMessage(hToolbar,TB_SETSTATE, (WPARAM)CM_EDIT_FIND,
    (LPARAM)MAKELONG (TBSTATE_CHECKED | TBSTATE_ENABLED,0));
```

   иначе кнопку "Найти" отобразить в доступном состоянии:

```
SendMessage(hToolbar,TB_SETSTATE, (WPARAM)CM_EDIT_FIND,
    (LPARAM)MAKELONG(TBSTATE_ENABLED,0));
```

   2.4. Так же управляют состоянием кнопки "Заменить":

```
if (GetMenuItem(hEditMenu,CM_EDIT_REPLC,FALSE)== MFS_CHECKED)
    SendMessage(hToolbar,TB_SETSTATE, (WPARAM)CM_EDIT_REPLC,
        (LPARAM)MAKELONG (TBSTATE_CHECKED | TBSTATE_ENABLED,0));
else
    SendMessage(hToolbar,TB_SETSTATE, (WPARAM)CM_EDIT_REPLC,
        (LPARAM)MAKELONG(TBSTATE_ENABLED,0));
```

Этим завершают обработку команды, которая поступает после выбора строки меню "Открыть" или нажатия кнопки панели инструментов "Открыть".

Намного меньше усилий требует обработка команды, которая поступает после выбора строки меню "Сохранить" или нажатия кнопки панели инструментов "Сохранить":

1. Если элемент меню "Сохранить" заблокирован, то завершить обработку:

```
if (GetMenuItem(hFileMenu, CM_FILE_SAVE, FALSE)==MFS_GRAYED) return 0;
```

2. Иначе удалить раздел меню "Правка":

```
RemoveMenu( hMainMenu, 1, MF_BYPOSITION );
```

3. Заблокировать строку меню "Сохранить":

```
SetMenuItem( hFileMenu, CM_FILE_SAVE, MFS_GRAYED, FALSE);
```

4. Разблокировать строку меню "Открыть":

```
SetMenuItem( hFileMenu, CM_FILE_OPEN, MFS_ENABLED, FALSE);
```

5. Перерисовать полосу меню:

```
DrawMenuBar(hwnd);
```

6. Разблокировать кнопку "Открыть":

```
SendMessage(hToolbar,TB_SETSTATE, (WPARAM)CM_FILE_OPEN,
      (LPARAM)MAKELONG(TBSTATE_ENABLED,0));
```

7. Заблокировать кнопку "Сохранить":

```
SendMessage(hToolbar,TB_SETSTATE, (WPARAM)CM_FILE_SAVE,
      (LPARAM)MAKELONG(TBSTATE_INDETERMINATE,0));
```

8. Скрыть кнопку "Найти":

```
SendMessage(hToolbar,TB_SETSTATE, (WPARAM)CM_EDIT_FIND,
      (LPARAM)MAKELONG(TBSTATE_HIDDEN,0));
```

9. Скрыть кнопку "Заменить":

```
SendMessage(hToolbar,TB_SETSTATE, (WPARAM)CM_EDIT_REPLC,
      (LPARAM)MAKELONG(TBSTATE_HIDDEN,0));
```

Этим завершают обработку команды, которая поступает после выбора строки меню "Сохранить" или нажатия кнопки панели инструментов "Сохранить".

Обработку команд от строк и кнопок "Найти" и "Заменить" рассмотрим на примере команды, которая поступает после выбора строки меню "Заменить" или нажатия кнопки панели инструментов "Заменить". Эта кнопка при первом нажатии должна сохранить нажатое состояние, а при втором нажатии – вернуться в исходное состояние. При этом кнопка "Заменить" дублирует и отображает состояние элемента меню "Заменить". Рассмотрим, как это достигается.

Если строка меню "Заменить" отмечена галочкой, то удалить признак отметки со строки меню "Заменить":

```
SetMenuItem( hEditMenu, CM_EDIT_REPLC, MFS_UNCHECKED, FALSE);
```

и перевести кнопку "Заменить" в отжатое состояние:

```
SendMessage( hToolbar,TB_SETSTATE,  (WPARAM)CM_EDIT_REPLC,
    (LPARAM)MAKELONG(TBSTATE_ENABLED,0));
```

иначе отметить галочкой строку меню "Заменить":

```
SetMenuItem( hEditMenu,CM_EDIT_REPLC,MFS_CHECKED,FALSE);
```

и перевести кнопку "Заменить" в нажатое состояние:

```
SendMessage( hToolbar, TB_SETSTATE, (WPARAM)CM_EDIT_REPLC,
    (LPARAM)MAKELONG (TBSTATE_CHECKED | TBSTATE_ENABLED,0));
```

Здесь состояние TBSTATE_CHECKED | TBSTATE_ENABLED выбрано таким, что кнопка при отмеченном состоянии остается и доступной для нажатия.

Других особенностей в работе этого приложения нет.

5.1.3. Вывод подсказок в панели инструментов

При остановке курсора мыши над любой кнопкой панели инструментов можно отображать текст подсказки, связанный с этой кнопкой. Для этого используют механизм уведомительных сообщений, при котором у функции окна можно запросить текст и отобразить его.

Панель инструментов функции родительского окна посылает уведомительное сообщение WM_NOTIFY о том, что с ней что-то сделали или требуется дополнительная информация. Для идентификации отправителя и истолкования сообщения используют параметр lParam. Он указывает на структуру типа NMHDR, которая содержит код и дополнительную информацию сообщения. После обработки сообщения WM_NOTIFY можно возвращать любое значение.

Структура NMHDR описана следующим образом:

```
typedef struct
{   HWND hwndFrom;
    UINT     idFrom;
    UINT     code;
}   NMHDR;
```

Здесь **hwndFrom** – дескриптор пославшего сообщение органа управления, **idFrom** – идентификатор этого органа управления, **code** – специфический для органа управления код сообщения.

Структура NMHDR обычно является частью структуры более высокого уровня. В случае панели инструментов она является частью структуры TOOLTIPTEXT:

```
typedef struct
{   NMHDR       hdr;
    LPTSTR      lpszText;
    char        szText[80];
    HINSTANCE   hinst;
    UINT        uFlags;
}   TOOLTIPTEXT;
```

Назначение полей этой структуры:

1. **hdr** – структура типа NMHDR.
2. **lpszText** – указатель на строку, которая содержит или получает текст для панели инструментов.
3. **szText** – буфер для получения текста от панели инструментов.
4. **hinst** – дескриптор приложения, которое содержит строковый ресурс. Если lpszText указывает на строку, hinst=NULL.
5. **uFlags** – флажок, который указывает, как интерпретировать значение поля idFrom структуры hdr. Если uFlags=TTF_IDISHWND, то idFrom – дескриптор, иначе idFrom – идентификатор кнопки панели инструментов.

Эти структуры обычно описывают и инициируют после получения сообщения WM_NOTIFY. Например, следующим образом:

1. Описывают указатель на структуру типа TOOLTIPTEXT:

```
LPTOOLTIPTEXT  TTStr;
```

2. Связывают указатель TTStr с параметром lParam:

```
TTStr=(LPTOOLTIPTEXT)lParam;
```

3. Далее анализируют код сообщения и принимают соответствующее решение.

Например, в случае организации подсказок, анализ сводится к проверке:

```
if (TTStr->hdr.code!=TTN_NEEDTEXT) return 0;
```

То есть обработку завершают, если код не равен TTN_NEEDTEXT. Сообщение TTN_NEEDTEXT говорит, что для панели инструментов требуется текст.

4. Представим, что принято уведомительное сообщение от кнопки панели о том, что требуется некий текст. Тогда определяют идентификатор этой кнопки и записывают соответствующий текст. Эту процедуру можно реализовать многими способами. Например:

```
switch (TTStr->hdr.idFrom)
{   case CM_FILE_NEW:
    {   TTStr->lpszText="Создать новый документ";
```

```
        return 0;
    }
    ...
    default: return 0;
}
```

В качестве подсказки можно посылать любые тексты (даже с управляющими символами типа \n).

Задача. В рассмотренном ранее приложении (листинг 4.6) к командам меню подключить кнопки панели инструментов и обеспечить выдачу подсказок в момент остановки курсора мыши над кнопкой панели.

Листинг 5.3. Выдача подсказок к кнопкам панели инструментов.

```
#include <windows.h>
#include <commctrl.h>
#pragma comment (lib,"comctl32.lib")
#define  ID_TOOLBAR      100
#define  CM_FILE_NEW     1000
#define  CM_FILE_OPEN    1001
#define  CM_FILE_SAVE    1002
#define  CM_FILE_QUIT    1003
#define  CM_EDIT_CUT     2000
#define  CM_EDIT_PASTE   2001
#define  CM_EDIT_COPY    2002
#define  CM_EDIT_DEL     2003
#define  CM_HELP_HELP    3000
#define  CM_HELP_ABOUT   3001

BOOL        RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT     CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HACCEL      CreateAccelTable( void );
HINSTANCE   hInstance;
char        szClass[ ]="NotifyClass";

int WINAPI  WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass, COLOR_WINDOW) )
        return FALSE;
    hwnd = CreateWindow(szClass," Выдача посказок ",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
```

```
        0,0,hInstance,NULL);
    if (!hwnd) return FALSE;
    HACCEL hAccel=CreateAccelTable();
    while(GetMessage(&msg, 0, 0, 0))
    {   if (!hAccel || !TranslateAccelerator(hwnd, hAccel, &msg))
        { TranslateMessage(&msg); DispatchMessage(&msg); }
    }
    DestroyAcceleratorTable(hAccel);
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
                            UINT brBackground)
{   WNDCLASS wc;  wc.style=wc.cbClsExtra=wc.cbWndExtra=0;
    wc.lpfnWndProc = Proc;     wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground+1);
    wc.lpszMenuName=NULL; wc.lpszClassName=szName;
    return (RegisterClass(&wc) != 0);
}

BOOL CreateMenuItem(HMENU hMenu, char *str, UINT uIns,
        UINT uCom, HMENU hSubMenu, BOOL flag, UINT fType)
{   MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE | MIIM_TYPE | MIIM_SUBMENU | MIIM_ID;
    mii.fType = fType;        mii.fState= MFS_ENABLED;
    mii.dwTypeData = str;  mii.cch=sizeof(str);
    mii.wID=uCom;             mii.hSubMenu=hSubMenu;
    return InsertMenuItem(hMenu, uIns, flag, &mii);
}

HACCEL   CreateAccelTable( void )
{   //Массив акселераторов
    ACCEL Accel[8];
    //Создать
    Accel[0].fVirt= FVIRTKEY | FCONTROL;
    Accel[0].key  = 0x4e;   Accel[0].cmd  = CM_FILE_NEW;
    //Открыть
    Accel[1].fVirt= FVIRTKEY | FCONTROL;
    Accel[1].key  = 0x4f;   Accel[1].cmd  = CM_FILE_OPEN;
    //Сохранить
```

```
    Accel[2].fVirt= FVIRTKEY | FCONTROL;
    Accel[2].key = 0x53;   Accel[2].cmd = CM_FILE_SAVE;
    //Выход
    Accel[3].fVirt= FVIRTKEY | FALT;
    Accel[3].key = 0x73;   Accel[3].cmd = CM_FILE_QUIT;
    //Вырезать
    Accel[4].fVirt= FVIRTKEY | FCONTROL;
    Accel[4].key = 0x58;   Accel[4].cmd = CM_EDIT_CUT;
    //Вклеить
    Accel[5].fVirt= FVIRTKEY | FCONTROL;
    Accel[5].key = 0x56;   Accel[5].cmd = CM_EDIT_PASTE;
    //Копировать
    Accel[6].fVirt= FVIRTKEY | FCONTROL;
    Accel[6].key = 0x43;   Accel[6].cmd = CM_EDIT_COPY;
    //Удалить
    Accel[7].fVirt= FVIRTKEY;
    Accel[7].key = 0x2e;   Accel[7].cmd = CM_EDIT_DEL;
    return CreateAcceleratorTable((LPACCEL)Accel,8);
}

HWND CreateToolBar(HWND hwnd, DWORD dwStyle, UINT uCom)
{   static TBBUTTON but[10];
    but[0].fsStyle=TBSTYLE_SEP;
    //Создать
    but[1].iBitmap=STD_FILENEW;
    but[1].idCommand=CM_FILE_NEW;
    but[1].fsState=TBSTATE_ENABLED;
    but[1].fsStyle=TBSTYLE_BUTTON;
    //Открыть
    but[2].iBitmap=STD_FILEOPEN;
    but[2].idCommand=CM_FILE_OPEN;
    but[2].fsState=TBSTATE_ENABLED;
    but[2].fsStyle=TBSTYLE_BUTTON;
    //Сохранить
    but[3].iBitmap=STD_FILESAVE;
    but[3].idCommand=CM_FILE_SAVE;
    but[3].fsState=TBSTATE_ENABLED;
    but[3].fsStyle=TBSTYLE_BUTTON;
    //Вырезать
    but[4].iBitmap=STD_CUT;
    but[4].idCommand=CM_EDIT_CUT;
```

```
    but[4].fsState=TBSTATE_ENABLED;
    but[4].fsStyle=TBSTYLE_BUTTON;
    //Копировать
    but[5].iBitmap=STD_COPY;
    but[5].idCommand=CM_EDIT_COPY;
    but[5].fsState=TBSTATE_ENABLED;
    but[5].fsStyle=TBSTYLE_BUTTON;
    //Вставить
    but[6].iBitmap=STD_PASTE;
    but[6].idCommand=CM_EDIT_PASTE;
    but[6].fsState=TBSTATE_ENABLED;
    but[6].fsStyle=TBSTYLE_BUTTON;
    //Удалить
    but[7].iBitmap=STD_DELETE;
    but[7].idCommand=CM_EDIT_DEL;
    but[7].fsState=TBSTATE_ENABLED;
    but[7].fsStyle=TBSTYLE_BUTTON;
    //Запрос помощи
    but[8].iBitmap=STD_HELP;
    but[8].idCommand=CM_HELP_HELP;
    but[8].fsState=TBSTATE_ENABLED;
    but[8].fsStyle=TBSTYLE_BUTTON;
    return CreateToolbarEx(hwnd, dwStyle, uCom, 0,
        HINST_COMMCTRL, IDB_STD_LARGE_COLOR,
        but,9, 0,0,0,0, sizeof(TBBUTTON));
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam )
{   static HMENU  hMainMenu, hPopUpFile, hPopUpEdit,
                hPopUpHelp;
    static HWND   hToolbar;
    switch (msg)
    {   case WM_SIZE:
        {   MoveWindow(hToolbar,0, 0, 0, 0,TRUE);   return 0; }
        case WM_CREATE:
        {   hMainMenu=CreateMenu();
            hPopUpFile=CreatePopupMenu();
            int i=0; //Инициализация позиции в меню
            CreateMenuItem(hPopUpFile,"&Новый\t Ctrl+N",
                i++,CM_FILE_NEW, NULL, FALSE, MFT_STRING);
```

```
    CreateMenuItem(hPopUpFile,"&Открыть\t Ctrl+O",
        i++,CM_FILE_OPEN, NULL, FALSE, MFT_STRING);
    CreateMenuItem(hPopUpFile,"&Сохранить\t Ctrl+S",
        i++,CM_FILE_SAVE, NULL, FALSE, MFT_STRING);
    CreateMenuItem(hPopUpFile,NULL,i++,0,
        NULL, FALSE, MFT_SEPARATOR);
    CreateMenuItem(hPopUpFile,"Выход\t Alt+F4",
        i++,CM_FILE_QUIT, NULL, FALSE, MFT_STRING);
    hPopUpEdit=CreatePopupMenu();
    i=0; //Инициализация позиции в меню
    CreateMenuItem(hPopUpEdit,"&Вырезать\t Ctrl+X",
        i++,CM_EDIT_CUT, NULL, FALSE, MFT_STRING);
    CreateMenuItem(hPopUpEdit,"B&ставить\t Ctrl+V",
        i++,CM_EDIT_PASTE, NULL, FALSE, MFT_STRING);
    CreateMenuItem(hPopUpEdit,"&Копировать\t Ctrl+C",
        i++, CM_EDIT_COPY, NULL, FALSE, MFT_STRING);
    CreateMenuItem(hPopUpEdit,"&Удалить\t Delete",
        i++, CM_EDIT_DEL, NULL, FALSE, MFT_STRING);
    hPopUpHelp=CreatePopupMenu();
    i=0; //Инициализация позиции в меню
    CreateMenuItem(hPopUpHelp,"&Помощь",
        i++,CM_HELP_HELP, NULL, FALSE, MFT_STRING);
    CreateMenuItem(hPopUpHelp,NULL,i++,0,
        NULL, FALSE, MFT_SEPARATOR);
    CreateMenuItem(hPopUpHelp,"&О программе",
        i++,CM_HELP_ABOUT, NULL, FALSE, MFT_STRING);
    //Подключаем временные меню к главному меню
    i=0; //Инициализация позиции в меню
    CreateMenuItem(hMainMenu,"&Файл", i++, 0,
        hPopUpFile,FALSE, MFT_STRING);
    CreateMenuItem(hMainMenu,"&Правка", i++, 0,
        hPopUpEdit,FALSE, MFT_STRING);
    CreateMenuItem(hMainMenu,"&Помощь", i++, 0,
        hPopUpHelp,FALSE, MFT_STRING);
    SetMenu(hwnd,hMainMenu); DrawMenuBar(hwnd);

    DWORD dwStyle=WS_CHILD | WS_VISIBLE |
        TBSTYLE_TOOLTIPS | WS_DLGFRAME;
    hToolbar=CreateToolBar(hwnd, dwStyle, ID_TOOLBAR);
    return 0;
}
```

```
case WM_COMMAND:
{   switch (LOWORD(wParam))
    {   case CM_FILE_NEW:
        case CM_FILE_OPEN:
        case CM_FILE_SAVE:
        case CM_EDIT_CUT:
        case CM_EDIT_PASTE:
        case CM_EDIT_COPY:
        case CM_EDIT_DEL:
        case CM_HELP_HELP:
        case CM_HELP_ABOUT: { return 0; }
        case CM_FILE_QUIT:
         { DestroyWindow(hwnd); return 0;}
    }
}
case WM_NOTIFY:
{   LPTOOLTIPTEXT  TTStr;
    TTStr=(LPTOOLTIPTEXT)lParam;
    if (TTStr->hdr.code!=TTN_NEEDTEXT) return 0;

    switch (TTStr->hdr.idFrom)
    {   case CM_FILE_NEW:
        {   TTStr->lpszText="Создать новый документ";
            return 0;
        }
        case CM_FILE_OPEN:
        {   TTStr->lpszText="Открыть существующий документ";
            return 0;
        }
        case CM_FILE_SAVE:
        {   TTStr->lpszText="Сохранить текущий документ";
            return 0;
        }
        case CM_EDIT_CUT:
        {   TTStr->lpszText="Вырезать и запомнить выделенный объект";
            return 0;
        }
        case CM_EDIT_PASTE:
        {   TTStr->lpszText="Вставить копированный объект";
            return 0;
        }
```

```
            case CM_EDIT_COPY:
            {   TTStr->lpszText="Копировать выделенный объект";
                return 0;
            }
            case CM_EDIT_DEL:
            {   TTStr->lpszText="Удаление выделенного объекта";
                return 0;
            }
            case CM_HELP_HELP:
            {   TTStr->lpszText=
                    "Добро пожаловать в справочную систему:\n\n"
                    "1. Найдите ответы на возникающие вопросы\n\n"
                    "2. Просмотрите электронную версию справочника";
                return 0;
            }
            case CM_HELP_ABOUT:
            {   TTStr->lpszText="Сообщение о программе";
                return 0;
            }
            case CM_FILE_QUIT:
            {   TTStr->lpszText="Завершить работу приложения";
                return 0;
            }
            default: return 0;
        }
        return 0;
    }
    case WM_DESTROY: {    PostQuitMessage(0); return 0; }
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Рассмотрим, какие изменения потребовались для решения данной задачи.

1. Описан идентификатор панели инструментов:

```
#define ID_TOOLBAR      100
```

2. Описана функция создания панели инструментов:

```
HWND CreateToolBar(HWND hwnd, DWORD dwStyle, UINT uCom);
```

Она от предыдущих версий отличается другим составом кнопок и тем, что отсутствует управление состоянием кнопок.

3. Для того чтобы кнопки панели инструментов посылали сообщение WM_NOTIFY, изменен стиль dwStyle окна панели инструментов:

```
DWORD dwStyle=WS_CHILD | WS_VISIBLE |
              TBSTYLE_TOOLTIPS | WS_DLGFRAME;
```

Здесь применен стиль TBSTYLE_TOOLTIPS. Затем стиль dwStyle используют при вызове функции CreateToolBar для создания панели инструментов:

```
hToolbar=CreateToolBar(hwnd, dwStyle, ID_TOOLBAR);
```

4. Обработка уведомительного сообщения WM_NOTIFY.

   4.1. Запоминают значение параметра lParam. Для этого описывают переменную-указатель на структуру типа TOOLTIPTEXT:

```
LPTOOLTIPTEXT  TTStr;
```

Затем в эту переменную записывают значение параметра lParam:

```
TTStr=(LPTOOLTIPTEXT)lParam;
```

   4.2. Если уведомительное сообщение WM_NOTIFY не требует текста подсказки, завершают обработку:

```
if (TTStr->hdr.code!=TTN_NEEDTEXT) return 0;
```

   4.3. Оператор варианта по идентификатору кнопки, которая послала сообщение WM_NOTIFY, записывает тот или иной текст в структуру TTStr:

```
switch (TTStr->hdr.idFrom)
{   case CM_FILE_NEW:
    {   TTStr->lpszText=  "Создать новый документ";
        return 0;
    }
    ...
    case CM_FILE_QUIT:
    {   TTStr->lpszText="Завершить работу приложения";
        return 0;
    }
    default: return 0;
}
```

Как видно из листинга, здесь перечислены все команды меню. Хотя не для всех команд созданы кнопки на панели инструментов. В любом случае подсказка соответствует той кнопке, над которой остановился курсор мыши. Подсказки будут появляться в том случае, если кнопка отображена на панели. При этом операционную систему не интересует состояние кнопки: достаточно того, что кнопка посылает уведомительное сообщение WM_NOTIFY.

Другие изменения в тексте приложения не обязательны.

Данный способ обработки сообщения WM_NOTIFY отличается простотой, наглядностью и информативностью сообщений. Например, можно выдать большее сообщение:

```
TTStr->lpszText=  "Добро пожаловать в справочную систему:\n\n"
    "1. Найдите ответы на возникающие вопросы.\n\n"
    "2. Просмотрите электронную версию справочника.";
```

5.2. Строка состояния

Строку состояния используют для отображения текстов о текущем состоянии приложения. Функции для работы со строкой состояния, как и функции для работы с панелью инструментов, объявлены в файле commctrl.h и описаны в файле comctl32.dll.

5.2.1. Создание строки состояния

Для создания окна строки состояния вызывают специальную функцию CreateStatusWindow:

```
HWND CreateStatusWindow( LONG style, LPCTSTR lpszText,
                         HWND hwndParent, UINT wID );
```

Она в нижней части окна hwndParent создает окно строки состояния и возвращает его дескриптор.

Параметры:

1. **style** – стиль окна строки состояния. Обязательно содержит комбинацию стилей WS_CHILD | WS_VISIBLE.
2. **lpszText** – указатель на текст, изначально отображаемый в строке состояния, может быть равен NULL.
3. **hwndParent** – дескриптор родительского окна.
4. **wID** – идентификатор окна строки состояния. Процедура окна использует wID для идентификации сообщений от строки состояния.

Задача. Создать строку состояния из трех секций. При нажатии левой и правой клавиш мыши в секциях отображать различный текст.

Листинг 5.4. Приложение со строкой состояния.

```
#include <windows.h>
#include <commctrl.h>
#pragma comment (lib,"comctl32.lib")
#define ID_STATUS                200

BOOL        RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT     CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
```

```
HINSTANCE   hInstance;
char        szClass[ ] = "StatusClass";

int WINAPI  WinMain(HINSTANCE hInstance,
        HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hMainWnd;  ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass, COLOR_WINDOW) )
        return FALSE;
    hMainWnd = CreateWindow( szClass, "Строка состояния",
        WS_VISIBLE | WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hMainWnd) return FALSE;
    while (GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
                        UINT brBackground)
{   WNDCLASS wc; wc.style=wc.cbClsExtra=wc.cbWndExtra = 0;
    wc.lpfnWndProc = Proc;   wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = NULL;     wc.lpszClassName = szName;
    return (RegisterClass(&wc) != 0);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam )
{   static  HWND hStatus;
    static  int      pParts[3];
    static  short    cx;
    switch (msg)
    {   case WM_SIZE:
        {   MoveWindow(hStatus, 0, 0, 0, 0, TRUE);
            cx=LOWORD(lParam);
            pParts[0]=cx-200; pParts[1]=cx-100;    pParts[2]=cx;
            SendMessage(hStatus, SB_SETPARTS, 3, (LPARAM)pParts);
            return 0;
        }
```

```
        case WM_CREATE:
        {   hStatus=CreateStatusWindow( WS_CHILD | WS_VISIBLE, "Готово",
                hwnd, ID_STATUS);
            return 0;
        }
        case WM_LBUTTONDOWN:
        {   SendMessage(hStatus,SB_SETTEXT,0,
                (LONG)"Первая секция строки состояния");
            SendMessage(hStatus,SB_SETTEXT,1,
                (LONG)"Сообщение 2");
            SendMessage(hStatus,SB_SETTEXT,2,
                (LONG)"Сообщение 3");
            return 0;
        }
        case WM_RBUTTONDOWN:
        {   SendMessage(hStatus,SB_SETTEXT,0,
                (LONG)"Сообщение 1");
            SendMessage(hStatus,SB_SETTEXT,1, (LONG)"");
            return 0;
        }
        case WM_DESTROY: { PostQuitMessage(0); return 0; }
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Проанализируем это приложение.

1. Функция окна приложения содержит описания дескриптора для окна строки статуса hStatus и массива pParts:

```
static   HWND hStatus;
static   int     pParts[3];
```

Массив pParts нужен для разбиения окна строки состояния на 3 секции (имя массива и количество секций могут быть другими).

2. На этапе создания окна приложения создают и окно строки состояния:

```
hStatus=CreateStatusWindow(WS_CHILD | WS_VISIBLE,
    "Готово", hwnd, ID_STATUS);
```

Здесь второй аргумент "Готово" будет по умолчанию отображен в первой секции. Этот аргумент может быть указан как NULL. Тогда строка состояния изначально не содержит никакого текста.

3. При любых изменениях размеров окна перемещают окно строки состояния:

```
MoveWindow( hStatus, 0, 0, 0, 0, TRUE );
```

Как и в случае панели инструментов, все координаты перемещения могут быть равны нулю. Предопределенная функция окна строки состояния всегда размещает строку состояния в нижней части рабочей области родительского окна. Ширина строки состояния при этом равна ширине рабочей области.

Далее определены координаты правых границ секций:

```
cx=LOWORD(lParam);
pParts[0]=cx-200; pParts[1]=cx-100;    pParts[2]=cx;
```

Например, правая граница первой секции равна cx-200.

Окну строки состояния сообщают о количестве и правых границах секций:

```
SendMessage( hStatus, SB_SETPARTS, 3, (LPARAM)pParts );
```

4. После нажатия левой клавиши мыши над рабочей областью окна приложения во все секции строки состояния записывают тексты сообщений: "Первая секция строки состояния", "Сообщение 2" и "Сообщение 3":

```
SendMessage( hStatus, SB_SETTEXT, 0,
            (LONG)"Первая секция строки состояния" );
SendMessage( hStatus, SB_SETTEXT, 1, (LONG)"Сообщение 2" );
SendMessage( hStatus, SB_SETTEXT, 2, (LONG)"Сообщение 3" );
```

Здесь третий аргумент вызова функции SendMessage равен номеру секции. Причем номер первой секции равен 0.

Аналогично обрабатывают нажатие правой клавиши мыши.

5.2.2. Сообщения о меню в строке состояния

Строку состояния часто используют для выдачи уведомительных сообщений о строках и разделах меню в процессе перемещения курсора по меню.

В процессе перемещения по строкам меню функция окна, к которому подключено меню, получает сообщение WM_MENUSELECT. При этом младшее слово параметра wParam равно идентификатору команды или позиции (если при выборе этой строки отображается временное меню) строки. Параметр lParam содержит дескриптор меню, по которому перемещается курсор.

Воспользуемся этим для выдачи сообщений в процессе перемещения по строкам меню. В качестве примера возьмем задачу из параграфа 4.2.3, решение которой представлено в листинге 4.3, добавим к этому листингу обработку сообщения WM_MENUSELECT и подключим к окну строку состояния из двух частей.

Задача. Главное меню изначально содержит раздел "Файлы". При выборе строки "Файлы" отображается временное меню со строками "Открыть", "Сохранить", разделительной линией и строкой "Выход". Причем строка "Сохранить" заблокирована.

При выборе строки "Открыть" заблокировать ее. Затем разблокировать строку "Сохранить" и вставить разделы "Правка" и "Вид" в главное меню.

При выборе строки "Правка" отображается временное меню со строками "Найти" и "Заменить". Причем эти строки при их выборе должны быть отмечены как независимые флажки.

При выборе строки "Вид" отображается временное меню со строками "Обычный" и "Структура". Причем эти строки при их выборе должны быть отмечены как зависимые переключатели.

При выборе строки "Сохранить" заблокировать ее. Затем разблокировать строку "Открыть" и из главного меню удалить разделы "Правка" и "Вид".

В процессе перемещения по строкам меню сообщать о разделе главного меню и о той строке, над которой находится курсор.

Листинг 5.5. Сообщения о меню в строке состояния.

```
#include <windows.h>
#include <commctrl.h>
#pragma comment (lib,"comctl32.lib")
#define ID_STATUS        200
#define CM_FILE_OPEN     1001
#define CM_FILE_SAVE     1002
#define CM_FILE_QUIT     1003
#define CM_EDIT_FIND     2001
#define CM_EDIT_REPLC    2002
#define CM_VIEW_NORM     3001
#define CM_VIEW_STRC     3002

BOOL        RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT     CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char        szClass[ ]="SetMenuInfo";

int WINAPI WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd;  ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass,COLOR_WINDOW) )
        return FALSE;
```

```
    hwnd = CreateWindow(szClass, "Сообщения о строках меню",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0))
    {   TranslateMessage(&msg); DispatchMessage(&msg); }
    return msg.wParam;
}
BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc; wc.style=wc.cbClsExtra=wc.cbWndExtra = 0;
    wc.lpfnWndProc = Proc;    wc.hInstance=hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = NULL;    wc.lpszClassName = szName;
    return (RegisterClass(&wc) != 0);
}

BOOL SetMenuItem(HMENU hMenu, UINT uIns, UINT fState, BOOL flag)
{   MENUITEMINFO mii;  mii.cbSize  = sizeof(MENUITEMINFO);
    mii.fMask                      = MIIM_STATE | MIIM_ID;
    mii.fState                     = fState;    mii.wID = uIns;
    return SetMenuItemInfo( hMenu, uIns, flag, &mii );
}

UINT GetMenuItem(HMENU hMenu, UINT uIns, BOOL flag)
{   MENUITEMINFO mii;  mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE;
    GetMenuItemInfo(hMenu, uIns, flag, &mii);
    return mii.fState;
}

BOOL CreateMenuItem(HMENU hMenu, char *str, UINT uIns,
    UINT uCom, HMENU hSubMenu, BOOL flag, UINT fType)
{   MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE | MIIM_TYPE | MIIM_SUBMENU | MIIM_ID;
    mii.fType = fType;          mii.fState= MFS_ENABLED;
    mii.dwTypeData = str;       mii.cch=sizeof(str);
    mii.wID=uCom;               mii.hSubMenu=hSubMenu;
```

```
    return InsertMenuItem(hMenu, uIns, flag, &mii);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam )
{   static  HMENU   hMainMenu, hFileMenu, hEditMenu,
                hViewMenu;
    static  HWND hStatus;
    static  int     pParts[2];
    static  short   cx;
    switch (msg)
    {   case WM_SIZE:
        {   MoveWindow(hStatus, 0, 0, 0, 0, TRUE);
            cx=LOWORD(lParam);
            pParts[0]=cx/2; pParts[1]=cx;
            SendMessage(hStatus, SB_SETPARTS,2,
                (LPARAM)pParts);
            return 0;
        }
        case WM_CREATE:
        {   hMainMenu=CreateMenu();
            //Создаем временное меню для раздела "Файлы"
            hFileMenu=CreatePopupMenu();
            int i=0; //Инициализация позиции в меню hFileMenu
            CreateMenuItem(hFileMenu,"&Открыть",i++,
                CM_FILE_OPEN, NULL, FALSE, MFT_STRING);
            CreateMenuItem(hFileMenu,"&Сохранить",i++,
                CM_FILE_SAVE, NULL, FALSE, MFT_STRING);
            CreateMenuItem(hFileMenu,NULL,i++,0,
                NULL, FALSE, MFT_SEPARATOR);
            CreateMenuItem(hFileMenu,"&Выход",i++,
                CM_FILE_QUIT, NULL, FALSE, MFT_STRING);
            //Создаем временное меню для раздела "Правка"
            hEditMenu=CreatePopupMenu();
            i=0; //Инициализация позиции в меню hEditMenu
            CreateMenuItem(hEditMenu,"&Найти",i++,
                CM_EDIT_FIND, NULL, FALSE, MFT_STRING);
            CreateMenuItem(hEditMenu,"&Заменить",i++,
                CM_EDIT_REPLC,
                NULL, FALSE, MFT_STRING);
            //Создаем временное меню для раздела "Вид"
```

```
    hViewMenu=CreatePopupMenu();
    i=0; //Инициализация позиции в меню hViewMenu
    CreateMenuItem(hViewMenu,"&Обычный",i++,
        CM_VIEW_NORM, NULL, FALSE, MFT_RADIOCHECK);
    CreateMenuItem(hViewMenu,"&Структура",i++,
        CM_VIEW_STRC, NULL, FALSE, MFT_RADIOCHECK);
    //Подключаем временные меню к главному меню
    i=0; //Инициализация позиции в меню hMainMenu
    CreateMenuItem(hMainMenu,"&Файл",i++, 0,
        hFileMenu,FALSE, MFT_STRING);
    SetMenu(hwnd,hMainMenu);
    SetMenuItem(hFileMenu,
        CM_FILE_SAVE, MFS_GRAYED, FALSE);
    DrawMenuBar(hwnd);

    hStatus=CreateStatusWindow( WS_CHILD | WS_VISIBLE,
        "Готово", hwnd, ID_STATUS);
    return 0;
}
case WM_COMMAND:
{   switch (LOWORD(wParam))
    {   case CM_FILE_OPEN:
        {   CreateMenuItem(hMainMenu,
                "&Правка",1,0, hEditMenu, FALSE, MFT_STRING);
            CreateMenuItem(hMainMenu,
                "&Вид",2,0, hViewMenu, FALSE, MFT_STRING);
            SetMenuItem(hFileMenu,
                CM_FILE_OPEN, MFS_GRAYED, FALSE);
            SetMenuItem(hFileMenu,
                CM_FILE_SAVE, MFS_ENABLED, FALSE);
            DrawMenuBar(hwnd);
            return 0;
        }
        case CM_FILE_SAVE:
        {   RemoveMenu(hMainMenu,1, MF_BYPOSITION);
            RemoveMenu(hMainMenu,1, MF_BYPOSITION);
            SetMenuItem(hFileMenu,
                CM_FILE_SAVE, MFS_GRAYED, FALSE);
            SetMenuItem(hFileMenu,
                CM_FILE_OPEN, MFS_ENABLED, FALSE);
            DrawMenuBar(hwnd);
```

```
            return 0;
        }
        case CM_EDIT_FIND:
        {   if (GetMenuItem(hEditMenu,
                CM_EDIT_FIND, FALSE)==MFS_CHECKED)
                SetMenuItem(hEditMenu,
                    CM_EDIT_FIND, MFS_UNCHECKED, FALSE);
            else
                SetMenuItem(hEditMenu,
                    CM_EDIT_FIND, MFS_CHECKED, FALSE);
            return 0;
        }
        case CM_EDIT_REPLC:
        {   if (GetMenuItem(hEditMenu,
                CM_EDIT_REPLC, FALSE)==MFS_CHECKED)
                SetMenuItem(hEditMenu,
                    CM_EDIT_REPLC, MFS_UNCHECKED, FALSE);
            else
                SetMenuItem(hEditMenu,
                    CM_EDIT_REPLC, MFS_CHECKED, FALSE);
            return 0;
        }
        case CM_VIEW_NORM:
        {   SetMenuItem(hViewMenu,
                CM_VIEW_NORM, MFS_CHECKED, FALSE);
            SetMenuItem(hViewMenu,
                CM_VIEW_STRC, MFS_UNCHECKED, FALSE);
            return 0;
        }
        case CM_VIEW_STRC:
        {   SetMenuItem(hViewMenu,
                CM_VIEW_NORM, MFS_UNCHECKED, FALSE);
            SetMenuItem(hViewMenu,
                CM_VIEW_STRC, MFS_CHECKED, FALSE);
            return 0;
        }
        case CM_FILE_QUIT:
        {   DestroyWindow(hwnd); return 0;   }
    }
    return 0;
}
```

```
case WM_MENUSELECT:
{   if ( (HMENU)lParam==hMainMenu )
    {   switch ( LOWORD(wParam) )
        {   case 0:
                SendMessage(hStatus, SB_SETTEXT,0,
                    (LONG)"Открыть/закрыть документ или выход");
                SendMessage(hStatus, SB_SETTEXT,1, NULL);
                return 0;
            case 1:
                SendMessage(hStatus, SB_SETTEXT,0,
                    (LONG)"Найти и/или заменить фрагмент");
                SendMessage(hStatus, SB_SETTEXT,1, NULL);
                return 0;
            case 2:
                SendMessage(hStatus, SB_SETTEXT,0,
                    (LONG)"Смена способа отображения документа");
                SendMessage(hStatus, SB_SETTEXT,1, NULL);
                return 0;
        }
    }
    switch (LOWORD(wParam))
    {   case CM_FILE_OPEN:
        {   SendMessage(hStatus,SB_SETTEXT,1,
                (LONG)"Открыть новый документ");
            return 0;
        }
        case CM_FILE_SAVE:
        {   SendMessage(hStatus,SB_SETTEXT,1,
                (LONG)"Закрыть текущий документ");
            return 0;
        }
        case CM_EDIT_FIND:
        {   SendMessage(hStatus,SB_SETTEXT,1,
                (LONG)"Найти фрагмент по выделенному образцу");
            return 0;
        }
        case CM_EDIT_REPLC:
        {   SendMessage(hStatus,SB_SETTEXT,1,
    (LONG)"Заменить найденный фрагмент на указанный образец");
            return 0;
        }
```

```
            case CM_VIEW_NORM:
            {   SendMessage(hStatus,SB_SETTEXT,1,
                    (LONG)"Показать текущий документ в обычном виде");
                return 0;
            }
            case CM_VIEW_STRC:
            {   SendMessage(hStatus,SB_SETTEXT,1,
                    (LONG)"Показать структуру текущего документа");
                return 0;
            }
            case CM_FILE_QUIT:
            {   SendMessage(hStatus,SB_SETTEXT,1,
                    (LONG)"Завершить работу приложения");
                return 0;
            }
        }
        return 0;
    }
    case WM_DESTROY:
    {   DestroyMenu(hEditMenu); DestroyMenu(hViewMenu);
        PostQuitMessage(0);
        return 0;
    }
}
return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Рассмотрим основные дополнения к листингу 4.3.

1. Дополнения, обусловленные созданием строки состояния, были изучены в приложении в листинге 5.4. Только теперь строка состояния включает две части.
2. Обработка сообщения WM_MENUSELECT состоит из трех частей.
   2.1. Отдельно обрабатывают перемещение по названиям разделов главного меню. Если параметр lParam равен дескриптору главного меню, то определяют значение позиции курсора. Это значение содержится в LOWORD(wParam), если строка связана с разделом меню. Мы знаем, какой раздел соответствует каждой позиции. Например, в первой позиции (ей соответствует LOWORD (wParam)=0) находится раздел "Файл". В этом случае в первой части строки состояния отображают текст "Открыть/закрыть документ или выход":

```
SendMessage( hStatus, SB_SETTEXT, 0,
    (LONG)"Открыть/закрыть документ или выход");
```

и стирают текст во второй части строки состояния:

```
SendMessage( hStatus, SB_SETTEXT, 1, NULL );
```

Точно так же обрабатывают другие позиции курсора в главном меню. Если строки временных меню указывают на разделы меню, то для них также нужно предусмотреть обработку перемещения по позициям. Обратите внимание, что условный оператор

```
if ( (HMENU)lParam==hMainMenu )
{   switch ( LOWORD(wParam) )
    {   case 0:
        ...
    }
}
```

завершает обработку сообщения оператором возврата, только если курсор установлен на строке раздела меню. То есть не все строки меню с дескриптором lParam должны указывать на разделы меню.

2.2. Здесь мы оказываемся в том случае, если младшее слово параметра wParam может быть равно идентификатору команды строки меню. Тогда достаточно определить этот идентификатор и послать соответствующий текст сообщения во вторую часть строки состояния. Например, если курсор находится над строкой "Открыть" раздела "Файл", то во второй части строки состояния окажется текст "Открыть новый документ":

```
SendMessage(hStatus,SB_SETTEXT,1,
    (LONG)"Открыть новый документ");
```

Точно так же поступают для других идентификаторов команд.

2.3. Обработку завершают возвратом значения 0.

После запуска приложения на экране появится точно такое же окно и с таким же меню, что и после запуска приложения листинга 4.3. Только в этом случае добавлена строка состояния из двух частей. В первой части отображен текст "Готово".

Если нажать строку "Файл", в первой части строки состояния появляется текст "Открыть/закрыть документ или выход", а вторая часть остается пустой. Если курсор перемещается на строку "Открыть", то текст в первой части не изменится, а во второй части появится текст "Открыть новый документ". Текст во второй части строки состояния будет менять-

ся при перемещении по строкам этого временного меню, а текст в первой части будет постоянным и указывать на текущий раздел меню.

Точно так же будет происходить информирование о строках меню после подключения других разделов к главному меню. Но вторая часть строки состояния будет пустой, если курсор переходит на название раздела меню. После выбора строки меню содержание частей строки состояния остается таким, каким оно было в момент выбора. То есть строка состояния напоминает о последней выбранной команде меню.

Контрольные вопросы

1. Перечислите основные шаги создания панели инструментов.
2. Что отличает состояние кнопок панели инструментов от состояния строк меню?
3. Какой константой определяют стиль окна панели инструментов, если все кнопки панели должны посылать уведомительные сообщения?
4. Перечислите основные шаги создания строки состояния.

Упражнения

Упражнения этой главы являются продолжением упражнений 4-й главы.

1. В строке состояния сообщать о действиях со строками меню и текущие координаты курсора мыши.
2. О действиях со строками меню сообщать в строке состояния.
3. Команды отображаемых строк меню продублировать кнопками панели инструментов и отмечать состояние кнопок.
4. Команды отображаемых строк меню продублировать кнопками панели инструментов и обеспечить подсказки к кнопкам.
5. Команды отображаемых строк меню продублировать кнопками панели инструментов и выдавать сообщения о строках меню при перемещении по ним.
6. О действиях с рисунками сообщать в строке состояния.
7. Команды меню окна приложения продублировать кнопками панели инструментов. В дополнительном окне выдавать сообщения о строках меню.
8. Команды меню окна приложения продублировать кнопками панели инструментов. В дополнительном окне выдавать сообщения о строках меню.
9. О действиях с фигурами и состоянии соответствующих строк меню сообщать в строке состояния.

10. Команды отображаемых строк меню продублировать кнопками панели инструментов и выдавать сообщения о строках меню при перемещении по ним.

11. О действиях с курсором мыши сообщать в строке состояния.

12. В строке состояния отобразить путь к текущей строке меню.

13. Строки состояния всех окон отображают информацию об активности окон и перемещении по строкам меню.

14. В строке состояния сообщать о строках меню, включая состояние строк.

15. Обо всем, что происходит в трех окнах, сообщать в строке состояния окна приложения.

16. Команды отображаемых строк меню продублировать кнопками панели инструментов и выдавать сообщения о строках меню при перемещении по ним.

17. Команды меню окон продублировать кнопками панелей инструментов.

18. О действиях с фигурами и состоянии соответствующих строк меню сообщать в строке состояния.

19. Стандартные команды меню продублировать кнопками панели инструментов. В дополнительном окне выдавать сообщения о строках меню.

20. Стандартные команды меню продублировать кнопками панели инструментов.

21. Команды меню продублировать кнопками панели инструментов.

22. Команды меню продублировать кнопками панели инструментов.

23. Строка состояния содержит текущую информацию о пользователе и данных, даже если выбор был сделан с помощью акселератора.

24. Строка состояния отражает текущую информацию о фигуре и цвете.

25. Строка состояния отражает путь перемещения курсора по строкам меню.

Глава 6
Диалоговые панели

6.1. Характеристики диалоговых панелей

Диалоговые панели обеспечивают оперативный обмен информацией между пользователем и приложением. Они представляют собой перекрывающиеся или временные окна и создаются на базе предопределенного в Windows **класса диалоговых панелей**. Функция окна такого класса обеспечивает специфическое поведение органов управления панели. Например, она обеспечивает передачу фокуса ввода от одного органа управления другому или между группами при нажатии клавиши Tab.

Внешний вид диалоговой панели задают шаблоном панели. Шаблон можно описать тремя способами:

1. **Включая текст описания шаблона** в файл ресурсов приложения. Для изменения внешнего вида панели достаточно редактировать файл ресурсов, не изменяя текста приложения.

2. **Используя редактор ресурсов** интегрированной среды разработки приложения. Эти редакторы позволяют с помощью мыши и клавиатуры нарисовать панель и сохранить текст описания шаблона в файле ресурсов.

3. **Создавая шаблон в памяти** в процессе работы приложения. Приложение в зависимости от содержания требуемых и предоставляемых данных и способа обмена изменяет вид панели. Функция окна диалоговой панели при этом содержит обработку необходимого и достаточного набора сообщений. Команды, которые не задействованы в текущей конфигурации панели, обычно блокируют. С ростом потребности оперативного обмена данными с различными источниками информации этот способ находит все более широкое применение.

В операционной системе есть и **стандартные диалоговые панели**. Приложения их вызывают для работы с файлами, выбора цветов и шрифта, работы с принтерами и текстовыми строками и т. д.

Здесь будут рассмотрены третий способ создания и использования диалоговых панелей и примеры работы со стандартными диалоговыми панелями для работы с файлами, выбора цветов и шрифта.

6.1.1. Единицы диалоговой панели

Диалоговые панели используют специальную систему измерения размеров панели и органов управления на ее поверхности. В этой системе единицы ширины и высоты определяют шириной и высотой используемого в панели шрифта.

Одна **диалоговая единица** ширины составляет четверть средней ширины символов, а высоты – восьмую часть высоты символов системного шрифта.

Размер диалоговой единицы получают при помощи функции GetDialogBaseUnits:

```
LONG GetDialogBaseUnits(VOID);
```

Она возвращает 32-разрядное значение, младшее слово которого равно единице ширины, а старшее слово – единице высоты.

Удобно пользоваться средними значениями ширины и высоты шрифта в диалоговых единицах. Эти значения можно вычислить следующим образом:

```
//Определяем среднюю ширину и высоту шрифта в пикселях
int cxChar, cyChar;
{   TEXTMETRIC  tm; HDC hdc=GetDC(hwnd);
    GetTextMetrics(hdc,&tm);  ReleaseDC(hwnd,hdc);
    cxChar=tm.tmAveCharWidth+1;
    cyChar=tm.tmHeight+tm.tmExternalLeading;
}
//Определяем диалоговые единицы ширины и высоты
DWORD    dlgunit =GetDialogBaseUnits();
int      dlgwunit=LOWORD(dlgunit), dlghunit=HIWORD(dlgunit);
//Пересчитываем габариты символов шрифта
cxChar=cxChar*4/dlgwunit;
cyChar=cyChar*8/dlghunit;
```

Теперь значения переменных cxChar и cyChar представляют соответственно среднюю ширину и высоту символов шрифта в диалоговых единицах. Обычно содержание элементов панелей задают с помощью текстовых строк. Зная расположение этих элементов на панели, можно корректно определить габариты диалоговой панели и вычислить координаты размещения элементов.

6.1.2. Стили диалоговой панели

Диалоговые панели бывают следующих трех типов:

1. **Модальные**. Приложение приостанавливает свою работу на время активизации собственной модальной диалоговой панели. Функции всех окон (включая их дочерние окна) этого приложения блокируют. Поэтому панели нельзя создавать на базе дочерних окон.

2. **Системные модальные**. После запуска системной модальной диалоговой панели приостанавливают работу все приложения. Пока эта па-

нель находится на экране, нельзя переключиться на работу с другими приложениями.

3. **Немодальные**. Немодальная диалоговая панель не блокирует ничью работу. Обычно ее создают один раз. Затем при необходимости ее скрывают или отображают с помощью функции ShowWindow. Немодальную панель разрушают функцией DestroyWindow.

Диалоговые панели 1-го и 2-го типа для завершения собственного цикла обработки сообщений и разрушения панели в теле функции своего окна должны содержать вызов функции EndDialog.

Значение стиля панели задают комбинацией констант с префиксом WS_ и с префиксом DS_ из следующей таблицы.

Таблица стилей диалоговых панелей

Стиль	*Пояснение к стилю панели*
DS_3DLOOK	Этот стиль используют по умолчанию в приложениях Win32
DS_ABSALIGN	Координаты отсчитывать от левого верхнего угла экрана. Иначе отсчет ведется от левого верхнего угла рабочей области
DS_CENTER	Центрировать панель на экране
DS_CENTERMOUSE	Центрировать курсор мыши в панели
DS_CONTEXTHELP	В заголовке окна панели отобразить кнопку с вопросительным знаком. При ее нажатии курсор мыши принимает вид указателя с вопросительным знаком. Если затем нажать левую клавишу мыши над элементом панели, то функция окна панели получит сообщение WM_HELP. При его обработке нужно вызвать функцию WinHelp, указав третьим аргументом HELP_WM_HELP. Тогда отображается подсказка над этим органом управления
DS_CONTROL	Панель работает как дочернее окно другой панели, подобно странице блокнота панелей. Этот стиль позволяет переходить от одной панели к другой с помощью клавиатуры
DS_FIXEDSYS	Вместо шрифта SYSTEM_FONT загрузить шрифт SYSTEM_FIXED_FONT
DS_MODALFRAME	Модальная панель. Обычно комбинируют с константами WS_CAPTION и WS_SYSMENU

DS_NOFAILCREATE	Панель создавать, даже если возникли ошибки создания его органов управления
DS_NOIDLEMSG	Функции окна – владельца панели не посылать сообщение WM_ENTERIDLE. Иначе это сообщение будет послано в момент отображения панели
DS_RECURSE	Стиль панели подобен стилю органов управления
DS_SETFONT	Шаблон панели должен содержать размер и имя шрифта. Иначе будет использован системный шрифт
DS_ SETFOREGROUND	Панель всегда на переднем плане
DS_SYSMODAL	Системная модальная панель

В стиле панели чаще других используют константы WS_POPUP, WS_BORDER, WS_SYSMENU, WS_CAPTION и DS_MODALFRAME.

Если диалоговая панель имеет рамку, но не имеет заголовка, используют константу WS_DLGFRAME.

Константу DS_SETFONT задают, только если есть **гарантия наличия и поддержки требуемого шрифта**.

6.1.3. Функция окна диалоговой панели

Функция окна диалоговой панели описывается разработчиком и должна быть объявлена следующим образом:

```
BOOL CALLBACK DlgProc(HWND hdlg, UINT msg,
                      WPARAM wParam, LPARAM lParam);
```

На имя этой функции распространяют обычные требования на имена функций. Список формальных параметров подобен списку параметров функции окна. Только через первый параметр передают дескриптор диалогового окна hdlg, а не обычного окна hwnd.

Функция окна диалоговой панели не содержит вызов функции DefWindowProc для тех сообщений, которые она не обрабатывает. Если эта функция обрабатывает сообщение, то должна вернуть значение TRUE, а если нет – FALSE. Исключением является только возврат после обработки сообщения WM_INITDIALOG (см. ниже).

Функция окна диалоговой панели **не обрабатывает сообщения WM_CREATE, WM_PAINT и WM_DESTROY**.

Функция окна диалога получает **сообщение WM_INITDIALOG** непосредственно перед отображением панели. При обработке этого сооб-

щения инициализируют переменные для работы с данными и органы управления панели.

Параметр wParam при этом равен дескриптору органа управления, который получит фокус ввода сразу после отображения панели. Это первый незаблокированный орган управления, описанный в шаблоне панели со стилем WM_TABSTOP. Если при обработке этого сообщения фокус ввода не передают другому органу, **возвращают значение TRUE**. Если фокус ввода передают другому органу управления, **возвращают значение FALSE**.

Параметр lParam содержит значение, передаваемое приложением при создании диалоговой панели. Обычно этот параметр указывает на дополнительные данные для диалоговой панели. Здесь не рассматривается этот метод передачи дополнительных данных.

В функцию окна диалога **сообщение WM_COMMAND** поступает от органов управления панели. При этом **младшее слово параметра wParam** содержит идентификатор пославшего сообщение органа управления.

Нужно осторожно обрабатывать **сообщение WM_COMMAND с идентификатором IDOK или IDCANCEL**.

Функция окна диалога получает сообщение WM_COMMAND с идентификатором IDOK, если клавиша Enter нажата в момент, когда ни одна из кнопок панели не имеет фокуса ввода. Это справедливо для двух различных ситуаций:

1. Ни одна из кнопок не имеет стиль WS_DEFPUSHBUTTON.
2. Одна из кнопок создана со стилем WS_DEFPUSHBUTTON и идентификатором IDOK.

Функция окна диалога получает сообщение WM_COMMAND с идентификатором IDCANCEL, если нажата клавиша Esc или нажата кнопка с идентификатором IDCANCEL.

6.2. Создание диалоговой панели

Диалоговые панели – это окна определенного операционной системой класса окон. Существуют специальные функции для создания диалоговых панелей на базе этого класса и для их создания не вызывают функцию CreateWindow.

Создание и работа с диалоговыми панелями предполагает следующие шаги:

1. Описание шаблона диалоговой панели.
2. Описание функции окна диалоговой панели.
3. Вызов функции создания диалоговой панели.

4. Обмен данными между функцией окна, вызвавшего диалоговую панель, и функцией окна диалоговой панели.

Наиболее трудоемкими являются описание шаблона и функции окна диалоговой панели. А наибольшую сложность представляет реализация обмена данными между функцией окна, создавшего диалоговую панель, и функцией окна диалоговой панели. Сложность обусловлена тем, что полученные в теле функции окна панели данные разрушаются при разрушении диалоговой панели. Для решения этой задачи описывают глобальные объекты, которые принимают значения данных панели и сохраняют их после ее разрушения.

Для создания панели на базе шаблона в памяти вызывают функцию DialogBoxIndirect или CreateDialogIndirect.

6.2.1. Создание модальной диалоговой панели

Функция DialogBoxIndirect **создает диалоговую панель 1-го или 2-го типа**:

```
int DialogBoxIndirect( HINSTANCE hInstance,
    LPDLGTEMPLATE lpTemplate,
    HWND hWndParent, DLGPROC lpDialogFunc);
```

Она не возвращает управление, пока функция окна панели не завершила работу вызовом функции EndDialog. Функция DialogBoxIndirect отображает панель (даже если не указан стиль WS_VISIBLE) и активизирует собственный цикл обработки сообщений.

Параметры:

1. **hInstance** – дескриптор текущей копии приложения.
2. **lpTemplate** – указатель на шаблон диалоговой панели.
3. **hWndParent** – дескриптор родительского окна панели.
4. **lpDialogFunc** – указатель на функцию окна диалоговой панели.

Если функция окна панели успешно завершает свою работу, то функция DialogBoxIndirect **возвращает** значение nResult, заданное в вызове функции EndDialog. Иначе возвращаемое значение равно -1.

Для **разрушения** созданной функцией DialogBoxIndirect панели вызывают функцию EndDialog:

```
BOOL EndDialog( HWND hdlg, int nResult );
```

Первым аргументом вызова функции EndDialog является дескриптор окна разрушаемой панели. Это значение первого параметра в заголовке функции окна панели.

Второй аргумент вызова функции EndDialog (nResult) определяет возвращаемое функцией DialogBoxIndirect значение.

Функция окна может вызвать эту функцию даже при обработке сообщения WM_INITDIALOG. В случае успешного разрушения панели функция EndDialog возвращает ненулевое значение. При ее вызове управление возвращают системе. Если цикл обработки сообщений панели закрыт, система разрушает панель и передает значение nResult как результат разрушения панели.

6.2.2. Создание немодальной диалоговой панели

Для **создания немодальной диалоговой панели** вызывают функцию CreateDialogIndirect:

```
HWND CreateDialogIndirect(HINSTANCE hInstance,
        LPCDLGTEMPLATE lpTemplate,
        HWND hWndParent, DLGPROC lpDialogFunc );
```

Список ее формальных параметров подобен списку параметров функции DialogBoxIndirect. Но она **возвращает дескриптор окна созданной панели**. Функция CreateDialogIndirect отображает панель, только если ее стиль в шаблоне содержит константу WS_VISIBLE. Другие замечания по поводу создания и работы с немодальными диалоговыми панелями были высказаны выше.

6.2.3. Шаблон диалоговой панели

Шаблон диалоговой панели создают с помощью структур двух типов: DLGTEMPLATE и DLGITEMTEMPLATE. В начало шаблона записывают структуру типа DLGTEMPLATE. Она задает габариты, стиль и количество органов управления панели. Затем в шаблон записывают значения нескольких структур типа DLGITEMTEMPLATE. Их количество равно количеству органов управления панели.

Структура DLGTEMPLATE описана следующим образом:

```
typedef struct
{   DWORD   style;
        DWORD   dwExtendedStyle;
        WORD cdit;
        short       x;
        short       y;
        short       cx;
        short       cy;
}   DLGTEMPLATE;
```

Назначение полей структуры DLGTEMPLATE:

1. **style** задает стиль диалоговой панели. Значение этого поля задают с помощью констант с префиксом WS_ (типа WS_CAPTION и WS_SYSMENU) и DS_ (типа DS_MODALFRAME и DS_CONTROL).
2. **dwExtendedStyle** позволяет задавать расширенные стили для окна. Это поле не используется для создания шаблона диалоговой панели. Его значение может быть использовано для создания окон других стилей.
3. **cdit** равно количеству органов управления панели.
4. **x** задает координату левого края панели.
5. **y** задает координату верхнего края панели.
6. **cx** задает ширину панели.
7. **cy** задает высоту панели.

Поля x, y, cx, cy задают в единицах диалоговых панелей.

В шаблоне панели после структуры DLGTEMPLATE записывают строки с именем меню, с именем класса и с текстом заголовка панели. Если в стиле панели указана константа DS_SETFONT, то за этими строками записывают 16-разрядное значение размера шрифта и строку с именем шрифта. Массивы строк меню, имени класса, заголовка и имени шрифта должны быть выровнены по границе WORD. Для выравнивания удобно пользоваться функцией следующего вида:

```
LPWORD lpwAlign(LPWORD lpIn)
{   ULONG ul = (ULONG)lpIn;
    ul +=3; ul >>=2;    ul <<=2; return (LPWORD)ul;
}
```

Эта функция получает указатель lpIn на массив элементов типа WORD и возвращает ближайший указатель, выровненный по границе WORD.

Строка с именем меню идентифицирует ресурс меню для панели. Операционная система истолковывает содержимое этой строки в зависимости от значения ее первого элемента. При этом различают 3 значения первого элемента:

1. Равен 0x0000. Шаблон не подключает ресурса меню к панели.

2. Равен 0xFFFF. Шаблон подключает ресурс меню к панели. Порядковый номер этого ресурса меню содержится во втором элементе строки. Других элементов в строке не должно быть.

3. Равен любому другому значению. Строку обрабатывают как строку Unicode, которая содержит имя ресурса меню.

Обычно исходную строку задают как Ansi-строку. Для ее преобразования в строку Unicode 16-битовых символов можно использовать функцию следующего вида:

```
int nCopyAnsiToWideChar (LPWORD lpWCStr, LPSTR lpAnsiIn)
{   int cchAnsi=lstrlen(lpAnsiIn);
        return    MultiByteToWideChar(GetACP(), MB_PRECOMPOSED,
            lpAnsiIn, cchAnsi, lpWCStr, cchAnsi) + 1;
}
```

Параметр lpAnsiIn этой функции принимает исходную Ansi-строку. Функция копирует ее в строку 16-битовых символов lpWCStr и возвращает количество символов в строке lpWCStr + 1.

Строка имени класса задает класс окна панели. Операционная система истолковывает ее содержимое в зависимости от значения первого элемента. При этом различают 3 значения первого элемента:

1. Равен 0x0000. Система использует предопределенный класс диалоговых окон.

2. Равен 0xFFFF. Система использует предопределенный класс окон. Порядковый номер этого класса содержится во втором элементе строки. Других элементов в строке не должно быть.

3. Равен любому другому значению. Строку обрабатывают как строку Unicode, которая содержит имя зарегистрированного класса.

Строка заголовка задается строкой Unicode. Если ее первый элемент равен 0x0000, то панель не имеет заголовка, а строка не содержит других элементов.

Размер шрифта определяет размер шрифта панели. **Строка имени шрифта** – это строка Unicode с именем используемого в панели шрифта.

Пример. Разработать функцию для записи значения структуры типа DLGTEMPLATE в шаблон панели.

Решение этой задачи может быть описано в виде:

```
void DlgTemplate(PWORD&  p, DWORD lStyle, int items,
                 int x, int y, int cx, int cy, LPSTR txt)
{   *p++ = LOWORD(lStyle);     //В первые два слова
    *p++ = HIWORD(lStyle);     //записываем стиль панели
    *p++ = 0;  //В следующие две строки можно
    *p++ = 0;  //записать расширенный стиль окна
    *p++ = items;                    //Количество органов управления панели
    *p++ = x;  //Координата левого края панели
    *p++ = y;  //Координата верхнего края панели
    *p++ = cx; //Ширина панели
    *p++ = cy; //Высота панели
    *p++ = 0;  //Меню не подключается
    *p++ = 0;  //Используем стандартный класс
    //Преобразуем Ansi-строку заголовка в строку Unicode
```

```
    int nchar=nCopyAnsiToWideChar(p,TEXT(txt));
    p += nchar; //Смещаем указатель на количество символов
    //Выравниваем шаблон по границе WORD
    p=lpwAlign((LPWORD)p);
}
```

Как пользоваться этой функцией?

Заполнение начала шаблона панели значением структуры типа DLGTEMPLATE содержит следующие обязательные шаги:

1. Описать два указателя на массив элементов типа WORD:

```
WORD *p, *pdlgtemplate;
```

2. Выделить блок памяти для шаблона и приравнять оба указателя с его адресом:

```
pdlgtemplate=p=(PWORD)LocalAlloc(LPTR,500);
```

3. Вызвать функцию DlgTemplate:

```
DlgTemplate(
        p,          //Указатель на начало выделенного блока памяти
        lStyle,     //Стиль диалоговой панели
        7,          //Количество органов управления
        20,         //Координата левого края диалоговой панели
        20,         //Координата верхнего края диалоговой панели
        100,        //Ширина
        80,         //Высота
        TEXT(str)); //Заголовок
```

Остальные аргументы вызова определяют обычным способом. Например, стиль панели можно было бы описать следующим образом:

```
DWORD lStyle = DS_MODALFRAME |
                    WS_POPUP | WS_CAPTION | WS_SYSMENU;
```

Так же просто можно было описать строку заголовка:

```
char str[ ] = "Пример диалоговой панели";
```

Использование такого подхода позволяет не менее эффективно, чем с помощью ресурсов, описывать шаблоны панели. В конечном итоге вышеприведенный пример записывается в виде:

```
WORD *p, *pdlgtemplate;
pdlgtemplate=p=(PWORD)LocalAlloc(LPTR,500);
DlgTemplate(p, lStyle, 7, 20, 20, 100, 80, TEXT(str));
```

Кроме данного удобства, реализована возможность динамического создания диалоговых панелей.

После записи значения структуры типа DLGTEMPLATE в шаблон панели записывают указанное количество значений структур типа DLGITEMTEMPLATE. Каждая такая структура задает габариты и стиль одного органа управления панели. И каждая такая запись значения в шаблоне должна быть выровнена по границе DWORD.

Структура DLGITEMTEMPLATE описана следующим образом:

```
typedef struct
{   DWORD   style;
    DWORD   dwExtendedStyle;
    short   x;
    short   y;
    short   cx;
    short   cy;
    WORD    id;
}   DLGITEMTEMPLATE;
```

Назначение полей этой структуры:

1. **style** определяет стиль органа управления. Значение этого поля может быть комбинаций значений стиля окна (типа WS_BORDER) и стиля органа управления (типа BS_PUSHBUTTON и ES_LEFT).
2. **dwExtendedStyle** задает расширенные стили для окна. Это поле для задания органов управления панели не используют. Но оно может быть использовано для создания других типов окон.
3. **x** задает координату, в диалоговых единицах, левого края органа управления в панели.
4. **y** задает координату, в диалоговых единицах, верхнего края органа управления в панели.
5. **cx** – ширина органа управления.
6. **cy** – высота органа управления.
7. **id** – идентификатор органа управления. Это значение содержится в младшем слове параметра wParam при поступлении в функцию окна панели сообщения WM_COMMAND от этого органа управления.

После каждой записи значения типа DLGITEMTEMPLATE следуют строки имени класса, заголовка и дополнительных данных создания органа управления. Причем каждая строка состоит из одного или более 16-разрядных значений. Строки имени класса и заголовка должны быть выровнены по границе WORD. А массив данных должен быть выровнен по границе DWORD.

Строка имени класса идентифицирует класс окна органа управления. Если ее первый элемент отличается от 0xFFFF, то система об-

рабатывает ее как строку Unicode с именем зарегистрированного класса. Иначе строка должна содержать еще один элемент, который определяет порядковый номер предопределенного класса. Порядковые номера предопределенных классов системы перечислены в следующей таблице:

Номер	*Имя класса*	*Номер*	*Имя класса*	*Номер*	*Имя класса*
0x0080	"button"	0x0082	"static"	0x0084	"scrollbar"
0x0081	"edit"	0x0083	"listbox"	0x0085	"combobox"

Строка заголовка содержит текст заголовка или идентификатор ресурса управления. Если ее первый элемент равен 0xFFFF, должен быть еще один элемент, который определяет порядковый номер ресурса или идентификатор статического органа управления – пиктограммы. Если первый элемент строки отличен от 0xFFFF, система обрабатывает ее как строку Unicode с текстом заголовка или начального текста (например, редактора).

Массив данных создания может иметь любой размер и формат. Если первое слово этого массива отлично от нуля, то оно равно размеру данных в байтах плюс 4 байта первого элемента. Данные должны быть согласованы с функцией окна органа управления, т. е. функция должна быть способна интерпретировать данные. При создании органа управления система передает указатель на эти данные в параметре lParam сообщения WM_CREATE.

Пример. Разработать функцию для записи значения структуры типа DLGITEMTEMPLATE в шаблон панели.

Решение этой задачи может быть описано в виде:

```
void DlgItemTemplate(PWORD& p, DWORD lStyle,
        int x, int y, int cx, int cy, WORD id, LPSTR classname, LPSTR txt)
{   *p++ = LOWORD(lStyle);          //В первые два слова
    *p++ = HIWORD(lStyle);          //записываем стиль органа управления
    *p++ = 0;           //В следующие две строки можно
    *p++ = 0;           //записать расширенный стиль окна
    *p++ = x;           //Координата левого края органа управления
    *p++ = y;           //Координата верхнего края органа управления
    *p++ = cx;          //Ширина органа управления
    *p++ = cy;          //Высота органа управления
    *p++ = id;          //Идентификатор органа управления
    //Преобразуем Ansi-строку имени класса в строку Unicode
    nt nchar= nCopyAnsiToWideChar( p, TEXT( classname ) );
    p += nchar;         //Смещаем указатель на количество символов
    //Преобразуем Ansi-строку заголовка в строку Unicode
```

```
    if ( lstrlen( txt ) > 0 )
        nchar = nCopyAnsiToWideChar( p, TEXT( txt ) );
    else
        nchar = nCopyAnsiToWideChar(p,TEXT(" "));
    p += nchar;          //Смещаем указатель на количество символов
    *p++ = 0;            //Дополнительные данные не используем
    //Выравниваем шаблон по границе WORD
    p=lpwAlign( (LPWORD)p );
}
```

Как видно, эта функция полностью повторяет описание структуры DLGITEMTEMPLATE.

Как пользоваться этой функцией?

Следующий фрагмент описывает запись структуры данных статического элемента (здесь это текст "Сейчас следует:") в шаблон панели.

```
char statictext[ ]="Сейчас следует:";   //Текст органа управления
Height=height;                          //Высота статического элемента
lStyle = SS_LEFT | WS_CHILD | WS_VISIBLE;
DlgItemTemplate(
        p,                  //Указываем на то место блока памяти,
                            //куда записать шаблон органа управления
        lStyle,             //Стиль
        left,               //Координата левого края органа управления
        top,                //Координата верхнего края органа управления
        (lstrlen(statictext))*cxChar*4/dlgwunit,  //Ширина
        Height,             //Высота
        ID_STATIC1,         //Идентификатор органа управления
        TEXT("static"),     //Имя класса
        TEXT(statictext));  //Заголовок
```

Задача. В центре экрана создать модальную диалоговую панель с заголовком "Изменение режима работы приложения". Панель содержит группу с именем "Сейчас следует: " зависимых переключателей с именами "завершить работу приложения", "перезагрузить данные" и "сменить имя пользователя", а также 3 кнопки: "Да", "Отмена" и "Справка".

Листинг 6.1. Создание диалоговой панели.

```
#include <windows.h>
#include "CreateDlg.h"

#define  ID_STATIC      2000
#define  ID_BUTTON1     2001
#define  ID_BUTTON2     2002
```

```
#define  ID_BUTTON3   2003
#define  ID_HELP      2004

BOOL        RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT     CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
LRESULT     CALLBACK DlgProc(HWND,UINT,WPARAM, LPARAM);
int         CreateDlg(HWND);
HINSTANCE   hInstance;
char        szClass[ ]="WindowAppClass";

int WINAPI  WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                    LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd; ::hInstance=hInstance;
    if ( !RegClass(WndProc,szClass,COLOR_DESKTOP) )
          return FALSE;
    int  wScreen=GetSystemMetrics(SM_CXSCREEN);
    int  hScreen=GetSystemMetrics(SM_CYSCREEN);
    hwnd = CreateWindow(szClass, "Создание диалоговой панели",
       WS_OVERLAPPEDWINDOW | WS_VISIBLE,
       0, 0, wScreen, hScreen, 0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0))
    { TranslateMessage(&msg);   DispatchMessage(&msg);}
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc;  wc.style=wc.cbClsExtra=wc.cbWndExtra=0;
    wc.lpfnWndProc=Proc; wc.hInstance = hInstance;
    wc.hIcon=LoadIcon(NULL,IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName  = (LPCTSTR)NULL;   wc.lpszClassName=szName;
    return (RegisterClass(&wc) != 0);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam )
{   switch (msg)
    {   case WM_LBUTTONDOWN:{CreateDlg(hwnd); return 0; }
        case WM_DESTROY:  {PostQuitMessage(0);  return 0; }
```

```
    }   return DefWindowProc(hwnd, msg, wParam, lParam);
}

int CreateDlg( HWND hwnd )
{   char const caption[ ]="  Изменение режима работы приложения";
    char const stattxt[ ]="Сейчас следует:";
    char const modeoff[ ]="завершить работу приложения";
    char const modedat[ ]="перезагрузить данные";
    char const modepsw[ ]="сменить имя пользователя";
    //Выделяем блок памяти для записи шаблона
    WORD  *p, *pdlgtemplate;
    pdlgtemplate=p=(PWORD)LocalAlloc(LPTR,2000);
    //Определяем среднюю ширину cxChar и высоту cyChar шрифта
    int cxChar, cyChar;
    {   TEXTMETRIC  tm;      HDC hdc=GetDC(hwnd);
        GetTextMetrics(hdc,&tm);  ReleaseDC(hwnd,hdc);
        cxChar=tm.tmAveCharWidth+1;
        cyChar=tm.tmHeight+tm.tmExternalLeading;
    }
    //Определяем диалоговые единицы ширины и высоты
    DWORD    dlgunit =GetDialogBaseUnits();
    int      dlgwunit=LOWORD(dlgunit),
             dlghunit=HIWORD(dlgunit);
    //Пересчитываем габариты символов шрифта
    cxChar=cxChar*4/dlgwunit;cyChar=cyChar*8/dlghunit;

    int  wDlg, hDlg, wItem, hItem, left, top;

    //Записываем в шаблон данные панели
    DWORD lStyle = DS_CENTER | DS_MODALFRAME |
                   WS_POPUPWINDOW | WS_CAPTION;
    wDlg=lstrlen(caption)*cxChar;      hDlg=cyChar*10;
    DlgTemplate(p, lStyle, 7, 0, 0, wDlg, hDlg, (LPSTR)caption);

    //Далее добавляем записи органов управления
    //1
    hItem=cyChar; top=left=hItem/2;   hItem+=left;
    wItem=(wDlg-left-left);
    lStyle = WS_CHILD | WS_VISIBLE | BS_GROUPBOX | WS_TABSTOP;
    DlgItemTemplate(p, lStyle, left, top, wItem, 4*hItem+left,
        ID_STATIC, (LPSTR)"button",(LPSTR)stattxt);
```

```
    //2
    wItem=lstrlen(modeoff)*cxChar+10;    top+=hItem;
    lStyle = BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE |  WS_TABSTOP;
    DlgItemTemplate(p, lStyle, hItem, top, wItem, hItem,
        ID_BUTTON1, (LPSTR)"button", (LPSTR)modeoff);
    //3
    wItem=lstrlen(modedat)*cxChar+10;    top+=hItem;
    lStyle = BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE;
    DlgItemTemplate(p, lStyle, hItem, top, wItem, hItem,
        ID_BUTTON2, "button", (LPSTR)modedat);
    //4
    wItem=lstrlen(modepsw)*cxChar+10;   top+=hItem;
    lStyle = BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE;
    DlgItemTemplate(p, lStyle, hItem, top, wItem, hItem,
        ID_BUTTON3, (LPSTR)"button", (LPSTR)modepsw);
    //5
    wItem=(wDlg-left-left-hItem-hItem)/3;
    top+=hItem+hItem/2+left;
    lStyle = BS_DEFPUSHBUTTON | WS_VISIBLE | WS_CHILD | WS_TABSTOP;
    DlgItemTemplate(p, lStyle, left, top, wItem, hItem,
        IDOK, (LPSTR)"button",(LPSTR)"Да");
    //6
    lStyle = BS_PUSHBUTTON | WS_VISIBLE | WS_CHILD | WS_TABSTOP;
    DlgItemTemplate(p, lStyle, left+wItem+hItem, top, wItem, hItem,
        IDCANCEL, (LPSTR)"button",(LPSTR)"Отмена");
    //7
    DlgItemTemplate(p, lStyle, left+wItem+hItem+wItem+hItem, top,
            wItem, hItem, ID_HELP, (LPSTR)"button",(LPSTR)"Справка");

    //Создаем модальное диалоговое окно
    DialogBoxIndirect(hInstance,
            (LPDLGTEMPLATE)pdlgtemplate,hwnd, (DLGPROC)DlgProc);
    //Освобождаем занятый под шаблон блок памяти
    LocalFree(LocalHandle(pdlgtemplate));
    return 0;
}

LRESULT CALLBACK DlgProc(HWND hdlg, UINT msg,
                    WPARAM wParam, LPARAM lParam)
{   switch (msg)
    {   case WM_COMMAND:
        {   switch (LOWORD(wParam))
```

```
            {   case IDOK:
                case IDCANCEL:
                {   EndDialog(hdlg,TRUE);
                    return TRUE;
                }
            }
        }
    }
    return FALSE;
}
```

При запуске этого приложения весь экран займет окно приложения с заголовком "Создание диалоговой панели". После нажатия левой клавиши мыши в центре экрана появляется диалоговая панель, изображенная на рис. 6.1.

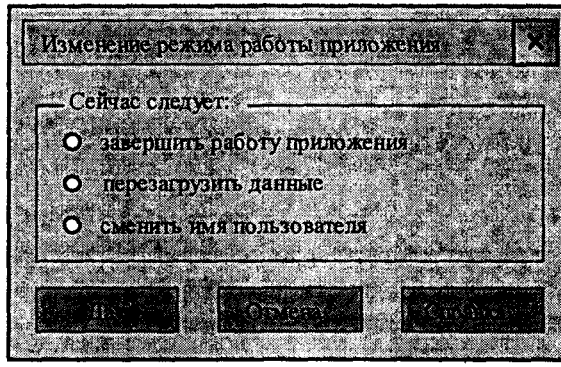

Рис. 6.1. Пример диалоговой панели к листингу 6.1

Изначально в группе переключателей не выбран ни один переключатель. Это сделано с целью избежать случайного двойного нажатия левой клавиши мыши и, как следствие, выбора значения переключателя по умолчанию. Такой подход к реализации диалоговых панелей часто помогает пользователю избежать случайных ошибок.

Рассмотрим те части текста приложения, которые относятся к созданию диалоговой панели и работе с ней.

1. Все функции, которые предназначены для записи шаблона в оперативную память, описаны в отдельном файле (CreateDlg.h). Имя файла и способ включения его в проект могут быть любыми другими. Здесь используется наиболее простой способ:

```
#include "CreateDlg.h"
```

Перед компиляцией проекта препроцессор включит текст файла CreateDlg.h в текст, представленный в листинге 6.1.

В этом примере и в дальнейшем текст файла CreateDlg.h имеет следующий вид:

```
//Объявление функций
LPWORD lpwAlign (LPWORD);
int nCopyAnsiToWideChar (LPWORD, LPSTR);
void DlgTemplate(PWORD&  p, DWORD lStyle, int items,
                int x, int y, int cx, int cy, LPSTR txt);
void DlgItemTemplate(PWORD& p, DWORD lStyle,
                int x, int y, int cx, int cy, WORD id, LPSTR classname, LPSTR txt);

LPWORD lpwAlign(LPWORD lpIn)
{   ULONG ul=(ULONG)lpIn;
    ul +=3; ul >>=2; ul <<=2; return (LPWORD)ul;
}

int nCopyAnsiToWideChar (LPWORD lpWCStr, LPSTR lpAnsiIn)
{   int cchAnsi=lstrlen(lpAnsiIn);
    return   MultiByteToWideChar(GetACP(), MB_PRECOMPOSED,
                lpAnsiIn, cchAnsi, lpWCStr, cchAnsi) + 1;
}

void DlgTemplate(PWORD&  p, DWORD lStyle, int items,
                        int x, int y, int cx, int cy, LPSTR txt)
{   *p++ = LOWORD(lStyle);   *p++ = HIWORD(lStyle);
    *p++ = 0;                *p++ = 0;
    *p++ = items;            *p++ = x;   *p++ = y;
    *p++ = cx;               *p++ = cy;
    *p++ = 0;                *p++ = 0;
    int nchar=nCopyAnsiToWideChar(p,TEXT(txt));
    p += nchar;          p=lpwAlign((LPWORD)p);
}

void DlgItemTemplate(PWORD& p, DWORD lStyle,
        int x, int y, int cx, int cy, WORD id, LPSTR classname, LPSTR txt)
{   *p++ = LOWORD(lStyle);   *p++ = HIWORD(lStyle);
    *p++ = 0;   *p++ = 0;        *p++ = x; *p++ = y;
    *p++ = cx;  *p++ = cy;       *p++ = id;
    int nchar= nCopyAnsiToWideChar(p, TEXT(classname));
    p += nchar;
    if (lstrlen(txt)>0)     nchar=nCopyAnsiToWideChar(p,TEXT(txt));
```

```
    else                nchar=nCopyAnsiToWideChar(p,TEXT(" "));
    p += nchar; *p++ = 0;       p=lpwAlign((LPWORD)p);
}
```

Как видим, все функции файла CreateDlg.h хорошо знакомы по вышеприведенным пояснениям.

2. Для некоторых органов управления описаны идентификаторы:

```
#define   ID_STATIC      2000
#define   ID_BUTTON1     2001
#define   ID_BUTTON2     2002
#define   ID_BUTTON3     2003
#define   ID_HELP        2004
```

3. Функция окна приложения при нажатии левой клавиши мыши над рабочей областью вызывает функцию CreateDlg:

```
case WM_LBUTTONDOWN:  {   CreateDlg(hwnd);  return 0;   }
```

4. Функция CreateDlg – обычная функция языка Си и выполняет следующие действия:

4.1. Описывает строки текстов органов управления:

```
char const caption[ ]="  Изменение режима работы приложения";
char const stattxt[ ]="Сейчас следует:";
char const modeoff[ ]="завершить работу приложения";
char const modedat[ ]="перезагрузить данные";
char const modepsw[ ]="сменить имя пользователя";
```

Эти описания в тело функции CreateDlg включены в качестве примера. В других задачах строки данных могут быть любыми другими и заданы другим способом.

4.2. Выделяет блок памяти для записи шаблона. С этой целью описаны указатели на массив элементов типа WORD:

```
WORD *p, *pdlgtemplate;
```

Причем они изначально указывают на начало выделенного блока памяти:

```
pdlgtemplate=p=(PWORD)LocalAlloc(LPTR,2000);
```

4.3. Определяет среднюю ширину и высоту системного шрифта. Здесь мы не рассматриваем другие шрифты. При необходимости можно создать любой другой шрифт рассмотренными в главе 3 способами.

Процедура определения средней ширины и высоты символов шрифта в пикселях ничуть не изменилась:

```
int cxChar, cyChar;
{   TEXTMETRIC tm;     HDC hdc=GetDC(hwnd);
    GetTextMetrics(hdc,&tm);  ReleaseDC(hwnd,hdc);
    cxChar=tm.tmAveCharWidth+1;
    cyChar=tm.tmHeight+tm.tmExternalLeading;
}
```

4.4. Определяет диалоговые единицы ширины и высоты:

```
DWORD   dlgunit =GetDialogBaseUnits();
int     dlgwunit=LOWORD(dlgunit), dlghunit=HIWORD(dlgunit);
```

4.5. Пересчитывает среднюю ширину и высоту символов шрифта в диалоговых единицах:

```
cxChar=cxChar*4/dlgwunit;cyChar=cyChar*8/dlghunit;
```

Это завершает подготовительную работу и можно приступать к записи данных в выделенный блок памяти.

4.6. Запись данных панели в память содержит следующие шаги:

4.6.1. Описание стиля панели:

```
DWORD lStyle = DS_CENTER | DS_MODALFRAME |
        WS_POPUPWINDOW | WS_CAPTION;
```

Здесь константа DS_CENTER освобождает нас от необходимости расчета координат левого верхнего угла панели. Такой стиль гарантирует, что панель всегда будет расположена в центре экрана.

4.6.2. Ширину панели приближенно вычисляют с помощью самой длинной строки (например, строки заголовка caption):

```
wDlg=lstrlen(caption)*cxChar;
```

Здесь количество символов умножают на среднюю ширину строчных символов шрифта в диалоговых единицах.

4.6.3. Высота панели также определена очень простым способом:

```
hDlg=cyChar*10;
```

Теперь все готово для записи шаблона в память.

4.6.4. Непосредственную запись в память выполняет функция DlgTemplate:

```
DlgTemplate(p, lStyle, 7, 0, 0, wDlg, hDlg, (LPSTR)caption);
```

4.7. Запись данных органов управления панели отличается только расчетами координат и описаниями стилей:

```
hItem=cyChar; top=left=hItem/2;   hItem+=left;  wItem=(wDlg-left-left);
lStyle = WS_CHILD | WS_VISIBLE | BS_GROUPBOX | WS_TABSTOP;
```

```
DlgItemTemplate(p, lStyle, left, top, wItem, 4*hItem+left,
        ID_STATIC, (LPSTR)"button",(LPSTR)stattxt);
wItem=lstrlen(modeoff)*cxChar+10;     top+=hItem;
lStyle = BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP;
DlgItemTemplate(p, lStyle, hItem, top, wItem, hItem,
      ID_BUTTON1, (LPSTR)"button", (LPSTR)modeoff);
...
DlgItemTemplate(p, lStyle, left+wItem+hItem+wItem+hItem, top,
           wItem, hItem, ID_HELP, (LPSTR)"button",(LPSTR)"Справка");
```

4.8. Создание модальной диалоговой панели сводится к вызову функции DialogBoxIndirect:

```
DialogBoxIndirect(hInstance, (LPDLGTEMPLATE)pdlgtemplate,hwnd,
      (DLGPROC)DlgProc);
```

Последним аргументом вызова передают имя функции окна диалоговой панели.

4.9. Функция CreateDlg перед завершением своей работы освобождает занятый под шаблон блок памяти:

```
LocalFree(LocalHandle(pdlgtemplate));
```

5. Функция окна панели описана следующим образом:

```
LRESULT CALLBACK DlgProc(HWND hdlg, UINT msg,
                  WPARAM wParam, LPARAM lParam)
{   switch (msg)
    {   case WM_COMMAND:
        {   switch (LOWORD(wParam))
            {   case IDOK:
                case IDCANCEL:
                {   EndDialog(hdlg,TRUE);
                    return TRUE;
                }
            }
        }
    }
    return FALSE;
}
```

Как видим, она реагирует только на нажатие кнопок "Да", "Отмена" и клавиши Esc. В ответ на их нажатие панель завершает свою работу и возвращает управление функции CreateDlg.

Это все отличия в тексте приложения, которые понадобились для создания диалоговой панели.

6.2.4. Пример немодальной диалоговой панели

Преобразуем приложение листинга 6.1 для создания немодальной диалоговой панели.

Задача. В центре экрана создать немодальную диалоговую панель с заголовком "Изменение режима работы приложения". Панель содержит группу с именем "Сейчас следует: " зависимых переключателей с именами "завершить работу приложения", "перезагрузить данные" и "сменить имя пользователя", а также 3 кнопки: "Да", "Отмена" и "Справка".

Листинг 6.2. Создание немодальной диалоговой панели.

```
#include <windows.h>
#include "CreateDlg.h"
#define  ID_STATIC    2000
#define  ID_BUTTON1   2001
#define  ID_BUTTON2   2002
#define  ID_BUTTON3   2003
#define  ID_HELP      2004

BOOL        RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT     CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
LRESULT     CALLBACK DlgProc(HWND,UINT,WPARAM,LPARAM);
HWND        CreateDlg(HWND, WORD*);
HINSTANCE   hInstance;
char    szClass[ ]="WindowAppClass";
int WINAPI  WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                    LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd; ::hInstance=hInstance;
    if ( !RegClass(WndProc,szClass,COLOR_DESKTOP) )
        return FALSE;
    int wScreen=GetSystemMetrics(SM_CXSCREEN);
    int hScreen=GetSystemMetrics(SM_CYSCREEN);
    hwnd = CreateWindow(szClass, "Создание диалоговой панели",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        0, 0, wScreen, hScreen, 0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    while(GetMessage(&msg, 0, 0, 0))
    { TranslateMessage(&msg);   DispatchMessage(&msg);}
    return msg.wParam;
}
```

```
BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
                    UINT brBackground)
{   WNDCLASS wc;  wc.style=wc.cbClsExtra=wc.cbWndExtra=0;
    wc.lpfnWndProc=Proc; wc.hInstance = hInstance;
    wc.hIcon=LoadIcon(NULL,IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName  = (LPCTSTR)NULL;   wc.lpszClassName=szName;
    return (RegisterClass(&wc) != 0);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam )
{   static   WORD *pdlgtemplate;
    static   HWND hdlg;
    switch (msg)
    {   case WM_CREATE:
        {   pdlgtemplate=(PWORD)LocalAlloc(LPTR,2000);
            hdlg=CreateDlg(hwnd, pdlgtemplate);
            return 0;
        }
        case WM_LBUTTONDOWN:
        {   ShowWindow(hdlg,SW_SHOWNORMAL); return 0; }
        case WM_RBUTTONDOWN:
        {   ShowWindow(hdlg,SW_HIDE); return 0; }
        case WM_DESTROY:
        {   DestroyWindow(hdlg);
            LocalFree(LocalHandle(pdlgtemplate));
            PostQuitMessage(0);
            return 0;
        }
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}

HWND CreateDlg( HWND hwnd, WORD  *pdlgtemplate )
{   char const caption[ ]="  Изменение режима работы приложения";
    char const stattxt[ ]="Сейчас следует:";
    char const modeoff[ ]="завершить работу приложения";
    char const modedat[ ]="перезагрузить данные";
    char const modepsw[ ]="сменить имя пользователя";
```

```
WORD *p; p=pdlgtemplate;

int cxChar, cyChar;
{   TEXTMETRIC tm;     HDC hdc=GetDC(hwnd);
    GetTextMetrics(hdc,&tm);  ReleaseDC(hwnd,hdc);
    cxChar=tm.tmAveCharWidth+1;
    cyChar=tm.tmHeight+tm.tmExternalLeading;
}
DWORD   dlgunit =GetDialogBaseUnits();
int         dlgwunit=LOWORD(dlgunit),
            dlghunit=HIWORD(dlgunit);
cxChar=cxChar*4/dlgwunit;cyChar=cyChar*8/dlghunit;

int wDlg, hDlg, wItem, hItem, left, top;

DWORD lStyle;

//Записываем в шаблон данные панели
lStyle = DS_CENTER | DS_MODALFRAME |
                WS_POPUPWINDOW | WS_CAPTION;
wDlg=lstrlen(caption)*cxChar;  hDlg=cyChar*10;
DlgTemplate(p, lStyle, 7, 0, 0, wDlg, hDlg, (LPSTR)caption);

//Далее добавляем записи для органов управления
//1
hItem=cyChar; top=left=hItem/2;   hItem+=left;
wItem=(wDlg-left-left);
lStyle = WS_CHILD | WS_VISIBLE | BS_GROUPBOX | WS_TABSTOP;
DlgItemTemplate(p, lStyle, left, top, wItem, 4*hItem+left,
    ID_STATIC, (LPSTR)"button",(LPSTR)stattxt);
//2
wItem=lstrlen(modeoff)*cxChar+10;     top+=hItem;
lStyle = BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE |
                WS_TABSTOP;
DlgItemTemplate(p, lStyle, hItem, top, wItem, hItem,
    ID_BUTTON1, (LPSTR)"button", (LPSTR)modeoff);
//3
wItem=lstrlen(modedat)*cxChar+10;     top+=hItem;
lStyle = BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE;
DlgItemTemplate(p, lStyle, hItem, top, wItem, hItem,
    ID_BUTTON2, "button", (LPSTR)modedat);
//4
wItem=lstrlen(modepsw)*cxChar+10;   top+=hItem;
```

```
    lStyle = BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE;
    DlgItemTemplate(p, lStyle, hItem, top, wItem, hItem,
        ID_BUTTON3, (LPSTR)"button", (LPSTR)modepsw);
    //5
    wItem=(wDlg-left-left-hItem-hItem)/3;
    top+=hItem+hItem/2+left;
    lStyle = BS_DEFPUSHBUTTON | WS_VISIBLE | WS_CHILD | WS_TABSTOP;
    DlgItemTemplate(p, lStyle, left, top, wItem, hItem,
        IDOK, (LPSTR)"button",(LPSTR)"Да");
    //6
    lStyle = BS_PUSHBUTTON | WS_VISIBLE | WS_CHILD | WS_TABSTOP;
    DlgItemTemplate(p, lStyle, left+wItem+hItem, top, wItem, hItem,
        IDCANCEL, (LPSTR)"button",(LPSTR)"Отмена");
    //7
    DlgItemTemplate(p, lStyle,
        left+wItem+hItem+wItem+hItem, top, wItem, hItem,
        ID_HELP, (LPSTR)"button",(LPSTR)"Справка");

    //Создаем немодальное диалоговое окно
    HWND hdlg=CreateDialogIndirect(hInstance,
                    (LPDLGTEMPLATE)pdlgtemplate,hwnd,
                    (DLGPROC)DlgProc);
    return hdlg;
}

LRESULT CALLBACK DlgProc(HWND hdlg, UINT msg,
                    WPARAM wParam, LPARAM lParam)
{   switch (msg)
    {   case WM_COMMAND:
        {   switch (LOWORD(wParam))
            {   case IDOK:
                case IDCANCEL:
                {   EndDialog(hdlg,TRUE);  return TRUE; }
            }
        }
    }
    return FALSE;
}
```

Изменился процесс создания панели. Ранее блок памяти для записи шаблона выделялся и разрушался в теле функции CreateDlg. Теперь панель существует как обычное окно, создается один раз и может быть ис-

пользована сколь угодно раз. Для ее активизации достаточно вызвать привычную функцию ShowWindow.

1. Процесс создания и разрушения панели происходит в теле функции окна приложения (хотя это может быть и функция любого другого недочернего окна).

   1.1. При обработке сообщения WM_CREATE выделяют блок памяти для записи шаблона:

```
pdlgtemplate=(PWORD)LocalAlloc(LPTR,2000);
```

   Далее вызывают функцию CreateDlg:

```
hdlg=CreateDlg(hwnd, pdlgtemplate);
```

   Она возвращает дескриптор окна созданной диалоговой панели. Обратите внимание, теперь появился второй параметр функции CreateDlg. Через него функции CreateDlg передается указатель на выделенный блок памяти.

   1.2. Функция CreateDlg претерпела на первый взгляд незначительные, а на самом деле важные изменения. Она теперь не выделяет и не освобождает блок памяти для шаблона. Кроме этого, раньше она завершала работу сразу после разрушения диалоговой панели. Теперь функция возвращает дескриптор окна созданной панели и завершает работу. А созданная диалоговая панель может работать сколь угодно раз и сколь угодно долго.

   1.3. Освобождение выделенного блока памяти и разрушение окна панели происходят непосредственно перед завершением работы приложения:

```
case WM_DESTROY:
{  DestroyWindow(hdlg); LocalFree(LocalHandle(pdlgtemplate));
   PostQuitMessage(0);  return 0;
}
```

2. Работа с немодальной панелью пока представлена двумя способами:

   2.1. После нажатия левой клавиши мыши панель отображается на том месте экрана, где она находилась перед завершением своей работы:

```
case WM_LBUTTONDOWN:
{  ShowWindow(hdlg,SW_SHOWNORMAL); return 0;  }
```

   2.2. После нажатия правой клавиши мыши панель становится невидимой:

```
case WM_RBUTTONDOWN:
{  ShowWindow(hdlg,SW_HIDE);    return 0;  }
```

На окно этой панели можно переключаться любым способом, и оно не блокирует работу других окон. То есть это самая обычная немодальная диалоговая панель.

Обратите внимание, что **эта диалоговая панель**, в отличие от модальной панели, **пока не реагирует на клавиатурные сообщения**.

6.3. Сообщения и диалоговые панели

Органы управления панели обычным способом посылают сообщение WM_COMMAND в функцию окна панели. Здесь никаких изменений не произошло. Приложение также может посылать сообщения органам управления панели. Но здесь появились изменения. Они обусловлены тем, что органы управления на поверхности панели расположены с помощью шаблона, а не созданы как окна. Чтобы с помощью функции SendMessage посылать сообщения органам управления панели, нужно получить дескриптор их окна.

Функция GetDlgItem возвращает дескриптор окна органа управления панели hdlg:

```
HWND GetDlgItem( HWND hdlg, int nIDDlgItem );
```

Параметр nIDDlgItem задает идентификатор органа управления, дескриптор окна которого нужно вернуть. Этот аргумент равен тому значению, которое использовано в качестве идентификатора при записи данных органа управления в шаблон панели.

В случае успешного выполнения функция возвращает дескриптор окна данного органа управления, иначе – NULL.

Функцию GetDlgItem можно использовать для получения дескриптора любого дочернего окна, указав первым параметром дескриптор родительского окна. То есть первым аргументом вызова может быть дескриптор не только окна диалоговой панели. Дескриптор окна модальной панели можно указывать только в теле функции окна этой панели. Дескриптор окна немодальной диалоговой панели может быть указан в любом месте, если эта панель существует.

Полученным дескриптором можно пользоваться сколь угодно раз. Если, например, нужно многократно управлять состоянием переключателя с идентификатором ID_SWITCH, то получают дескриптор его окна и посылают ему сообщения:

```
HWND hSwitch = GetDlgItem( hdlg, IDC_SWITCH );
...
SendMessage( hSwitch, BM_SETCHECK, TRUE, 0L );
```

Не запрещается использовать и сложный вызов:

```
SendMessage( GetDlgItem( hdlg, IDC_SWITCH ), BM_SETCHECK, TRUE, 0L );
```

Такой вызов обычно используют, если нужно только один раз послать сообщение органу управления.

Диалоговые панели создают на базе перекрывающихся или временных окон. Это означает, что **панели могут иметь меню и панель инструментов**. Создание, обработка сообщений элементов меню и управление состоянием строк меню в панели происходят обычным способом. То же самое справедливо и для панели инструментов.

Существует ряд объективных причин для отказа от работы с акселераторами в панелях. Во-первых, предопределенный класс диалоговых окон уже содержит обработку клавиатурных сообщений. Эту обработку в целях сохранения единых правил работы пользователя с панелями не следует менять без особой надобности. Во-вторых, возникают большие сложности программной реализации цикла обработки сообщений.

При работе с немодальной диалоговой панелью сообщения для органов управления панели проходят через общую очередь сообщений приложения. Чтобы отдельно обработать сообщение для диалоговой панели, вызывают функцию IsDialogMessage. Она обрабатывает сообщения, предназначенные для окна hdlg:

```
BOOL IsDialogMessage( HWND hdlg, LPMSG lpMsg );
```

Параметр lpMsg указывает на структуру с сообщением.

Функция IsDialogMessage передает это сообщение функции окна hdlg. Если функция окна обработала сообщение, то функция IsDialogMessage возвращает ненулевое значение, иначе возвращает 0. Обработанное функцией IsDialogMessage сообщение не нужно передавать функциям TranslateMessage и DispatchMessage.

Эта функция предназначена для работы с немодальными диалоговыми панелями. Но она так же работает с любым окном, содержащим органы управления. При этом обеспечивается такой же режим работы с клавиатурой, который используется в работе с модальными панелями.

Функция IsDialogMessage проверяет сообщения клавиатуры и преобразует их в команды для указанного окна. Например, нажатие клавиши Tab переводит фокус ввода на следующий орган или группу органов управления, а нажатие стрелок перемещения передает фокус ввода соседнему в группе органу управления.

Цикл обработки сообщений при этом может быть преобразован в следующий вид:

```
while ( GetMessage(&msg, 0, 0, 0) )
{   if ( !hdlg || !IsDialogMessage(hdlg, &msg) )
    {   TranslateMessage(&msg);  DispatchMessage(&msg);  }
}
```

Глобальная переменная hdlg используется для хранения дескриптора окна немодальной панели. Окно панели может быть создано позже. Поэтому переменная hdlg должна быть изначально инициирована нулем, например следующим образом:

```
static HWND hdlg;
```

После создания немодальной панели в цикле обработки сообщений будет вызываться функция IsDialogMessage. Она будет обрабатывать сообщения, предназначенные для панели hdlg. Другие сообщения будут обработаны функциями TranslateMessage и DispatchMessage.

Задача. Окно приложения содержит меню из одного раздела "Файл" со строками "Открыть", "Сохранить" и "Выход". При выборе строки "Открыть" отобразить немодальную панель с таким же меню. Панель также содержит статический объект с текстом "Выбранная строка – " и, чуть правее, пустой статический объект. Ниже первого статического объекта создать пустой список. Ниже правого статического органа расположить кнопку по умолчанию "Готово" и еще ниже – кнопку "Выход". При выборе строки "Открыть" меню панели в список добавить несколько строк. При нажатии кнопки "Готово" во втором статическом объекте отобразить выбранную в списке строку. При выборе кнопки "Выход" панель завершает работу. Панель отображать после выбора строки "Открыть" и скрыть после выбора строки "Сохранить" меню приложения. Команды элементов меню продублировать акселераторами.

Листинг 6.3. Диалоговая панель с меню.

```
#include <windows.h>
#include "CreateDlg.h"

#define  CM_FILE_OPEN   1001
#define  CM_FILE_SAVE   1002
#define  CM_FILE_QUIT   1003
#define  ID_STATIC      2000
#define  ID_LISTBOX     2001

BOOL        RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT     CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
LRESULT     CALLBACK DlgProc(HWND,UINT,WPARAM,LPARAM);
HWND        CreateDlg(HWND, WORD*);
HACCEL      CreateAccelTable( void );
HINSTANCE   hInstance;
char        szClass[ ]="WindowAppClass";
static      HWND hdlg;
```

```
int WINAPI  WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                    LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd; ::hInstance=hInstance;
    if  ( !RegClass(WndProc,szClass,COLOR_DESKTOP) )
                return FALSE;
    int  wScreen=GetSystemMetrics(SM_CXSCREEN);
    int  hScreen=GetSystemMetrics(SM_CYSCREEN);
    hwnd = CreateWindow(szClass, "Создание диалоговой панели",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        0, 0, wScreen, hScreen, 0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    HACCEL hAccel=CreateAccelTable();
    while ( GetMessage(&msg, 0, 0, 0) )
    {   if ( !hdlg || !IsDialogMessage(hdlg, &msg) )
            if  ( !hAccel ||
                    !TranslateAccelerator(hwnd, hAccel, &msg) )
            {       TranslateMessage(&msg); DispatchMessage(&msg);
            }
    }
    DestroyAcceleratorTable(hAccel);
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
                    UINT brBackground)
{   WNDCLASS wc;  wc.style=wc.cbClsExtra=wc.cbWndExtra=0;
    wc.lpfnWndProc=Proc; wc.hInstance = hInstance;
    wc.hIcon=LoadIcon(NULL,IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName = (LPCTSTR)NULL;
    wc.lpszClassName=szName;
    return (RegisterClass(&wc) != 0);
}

BOOL CreateMenuItem(HMENU hMenu, char *str, UINT uIns,
            UINT uCom, HMENU hSubMenu, BOOL flag, UINT fType)
{   MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE | MIIM_TYPE | MIIM_SUBMENU | MIIM_ID;
    mii.fType = fType;          mii.fState= MFS_ENABLED;
    mii.dwTypeData = str;       mii.cch=sizeof(str);
    mii.wID=uCom;               mii.hSubMenu=hSubMenu;
```

```
    return InsertMenuItem(hMenu, uIns, flag, &mii);
}

HACCEL   CreateAccelTable( void )
{   static ACCEL  Accel[3];
    Accel[0].fVirt= FVIRTKEY | FCONTROL;
    Accel[0].key  = 0x4f;    Accel[0].cmd  = CM_FILE_OPEN;
    Accel[1].fVirt= FVIRTKEY | FCONTROL;
    Accel[1].key  = 0x53;    Accel[1].cmd  = CM_FILE_SAVE;
    Accel[2].fVirt= FVIRTKEY | FALT;
    Accel[2].key  = 0x73;    Accel[2].cmd  = CM_FILE_QUIT;
    return CreateAcceleratorTable((LPACCEL)Accel,8);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                          WPARAM wParam, LPARAM lParam )
{   static   WORD *pdlgtemplate;
    static   HMENU    hMainMenu, hFileMenu;
    switch (msg)
    {   case WM_CREATE:
        {   hMainMenu=CreateMenu();
            hFileMenu=CreatePopupMenu();   int i=0;
            CreateMenuItem(hFileMenu,"&Открыть",i++,
                CM_FILE_OPEN, NULL, FALSE, MFT_STRING);
            CreateMenuItem(hFileMenu,"&Сохранить",i++,
                CM_FILE_SAVE, NULL, FALSE, MFT_STRING);
            CreateMenuItem(hFileMenu,NULL,i++,0,
                NULL, FALSE, MFT_SEPARATOR);
            CreateMenuItem(hFileMenu,"&Выход",i++,
                CM_FILE_QUIT, NULL, FALSE, MFT_STRING);
            CreateMenuItem(hMainMenu,"&Файл", 0, 0,
                hFileMenu,FALSE, MFT_STRING);
            SetMenu(hwnd,hMainMenu); DrawMenuBar(hwnd);

            pdlgtemplate=(PWORD)LocalAlloc(LPTR,2000);
            hdlg=CreateDlg(hwnd, pdlgtemplate);
            return 0;
        }
        case WM_COMMAND:
        {   switch (LOWORD(wParam))
            {   case CM_FILE_OPEN:
                {   ShowWindow(hdlg,SW_SHOWNORMAL);
```

```
                return 0;
            }
            case CM_FILE_SAVE:
            {   ShowWindow(hdlg,SW_HIDE);
                return 0;
            }
            case CM_FILE_QUIT:
            {   DestroyWindow(hwnd); return 0;   }
        }
        return 0;
    }
    case WM_DESTROY:
    {   DestroyWindow(hdlg);
        LocalFree(LocalHandle(pdlgtemplate));
        PostQuitMessage(0);
        return 0;
    }
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}

HWND CreateDlg( HWND hwnd, WORD  *pdlgtemplate )
{   char const caption[ ]="  Немодальная диалоговая панель";
    char const stattxt[ ]="Выбранная строка  -";

    WORD  *p;  p=pdlgtemplate;

    //Определяем средние ширину cxChar и высоту cyChar шрифта
    int cxChar,  cyChar;
    {   TEXTMETRIC  tm;      HDC hdc=GetDC(hwnd);
        GetTextMetrics(hdc,&tm);  ReleaseDC(hwnd,hdc);
        cxChar=tm.tmAveCharWidth+1;
        cyChar=tm.tmHeight+tm.tmExternalLeading;
    }
    //Определяем диалоговые единицы ширины и высоты
    DWORD    dlgunit =GetDialogBaseUnits();
    int      dlgwunit=LOWORD(dlgunit), dlghunit=HIWORD(dlgunit);
    //Пересчитываем габариты символов шрифта
    cxChar=cxChar*4/dlgwunit;cyChar=cyChar*8/dlghunit;

    int  wDlg, hDlg, wItem, hItem, left, top;

    DWORD lStyle;
```

```
    //Записываем в шаблон данные панели
    lStyle = DS_CENTER | DS_MODALFRAME |
                    WS_POPUPWINDOW | WS_CAPTION;
    wDlg=lstrlen(caption)*cxChar;  hDlg=cyChar*9;
    DlgTemplate(p, lStyle, 5, 0, 0, wDlg, hDlg, (LPSTR)caption);

    //Добавляем записи для органов управления
    //1
    hItem=cyChar; top=left=hItem/2;    hItem+=left;
    wItem=lstrlen(stattxt)*cxChar;
    lStyle = SS_LEFT | WS_CHILD | WS_VISIBLE;
    DlgItemTemplate(p, lStyle, left, top, wItem, hItem,
        -1, (LPSTR)"static",(LPSTR)stattxt);
    //2
    DlgItemTemplate(p, lStyle, left+wItem, top, wItem, hItem,
        ID_STATIC, (LPSTR)"static",(LPSTR)"");
    //3
    top+=hItem;
    lStyle = WS_CHILD | WS_VISIBLE | WS_TABSTOP |
                    LBS_STANDARD | LBS_DISABLENOSCROLL;
    DlgItemTemplate(p, lStyle, left, top, wItem, hItem*4,
            ID_LISTBOX, (LPSTR)"listbox",(LPSTR)"\0");
    //4
    wItem=wDlg-wItem-hItem; left+=left+lstrlen(stattxt)*cxChar;
    lStyle = BS_DEFPUSHBUTTON | WS_VISIBLE | WS_CHILD | WS_TABSTOP;
    DlgItemTemplate(p, lStyle, left, top, wItem, hItem,
        IDOK, (LPSTR)"button",(LPSTR)"Готово");
    //5
    top += hItem+hItem;
    lStyle = BS_PUSHBUTTON | WS_VISIBLE | WS_CHILD | WS_TABSTOP;
    DlgItemTemplate(p, lStyle, left, top, wItem, hItem,
        IDCANCEL, (LPSTR)"button",(LPSTR)"Выход");
    //Создаем немодальное диалоговое окно
    HWND hdlg=CreateDialogIndirect(hInstance,
                (LPDLGTEMPLATE)pdlgtemplate,hwnd,
                (DLGPROC)DlgProc);
    return hdlg;
}

LRESULT CALLBACK DlgProc(HWND hdlg, UINT msg,
                         WPARAM wParam, LPARAM lParam)
```

```
{   static   HMENU    hMainMenu, hFileMenu;
    switch (msg)
    {   case WM_INITDIALOG:
        {   hMainMenu=CreateMenu();
            hFileMenu=CreatePopupMenu();
            CreateMenuItem(hFileMenu,"&Открыть", 0,
                CM_FILE_OPEN, NULL, FALSE, MFT_STRING);
            CreateMenuItem(hFileMenu,"&Сохранить", 1,
                CM_FILE_SAVE, NULL, FALSE, MFT_STRING);
            CreateMenuItem(hFileMenu,NULL,2,0,
                NULL, FALSE, MFT_SEPARATOR);
            CreateMenuItem(hFileMenu,"&Выход",3,
                CM_FILE_QUIT, NULL, FALSE, MFT_STRING);
            CreateMenuItem(hMainMenu,"&Файл", 0, 0,
                hFileMenu,FALSE, MFT_STRING);
            SetMenu(hdlg,hMainMenu); DrawMenuBar(hdlg);
            return TRUE;
        }
        case WM_COMMAND:
        {   switch (LOWORD(wParam))
            {   case IDOK:
                {   int uSelectedItem;
                    char Buffer[256];
                    //Берем дескриптор окна-списка
                    HWND hListBox=GetDlgItem(hdlg,ID_LISTBOX);
                    //Определяем номер выделенной строки
                    uSelectedItem=(int)SendMessage(hListBox,
                                LB_GETCURSEL,0,0L);
                    //Если выделена строка
                    if(uSelectedItem != LB_ERR)
                    {   SendMessage(hListBox, LB_GETTEXT,
                            uSelectedItem,(LPARAM)Buffer);
                        SetWindowText(GetDlgItem(
                            hdlg,ID_STATIC), (LPCTSTR)Buffer);
                    }
                    return TRUE;
                }
                case CM_FILE_OPEN:
                {   HWND hListBox= GetDlgItem(hdlg,ID_LISTBOX);
                    //Отменяем режим перерисовки списка
                    SendMessage(hListBox,
```

```
                        WM_SETREDRAW, FALSE, 0L);
                    // Добавляем в список несколько строк
                    SendMessage(hListBox,LB_ADDSTRING,
                        0, (LPARAM)(LPSTR)"Зеленый");
                    SendMessage(hListBox,LB_ADDSTRING,
                        0, (LPARAM)(LPSTR)"Красный");
                    SendMessage(hListBox, LB_ADDSTRING,
                        0, (LPARAM)(LPSTR)"Розовый");
                    SendMessage(hListBox, LB_ADDSTRING,
                        0, (LPARAM)(LPSTR)"Пурпурный");
                    SendMessage(hListBox, LB_ADDSTRING,
                        0, (LPARAM)(LPSTR)"Синий");
                    SendMessage(hListBox, LB_ADDSTRING,
                        0, (LPARAM)(LPSTR)"Желтый");
                    SendMessage(hListBox, LB_ADDSTRING,
                        0, (LPARAM)(LPSTR)"Фиолетовый");
                    SendMessage(hListBox, LB_ADDSTRING,
                        0, (LPARAM)(LPSTR)"Черный");
                    SendMessage(hListBox, LB_ADDSTRING,
                        0, (LPARAM)(LPSTR)"Белый");
                    // Включаем режим перерисовки списка
                    SendMessage(hListBox, WM_SETREDRAW, TRUE, 0L);
                    // Перерисовываем список
                    InvalidateRect(hListBox, NULL, TRUE);
                    return TRUE;
                }
                case IDCANCEL:
                {   EndDialog(hdlg,TRUE);
                    ёreturn TRUE;
                }
            }
        }
    }
    return FALSE;
}
```

При запуске этого приложения весь экран займет окно приложения с заголовком "Создание диалоговой панели" и с главным меню. Меню содержит раздел "Файл" со строками "Открыть", "Сохранить" и "Выход". После выбора строки "Открыть" в центре экрана появляется диалоговая панель (рис. 6.2).

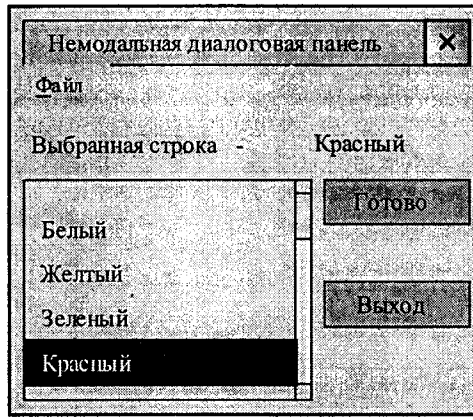

Рис. 6.2. Пример диалоговой панели к листингу 6.3

Панель имеет главное меню, которое содержит раздел "Файл" со строками "Открыть", "Сохранить" и "Выход". С этим меню работают точно так же, как и с меню приложения. Отличие лишь в том, что команды меню панели не могут быть сгенерированы с помощью акселераторов. Изначально список на панели пуст. Соответственно, пусто и место над кнопкой "Готово".

Если выбрать строку "Открыть" меню панели, то в список будут добавлены строки "Белый", "Желтый", "Зеленый", "Красный", "Пурпурный" и т. д. Эти строки могут быть добавлены в произвольном порядке, но список их сортирует в алфавитном порядке.

Если в списке выбрать строку (например, "Красный") и нажать кнопку "Готово", то правее статического органа "Выбранная строка –" отобразится выбранная строка списка.

Если нажать кнопку "Выход", панель завершит работу.

Если выбрать строку "Сохранить" главного меню приложения, панель становится невидимой.

Экспериментируя с приложением, можно увидеть, что эта немодальная панель обрабатывает сообщения от клавиатуры. Это достигнуто следующими изменениями в теле главного цикла обработки сообщений приложения:

```
while ( GetMessage(&msg, 0, 0, 0) )
{   if ( !hdlg || !IsDialogMessage(hdlg, &msg) )
        if  ( !hAccel || !TranslateAccelerator(hwnd, hAccel, &msg) )
        {   TranslateMessage(&msg); DispatchMessage(&msg); }
}
```

Рассмотрим изменения в тексте приложения.

Видно, что отличие внешнего вида панели обусловлено изменениями в теле функций CreateDlg и DlgProc. Причем в CreateDlg изменился лишь состав органов управления панели.

Функция DlgProc теперь выполняет следующие действия:

1. Обрабатывает сообщение WM_INITDIALOG. Здесь обработка сведена к созданию меню панели:

```
hMainMenu=CreateMenu();
hFileMenu=CreatePopupMenu();
CreateMenuItem(hFileMenu,"&Открыть", 0, CM_FILE_OPEN,
                NULL, FALSE, MFT_STRING);
...
CreateMenuItem(hFileMenu,"&Выход",3, CM_FILE_QUIT,
                NULL, FALSE, MFT_STRING);
CreateMenuItem(hMainMenu,"&Файл", 0, 0, hFileMenu, FALSE,
                MFT_STRING);
SetMenu(hdlg, hMainMenu);   DrawMenuBar(hdlg);
```

Как видно, процедура создания меню не изменилась.

2. Обрабатывает сообщение от строки "Открыть" меню панели:

```
case CM_FILE_OPEN:
{   HWND hListBox=GetDlgItem(hdlg,ID_LISTBOX);
    SendMessage(hListBox, WM_SETREDRAW, FALSE, 0L);
    SendMessage(hListBox,LB_ADDSTRING,
                0, (LPARAM)(LPSTR)"Зеленый");
...
    SendMessage(hListBox, LB_ADDSTRING,
                0, (LPARAM)(LPSTR)"Белый");
    SendMessage(hListBox, WM_SETREDRAW, TRUE, 0L);
    InvalidateRect(hListBox, NULL, TRUE);
    return TRUE;
}
```

2.1. При этом определяет дескриптор окна списка:

```
HWND hListBox=GetDlgItem(hdlg,ID_LISTBOX);
```

2.2. Отменяет режим автоматической перерисовки списка:

```
SendMessage(hListBox, WM_SETREDRAW, FALSE, 0L);
```

Это нужно для того, чтобы после добавления каждой строки список не перерисовывал свое содержимое.

2.3. Добавляет строки в список:

```
SendMessage(hListBox,LB_ADDSTRING, 0, (LPARAM)(LPSTR)"Зеленый");
```

2.4. Включает режим перерисовки списка:

```
SendMessage(hListBox, WM_SETREDRAW, TRUE, 0L);
```

2.5. Сообщает о необходимости перерисовки списка:

```
InvalidateRect(hListBox, NULL, TRUE);
```

3. Обрабатывает сообщения от кнопок:

3.1. После нажатия кнопки "Готово" определяет номер выбранного элемента списка:

```
uSelectedItem=(int)SendMessage(hListBox, LB_GETCURSEL,0,0L);
```

Если строка выделена, то выбирает строку с этим номером:

```
SendMessage(hListBox,LB_GETTEXT,uSelectedItem,(LPARAM)Buffer);
```

и устанавливает ее в окне органа с идентификатором ID_STATIC:

```
SetWindowText( GetDlgItem( hdlg, ID_STATIC ), (LPCTSTR)Buffer );
```

3.2. После нажатия кнопки "Выход" панель завершает работу:

```
EndDialog(hdlg,TRUE);
```

6.4. Блокнот диалоговых панелей

Диалоговые панели можно объединять в блокноты, где каждая страница представляет собой одну панель. Для создания блокнота вызывают функцию PropertySheet:

```
int PropertySheet( LPCPROPSHEETHEADER lppsph );
```

Она на базе данных структуры типа PROPSHEETHEADER создает блокнот. На эту структуру указывает параметр lppsph. В случае успешного выполнения возвращает положительное значение, иначе возвращаемое значение равно -1.

Структура PROPSHEETHEADER определяет базовые характеристики блокнота. Она описана следующим образом:

```
typedef struct
{   DWORD          dwSize;
    DWORD          dwFlags;
    HWND           hwndParent;
    HINSTANCE      hInstance;
    union
    {   HICON      hIcon;
        LPCTSTR    pszIcon;
    };
    LPCTSTR        pszCaption;
    UINT           nPages
    union
```

```
    { UINT        nStartPage
      LPCTSTR     pStartPage;
    };
    union
    { LPCPROPSHEETPAGE          ppsp;
      HPROPSHEETPAGE FAR        *phpage;
    };
    PFNPROPSHEETCALLBACK        pfnCallback;
} PROPSHEETHEADER;
```

Назначение полей структуры PROPSHEETHEADER:

1. **dwSize** – размер структуры PROPSHEETHEADER в байтах.
2. **dwFlags** определяет, какие элементы структуры используются. Значение этого поля часто задают комбинацией следующих констант:

Константа	*Пояснение*
PSH_DEFAULT	Все элементы установить по умолчанию
PSH_MODELESS	Блокнот создать в немодальном режиме
PSH_NOAPPLYNOW	Блокнот без кнопки "Применить"
PSH_PROPSHEETPAGE	При создании страниц блокнота использовать поле ppsp и игнорировать phpage
PSH_PROPTITLE	В области заголовка блокнота перед строкой, заданной в поле pszCaption, ставится строка "Свойства:"
PSH_USECALLBACK	При инициализации блокнота вызвать функцию, заданную полем pfnCallback
PSH_USEHICON	В заголовке блокнота использовать пиктограмму, на которую указывает поле hIcon
PSH_USEPSTARTPAGE	При отображении блокнота использовать поле pStartPage и игнорировать nStartPage

3. **hwndParent** – дескриптор окна-владельца.
4. **hInstance** – дескриптор экземпляра приложения, содержащего загружаемую пиктограмму или ресурс строки заголовка.
5. **hIcon** – дескриптор пиктограммы заголовка блокнота. Если поле dwFlags не содержит значение PSH_USEHICON, этот член игнорируется.
6. **pszIcon** – идентификатор ресурса или строки с именем ресурса пиктограммы. Если dwFlags не содержит значение PSH_USEICONID, этот член игнорируется.
7. **pszCaption** – идентификатор ресурса или строки с заголовком блокнота.

8. **nPages** – количество элементов в массиве phpage.
9. **nStartPage** – номер изначально раскрываемой страницы.
10. **pStartPage** – имя изначально раскрываемой страницы.
11. **ppsp** – указатель на массив структур типа PROPSHEETPAGE, который описывает страницы блокнота. Если dwFlags не включает значение PSH_PROPSHEETPAGE, этот член игнорируется.
12. **phpage** – указатель на массив дескрипторов окон страниц блокнота. Каждый дескриптор должен быть уже создан вызовом функции CreatePropertySheetPage. Если dwFlags включает значение PSH_PROPSHEETPAGE, этот член игнорируется.
13. **pfnCallback** – указатель на функцию повторного вызова, которая будет вызвана системой на этапе инициализации блокнота. Если dwFlags не включает PSP_USECALLBACK, этот член игнорируется.

Если поле dwFlags содержит значение PSH_MODELESS, то функция PropertySheet возвращает дескриптор окна блокнота.

Следующие возвращаемые функцией PropertySheet значения имеют специальное назначение:

Значение	*Пояснение*
ID_PSREBOOTSYSTEM	Страница послала блокноту сообщение PSM_REBOOTSYSTEM – внесенные изменения имеют силу только после перезагрузки компьютера
ID_PSRESTARTWINDOWS	Страница послала блокноту сообщение PSM_RESTARTWINDOWS – внесенные изменения имеют силу только после перезагрузки операционной системы

В случае создания немодального блокнота в цикле обработки сообщений для трансляции и передачи сообщений блокноту вызывают функцию PropSheet_IsDialogMessage:

```
PropSheet_IsDialogMessage( hdlg, &msg );
```

Параметр hdlg – дескриптор окна блокнота, msg – структура, которая содержит сообщение.

В случае обработки сообщения возвращаемое значение – TRUE.

Страницу блокнота описывает структура PROPSHEETPAGE:

```
typedef struct
{  DWORD                wSize;
   DWORD                dwFlags;
```

```
    HINSTANCE           hInstance;
    union
    {   LPCTSTR         pszTemplate;
        LPCDLGTEMPLATE  pResource;
    };
    union
    {   HICON           hIcon;
        LPCTSTR         pszIcon;
    };
    LPCTSTR             pszTitle;
    DLGPROC             pfnDlgProc;
    LPARAM              lParam;
    LPFNPSPCALLBACK     pfnCallback;
    UINT FAR            *pcRefParent;
} PROPSHEETPAGE;
```

Назначение полей структуры PROPSHEETPAGE:

1. **dwSize** равен размеру этой структуры в байтах плюс размер данных, задаваемых приложением в конце структуры.
2. **dwFlags** задает набор свойств страницы. Его значение может быть комбинацией констант из нижеследующей таблицы:

Значение	*Пояснение*
PSP_DEFAULT	Все поля структуры задать по умолчанию
PSP_DLGINDIRECT	Создать страницу из шаблона в памяти. При этом на блок памяти указывает поле с именем pResource. Иначе на шаблон указывает поле с именем pszTemplate
PSP_HASHELP	Страница имеет кнопку контекстной помощи
PSP_ USECALLBACK	Страница имеет функцию повторного вызова, которая будет вызвана системой при создании и разрушении страницы. Имя этой функции содержится в поле pfnCallback
PSP_USEHICON	Заголовок страницы содержит пиктограмму
PSP_USEICONID	Значение поля pszIcon использовать как имя ресурса пиктограммы для заголовка
PSP_ USEREFPARENT	Считать количество обращений к странице и хранить это количество в поле pcRefParent
PSP_USETITLE	Вместо заголовка, указанного в шаблоне, использовать заголовок, указанный в поле с именем pszTitle

3. **hInstance** – дескриптор экземпляра приложения, из которого загружать шаблон панели, пиктограмму или ресурс строки заголовка.
4. **pszTemplate** – идентификатор ресурса или адрес строки с именем шаблона панели. Этот член игнорируется, если шаблон создан в памяти и dwFlags содержит константу PSP_DLGINDIRECT.
5. **pResource** – указатель на блок памяти с шаблоном панели. Если поле dwFlags не содержит значение PSP_DLGINDIRECT, значение поля pResource игнорируется.
6. **hIcon** – дескриптор пиктограммы заголовка страницы. Если поле dwFlags не содержит значение PSP_USEHICON, значение поля hIcon игнорируется.
7. **pszIcon** – идентификатор ресурса или строки с именем пиктограммы. Если поле dwFlags не содержит значение PSP_USEICONID, значение поля pszIcon игнорируется.
8. **pszTitle** – заголовок страницы. Он отменяет заголовок, задаваемый шаблоном панели. Если поле dwFlags не содержит значение PSP_USETITLE, этот член игнорируется.
9. **pfnDlgProc** – указатель на функцию окна панели этой страницы. Эта функция окна не должна вызывать функцию EndDialog.
10. **lParam** – задаваемые приложением данные.
11. **pfnCallback** – указатель на функцию повторного вызова. Если поле dwFlags не содержит значение PSP_USECALLBACK, этот член игнорируется.
12. **pcRefParent** – счетчик количества ссылок к странице. Если поле dwFlags не содержит значение PSP_USERREFPARENT, этот член игнорируется.

Алгоритм создания блокнота состоит из следующих шагов:

1. Описать шаблоны всех диалоговых панелей, которые нужно включить в блокнот.
2. Задать массив для описания страниц блокнота. Например:

```
static PROPSHEETPAGE pPage[3];
```

3. Инициализировать страницы блокнота. Например:

```
pPage[0].dwSize = sizeof(PROPSHEETPAGE);
...
pPage[0].pfnDlgProc = DlgProc0;
pPage[0].pszTitle = "Закладка0";
```

4. Описать заголовок блокнота:

```
static PROPSHEETHEADER pHeader;
```

5. Инициализировать заголовок блокнота. Например:

```
pHeader.dwSize = sizeof(PROPSHEETHEADER);
...
pHeader.pszCaption = "Заголовок блокнота";
pHeader.nPages=sizeof(pPage) / sizeof(PROPSHEETPAGE);
```

6. Создать и отобразить блокнот:

```
PropertySheet(&pHeader);
```

Задача. Объединить диалоговые панели из листингов 6.2 и 6.3 в одном модальном блокноте.

Листинг 6.4. Блокнот из двух страниц диалоговых панелей.

```
#include <windows.h>
#include <prsht.h>
#include "CreateDlg.h"

#define  ID_STATIC0      1000
#define  ID_BUTTON1      1001
#define  ID_BUTTON2      1002
#define  ID_BUTTON3      1003
#define  ID_HELP         1004
#define  ID_STATIC1      2003
#define  ID_LISTBOX      2004
#define  ID_CREATEDLG    100

BOOL        RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT     CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
LRESULT     CALLBACK WndForDlg( HWND,UINT,WPARAM,LPARAM);
LRESULT     CALLBACK DlgProc0(HWND,UINT,WPARAM,LPARAM);
LRESULT     CALLBACK DlgProc1(HWND,UINT,WPARAM,LPARAM);
WORD*       CreateDlg0(HWND);
WORD*       CreateDlg1(HWND);

HINSTANCE   hInstance;
char        szClass[ ]="WindowAppClass";
char        szForDlg[ ]="PopupWindow";
in          wScreen, hScreen;
static      HWND hForDlg;

int WINAPI  WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                    LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd; ::hInstance=hInstance;
    if  ( !RegClass(WndProc,szClass,COLOR_DESKTOP) )
```

```
                    return FALSE;
    if ( !RegClass(WndForDlg,szForDlg,0) ) return FALSE;
    wScreen=GetSystemMetrics(SM_CXSCREEN);
    hScreen=GetSystemMetrics(SM_CYSCREEN);
    hwnd = CreateWindow(szClass, "Создание диалоговой панели",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        0, 0, wScreen, hScreen, 0, 0, hInstance, NULL);
    if (!hwnd) return FALSE;
    hForDlg=CreateWindow(szForDlg, NULL, WS_POPUP,
            0, 0, 10, 10, hwnd, 0, hInstance, NULL);
    while(GetMessage(&msg, 0, 0, 0))
    {   TranslateMessage(&msg); DispatchMessage(&msg); }
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
                    UINT brBackground)
{   WNDCLASS wc;  wc.style=wc.cbClsExtra=wc.cbWndExtra=0;
    wc.lpfnWndProc=Proc; wc.hInstance = hInstance;
    wc.hIcon=LoadIcon(NULL,IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName  = (LPCTSTR)NULL;   wc.lpszClassName=szName;
    return (RegisterClass(&wc) != 0);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam,  LPARAM lParam )
{   switch (msg)
    {   case WM_LBUTTONDOWN:
        {   SendMessage(hForDlg, WM_COMMAND,
                    ID_CREATEDLG, 0L);
            return 0;
        }
            case WM_DESTROY:  {PostQuitMessage(0);  return 0; }
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}

int CALLBACK PSCallback( HWND hdlg, UINT uMsg,
                    LPARAM lParam )
{   if (uMsg==PSCB_INITIALIZED)
```

```
    {   RECT rc; GetWindowRect(hdlg, &rc);
        int l=rc.left, t=rc.top, w=rc.right-l, h=rc.bottom-t;
        MoveWindow(hForDlg, (wScreen-w)/2, (hScreen-h)/2, w, h, TRUE);
    }
    return 0;
}

LRESULT CALLBACK WndForDlg( HWND hwnd, UINT msg,
                            WPARAM wParam, LPARAM lParam )
{   switch (msg)
    {   case WM_COMMAND:
        switch (LOWORD(wParam))
        {   case ID_CREATEDLG:
            {   static PROPSHEETPAGE   pPage[2];
                static PROPSHEETHEADER pHeader;

                pPage[0].dwSize = sizeof(PROPSHEETPAGE);
                pPage[0].dwFlags =
                        PSP_USETITLE | PSP_DLGINDIRECT;
                pPage[0].pResource =
                        (DLGTEMPLATE*)CreateDlg0(hwnd);
                pPage[0].pfnDlgProc = (DLGPROC)DlgProc0;
                pPage[0].pszTitle = "Режимы";

                pPage[1].dwSize = sizeof(PROPSHEETPAGE);
                pPage[1].dwFlags = PSP_USETITLE | PSP_DLGINDIRECT;
                pPage[1].pResource = (DLGTEMPLATE*)CreateDlg1(hwnd);
                pPage[1].pfnDlgProc = (DLGPROC)DlgProc1;
                pPage[1].pszTitle = "Список";

                pHeader.dwSize =  sizeof(PROPSHEETHEADER);
                pHeader.dwFlags =
                                PSH_PROPSHEETPAGE | PSH_USECALLBACK;
                pHeader.hwndParent = hwnd;
                pHeader.pszCaption = "Изменение свойств приложения";
                pHeader.nPages = sizeof(pPage)/sizeof(PROPSHEETPAGE);
                pHeader.ppsp = (LPCPROPSHEETPAGE)pPage;
                pHeader.pfnCallback= (PFNPROPSHEETCALLBACK)PSCallback;

                PropertySheet(&pHeader);
                LocalFree(LocalHandle(pPage[0].pResource));
                LocalFree(LocalHandle(pPage[1].pResource));
```

```
                return 0;
            }
            return 0;
        }
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
WORD* CreateDlg0( HWND hwnd )
{   char const caption[ ]=" Изменение режима работы приложения";
    char const stattxt[ ]="Укажите один из режимов:";
    char const modeoff[ ]="завершить работу приложения";
    char const modedat[ ]="перезагрузить данные";
    char const modepsw[ ]="сменить имя пользователя";

    //Выделяем блок памяти для записи шаблона
    WORD *p, *pdlgtemplate;
    pdlgtemplate=p=(PWORD)LocalAlloc(LPTR,4000);

    int cxChar, cyChar;
    {   TEXTMETRIC tm;      HDC hdc=GetDC(hwnd);
        GetTextMetrics(hdc,&tm);  ReleaseDC(hwnd,hdc);
        cxChar=tm.tmAveCharWidth+1;
        cyChar=tm.tmHeight+tm.tmExternalLeading;
    }
    DWORD   dlgunit =GetDialogBaseUnits();
    int     dlgwunit=LOWORD(dlgunit), dlghunit=HIWORD(dlgunit);
    cxChar=cxChar*4/dlgwunit;cyChar=cyChar*8/dlghunit;

    int  wDlg, hDlg, wItem, hItem, left, top;
    DWORD lStyle;

    //Записываем в шаблон данные панели
    lStyle = DS_MODALFRAME | WS_POPUPWINDOW | WS_CAPTION;
    wDlg=lstrlen(caption)*cxChar;  hDlg=cyChar*8;
    DlgTemplate(p, lStyle, 7, 0, 0, wDlg, hDlg, (LPSTR)caption);

    //Далее добавляем записи для органов управления
    //1
    hItem=cyChar; top=left=hItem/2;   hItem+=left;
    wItem=(wDlg-left-left);
    lStyle = WS_CHILD | WS_VISIBLE | BS_GROUPBOX | WS_TABSTOP;
    DlgItemTemplate(p, lStyle, left, top, wItem, 4*hItem+left,
        ID_STATIC0, (LPSTR)"button",(LPSTR)stattxt);
```

```
    //2
    wItem=lstrlen(modeoff)*cxChar+10;    top+=hItem;
    lStyle = BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP;
    DlgItemTemplate(p, lStyle, hItem, top, wItem, hItem,
        ID_BUTTON1, (LPSTR)"button", (LPSTR)modeoff);
    //3
    wItem=lstrlen(modedat)*cxChar+10;    top+=hItem;
    lStyle = BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE;
    DlgItemTemplate(p, lStyle, hItem, top, wItem, hItem,
        ID_BUTTON2, "button", (LPSTR)modedat);
    //4
    wItem=lstrlen(modepsw)*cxChar+10;   top+=hItem;
    lStyle = BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE;
    DlgItemTemplate(p, lStyle, hItem, top, wItem, hItem,
        ID_BUTTON3, (LPSTR)"button", (LPSTR)modepsw);
    //5
    wItem=(wDlg-left-left-hItem-hItem)/3;
    top+=hItem+hItem/2+left;
    lStyle = BS_DEFPUSHBUTTON | WS_VISIBLE | WS_CHILD | WS_TABSTOP;
    DlgItemTemplate(p, lStyle, left, top, wItem, hItem,
        IDOK, (LPSTR)"button",(LPSTR)"Да");
    //6
    lStyle = BS_PUSHBUTTON | WS_VISIBLE | WS_CHILD | WS_TABSTOP;
    DlgItemTemplate(p, lStyle, left+wItem+hItem, top, wItem, hItem,
        IDCANCEL, (LPSTR)"button",(LPSTR)"Отмена");
    //7
    DlgItemTemplate(p, lStyle, left+wItem+hItem+wItem+hItem,
        top, wItem, hItem, ID_HELP, (LPSTR)"button",(LPSTR)"Справка");
    return pdlgtemplate;
}

WORD* CreateDlg1( HWND hwnd )
{   char const caption[ ]="Модальная диалоговая панель";
    char const stattxt[ ]="Выбранная строка  -";

    WORD  *p, *pdlgtemplate;
    pdlgtemplate=p=(PWORD)LocalAlloc(LPTR,4000);

    int cxChar, cyChar;
    {   TEXTMETRIC  tm;      HDC hdc=GetDC(hwnd);
        GetTextMetrics(hdc,&tm);  ReleaseDC(hwnd,hdc);
        cxChar=tm.tmAveCharWidth+1;
        cyChar=tm.tmHeight+tm.tmExternalLeading;
    }
```

```
    DWORD    dlgunit =GetDialogBaseUnits();
    int      dlgwunit=LOWORD(dlgunit), dlghunit=HIWORD(dlgunit);
    cxChar=cxChar*4/dlgwunit;cyChar=cyChar*8/dlghunit;

    int  wDlg, hDlg, wItem, hItem, left, top;
    DWORD lStyle;

    //Записываем в шаблон данные панели
    lStyle = DS_MODALFRAME | WS_POPUPWINDOW |
                          WS_CAPTION;
    wDlg=lstrlen(caption)*cxChar;  hDlg=cyChar*9;
    DlgTemplate(p, lStyle, 5, 0, 0, wDlg, hDlg, (LPSTR)caption);

    //Добавляем записи для органов управления
    //1
    hItem=cyChar; top=left=hItem/2;    hItem+=left;
    wItem=lstrlen(stattxt)*cxChar;
    lStyle = SS_LEFT | WS_CHILD | WS_VISIBLE;
    DlgItemTemplate(p, lStyle, left, top, wItem, hItem,
        -1, (LPSTR)"static",(LPSTR)stattxt);
    //2
    DlgItemTemplate(p, lStyle, left+wItem, top, wItem, hItem,
        ID_STATIC1, (LPSTR)"static",(LPSTR)"");
    //3
    top+=hItem;
    lStyle = WS_CHILD | WS_VISIBLE | WS_TABSTOP |
                    LBS_STANDARD |  LBS_DISABLENOSCROLL;
    DlgItemTemplate(p, lStyle, left, top, wItem, hItem*4,
            ID_LISTBOX, (LPSTR)"listbox",(LPSTR)"\0");
    //4
    wItem=wDlg-wItem-hItem;
    left+=left+lstrlen(stattxt)*cxChar;
    lStyle = BS_DEFPUSHBUTTON | WS_VISIBLE | WS_CHILD |
                          WS_TABSTOP;
    DlgItemTemplate(p, lStyle, left, top, wItem, hItem,
        IDOK, (LPSTR)"button",(LPSTR)"Готово");
    //5
    top += hItem+hItem;
    lStyle = BS_PUSHBUTTON | WS_VISIBLE | WS_CHILD | WS_TABSTOP;
    DlgItemTemplate(p, lStyle, left, top, wItem, hItem,
        IDCANCEL, (LPSTR)"button",(LPSTR)"Выход");
    return pdlgtemplate;
}
```

```
LRESULT CALLBACK DlgProc0(HWND hdlg, UINT msg,
                          WPARAM wParam, LPARAM lParam)
{   switch (msg)
    {   case WM_COMMAND:
        {   switch (LOWORD(wParam))
            {   case IDOK:
                case IDCANCEL:  {return TRUE; }
            }   return FALSE;
        }
    }   return FALSE;
}

LRESULT CALLBACK DlgProc1(HWND hdlg, UINT msg,
                          WPARAM wParam, LPARAM lParam)
{   switch (msg)
    {   case WM_INITDIALOG:
        {   HWND hListBox=GetDlgItem(hdlg,ID_LISTBOX);
            SendMessage(hListBox, WM_SETREDRAW, FALSE, 0L);
            SendMessage(hListBox, LB_ADDSTRING, 0,
                        (LPARAM)(LPSTR)"Зеленый");
            SendMessage(hListBox, LB_ADDSTRING, 0,
                        (LPARAM)(LPSTR)"Красный");
            SendMessage(hListBox, LB_ADDSTRING, 0,
                        (LPARAM)(LPSTR)"Розовый");
            SendMessage(hListBox, LB_ADDSTRING, 0,
                        (LPARAM)(LPSTR)"Пурпурный");
            SendMessage(hListBox, LB_ADDSTRING, 0,
                        (LPARAM)(LPSTR)"Синий");
            SendMessage(hListBox, LB_ADDSTRING, 0,
                        (LPARAM)(LPSTR)"Желтый");
            SendMessage(hListBox, LB_ADDSTRING, 0,
                        (LPARAM)(LPSTR)"Белый");
            SendMessage(hListBox,WM_SETREDRAW,TRUE,0L);
            InvalidateRect(hListBox, NULL, TRUE); return TRUE;
        }
        case WM_NOTIFY:
        {   LPNMHDR pn=(LPNMHDR)lParam;
            switch (pn->code)
            {   case PSN_SETACTIVE:
                {   MessageBox(hdlg, "PSN_SETACTIVE",
                        "WM_NOTIFY",MB_OK);
                    return FALSE;
                }
```

```
            case PSN_APPLY:
            {   MessageBox(hdlg,"PSN_APPLY",
                    "WM_NOTIFY",MB_OK);
                return FALSE;
            }
            case PSN_QUERYCANCEL:
            {   MessageBox(hdlg, "PSN_QUERYCANCEL",
                    "WM_NOTIFY",MB_OK);
                return FALSE;
            }
            case PSN_KILLACTIVE:
            {   MessageBox(hdlg, "PSN_KILLACTIVE",
                    "WM_NOTIFY",MB_OK);
                return FALSE;
            }
        }   return FALSE;
    }
    case WM_COMMAND:
    {   switch (LOWORD(wParam))
        {   case IDOK:
            {   int uSelectedItem;  char Buffer[256];
                HWND hListBox=GetDlgItem(hdlg, ID_LISTBOX);
                uSelectedItem=(int)SendMessage(
                    hListBox,LB_GETCURSEL,0,0L);
                if(uSelectedItem != LB_ERR)
                {   SendMessage(hListBox,LB_GETTEXT,
                        uSelectedItem,(LPARAM)Buffer);
                    SetWindowText(GetDlgItem(hdlg,
                        ID_STATIC1), (LPCTSTR)Buffer);
                }
                SendMessage(GetParent(hdlg),
                    PSM_CHANGED, (WPARAM)(HWND)hdlg, 0L);
                return TRUE;
            }
            return FALSE;
        }
        return FALSE;
    }
}   return FALSE;
}
```

Все части текста этого приложения детально рассмотрены ранее. Интерес представляют цели, ради достижения которых добавлены некоторые объекты.

1. В установках проекта добавлена библиотека comctl32.lib.
2. В текст приложения включен файл prsht.h:

```
#include <prsht.h>
```

Эти два пункта обусловлены тем, что блокнот принадлежит к объектам расширенных возможностей Win32.

3. Изменены функции создания диалоговых панелей:

```
WORD*   CreateDlg0(HWND);
WORD*   CreateDlg1(HWND);
```

Теперь они в памяти создают шаблон панели и возвращают указатель на этот блок памяти.

4. Появился глобальный дескриптор вспомогательного окна:

```
static  HWND hForDlg;
```

В теле функции этого окна с именем WndForDlg создается блокнот. Причем окно hForDlg за время работы приложения ни разу не появляется на экране.

Перечислим основные причины использования этого окна:

4.1. В теле функции окна можно хранить статически описанные локальные данные для работы с панелью. Например, уже на этапе создания окна hForDlg можно было создать шаблоны панелей и инициировать другие данные блокнота.

4.2. В функцию окна можно посылать сообщения. Здесь в функцию окна посылается единственное сообщение WM_COMMAND с параметром LOWORD(wParam)=ID_CREATEDLG от функции окна приложения.

4.3. Блокнот по умолчанию располагается в левом верхнем углу окна-владельца. Окно hForDlg уже создано, и его можно перемещать куда угодно. То есть, перемещая окно hForDlg, можно управлять расположением блокнота. Здесь с помощью такого приема блокнот отображается в центре экрана. Это происходит на этапе инициализации блокнота, в теле функции повторного вызова:

```
int CALLBACK PSCallback(HWND hdlg,UINT uMsg, LPARAM lParam)
{   if (uMsg==PSCB_INITIALIZED)
    {   RECT rc; GetWindowRect(hdlg, &rc);
        int l=rc.left, t=rc.top, w=rc.right-l, h=rc.bottom-t;
        MoveWindow(hForDlg,
            (wScreen-w)/2, (hScreen-h)/2, w, h, TRUE);
    }   return 0;
}
```

В теле этой функции определяют габариты окна hdlg блокнота и окно-владелец hForDlg перемещают так, что окно блокнота оказывается приблизительно в центре экрана. Размеры окна-владельца не имеют значения. Например, на месте w и h можно было записать 1:

```
MoveWindow(hForDlg, (wScreen-w)/2, (hScreen-h)/2, 1, 1, TRUE);
```

5. Функция окна приложения функции окна hForDlg посылает сообщение WM_COMMAND с параметром ID_CREATEDLG:

```
SendMessage(hForDlg, WM_COMMAND, ID_CREATEDLG, 0L);
```

6. Функция окна hForDlg при обработке этого сообщения создает блокнот из двух страниц. Панели для этих страниц рассмотрены выше. Отличие появилось во второй диалоговой панели (листинг 6.3). Блокнот не допускает использования главного меню, расположенного ниже заголовка панели. Это ясно, поскольку у панели на странице блокнота нет области заголовка. Вместо заголовка панели в блокноте есть только вкладыши или закладки страниц.
   Обратите внимание, что блокнот – это панель, созданная на базе шаблона в памяти. Этот шаблон в памяти создается, и указатель на него можно получить в теле функции повторного вызова.
   Есть еще одна особенность – блокнот сам выбирает размеры панелей страниц. Но это не означает, что не нужно задавать размеры панелей страниц в шаблонах.
7. Основные изменения коснулись функций окон панелей. В качестве примера была разработана функция окна второй страницы.
   Рассмотрим ее более подробно.
   7.1. На этапе инициализации страницы диалоговой панели заполняется список:

```
SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)(LPSTR)"Зеленый");
...
SendMessage(hListBox, LB_ADDSTRING, 0, (LPARAM)(LPSTR)"Белый");
```

Внимание! Инициализация выполняется только при первом раскрытии страницы.

   7.2. Обработка сообщений от органов управления мало отличается от прежней обработки.
   Например, после нажатия кнопки "Готово" из списка берется выбранная строка и помещается во второй статический орган. В этой части изменений нет.
   Но страница должна взаимодействовать с кнопками блокнота. В качестве примера после установки текста второго статического органа сообщаем блокноту о том, что произошли изменения:

```
SendMessage(GetParent(hdlg), PSM_CHANGED, (WPARAM)(HWND)hdlg, 0L);
```

В этом операторе функция GetParent возвращает дескриптор окна блокнота, т. е. дескриптор родительского окна для окна hdlg страницы. Окно блокнота получит сообщение PSM_CHANGED и, в ответ, разблокирует кнопку "Применить".

Обратите внимание, что нет вызова функции EndDialog. Если в теле функции окна страницы вызвать эту функцию, то панель для этой страницы будет пустой.

7.3. Есть еще одна разновидность взаимодействия между блокнотом и функцией окна страницы. Блокнот посылает уведомительное сообщение WM_NOTIFY функции окна раскрытой страницы. При этом параметр lParam указывает на структуру NMHDR, содержащую информацию о сообщении:

```
LPNMHDR pn=(LPNMHDR)lParam;
```

Код уведомительного сообщения зависит от произошедшего события. Например, при активизации страницы уведомительное сообщение содержит код PSN_SETACTIVE, а при потере активности код равен PSN_KILLACTIVE. При нажатии кнопок OK или "Применить" код сообщения равен PSN_APPLY, при нажатии "Отмена" код равен PSN_QUERYCANCEL. При этом после нажатия каждой кнопки поступает несколько уведомительных сообщений с различными кодами. Например, после нажатия кнопки OK сначала поступает сообщение с кодом PSN_KILLACTIVE, а потом – с кодом PSN_APPLY.

6.5. Стандартные диалоговые панели

6.5.1. Панели для открытия или сохранения файлов

В Win32 API существуют две функции для организации пользовательского интерфейса **при выборе имен для открытия или создания файлов** и **при выборе имени для сохранения файла** Это функции GetOpenFileName и GetSaveFileName.

Функция GetOpenFileName создает стандартную панель выбора имен для открытия или создания файлов. Эта панель обеспечивает просмотр дисков, каталогов и списка имен файлов и выбор одного или нескольких имен файлов из списка.

Функция GetOpenFileName объявлена следующим образом:

```
BOOL GetOpenFileName( LPOPENFILENAME lpofn );
```

Параметр lpofn указывает на структуру типа OPENFILENAME, которая содержит данные для инициализации панели. После завершения работы функции GetOpenFileName эта структура будет дополнена данными о выбранных именах файлов.

Если пользователь выбрал имя файла и нажал кнопку OK, возвращаемое значение отлично от нуля. При этом буфер, указанный полем lpstrFile структуры будет содержать полное имя выбранного файла. Иначе возвращаемое значение – нуль.

Функция GetSaveFileName создает стандартную панель выбора имени для сохранения файла. Эта панель обеспечивает просмотр дисков, каталогов и списка имен файлов и выбор одного имени файла. Эта функция объявлена следующим образом:

```
BOOL GetSaveFileName( LPOPENFILENAME lpofn );
```

Параметр lpofn указывает на структуру типа OPENFILENAME, которая содержит данные для инициализации панели. После завершения работы функции GetSaveFileName эта структура будет дополнена данными о выбранном имени файла.

Если пользователь выбрал имя файла и нажал кнопку OK, возвращаемое значение отлично от нуля. При этом буфер, указанный полем lpstrFile структуры, будет содержать полное имя выбранного файла. Иначе возвращаемое значение – нуль.

Структура OPENFILENAME описана следующим образом:

```
typedef struct
{   DWORD           lStructSize;
    HWND            hwndOwner;
    HINSTANCE       hInstance;
    LPCTSTR         lpstrFilter;
    LPTSTR          lpstrCustomFilter;
    DWORD           nMaxCustFilter;
    DWORD           nFilterIndex;
    LPTSTR          lpstrFile;
    DWORD           nMaxFile;
    LPTSTR          lpstrFileTitle;
    DWORD           nMaxFileTitle;
    LPCTSTR         lpstrInitialDir;
    LPCTSTR         lpstrTitle;
    DWORD           Flags;
    WORD            nFileOffset;
    WORD            nFileExtension;
    LPCTSTR         lpstrDefExt;
    DWORD           lCustData;
    LPOFNHOOKPROC   lpfnHook;
    LPCTSTR         lpTemplateName;
}   OPENFILENAME;
```

Назначение полей структуры OPENFILENAME:

1. **lStructSize** содержит размер структуры OPENFILENAME в байтах.
2. **hwndOwner** – дескриптор окна – владельца панели.
3. **hInstance** указывает на блок памяти с шаблоном панели. Это поле игнорируется, если не указан флаг OFN_ENABLETEMPLATE или OFN_ ENABLETEMPLATEHANDLE.

Если Flags содержит значение OFN_EXPLORER, используется предопределенный шаблон панели в стиле Explorer. Иначе создается панель старого стиля.

4. **lpstrFilter** содержит адрес текстовой строки, задающей фильтр имен файлов. Фильтр состоит из одной или нескольких пар текстовых строк. Например, "Текстовые файлы\0*.txt\0". Символ '\0' означает конец строки. Первая строка пары (здесь "Текстовые файлы") поясняет назначение фильтра, а вторая (здесь "*.txt") задает образец имен файлов. В одной паре можно задать различные образцы, разделяя их точкой с запятой (например, "*.txt; *.doc; *.bak"). Если lpstrFilter=NULL, используется фильтр lpstrCustomFilter.
5. **lpstrCustomFilter** указывает на статический буфер, в котором будет храниться примененный пользователем фильтр. Этот буфер должен быть длиной не менее 40 символов.
6. **nMaxCustFilter** содержит размер буфера lpstrCustomFilter.
7. **nFilterIndex** задает номер пары строк в поле lpstrFilter для использования в качестве фильтра. После выбора имени файла в поле nFilterIndex сохраняется номер использованного фильтра.
8. **lpstrFile** содержит адрес строки, в которую будет записано полное имя выбранного файла. Если выбран список имен файлов, эта строка будет содержать путь, завершенный символом '\0', затем следуют имена файлов, завершенные символом '\0'. Последнее в списке имя завершается двумя символами '\0'.
9. **nMaxFile** содержит размер в байтах буфера, расположенного по адресу, указанному в поле lpstrFile.
10. **lpstrFileTitle** содержит адрес буфера, в который после выбора будет записано имя файла без указания пути. Это поле может быть использовано для отображения имени выбранного файла.
11. **nMaxFileTitle** содержит размер указанного выше буфера. Этот член игнорируется, если lpstrFileTitle=NULL.
12. **lpstrInitialDir** указывает на строку, содержащую начальный путь поиска имен файлов.

13. **lpstrTitle** указывает на строку, содержащую нестандартный заголовок панели.
14. **nFileOffset** будет содержать смещение первого символа имени файла от начала буфера lpstrFile. Например, если lpstrFile указывает на строку "c:\dir1\dir2\file.ext", то поле nFileOffset содержит смещение имени file.ext от начала строки, равное 13.
15. **nFileExtension** будет содержать смещение первого символа расширения имени файла от начала буфера lpstrFile. Например, если lpstrFile указывает на строку "c:\dir1\dir2\file.ext", то это поле содержит смещение расширения ext от начала строки, равное 18.
16. **lpstrDefExt** указывает на буфер, который содержит расширение имени файла по умолчанию. Это расширение добавляется к выбранному имени файла, если расширение не было указано.
17. Значение **lCustData** передается функции фильтра через параметр lParam.
18. **Flags** задает режимы выбора и внешний вид панели. Следующая таблица содержит список часто применяемых флагов, комбинации которых используют для заполнения этого поля:

Флаг	*Пояснение*
OFN_ALLOWMULTISELECT	Можно выбрать список имен файлов
OFN_CREATEPROMPT	При выборе имени не существующего файла панель запросит, нужно ли создать файл
OFN_ENABLEHOOK	Использовать функцию фильтра, указанную в поле lpfnHook
OFN_EXPLORER	Этот флаг указывают только с флагом OFN_ENABLEHOOK или OFN_ALLOWMULTISELECT
OFN_ EXTENSIONDIFFERENT	Проверить ситуации, когда расширение заданного имени файла отличается от расширения, указанного в lpstrDefExt
OFN_FILEMUSTEXIST	Предупреждать при вводе имени несуществующего файла. Используется с флагом OFN_PATHMUSTEXIST
OFN_HIDEREADONLY	Скрыть кнопку "Только чтение"
OFN_NOCHANGEDIR	Поиск файлов начинать в первоначально заданном каталоге

Флаг	Пояснение
OFN_NONETWORKBUTTON	Скрыть и отключить кнопку Network
OFN_ NOREADONLYRETURN	Запретить выбор файла с атрибутом "только чтение" и в защищенном от записи каталоге
OFN_OVERWRITEPROMPT	При сохранении файла предупреждать, если файл с указанным именем уже существует
OFN_PATHMUSTEXIST	Можно вводить только существующее полное имя файла
OFN_READONLY	Включить кнопку "Только чтение"
OFN_SHAREAWARE	Игнорировать факт совместного доступа к файлу по сети
OFN_SHOWHELP	Показать кнопку "Справка"

19. **lpfhHook** указывает на функцию фильтра, обрабатывающую сообщения для панели.
20. **lpTemplatename** идентифицирует ресурс, содержащий шаблон панели, используемый вместо стандартного шаблона. Для применения альтернативного шаблона в поле Flags следует установить флаг OFN_ ENABLETEMPLATE.

Задача. Главное меню приложения содержит раздел "Файл" со строками "Открыть", "Сохранить" и "Выход". При выборе строки "Открыть" создать стандартную панель для выбора списка имен для открытия или создания файлов. Отобразить этот список. При выборе строки "Сохранить" создать стандартную панель для выбора имени для сохранения файла. Отобразить это имя. Панели должны работать с одними и теми же допустимыми начальными установками.

Листинг 6.5. Стандартные панели для работы с файлами.

```
#include <windows.h>
#define CM_FILE_OPEN   1001
#define CM_FILE_SAVE   1002
#define CM_FILE_QUIT   1003

BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char        szClass[ ]  ="OpenSaveFile";
char        szTitle[ ]  ="Открытие и закрытие файла";
```

```
int WINAPI  WinMain(HINSTANCE hInstance,
        HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd; ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass, COLOR_WINDOW) )
        return FALSE;
    if  ( !( hwnd = CreateWindow(szClass, szTitle,
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL ) ) )  return FALSE;
    while ( GetMessage(&msg, 0, 0, 0) )  DispatchMessage(&msg);
    return msg.wParam;
}
BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc; wc.style=wc.cbClsExtra=wc.cbWndExtra=0;
    wc.lpfnWndProc = Proc;     wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName=(LPCTSTR)NULL;
    wc.lpszClassName=szName;   return (RegisterClass(&wc) != 0);
}
BOOL CreateMenuItem(HMENU hMenu, char *str, UINT uIns,
UINT uCom, HMENU hSubMenu, BOOL flag, UINT fType)
{   static MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE | MIIM_TYPE | MIIM_SUBMENU | MIIM_ID;
    mii.fType = fType;  mii.fState= MFS_ENABLED;
    mii.dwTypeData = str;  mii.cch=sizeof(str);
    mii.wID=uCom;      mii.hSubMenu=hSubMenu;
    return InsertMenuItem(hMenu, uIns, flag, &mii);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam )
{   static   HMENU    hMainMenu, hFileMenu;
    static   OPENFILENAME   ofn;
    static   char    szFile[256];
    static   char    szFileTitle[256];
    static   char    CustomFilter[256];
    switch (msg)
    {   case WM_CREATE:
```

```
{   hMainMenu=CreateMenu();
    hFileMenu=CreatePopupMenu();   int i=0;
    CreateMenuItem( hFileMenu, "&Открыть", i++,
        CM_FILE_OPEN, NULL, FALSE, MFT_STRING);
    CreateMenuItem( hFileMenu, "&Сохранить", i++,
        CM_FILE_SAVE, NULL, FALSE, MFT_STRING);
    CreateMenuItem( hFileMenu, NULL, i++,
        0, NULL, FALSE, MFT_SEPARATOR);
    CreateMenuItem( hFileMenu, "&Выход", i++,
        CM_FILE_QUIT, NULL, FALSE, MFT_STRING);
    i=0;
    CreateMenuItem(hMainMenu,"&Файл",i++, 0,
        hFileMenu,FALSE, MFT_STRING);
    SetMenu(hwnd,hMainMenu); DrawMenuBar(hwnd);

    ofn.lStructSize       = sizeof(OPENFILENAME);
    ofn.hwndOwner         = hwnd;
    ofn.nFilterIndex      = 1;
    ofn.lpstrFile         = szFile;
    ofn.nMaxFile          = sizeof(szFile);
    ofn.lpstrFileTitle    = szFileTitle;
    ofn.nMaxFileTitle     = sizeof(szFileTitle);
    ofn.lpstrCustomFilter =CustomFilter;
    return 0;
}
case WM_COMMAND:
{   switch (LOWORD(wParam))
    {   case CM_FILE_OPEN:
        {   ofn.Flags = OFN_EXPLORER |
                OFN_CREATEPROMPT |
                OFN_ALLOWMULTISELECT;
            ofn.lpstrTitle=
        "Панель выбора имени открываемого файла";
            char szFilter[256] =
                "Файлы C++ и C\0*.cpp;*.c\0"
                "Заголовочные файлы\0*.h;*.hpp\0"
                "Все файлы\0*.*\0";
            ofn.lpstrFilter=szFilter;
            szFileTitle[0]='\0';
            szFile[0]='\0';
            if ( GetOpenFileName(&ofn) )
```

```
            {   char str[512];
                strcpy(str,"Список имен файлов:\t");
                for (int i=0; i<255; i++)
            {   if (szFile[i]=='\0' &&
                szFile[i+1]=='\0') break;
                    if (szFile[i]=='\0') szFile[i]='\n';
                }
                strcat(str,szFile);
                strcat(str, "\n\nИмя файла без указания пути:\t");
                strcat(str,szFileTitle);
                MessageBox(hwnd,str,
                    "Результаты выбора имени открываемого файла",
                        MB_OK);
                }
                return 0;
            }
            case CM_FILE_SAVE:
            {   ofn.Flags = OFN_OVERWRITEPROMPT;
                ofn.lpstrTitle=
                    "Панель выбора имени для сохранения файла";
                char szFilter[256] = "Все файлы\0*.*\0";
                ofn.lpstrFilter=szFilter;
                szFile[0]='\0';
                if (GetSaveFileName(&ofn))
                {   char str[512];
                    strcpy(str,"Полное имя файла:\t");
                    strcat(str,szFile);
                    strcat(str, "\n\nИмя файла без указания пути:\t");
                    strcat(str,szFileTitle);
                    MessageBox(hwnd,str,
                        "Результаты выбора имени для сохранения файла",
                        MB_OK);
                }
                return 0;
            }
            case CM_FILE_QUIT:
            {   DestroyWindow(hwnd); return 0;   }
        }
        return 0;
    }
    case WM_DESTROY: { PostQuitMessage(0); return 0; }
```

```
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Задача создания панелей полностью решена в теле функции окна приложения. Рассмотрим основные шаги.

1. Описана структура типа OPENFILENAME:

```
static  OPENFILENAME  ofn;
```

2. Описаны строки для работы со структурой ofn:

```
static  char   szFile[256];
static  char   szFileTitle[256];
static  char   CustomFilter[256];
```

Они предназначены для работы с полями структуры соответственно с именами lpstrFile, lpstrFileTitle и lpstrCustomFilter.

3. При обработке сообщения WM_CREATE инициализированы те поля структуры ofn, которые не будут изменены в процессе работы с панелями:

```
ofn.lStructSize       = sizeof(OPENFILENAME);
ofn.hwndOwner         = hwnd;
ofn.nFilterIndex      = 1;
ofn.lpstrFile         = szFile;
ofn.nMaxFile          = sizeof(szFile);
ofn.lpstrFileTitle    = szFileTitle;
ofn.nMaxFileTitle     = sizeof(szFileTitle);
ofn.lpstrCustomFilter =CustomFilter;
```

4. При выборе строки меню "Открыть" создается панель для выбора списка имен для открытия файлов. С этой целью выполняют следующие действия:

4.1. Задают значение поля Flags:

```
ofn.Flags = OFN_EXPLORER | OFN_CREATEPROMPT |
                           OFN_ALLOWMULTISELECT;
```

Ясно, что набор флажков для открытия и сохранения файлов должен быть различным.

4.2. Задают текст заголовка панели:

```
ofn.lpstrTitle="Панель выбора имени открываемого файла";
```

Если не указывать этот текст, система в заголовке использует стандартный текст "Открытие файла". По правилам хорошего тона, в начале каждого шага пользователя нужно вводить пояснение последствия шага,

который он сейчас сделает. Например, текст заголовка мог бы подсказывать пользователю, что и для чего он это делает.

4.3. Задают фильтры для списка имен файлов:

```
char szFilter[256] = "Файлы C++ и C\0*.cpp;*.c\0"
        "Заголовочные файлы\0*.h;*.hpp\0"
        "Все файлы\0*.*\0";
ofn.lpstrFilter=szFilter;
```

4.4. Освобождают вспомогательные буферы от последствий предыдущего создания панели:

```
szFileTitle[0]='\0';  szFile[0]='\0';
```

4.5. Создают панель:

```
GetOpenFileName(&ofn)
```

4.6. Если работа с панелью закончилась выбором списка имен файлов, то формируют сообщение о списке имен:

```
char str[512];   strcpy(str,"Список имен файлов:\t");
for (int i=0; i<255; i++)
{   if (szFile[i]=='\0' && szFile[i+1]=='\0') break;
    if (szFile[i]=='\0') szFile[i]='\n';
}
strcat(str,szFile);   strcat(str,"\n\nИмя файла без указания пути:\t");
strcat(str,szFileTitle);
MessageBox(hwnd,str,
    "Результаты выбора имени открываемого файла", MB_OK);
```

Поясним смысл цикла:

```
for (int i=0; i<255; i++)
{   if (szFile[i]=='\0' && szFile[i+1]=='\0') break;
    if (szFile[i]=='\0') szFile[i]='\n';
}
```

Если пользователь выбрал список имен файлов, то массив символов szFile содержит путь, завершенный символом '\0', затем следуют имена файлов, завершенные символом '\0'. Последнее в списке имя завершается двумя символами '\0'.

Поэтому анализируют символы этого массива и прерывают анализ, если подряд следуют два символа '\0':

```
if (szFile[i]=='\0' && szFile[i+1]=='\0') break;
```

Иначе символ '\0' заменяют символом '\n':

```
if (szFile[i]=='\0') szFile[i]='\n';
```

Последняя замена понадобилась потому, что функции отображают содержимое строк только до символа '\0'.

5. При выборе строки меню "Сохранить" создается панель для выбора имени для сохранения файла.

Внимание! В этом приложении не производится непосредственная работа с файлами. Вся работа панелей заключается в выборе имен файлов. ***Работу с файлами следует начинать, только если есть гарантии того, что не будет случайных потерь файлов.***

6.5.2. Панель для выбора цветов

Для создания стандартной панели выбора цветов вызывают функцию ChooseColor:

```
BOOL ChooseColor( LPCHOOSECOLOR lpcc );
```

Параметр lpcc указывает на структуру типа CHOOSECOLOR, которая содержит данные инициализации панели. После выбора цвета в панели эта структура дополняется данными о выбранных цветах.

Если пользователь нажал кнопку OK панели, возвращаемое значение отлично от нуля, а поле rgbResult структуры будет содержать RGB-значение выбранного цвета.

В случае отмены выбора или при ошибке возвращаемое значение равно нулю.

Эта панель не работает с палитрой цветов.

Структура CHOOSECOLOR описана следующим образом:

```
typedef struct
{   DWORD           lStructSize;
    HWND            hwndOwner;
    HWND            hInstance;
    COLORREF        rgbResult;
    COLORREF*       lpCustColors;
    DWORD           Flags;
    LPARAM          lCustData;
    LPCCHOOKPROC    lpfnHook;
    LPCTSTR         lpTemplateName;
}   CHOOSECOLOR;
```

Поля структуры CHOOSECOLOR имеют следующее назначение:

1. **lStructSize** – длина структуры CHOOSECOLOR в байтах.
2. **hwndOwner** – дескриптор окна – владельца панели.
3. **hInstance** – дескриптор блока памяти с шаблоном панели (если установлен флаг CC_ENABLETEMPLATEHANDLE) или дескриптор эк-

земпляра приложения, содержащего шаблон панели (если установлен флаг CC_ENABLETEMPLATE). Иначе значение этого поля игнорируется.

4. **rgbResult** содержит значение начальной установки цвета (если установлен флаг CC_RGBINIT). Иначе значения rgbResult и начальной установки цвета нулевые. После нажатия кнопки OK на панели значение rgbResult будет заменено значением выбранного цвета.
5. **lpCustColors** – массив для набора из 16 дополнительных значений цвета, в который можно записать различные значения цвета, нажимая кнопку "Добавить в набор" на панели.
6. Поле **Flags** служит для инициализации панели. Его значение обычно строится в виде комбинации следующих флажков:

Флаг	*Значение*
CC_ANYCOLOR	В наборе базисных цветов отображать все доступные цвета
CC_FULLOPEN	При активизации панели отображать дополнительные органы управления. Иначе для отображения этих органов нужно нажать кнопку "Определить цвет"
CC_PREVENTFULLOPEN	Отключить кнопку "Определить цвет"
CC_RGBINIT	По умолчанию использовать значение цвета, указанное в поле rgbResult
CC_SOLIDCOLOR	В наборе базисных цветов отображать только сплошные цвета

7. **lCustData** задает данные для функции, идентифицированной полем lpfnHook.
8. **lpfnHook** указывает на функцию обработки прерываний, которая может обрабатывать сообщения для панели выбора цвета.
9. **lpTemplateName** указывает на строку с именем ресурса шаблона панели в экземпляре приложения, дескриптор которого задан полем hInstance.

Задача. Главное меню приложения содержит раздел "Цвет" со строками "Задать цвет" и "Выход". Верхнюю половину рабочей области занимает прямоугольник черного цвета, а нижнюю половину – 16 прямоугольников различных цветов спектра, начиная от красного, плавно переходя к зеленому и заканчивая синим цветом.

При выборе строки "Задать цвет" создать стандартную панель для выбора цветов. Выбрать набор 16 различных цветов для закрашивания

прямоугольников в нижней половине рабочей области. Для закрашивания верхнего прямоугольника использовать тот цвет, который был добавлен в набор последним.

В случае отмены выбора цвета набор 16 цветов оставить прежним, а верхний прямоугольник закрасить белым.

Перо для границ всех прямоугольников толщиной 5 пикселей. Составляющие цвета пера всегда вдвое меньше составляющих цвета закрашивания верхнего прямоугольника.

Листинг 6.6. Эксперименты с панелью выбора цветов.

```
#include <windows.h>

#define CM_GET_COLOR 1001
#define CM_FILE_QUIT    1003

BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
HINSTANCE   hInstance;
char        szClass[ ]="GetColors";
char        szTitle[ ]="Выбор цвета";

int WINAPI  WinMain(HINSTANCE hInstance,
            HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd; ::hInstance=hInstance;
    if  ( !RegClass(WndProc, szClass, COLOR_WINDOW) )
        return FALSE;
    if  ( !( hwnd = CreateWindow(szClass, szTitle,
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL ) ) )  return FALSE;
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName, UINT brBackground)
{   WNDCLASS wc; wc.style=CS_HREDRAW | CS_VREDRAW;
    wc.cbClsExtra=wc.cbWndExtra=0; wc.lpfnWndProc=Proc;
    wc.hInstance = hInstance; wc.lpszClassName=szName;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName=(LPCTSTR)NULL;
```

```
    return (RegisterClass(&wc) != 0);
}

BOOL CreateMenuItem(HMENU hMenu, char *str, UINT uIns,
            UINT uCom, HMENU hSubMenu, BOOL flag, UINT fType)
{   static MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE | MIIM_TYPE | MIIM_SUBMENU | MIIM_ID;
    mii.fType = fType;  mii.fState= MFS_ENABLED;
    mii.dwTypeData = str;  mii.cch=sizeof(str);
    mii.wID=uCom;      mii.hSubMenu=hSubMenu;
        return InsertMenuItem(hMenu, uIns, flag, &mii);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                    WPARAM wParam, LPARAM lParam )
{   static   HMENU    hMainMenu, hColorMenu;
    static   CHOOSECOLOR cc;
    static   COLORREF    clf, clfCust[16];
    static   short       cx, cy;
    switch (msg)
    {   case WM_SIZE:
        {   cx=LOWORD(lParam); cy=HIWORD(lParam);
            return 0;
        }
        case WM_CREATE:
        {   hMainMenu=CreateMenu();
            hColorMenu=CreatePopupMenu(); int i=0;
            CreateMenuItem( hColorMenu, "&Задать цвет", i++,
                CM_GET_COLOR, NULL, FALSE, MFT_STRING);
            CreateMenuItem( hColorMenu, NULL, i++,
                0, NULL, FALSE, MFT_SEPARATOR);
         CreateMenuItem( hColorMenu, "&Выход", i++,
                CM_FILE_QUIT, NULL, FALSE, MFT_STRING);
            i=0;
            CreateMenuItem(hMainMenu,"&Цвет",i++, 0,
                hColorMenu,FALSE, MFT_STRING);
            SetMenu(hwnd,hMainMenu); DrawMenuBar(hwnd);

            for (i=0; i<16; i++)
                clfCust[i]=RGB(255-i*15, 50+48*i-3*i*i, i*15);
            cc.lStructSize      = sizeof(CHOOSECOLOR);
            cc.lpCustColors     = clfCust;
```

```
        return 0;
    }
    case WM_PAINT:
    {   PAINTSTRUCT ps;
        HDC hdc = BeginPaint(hwnd, &ps);
        HBRUSH  hbrush  = CreateSolidBrush(clf);
        HBRUSH  hOldBrush=
                (   HBRUSH)SelectObject(hdc, hbrush);
        HPEN    hpen    = CreatePen(PS_SOLID, 5, clf/2);
        HPEN    hOldPen = (HPEN)SelectObject(hdc, hpen);
        short   w=cx/16, h=cy/2;
        Rectangle(hdc, 0, 0, cx, h);
        for (int i=0; i<16; i++)
        {   hbrush = CreateSolidBrush(clfCust[i]);
            SelectObject(hdc, hbrush);
            Rectangle(hdc, w*i, h, w*i+w, cy);
            SelectObject(hdc, hOldBrush);
            DeleteObject(hbrush);
        }
        SelectObject(hdc, hOldPen);   DeleteObject(hpen);
        EndPaint(hwnd, &ps);          return 0;
    }
    case WM_COMMAND:
    {   switch (LOWORD(wParam))
        {   case CM_GET_COLOR:
            {   if (!ChooseColor(&cc)) clf=RGB(255,255,255);
                else clf = cc.rgbResult;
                InvalidateRect(hwnd, NULL, TRUE);       return 0;
            }
            case CM_FILE_QUIT:
            {   DestroyWindow(hwnd); return 0;   }
        }   return 0;
    }
    case WM_DESTROY: { PostQuitMessage(0); return 0; }
}   return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Задача создания панели полностью решена в теле функции окна приложения. Рассмотрим основные шаги.

1. Описана структура типа CHOOSECOLOR:

```
static  CHOOSECOLOR cc;
```

2. Описаны переменные для работы со структурой cc:

```
static   COLORREF    clf, clfCust[16];
```

Они предназначены для работы с полями структуры соответственно с именами rgbResult и lpCustColors. Причем переменная clf будет всегда использована для закрашивания верхнего прямоугольника и выбора цвета пера. А массив clfCust будет использован для закрашивания 16 прямоугольников в нижней половине рабочей области.

3. При обработке сообщения WM_CREATE инициализированы те поля структуры cc, которые не будут изменены в процессе работы с панелями, и поле lpCustColors:

```
cc.lStructSize   = sizeof(CHOOSECOLOR);
cc.lpCustColors     = clfCust;
for (i=0; i<16; i++)  clfCust[i]=RGB(255-i*15, 50+48*i-3*i*i, i*15);
```

Последний цикл задан для записи в массив clfCust 16 приближенно вычисленных значений цветов спектра.

4. При обработке сообщения WM_PAINT установленные значения цветов переменных clf, clfCust используются для создания кистей:

```
hbrush = CreateSolidBrush(clf);
...
hbrush = CreateSolidBrush(clfCust[i]);
```

Цвет пера устанавливают таким образом, что все его составляющие вдвое меньше составляющих цвета clf:

```
hpen     = CreatePen(PS_SOLID, 5, clf/2);
```

5. При выборе строки меню "Задать цвет" создается панель для выбора набора цветов:

```
if  ( !ChooseColor(&cc) ) clf=RGB(255,255,255);
else clf = cc.rgbResult;
```

Если работа с панелью завершилась отменой выбора, то переменной clf присваивают значение белого цвета. Иначе эта переменная принимает значение поля rgbResult. В это поле панель записывает то значение цвета, которое было выбрано последним, независимо от того, какой элемент набора дополнительных цветов был выбран последним. В любом случае посылают сообщение о необходимости перерисовки всей рабочей области:

```
InvalidateRect(hwnd, NULL, TRUE);
```

6.5.3. Панель для выбора шрифта

Для выбора шрифта вызывают функцию ChooseFont. Эта функция создает стандартную панель выбора атрибутов логического шрифта. Ат-

рибуты определяют имя шрифта, стиль (полужирный, наклонный или обычный), размер, вид (подчеркнутый, зачеркнутый или цветной) и набор символов. В процессе выбора, еще до нажатия кнопки OK, пользователь в поле образца может увидеть результаты своего выбора.

Функция ChooseFont объявлена следующим образом:

```
BOOL ChooseFont( LPCHOOSEFONT lpcf );
```

Параметр lpcf указывает на структуру типа CHOOSEFONT, которая содержит данные инициализации панели. После нажатия кнопки OK эта структура будет дополнена данными о выбранном шрифте.

Иначе возвращаемое значение равно нулю.

Структура CHOOSEFONT описана следующим образом:

```
typedef struct
{   DWORD           lStructSize;
    HWND            hwndOwner;
    HDC             hDC;
    LPLOGFONT       lpLogFont;
    INT iPointSize;
    DWORD           Flags;
    DWORD           rgbColors;
    LPARAM          lCustData;
    LPCFHOOKPROC    lpfnHook;
    LPCTSTR         lpTemplateName;
    HINSTANCE       hInstance;
    LPTSTR          lpszStyle;
    WORD            nFontType;
    WORD            ___MISSING_ALIGNMENT__;
    INT nSizeMin;
    INT nSizeMax;
}   CHOOSEFONT;
```

Рассмотрим назначение полей структуры CHOOSEFONT:

1. **lStructSize** задает длину структуры CHOOSEFONT в байтах.
2. **hwndOwner** – дескриптор окна – владельца панели.
3. **Flags** – флаги инициализации панели. Можно использовать комбинацию значений из нижеследующей таблицы.

Флаг	*Пояснение*
CF_BOTH	В списке шрифтов перечислить экранные и принтерные шрифты. Поле hDC должно указывать на принтер
CF_TTONLY	В списке шрифтов перечислить лишь масштабируемые шрифты True Type

CF_EFFECTS	Можно выбирать цвет букв и подчеркнутые и перечеркнутые шрифты
CF_FIXEDPITCHONLY	В списке шрифтов перечислить лишь шрифты с фиксированной шириной
CF_FORCEFONTEXIST	Предупреждать при попытке выбора несуществующего шрифта
CF_ INITTOLOGFONTSTRUCT	Значение поля lpLogFont использовать при инициализации панели
CF_LIMITSIZE	Размеры шрифта ограничить значениями полей nSizeMin и nSizeMax
CF_NOOEMFONTS или CF_NOVECTORFONTS	Запретить выбор векторных шрифтов
CF_NOFACESEL	Игнорировать выбор в списке имен шрифтов
CF_NOSTYLESEL	Игнорировать выбор стиля шрифта
CF_NOSIZESEL	Игнорировать выбор размера шрифта
CF_NOSIMULATIONS	Запрещается эмуляция шрифтов
CF_NOVERTFONTS	В списке шрифтов перечислить лишь горизонтально ориентированные шрифты
CF_PRINTERFONTS	В списке шрифтов перечислить только шрифты принтера, контекст отображения которого задан в поле hDC
CF_SCALABLEONLY	В списке шрифтов перечислить лишь масштабируемые и векторные шрифты
CF_SCREENFONTS	В списке шрифтов перечислить лишь экранные шрифты системы
CF_USESTYLE	Строка "lpszStyle" указывает на буфер, который содержит описание стиля. Эта строка используется для инициализации списка стилей панели
CF_WYSIWYG	Можно выбирать шрифты, доступные и для отображения на экране, и для печати на принтере. Следует установить флаги CF_BOTH и CF_SCALABLEONLY

4. **hDC** – дескриптор контекста устройства, чьи шрифты будут перечислены в панели. Этот дескриптор указывают, только если задан флаг CF_BOTH или CF_PRINTERFONTS.
5. **lpLogFont** указывает на структуру типа LOGFONT. Если установлен флаг CF_INITTOLOGFONTSTRUCT и инициализирована эта структура, то параметры структуры будут использованы при установке начального состояния панели. После нажатия кнопки OK в эту структуру будут записаны данные выбранного шрифта.
6. **iPointSize** будет содержать размер букв выбранного шрифта в десятых долях пункта.
7. **rgbColors** – цвет символов шрифта, который будет выбран в комбинированном списке "Цвет текста" сразу после отображения панели. Должен быть установлен флаг CF_EFFECTS. После выбора шрифта это поле содержит значение выбранного цвета.
8. **lCustData** – данные, передаваемые функции фильтра, определенной содержимым поля lpfnHook.
9. **lpfnHook** – указатель на функцию фильтра, которая может обрабатывать сообщения, предназначенные для панели. Этот член игнорируется, если не установлен флаг CF_ENABLEHOOK.
10. **lpTemplateName** – указатель на строку с именем ресурса шаблона панели в модуле, идентифицированном полем hInstance.
11. **hInstance** – дескриптор блока памяти с шаблоном панели (если установлен флаг CF_ENABLETEMPLATEHANDLE) или дескриптор модуля, содержащего ресурс шаблона панели (если установлен флаг CF_ENABLETEMPLATE). Иначе это поле игнорируется.
12. **lpszStyle** – указатель на буфер, который содержит данные стиля. Если установлен флаг CF_USESTYLE, данные этого буфера инициализируют список стилей шрифта на панели. После выбора шрифта в этот буфер записывается выбранный стиль.
13. **nFontType** определяет тип выбранного шрифта. Может содержать комбинацию следующих констант:

Константа	*Пояснение к шрифту*
BOLD_FONTTYPE	Полужирный. Эта информация дублирована полем lfWeight
ITALIC_FONTTYPE	Наклонный. Эта информация дублирована полем lfItalic
PRINTER_FONTTYPE	Встроенный шрифт принтера

REGULAR_FONTTYPE	Обычная жирность. Эта информация дублирована полем lfWeight
SCREEN_FONTTYPE	Шрифт – экранный шрифт
SIMULATED_FONTTYPE	Шрифт моделирует GDI

14. **nSizeMin** – минимальный размер шрифта, который можно выбрать. Для использования этого поля необходимо установить флаг CF_LIMITSIZE.
15. **nSizeMax** – максимальный размер шрифта, который можно выбрать. Для использования этого поля необходимо установить флаг CF_LIMITSIZE.

Задача. Главное меню приложения содержит раздел "Шрифт" со строками "Задать шрифт" и "Выход". В центре рабочей области вывести текст "Пробный вывод" шрифтом по умолчанию в режиме прозрачного фона.

При выборе строки "Задать шрифт" создать панель для выбора шрифта. Выбранным шрифтом вывести текст "Пробный вывод".

В случае отмены выбора шрифт оставить прежним, фон сделать непрозрачным по умолчанию.

Листинг 6.7. Панель для выбора шрифта.

```
#include <windows.h>

#define CM_GET_FONT   1001
#define CM_FILE_QUIT   1003

BOOL  RegClass(WNDPROC, LPCTSTR, UINT);
LRESULT  CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);

HINSTANCE   hInstance;
char    szClass[ ]="GetFont";
char    szTitle[ ]="Выбор шрифта";

int WINAPI  WinMain(HINSTANCE hInstance,
        HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{   MSG  msg;  HWND hwnd; ::hInstance=hInstance;
    if ( !RegClass(WndProc, szClass, COLOR_WINDOW) )
        return FALSE;
    if  ( !( hwnd = CreateWindow(szClass, szTitle,
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        0, 0, hInstance, NULL ) ) )  return FALSE;
```

```
    while(GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg);
    return msg.wParam;
}

BOOL RegClass(WNDPROC Proc, LPCTSTR szName,
                    UINT brBackground)
{   WNDCLASS wc; wc.style=CS_HREDRAW | CS_VREDRAW;
    wc.cbClsExtra=wc.cbWndExtra=0; wc.lpfnWndProc=Proc;
    wc.hInstance = hInstance; wc.lpszClassName=szName;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(brBackground + 1);
    wc.lpszMenuName=(LPCTSTR)NULL;
    return (RegisterClass(&wc) != 0);
}

BOOL CreateMenuItem(HMENU hMenu, char *str, UINT uIns,
                UINT uCom, HMENU hSubMenu, BOOL flag, UINT fType)
{   static MENUITEMINFO mii; mii.cbSize=sizeof(MENUITEMINFO);
    mii.fMask = MIIM_STATE | MIIM_TYPE | MIIM_SUBMENU | MIIM_ID;
    mii.fType = fType;          mii.fState= MFS_ENABLED;
    mii.dwTypeData = str;       mii.cch=sizeof(str);
    mii.wID=uCom;               mii.hSubMenu=hSubMenu;
return InsertMenuItem(hMenu, uIns, flag, &mii);
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT msg,
                        WPARAM wParam, LPARAM lParam )
{   static  HMENU   hMainMenu, hFontMenu;
    static  short   cx, cy, bkMode;
    static  CHOOSEFONT  cf;
    static  LOGFONT lf;
    static  char    szFontStyle[LF_FACESIZE];
    switch (msg)
    {   case WM_SIZE:
        {   cx=LOWORD(lParam); cy=HIWORD(lParam);
            return 0;
        }
        case WM_CREATE:
        {   hMainMenu=CreateMenu();
            hFontMenu=CreatePopupMenu();  int i=0;
            CreateMenuItem( hFontMenu, "&Задать шрифт", i++,
```

```
        CM_GET_FONT, NULL, FALSE, MFT_STRING );
    CreateMenuItem( hFontMenu, NULL, i++,
        0, NULL, FALSE, MFT_SEPARATOR);
    CreateMenuItem( hFontMenu, "&Выход", i++,
        CM_FILE_QUIT, NULL, FALSE, MFT_STRING );
    i=0;
    CreateMenuItem(hMainMenu,"&Шрифт",i++, 0,
        hFontMenu,FALSE, MFT_STRING);
    SetMenu(hwnd,hMainMenu); DrawMenuBar(hwnd);

    cf.lStructSize = sizeof(CHOOSEFONT);
    cf.lpLogFont = &lf;
    cf.Flags =  CF_SCREENFONTS | CF_USESTYLE |
        CF_EFFECTS | CF_INITTOLOGFONTSTRUCT;
    cf.lpszStyle = (LPSTR)szFontStyle;
    bkMode=TRANSPARENT;
    return 0;
}
case WM_PAINT:
{   PAINTSTRUCT ps; char szBuf[ ]="Пробный вывод";
    HDC hdc = BeginPaint(hwnd, &ps);
    HFONT hFont     = CreateFontIndirect(&lf);
    HFONT hOldFont=(HFONT)SelectObject(hdc, hFont);
    SetTextColor(hdc, cf.rgbColors);
    SetTextAlign( hdc, TA_CENTER );
    SetBkMode(hdc, bkMode);
    TextOut(hdc, cx/2, cy/2, szBuf, lstrlen(szBuf));
    SelectObject(hdc, hOldFont); DeleteObject(hFont);
    EndPaint(hwnd, &ps);        return 0;
}
case WM_COMMAND:
{   switch (LOWORD(wParam))
    {   case CM_GET_FONT:
        {   if (ChooseFont(&cf)) bkMode=TRANSPARENT;
            else    bkMode=OPAQUE;
            InvalidateRect(hwnd, NULL, TRUE);  return 0;
        }
        case CM_FILE_QUIT:
        {   DestroyWindow(hwnd); return 0;   }
    }   return 0;
}   case WM_DESTROY:{PostQuitMessage(0); return 0;}
```

```
    }   return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Задача создания панели полностью решена в теле функции окна приложения. Рассмотрим основные шаги.

1. Описана структура типа CHOOSEFONT:

```
static  CHOOSEFONT    cf;
```

2. Описаны переменные для работы со структурой cf:

```
static  LOGFONT       lf;
static  char          szFontStyle[LF_FACESIZE];
```

Они предназначены для работы с полями структуры соответственно с именами lpLogFont и lpszStyle. Причем переменная lf будет использована для хранения атрибутов выбранного логического шрифта и инициализации органов управления панели. А переменная szFontStyle будет хранить данные стиля шрифта.

3. При обработке сообщения WM_CREATE инициализированы те поля структуры cf, которые используются для инициализации панели и для хранения атрибутов выбранного шрифта:

```
cf.lStructSize = sizeof(CHOOSEFONT);
cf.lpLogFont = &lf;
cf.Flags =  CF_SCREENFONTS | CF_USESTYLE |
            CF_EFFECTS | CF_INITTOLOGFONTSTRUCT;
cf.lpszStyle = (LPSTR)szFontStyle;
```

Здесь же устанавливают режим прозрачного фона вывода текста:

```
bkMode=TRANSPARENT;
```

4. При обработке сообщения WM_PAINT значение переменной lf используют для создания шрифта:

```
HFONT hFont       = CreateFontIndirect(&lf);
HFONT hOldFont    = (HFONT)SelectObject(hdc, hFont);
```

А значение поля rgbColors структуры cf – для установки цвета вывода текста:

```
SetTextColor(hdc, cf.rgbColors);
```

Остальные операторы, служащие для вывода текста, непосредственной связи с задачей выбора и создания шрифта не связаны:

```
SetTextAlign( hdc, TA_CENTER );
SetBkMode(hdc, bkMode);
TextOut(hdc, cx/2, cy/2, szBuf, lstrlen(szBuf));
```

5. При выборе строки меню "Задать шрифт" создают панель для выбора атрибутов шрифта:

```
if (ChooseFont(&cf)) bkMode=TRANSPARENT;
else    bkMode=OPAQUE;
```

Если работа с панелью завершилась выбором атрибутов шрифта, то переменной bkMode присваивается значение TRANSPARENT. Иначе эта переменная принимает значение OPAQUE. Как было сказано выше, значение переменной bkMode является вторым аргументом вызова функции SetBkMode.

В любом случае посылают сообщение о необходимости перерисовки всей рабочей области:

```
InvalidateRect(hwnd, NULL, TRUE);
```

Контрольные вопросы

1. Как определяют диалоговые единицы средней ширины и высоты символов шрифта?
2. Какие типы панелей существуют и чем они отличаются?
3. Чем отличается функция окна диалоговой панели от функций обычных окон?
4. Как создать и разрушить модальную панель на базе шаблона панели в памяти?
5. Какова структура шаблона панели в памяти?
6. Какие действия и в какой последовательности выполняют для создания и разрушения модальной панели?
7. Какие действия и в какой последовательности выполняют для создания и разрушения немодальной панели?
8. Какая последовательность действий используется для создания блокнота панелей?
9. Каким образом обеспечивается обработка клавиатурных сообщений в немодальной панели и в немодальном блокноте?
10. Где обрабатывают сообщения о нажатии кнопок блокнота?
11. Как создают стандартные диалоговые панели?

Упражнения

1. Создать модальную диалоговую панель для ввода размерности и элементов m*n-матрицы (m, n ≤ 10). После нажатия кнопки OK построчно отобразить элементы введенной матрицы.

2. Диалоговая панель содержит окно ввода, два списка и кнопки <<, >> и "Готово". После нажатия кнопки << (или кнопки >>) содержимое окна ввода или выбранную в правом (или в левом) списке строку пере-

слать в левый (или в правый) список. Работу завершить после нажатия кнопки "Готово".

3. Диалоговая панель содержит временное окно. В центре этого окна нарисовать эллиптическую диаграмму. Количество секторов и их процентные доли задать с помощью органов управления панели. Для выбора цветов закраски секторов использовать стандартную панель выбора цветов.

4. Диалоговая панель содержит два списка, кнопки <<, >>, OK и "Назад". После нажатия кнопки << (или кнопки >>) выбранные в правом (или в левом) списке строки переслать в левый (или в правый) список. При нажатии кнопки "Назад" отменить последний перенос. После нажатия кнопки OK завершить работу.

5. Диалоговая панель содержит список имен участников некоторого события и список их рейтингов. После выбора участника на панели нарисовать цилиндр, высота которого пропорциональна рейтингу участника. Ниже цилиндра вывести имя участника, выше – рейтинг.

6. В первой странице блокнота установить стиль выравнивания текста ("По левой границе", "По правой границе" и "По центру"), во второй странице – шрифт ("Обычный", "Полужирный", "Курсив" и "Полужирный Курсив"). После нажатия кнопки OK в центре окна – владельца блокнота выбранным стилем и шрифтом отобразить текст "Пробный вывод".

7. При выборе строки меню "Установить пароль" создать панель, где с помощью клавиатуры ввести имя пользователя и пароль, затем нажать кнопку OK. После этого создать другую диалоговую панель для подтверждения введенных данных. После подтверждения данных завершить работу панели и отобразить введенные данные.

8. Создать диалоговую панель для отображения набираемого номера телефона в статическом органе. Номер набирать с помощью расположенных на поверхности панели кнопок от "0" до "9".

9. Диалоговая панель содержит 3 вертикальные полосы прокрутки для регулирования значений rgb-составляющих цвета и 3 статических органа для отображения значений составляющих цвета. Ниже полос прокрутки находится временное окно, в котором отображается текст "Пробный вывод" текущим значением цвета на прозрачном фоне. После любого изменения составляющих отобразить текст этим цветом.

10. На диалоговой панели расположить временное окно. В центре этого окна нарисовать мишень из 10 полей и в каждом поле вывести его значение (от 1 до 10). Поля выделять различными цветами, выбранными с помощью стандартной диалоговой панели выбора цветов.

11. На месте нажатия правой клавиши мыши отобразить диалоговую панель с группой зависимых переключателей выбора одного из несколь-

ких размеров шрифта, группой независимых переключателей для выбора стиля шрифта и кнопками "Да" и "Отмена". После нажатия кнопки "Да" в рабочей области окна приложения выбранными атрибутами шрифта отобразить текст "Пробный вывод".

12. Первая страница блокнота запрашивает имя пользователя, пароль и его подтверждение. В случае успешной регистрации пользователя раскрыть вторую страницу со списком доступных этому пользователю данных. После нажатия кнопки OK блокнота отобразить имя пользователя и выбранные данные.

13. На диалоговой панели расположить список с именами констант системных цветов. Правее списка отобразить временное окно. Выбрать название системного цвета и нажать кнопку "Готово". После этого выбранным цветом закрасить рабочую область временного окна. В заголовке этого окна отобразить имя выбранной константы.

14. В центре рабочей области окна приложения друг под другом трижды отобразить один и тот же текст. В первый раз – шрифтом по умолчанию, второй – выбранным с помощью стандартной диалоговой панели, третий – шрифтом по умолчанию.

15. Создать диалоговую панель для задания текста выводимой строки, цвета букв, цвета фона текста и ориентации вывода. После выбора атрибутов вывода отобразить указанную строку с этими атрибутами.

16. В окне приложения отобразить прямоугольник, высоту, ширину и цвет которого задать с помощью диалоговой панели.

17. Создать макет калькулятора, содержащего поле ввода чисел, кнопки цифр от 0 до 9, десятичной точки, изменения знака числа, арифметических действий и "=" .

18. На панели расположить временное окно. В этом окне отобразить клетки для игры в крестики-нолики. При первом нажатии левой клавиши мыши в клетке нарисовать крестик, при втором нажатии – нолик. Количество клеток задавать с помощью органов управления панели.

19. В четырех комбинированных списках панели находятся фамилии, имена и отчества сотрудников и номера их телефонов. При выборе в любом списке (например, в списке фамилий) обеспечить выбор соответствующих данных в остальных списках. После нажатия клавиши OK в заголовке окна – владельца панели отобразить данные сотрудника.

20. Во временном окне создать пустые списки для хранения имен файлов и путей к этим файлам. С помощью стандартной панели выбрать имена файлов в любом каталоге и записать их данные в списки временного окна. Временное окно должно обрабатывать клавиатурные сообщения так же, как и модальные панели.

21. Блокнот содержит две страницы. В первой странице содержатся списки паспортных данных пациентов клиники. Во второй странице хранятся данные из их медицинских карт. При раскрытии любой страницы в ее списках должны быть выбраны данные, соответствующие выбранным данным другой страницы.

22. На панели расположить временное окно. В этом окне нарисовать графики функций 2*a*Cos(kt)*exp(-nt) и 0.5*a*Sin(kt)*exp(-nt). Значения параметров функций задать с помощью органов управления этой же панели.

23. Панель содержит раздел меню "Вид" со строками "Обычный", "Сжатый" и "Сокращенный". На обычной панели расположить временное окно с фоном "Рабочий стол", горизонтальную полосу прокрутки и несколько статических органов. На сжатой панели должны отсутствовать статические органы, а на сокращенной панели отсутствует и полоса прокрутки. При изменении вида изменить и размеры панели.

24. Диалоговая панель содержит список имен фигур (например, прямоугольник, треугольник...), окна ввода вещественных параметров фигур (например, высота, ширина...) и временное окно. После выбора фигуры, ввода ее параметров и нажатия кнопки "Готово" в рабочей области временного окна нарисовать выбранную фигуру.

25. Первая страница блокнота содержит названия книг, вторая – названия глав выбранной книги, третья – названия параграфов выбранной главы. Все страницы содержат окно ввода и кнопки "Добавить" и "Готово". После нажатия кнопки "Добавить" содержимое окна ввода добавить в список. После выбора строки в списке и нажатия кнопки "Готово" раскрыть следующую страницу. Если больше страниц нет, завершить работу блокнота.

Приложение 1

Таблица 1.1. Список значений стиля класса окон

Значение стиля класса	*Пояснение*
CS_ BYTEALIGNCLIENT	Рабочую область окна выравнивать по границе байта видеопамяти. Это ускоряет перерисовку окна, влияет на размеры окна по оси **x**
CS_ BYTEALIGNWINDOW	Все окно выравнивать по границе байта видеопамяти. Этот стиль влияет на размеры окна по оси **x**
CS_CLASSDC	Создать единый контекст отображения для всех окон
CS_DBLCLKS	Функция окна будет получать сообщения при двойном щелчке клавишей мыши над рабочей областью
CS_GLOBALCLASS	Доступный всем приложениям класс. При создании окон можно игнорировать дескриптор приложения
CS_HREDRAW	Рабочую область перерисовывать при изменении ширины окна
CS_NOCLOSE	Отключить команду Close в системном меню
CS_OWNDC	Создать свой контекст устройства для каждого окна
CS_PARENTDC	Окна будут пользоваться контекстом устройства создавшего их окна (родителя)
CS_SAVEBITS	Часть окна, затененного другим окном, сохранять в виде битового образа и использовать этот образ для воссоздания этой части при перемещении затеняющего окна
CS_VREDRAW	Рабочую область перерисовывать при изменении высоты окна

Таблица 1.2. Список значений определенных в системе иконок

Значения	*Назначение*
IDI_APPLICATION	Иконка, назначаемая окну приложения по умолчанию
IDI_ASTERISK	Символ i (используется в информативных сообщениях)
IDI_EXCLAMATION	Восклицательный знак (используется в предупреждениях)
IDI_HAND	Знак STOP (используется при ошибках).
IDI_QUESTION	Вопросительный знак (используется в подсказках)
NULL или IDI_WINLOGO	Эмблема Windows

Таблица 1.3. Список значений определенных в системе курсоров мыши

Значение	*Назначение*
IDC_APPSTARTING	Стандартная стрелка и малые песочные часы
IDC_ARROW	Стандартная стрелка
IDC_CROSS	Перекрестие
IDC_IBEAM	Текстовый курсор в виде I
IDC_NO	Перечеркнутый круг
IDC_SIZEALL	Четырехсторонняя стрелка
IDC_SIZENESW	Двусторонняя стрелка (на северо-восток и юго-запад)
IDC_SIZENS	Двусторонняя стрелка (на север и юг)
IDC_SIZENWSE	Двусторонняя стрелка (на северо-запад и юго-восток)
IDC_SIZEWE	Двусторонняя стрелка (на запад и восток)
IDC_UPARROW	Вертикальная стрелка
IDC_WAIT	Песочные часы

Таблица 1.4. Список значений системных цветов

Идентификатор цвета	*Назначение в системе*
COLOR_ACTIVEBORDER	Рамка активного окна
COLOR_ACTIVECAPTION	Заголовок активного окна
COLOR_APPWORKSPACE	Фон MDI-окна
COLOR_BACKGROUND, COLOR_DESKTOP	Рабочий стол
COLOR_BTNFACE	Цвет трехмерных элементов
COLOR_BTNHIGHLIGHT	Выбранная кнопка
COLOR_BTNSHADOW	Тень трехмерных элементов
COLOR_BTNTEXT	Текст надписи кнопки
COLOR_CAPTIONTEXT	Текст заголовка окна, кнопок изменения размера и полосы просмотра
COLOR_GRAYTEXT	Текст заблокированного элемента
COLOR_HIGHLIGHT	Выбранный элемент управления
COLOR_HIGHLIGHTTEXT	Текст выбранного элемента
COLOR_INACTIVEBORDER	Рамка неактивного окна
COLOR_INACTIVECAPTION	Заголовок неактивного окна
COLOR_ INACTIVECAPTION-TEXT	Текст заголовка для неактивного окна
COLOR_MENU	Фон меню
COLOR_MENUTEXT	Текст меню
COLOR_SCROLLBAR	Внутренняя область полосы просмотра
COLOR_WINDOW	Фон окна
COLOR_WINDOWFRAME	Рамка окна
COLOR_WINDOWTEXT	Текст в окне
COLOR_3DDKSHADOW	Темная тень трехмерных элементов
COLOR_BTNHILIGHT	Цвет граней выбранных трехмерных элементов
COLOR_3DLIGHT	Цвет граней трехмерных элементов
COLOR_INFOBK	Фон панели инструментов
COLOR_INFOTEXT	Текст на панели инструментов

Таблица 1.5. Список значений стилей окон

Имя константы	*Описание стиля*
WS_BORDER	Окно с рамкой
WS_CAPTION	Окно с заголовком (включает стиль WS_BORDER)
WS_CHILD или WS_CHILDWINDOW	Дочернее окно
WS_CLIPCHILDREN	Родительское окно не перерисовывает те области, которые затенены собственными дочерними окнами
WS_CLIPSIBLINGS	Дочернее окно не перерисовывает те области, которые затенены другими дочерними окнами того же родителя
WS_DISABLED	Создать заблокированное окно
WS_DLGFRAME	Окно с двойной рамкой без заголовка
WS_GROUP	Определяет первый элемент группы органов управления, которая состоит из этого элемента и последующих за ним элементов до элемента со стилем WS_GROUP. Первый элемент группы имеет стиль WS_TABSTOP, чтобы пользователь мог перемещаться между группами нажатием клавиш Tab. Внутри группы от элемента к элементу переходят клавишами направления
WS_HSCROLL	Окно с горизонтальной полосой просмотра
WS_ICONIC или WS_MINIMIZE	Изначально свернутое в пиктограмму окно
WS_MAXIMIZE	Окно максимально возможного размера
WS_MAXIMIZEBOX	Окно с кнопкой восстановления/максимизации размера. Используют со стилями WS_OVERLAPPED и WS_CAPTION

Имя константы	*Описание стиля*
WS_MINIMIZEBOX	Окно с кнопкой сворачивания окна в пиктограмму. Используют со стилями WS_OVERLAPPED и WS_CAPTION
WS_OVERLAPPED	Перекрывающееся окно, имеющее заголовок и рамку
WS_OVERLAPPEDWINDOW	Окно с комбинацией стилей WS_OVERLAPPED, WS_THICKFRAME, WS_SYSMENU, WS_MINIMIZEBOX и WS_MAXIMIZEBOX
WS_POPUP	Временное окно
WS_POPUPWINDOW	Комбинация стилей WS_POPUP, WS_BORDER и WS_SYSMENU. Для того чтобы сделать системное меню доступным, нужно добавить стиль WS_CAPTION
WS_SIZEBOX	Окно с толстой рамкой для изменения размера окна
WS_SYSMENU	Окно с системным меню. Объединяют с WS_CAPTION
WS_TABSTOP	Элемент управления, который получает фокус ввода с клавиатуры при нажатии клавиши Tab
WS_VISIBLE	Окно становится видимым сразу после создания
WS_VSCROLL	Окно с вертикальной полосой просмотра
MDIS_ALLCHILDSTYLES	Используется при создании дочерних MDI-окон и определяет окна, которые могут иметь любые комбинации стилей. По умолчанию дочерние MDI-окна имеют стили WS_MINIMIZE, WS_MAXIMIZE, WS_VSCROLL, WS_HSCROLL

Приложение 2

Таблица 2.1. Список имен определенных в системе классов

Имя класса	*Пояснение*	*Сообщение родительскому окну*
"button"	Окно, состояние которого изменяется при выборе и нажатии. Это нажимающиеся кнопки (push button), переключатели (radio button) и флажки (check box)	WM_COMMAND
"static"	Текстовое поле, прямоугольник выделения и другие элементы оформления	WM_NCHITTEST
"scrollbar"	Полоса прокрутки – прямоугольник с ползунком и стрелками направления в оба конца. Родительское окно обеспечивает калибровку и установку позиций ползунка. Полоса прокрутки имеет такие же вид и функцию, что и полосы просмотра окна, но может появиться где угодно. Она также обеспечивает изменение размеров окна	WM_HSCROLL и WM_VSCROLL
"edit"	Окно ввода текста из клавиатуры или редактор	WM_COMMAND
"listbox"	Список символьных строк. При просмотре в каждый момент подсвечивается одна строка и о ней передается сообщение родительскому окну	WM_COMMAND
"combobox"	Орган управления, состоящий из списка и поля выбора. Список может быть распахнут или свернут. Если список распахнут, при вводе символов в поле выбора список устанавливает в начало первое вхождение наиболее похожей строки списка	WM_COMMAND

Таблица 2.2. Список стилей окон класса "button"

Стили	*Пояснение*
BS_3STATE	Флажок (квадратик), который может находиться во включенном (перечеркнут), выключенном (не перечеркнут) и неактивном (отображается серым цветом) состоянии
BS_AUTO3STATE	То же, что и BS_3STATE, но вид квадратика автоматически изменяется при переключении, циклически отображая включенное, выключенное и неактивное состояние
BS_AUTOCHECKBOX	Флажок, который может находиться только во включенном или выключенном (неактивном) состоянии
BS_AUTORADIOBUTTON	Переключатель (окружность), который может находиться во включенном (внутри имеется жирная черная точка) или выключенном (внутри нет точки) состоянии. В группе может быть включен только один переключатель этой группы
BS_CHECKBOX	То же, что и BS_AUTOCHECKBOX, но с текстом справа от флажка
BS_DEFPUSHBUTTON	То же, что и BS_PUSHBUTTON, но кнопка имеет толстую черную рамку. Если такая кнопка находится в диалоговом окне, то она посылает сообщение родительскому окну при нажатии клавиши Enter, даже если она не имеет фокуса ввода. Этот стиль используется для задания кнопок по умолчанию
BS_GROUPBOX	Прямоугольник, в котором группируют органы управления. Он не принимает сообщения. Текст этого органа отображается в верхнем левом угле прямоугольника

Стили	*Пояснение*
BS_LEFTTEXT или BS_RIGHTBUTTON	Этот стиль дополняет стиль флажка или переключателя и помещает текст слева от элемента
BS_PUSHBUTTON	Стандартная кнопка без рамки
BS_RADIOBUTTON	Переключатель с текстом справа, который может находиться во включенном или выключенном состоянии
BS_ICON	Органом управления является иконка
BS_MULTILINE	Распределить текст в нескольких строках внутри прямоугольника кнопки
BS_NOTIFY	Орган управления посылает сообщения BN_DBLCLK, BN_KILLFOCUS и BN_SETFOCUS родительскому окну
BS_PUSHLIKE	Флажок или переключатель этого стиля ведет себя подобно кнопке – приподнимается или притапливается
BS_TEXT	Орган управления является текстом
BS_RIGHT, BS_LEFT, BS_TOP и BS_BOTTOM	Текст выровнен соответственно по правому, левому, верхнему или нижнему краю в прямоугольнике кнопки
BS_CENTER	Центрирует текст по горизонтали в прямоугольнике кнопки
BS_VCENTER	Центрирует текст по вертикали в прямоугольнике кнопки

Таблица 2.3. Список стилей окон класса "static"

Стили	*Описание*
SS_BLACKFRAME, SS_GRAYFRAME или SS_WHITEFRAME	Прямоугольня рамка системного цвета соответственно COLOR_WINDOWFRAME ("черного" цвета рамок окон), COLOR_BACKGROUND ("серого" цвета фона экрана) или COLOR_WINDOW ("белого" цвета рабочей области окон). Внутренняя область остается незакрашенной. Текст заголовка окна не используется. Соответствующий параметр функции CreateWindow указывают как NULL

Стили	*Описание*
SS_BLACKRECT, SS_GRAYRECT или SS_WHITERECT	Закрашенный прямоугольник системного цвета соответственно COLOR_ WINDOWFRAME ("черного" цвета рамок окон), COLOR_BACKGROUND ("серого" цвета фона экрана) или COLOR_WINDOW ("белого" цвета рабочей области окон). Текст заголовка окна не используют. Этот параметр функции CreateWindow указывают как NULL
SS_LEFT, SS_RIGHT или SS_CENTER	В заданном прямоугольнике указанный текст выводит функция DrawText, соответственно выравнивая его по левому, правому краю или центрируя и перенося слова в следующую строку. Не поместившаяся в прямоугольнике часть текста не отображается. Символы табуляции заменяются пробелами
SS_LEFTNOWORDWRAP	То же, что и SS_LEFT, но без переноса слов
SS_NOPREFIX	Этот стиль дополняет другие стили в тех случаях, когда необходимо отменить специальную обработку символа &. Обычно этот символ не выводится статическими органами управления на экран, а следующий за ним символ изображается подчеркнутым (для изображения символа & его надо повторить 2 раза подряд)
SS_NOTIFY	Окно этого стиля функции родительского окна посылает сообщения STN_CLICKED и STN_DBLCLK о нажатии или двойном щелчке левой клавишей мыши

Стили	Описание
SS_SIMPLE	Выводит текст в одну строку, используя функцию TextOut и выравнивая по левому краю. Символы табуляции не заменяет пробелами. При повторном выводе текста содержимое прямоугольника не стирается, поэтому новый текст не должен быть короче прежнего. Обычно комбинируют со стилем SS_NOPREFIX (используется более быстрая функция ExtTextOut)

Таблица 2.4. Список стилей окон класса "scrollbar"

Стили	Описание
SBS_BOTTOMALIGN или SBS_TOPALIGN	В нижней или верхней части прямоугольника, заданного аргументами вызова функции CreateWindow, создается горизонтальная полоса прокрутки, высота которой равна высоте системной полосы просмотра. Этот стиль используется вместе с SBS_HORZ
SBS_HORZ	Если не указан стиль SBS_BOTTOMALIGN или SBS_TOPALIGN, то позиция, высота и ширина горизонтальной полосы прокрутки определяются аргументами x, y, nWidth и nHeight вызова функции CreateWindow
SBS_LEFTALIGN или SBS_RIGHTALIGN	В левом или правом краю прямоугольника, заданного аргументами вызова функции CreateWindow, создается вертикальная полоса прокрутки, ширина которой равна ширине системной полосы просмотра. Этот стиль используется вместе с SBS_VERT

Стили	*Описание*
SBS_SIZEBOX	Создается прямоугольник серого цвета. Если установить курсор мыши внутрь прямоугольника, нажать левую клавишу и перемещать мышь, родительское окно получает сообщения, аналогичные сообщениям от рамки изменения размера окна. Если стили SBS_SIZEBOXBOTTOMRIGHTALIGN и SBS_SIZEBOXTOPLEFTALIGN не указаны, то прямоугольник задается аргументами x, y, nWidth и nHeight функции CreateWindow
SBS_SIZEBOXBOTTOM-RIGHTALIGN (SBS_SIZEBOXTOP-LEFTALIGN)	Аналогично SBS_SIZEBOX, но правый нижний (левый верхний) угол прямоугольника выравнивается по правому нижнему (левому верхнему) углу прямоугольника, определенного при вызове функции Create Window, а для высоты и ширины органа управления используются системные значения. Этот стиль применяется вместе с SBS_SIZEBOX
SBS_SIZEGRIP	Тот же самый, что и SBS_SIZEBOX, но с поднятым краем
SBS_VERT	Если не указан стиль SBS_RIGHTALIGN или SBS_LEFTALIGN, то позиция, высота и ширина вертикальной полосы прокрутки определяются аргументами вызова функции CreateWindow

Таблица 2.5. Список стилей окон класса "edit"

Стиль	*Назначение*
ES_AUTOHSCROLL	Автоматически сдвигать текст влево на 10 символов при достижении правой границы окна ввода. В многострочном редакторе для перехода в начало следующей строки нужно нажать клавишу Enter

Стиль	Назначение
ES_AUTOVSCROLL	Используется в многострочном редакторе для автоматического сдвига текста вверх на одну строку при достижении нижней границы окна ввода. Иначе при достижении нижней границы будет выдан звуковой сигнал
ES_CENTER	Центрировать строки по горизонтали
ES_LEFT	Выравнивать текст по левому краю окна ввода. Часто используется для задания однострочного редактора текста и комбинируется со стилем ES_AUTOHSCROLL
ES_LOWERCASE	Преобразовать вводимые символы в строчные
ES_MULTILINE	Многострочный редактор. Обычно комбинируют со стилями ES_WANTRETURN (для использования клавиши Enter в целях перехода на новую строку), ES_AUTOVSCROLL (для перелистывания текста по вертикали, иначе текст только сдвигается вверх при достижении нижнего края окна и подается звуковой сигнал), ES_AUTOHSCROLL (для перелистывания текста по горизонтали, иначе ввод автоматически переносится в начало следующей строки при достижении правого края окна). Для подключения полос просмотра многострочному редактору этот стиль комбинируют со значениями WS_HSCROLL и WS_VSCROLL. Функция окна многострочного редактора сама обрабатывает сообщения от полос просмотра
ES_NOHIDESEL	При потере фокуса ввода выделенный в окне фрагмент текста отображать в инверсном цвете. Если этот стиль не указан, выделение фрагмента пропадает до возврата фокуса ввода
ES_NUMBER	Разрешить ввод только цифр
ES_PASSWORD	Отображать звездочку (*) для каждого вводимого символа
ES_READONLY	Запрет редактирования текста в окне редактирования
ES_RIGHT	Выравнивать текст по правому краю окна

Стиль	*Назначение*
ES_UPPERCASE	Преобразовать вводимые символы в прописные
ES_WANTRETURN	При нажатии на клавишу ENTER перейти в начало следующей строки. Иначе в диалоговых окнах нажатие клавиши ENTER воспринимается как нажатие кнопки по умолчанию. Стиль используется в комбинации со стилем ES_MULTILINE и только в диалоговых панелях

Таблица 2.6. Список стилей окон класса "listbox"

Стиль	*Пояснение*
LBS_ DISABLENOSCROLL	Вертикальную полосу просмотра отображать в неактивном состоянии (но не скрывать), если в списке помещаются все строки. Без указания стиля LBS_DISABLENOSCROLL в аналогичной ситуации вертикальная полоса просмотра пропадает. Используется, если нежелательно изменение внешнего вида списка
LBS_EXTENDEDSEL	С помощью клавиши Shift или мыши можно выделять несколько расположенных рядом строк
LBS_HASSTRINGS	Список родительскому окну посылает сообщение WM_VKEYTOITEM с кодом виртуальной клавиши LOWORD(wParam), с номером текущей строки HIWORD (wParam) и дескриптором списка lParam
LBS_MULTICOLUMN	Многоколоночный список. Количество колонок задают сообщением LB_SETCOLUMNWIDTH
LBS_MULTIPLESEL	С помощью клавиши Shift или мыши можно выделять несколько строк в любом месте списка
LBS_ NOINTEGRALHEIGHT	Высота окна не обязательно кратна высоте строк
LBS_NOREDRAW	Не перерисовывать содержимое при добавлении или удалении строк

Стиль	Пояснение
LBS_NOSEL	Определяет, что окно списка содержит единицы, которые могут просматриваться, но не выбран
LBS_NOTIFY	Сообщать о двойном щелчке мышью по строке
LBS_ OWNERDRAWFIXED	Список строк одинаковой высоты, перерисовывается родительским окном
LBS_OWNERDRAW-VARIABLE	Список строк переменной высоты, перерисовывается родительским окном
LBS_SORT	Сортировать строки в алфавитном порядке
LBS_STANDARD	Комбинация стилей LBS_NOTIFY, LBS_SORT, WS_BORDER и WS_VSCROLL. Обычно используют для создания простейшего одноколоночного списка
LBS_USETABSTOPS	При отображении строк преобразовывать символы табуляции. По умолчанию один символ табуляции расширяется на 32 единицы ширины диалоговых панелей (или 8 единиц средней ширины текущего шрифта)
LBS_WANTKEY-BOARDINPUT	Родительское окно от списка получает сообщения WM_VKEYTOITEM или WM_CHARTOITEM при работе со списком при помощи клавиатуры. Это дает возможность приложению выполнить специальную обработку ввода с клавиатуры

Таблица 2.7. Список стилей окон класса "ComboBox"

Стиль	Пояснение
CBS_ AUTOHSCROLL	Автоматическая свертка текста по горизонтали в окне редактирования
CBS_ DISABLENOSCROLL	Вертикальную полосу просмотра отображать в неактивном состоянии, даже если в списке помещаются все строки
CBS_DROPDOWN	Подобен CBS_SIMPLE, но список остается в невидимом состоянии до тех пор, пока пользователь не нажмет пиктограмму, предназначенную для отображения списка

Стиль	*Пояснение*
CBS_ DROPDOWNLIST	Подобен CBS_DROPDOWN, но однострочный текстовый редактор может быть использован только для отображения текста, а не для редактирования
CBS_LOWERCASE	Преобразовать все символы в строчные
CBS_SIMPLE	Создается список, который всегда виден и расположен под окном однострочного редактора текста, содержащего выделенную в списке строку
CBS_SORT	Автоматически сортирует строки, введенные в окно списка
CBS_UPPERCASE	Преобразовать все символы в прописные

Приложение 3

Таблица 3.1. Список системных цветов в RGB-представлении

Красный цвет	*Зеленый цвет*	*Голубой цвет*	*Цвет в системной палитре*
0	0	0	Черный
80	0	0	Темно-красный
0	80	0	Темно-зеленый
80	80	0	Темно-желтый
0	0	80	Темно-голубой
80	0	80	Темно-малиновый
0	80	80	Темно-синий
192	192	192	Светло-серый
192	220	192	Светло-зеленый
166	202	240	Светло-голубой
255	251	240	Кремовый
160	160	164	Светло-серый
80	80	80	Серый
255	0	0	Красный
0	255	0	Зеленый
255	255	0	Желтый

Красный цвет	*Зеленый цвет*	*Голубой цвет*	*Цвет в системной палитре*
0	0	255	Синий
255	0	255	Малиновый
0	255	255	Голубой (циан)
255	255	255	Белый

Таблица 3.2. Список режимов рисования

Режимы рисования	*Цвет пикселя*
R2_BLACK	Черный
R2_COPYPEN	Цвет пера
R2_MASKNOTPEN или R2_MERGENOTPEN	Комбинация цвета пикселя до рисования и инверсии цвета пера
R2_MASKPEN или R2_MERGEPEN	Комбинация цвета пикселя до рисования и цвета пера
R2_MASKPENNOT или R2_MERGEPENNOT	Комбинация инверсии цвета пикселя до рисования и цвета пера
R2_NOP	Остается неизменным
R2_NOT	Инверсия цвета пикселя до рисования
R2_NOTCOPYPEN	Инверсия цвета пера
R2_NOTMASKPEN или R2_NOTMERGEPEN	Инверсия цвета пикселя до рисования
R2_NOTXORPEN	Инверсия цвета R2_XORPEN
R2_WHITE	Белый
R2_XORPEN	Операция ИСКЛЮЧАЮЩЕЕ ИЛИ к цветам пикселя до рисования и пера

Приложение 4

Таблица 4.1. Список кодов виртуальных клавиш

Символическое имя	*Код виртуальной клавиши*	*Клавиша на клавиатуре*
Не определено	0x0	
VK_LBUTTON (код мыши)	0x1	

Символическое имя	*Код виртуальной клавиши*	*Клавиша на клавиатуре*
VK_RBUTTON (код мыши)	0x2	
VK_CANCEL	0x3	<Control + Break>
VK_MBUTTON (код мыши)	0x4	
Не определено	0x5 – 0x7	
VK_BACK	0x8	Клавиша забоя <Backspace>
VK_TAB	0x9	<Tab>
Не определено	0xa – 0xb	
VK_CLEAR	0xc	Соответствует клавише <5> дополнительной клавиатуры при выключенном режиме <Num Lock>
VK_RETURN	0xd	<Enter>
Не определено	0xe – 0xf	
VK_SHIFT	0x10	<Shift>
VK_CONTROL	0x11	<Control>
VK_MENU	0x12	<Alt>
VK_PAUSE	0x13	<Pause>
VK_CAPITAL	0x14	<Caps Lock>
Не определено	0x15 – 0x1a	
VK_ESCAPE	1b	<Esc>
Не определено	0x1c – 0x1f	
VK_SPACE	0x20	Клавиша пробела
VK_PRIOR	0x21	<PgUp>
VK_NEXT	0x22	<PgDn>
VK_END	0x23	<End>
VK_HOME	0x24	<Home>
VK_LEFT	0x25	Клавиша перемещения влево <Left>
VK_UP	0x26	Клавиша перемещения вверх <Up>
VK_RIGHT	0x27	Клавиша перемещения вправо <Right>

Символическое имя	*Код виртуальной клавиши*	*Клавиша на клавиатуре*
VK_DOWN	0x28	Клавиша перемещения вниз <Down>
VK_SELECT	0x29	
VK_PRINT	0x2a	
VK_EXECUTE	0x2b	
VK_SNAPSHOT	0x2c	<PrtSc>
VK_INSERT	0x2d	<Insert>
VK_DELETE	0x2e	<Delete>
VK_HELP	0x2f	
Не определено	0x30	<0>
" "	0x31	<1>
" "	0x32	<2>
" "	0x33	<3>
" "	0x34	<4>
" "	0x35	<5>
" "	0x36	<6>
" "	0x37	<7>
" "	0x38	<8>
" "	0x39	<9>
" "	0x3a – 0x40	
" "	0x41	<A>
" "	0x42	<B>
" "	0x43	<C>
" "	0x44	<D>
" "	0x45	<E>
" "	0x46	<F>
" "	0x47	<G>
" "	0x48	<H>
" "	0x49	<I>
" "	0x4a	<J>
" "	0x4b	<K>
" "	0x4c	<L>
" "	0x4d	<M>

Символическое имя	*Код виртуальной клавиши*	*Клавиша на клавиатуре*
Не определено	0x4e	<N>
" "	0x4f	<O>
" "	0x50	<P>
" "	0x51	<Q>
" "	0x52	<R>
" "	0x53	<S>
" "	0x54	<T>
" "	0x55	<U>
" "	0x56	<V>
" "	0x57	<W>
" "	0x58	<X>
" "	0x59	<Y>
" "	0x5a	<Z>
" "	0x5b – 0x5f	
VK_NUMPAD0	0x60	<0> на цифровой клавиатуре
VK_NUMPAD1	0x61	<1> на цифровой клавиатуре
VK_NUMPAD2	0x62	<2> на цифровой клавиатуре
VK_NUMPAD3	0x63	<3> на цифровой клавиатуре
VK_NUMPAD4	0x64	<4> на цифровой клавиатуре
VK_NUMPAD5	0x65	<5> на цифровой клавиатуре
VK_NUMPAD6	0x66	<6> на цифровой клавиатуре
VK_NUMPAD7	0x67	<7> на цифровой клавиатуре
VK_NUMPAD8	0x68	<8> на цифровой клавиатуре
VK_NUMPAD9	0x69	<9> на цифровой клавиатуре
VK_MULTIPLAY	0x6a	<*> на цифровой клавиатуре
VK_ADD	0x6b	<+> на цифровой клавиатуре
VK_SEPARATOR	0x6c	
VK_SUBTRACT	0x6d	<-> на цифровой клавиатуре
VK_DECIMAL	0x6e	<.> на цифровой клавиатуре
VK_DIVIDE	0x6f	</> на цифровой клавиатуре
VK_F1	0x70	<F1>
VK_F2	0x71	<F2>
VK_F3	0x72	<F3>

Символическое имя	*Код виртуальной клавиши*	*Клавиша на клавиатуре*
VK_F4	0x73	<F4>
VK_F5	0x74	<F5>
VK_F6	0x75	<F6>
VK_F7	0x76	<F7>
VK_F8	0x77	<F8>
VK_F9	0x78	<F9>
VK_F10	0x79	<F10>
VK_F11	0x7a	<F11>
VK_F12	0x7b	<F12>
VK_F13	0x7c	
VK_F14	0x7d	
VK_F15	0x7e	
VK_F16	0x7f	
Не определено	0x80 – 0x8f	
VK_NUMLOCK	0x90	<Num Lock>
VK_SCROLL	0x91	<Scroll Lock>
Не определено	0x92 – 0xb9	
" "	0xba	;
" "	0xbb	+ =
" "	0xbc	, <
" "	0xbd	- _
" "	0xbe	. >
" "	0xbf	/ ?
" "	0xc0	` ~
" "	0xc1 – 0xda	
" "	0xdb	[{
" "	0xdc	\ |
" "	0xdd	] }
" "	0xde	' "

Оглавление