Утверждаю

Ректор университета

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.Н. Федонин

2021г.



**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

Брянский государственный технический университет

**ПРОГРАММИРОВАНИЕ WINDOWS-ПРИЛОЖЕНИЙ**

**РАЗРАБОТКА МНОГООКОННОГО ПРИЛОЖЕНИЯ WINDOWS**

**Методические указания**

**к выполнению лабораторной работы №3**

**для студентов очной формы обучения**

**по направлению подготовки**

**02.03.03 «Математическое обеспечение и администрирование информационных систем», профиль «Технология программирования»**

**Брянск 2020**

**2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

*Стили классов окон и оконные стили*

**На базе одного класса окна приложение может создать несколько окон**. Все эти окна могут быть сделаны в одном или нескольких стилях**.**

* **Стиль окна определяет внешний вид окна и его поведение.**
* Для класса окна также определяется понятие стиля **- стиль класса определяет внешний вид и поведение всех окон, созданных на базе данного класса**.

*Стиль класса окна*

**Стиль класса окна определяется при регистрации класса окна**. Он задается в виде отдельных битов, для которых определены символические константы с префиксом **CS\_**. **Определенный в классе окна стиль класса окна используется при создании всех окон на базе этого класса.**

**Чаще всего используются стили** **CS\_HREDRAW|CS\_VREDRAW|CS\_DBLCLKS**. Как уже говорилось, если для класса заданы стили **CS\_HREDRAW** и **CS\_VREDRAW**, то **при изменении размеров окна функция окна может получить сообщение** **WM\_PAINT**. В этом случае функция окна должна перерисовать часть окна или все окно.

**Стиль CS\_DBLCLKS используется при необходимости отслеживать двойные щелчки мышью.** При этом в функцию окна посылаются сообщения **WM\_LBUTTONDBLCLK** и **WM\_RBUTTONDBLCLK**. Если этот стиль не будет задан, функция окна получит только идущие парами сообщения об одиночном нажатии клавиш мыши **WM\_LBUTTONDOWN** и **WM\_RBUTTONDOWN.**

*Стили окон, окна основных стилей*

Определенный в классе окна **стиль класса окна используется при создании** всех окон на базе этого класса. **Для дальнейшего уточнения внешнего вида и поведения окна используется другая характеристика - стиль окна**. Стиль окна указывается при создании окна функцией **CreateWindow**. Для определения стиля окна используются символические константы с префиксом **WS\_**.

Чаще всего используемые **основные стили** – это перекрывающиеся окна (overlapped window), всплывающие (или временные, или выпадающие) окна (pop-up window), дочерние окна (child window).

*Перекрывающиеся окна*

**Перекрывающиеся окна** имеют заголовок (title bar), рамку и внутреннюю часть окна (client region). Дополнительно перекрывающиеся окна могут иметь системное меню, кнопки для максимального увеличения размера окна и свертки окна в пиктограмму, вертикальную и горизонтальную полосу просмотра и меню.

**Перекрывающиеся окна обычно используются в качестве главного окна приложения**.

**Для определения стиля перекрывающегося окна существует символическая константа WS\_OVERLAPPEDWINDOW**, определенная как поразрядное ИЛИ нескольких констант:

#define WS\_OVERLAPPEDWINDOW (WS\_OVERLAPPED| \

WS\_CAPTION|WS\_SYSMENU|WS\_THICKFRAME| \

WS\_MINIMIZEBOX|WS\_MAXIMIZEBOX)

**Приложение Windows может создавать несколько перекрывающихся окон, связанных между собой “узами родства” и “отношениями собственности”.** При создании окон при помощи функции **CreateWindow** в качестве восьмого параметра функции можно указать **дескриптор окна-владельца** (переменная типа HWND). Окно-владелец уже должно существовать на момент создания окна, имеющего владельца.

*Особенности перекрывающихся окон:*

* **если окно-хозяин сворачивается в пиктограмму,** все окна, которыми оно владеет, становятся **невидимыми;**
* если **сначала свернули в пиктограмму окно, которым владеет другое окно, а затем и окно-хозяина,** то пиктограмма подчиненного окна **исчезнет;**
* **при уничтожении окна-владельца** автоматически **уничтожаются** и все принадлежащие ему окна;
* **обычное перекрывающееся окно, не имеющее окна-владельца**, может располагаться **в любом месте экрана** и принимать любые размеры; **подчиненные окна** располагаются всегда **над поверхностью окна-владельца**, перекрывая его изображение;
* **координаты создаваемых функцией CreateWindow перекрывающихся окон указываются по отношению ко всему экрану,** т.е. при создании окна с координатами (0,0) оно будет расположено в верхнем левом углу экрана;
* **при изменении размеров перекрывающегося окна функция окна получает сообщение WM\_SIZE**, в параметрах которого указаны новые размеры окна.

*Временные окна*

Другим базовым стилем является стиль **временных окон**, которые обычно используются **для вывода информационных сообщений и остаются на экране непродолжительное время**. Временные окна, в отличие от перекрывающихся, могут не иметь заголовка. Если для временного меню заголовок определен, тогда оно может иметь и системное меню.

Часто для **создания выпадающих окон, имеющих рамку, используется стиль WS\_POPUPWINDOW,** определенный как поразрядное ИЛИ нескольких констант (для того чтобы к временному окну добавить **системное меню и заголовок**, следует использовать комбинацию **WS\_POPUPWINDOW | WS\_CAPTION**):

#define WS\_POPUPWINDOW (WS\_POPUP|WS\_BORDER|WS\_SYSMENU)

*Особенности временных окон:*

* **временные окна могут иметь окно-владельца и могут сами владеть другими окнами.** Замечания относительно владения перекрывающимися окнами справедливы и для временных окон;
* начало системы координат, используемой при создании временных окон, находится в левом верхнем углу экрана, поэтому **при создании временных окон применяются экранные координаты, как и при создании перекрывающихся окон**;
* **при изменении размеров временного окна функция окна получает сообщение WM\_SIZE**, в параметрах которого указаны новые размеры окна.

*Дочерние окна*

**Дочерние окна** чаще всего используются приложениями Windows. Эти окна нужны **для создания таких органов управления, как кнопки или списки**. Все органы управления - дочерние окна.

**Стиль дочернего окна определяется константой WS\_CHILDWINDOW.** В отличие от перекрывающихся и временных окон дочерние окна, как правило, не имеют рамки, заголовка, кнопок минимизации и максимального увеличения размеров окна, а также полос просмотра. Дочерние окна сами рисуют все, что в них должно быть изображено.

*Особенности дочерних окон:*

* **дочерние окна должны иметь окно-родителя**; только **дочерние окна могут иметь родителей, перекрывающиеся и временные окна могут иметь окно-хозяина, но не родителя**; у дочерних окон могут быть “братья” (или “сестры”);
* дочерние окна **всегда располагаются на поверхности окна-родителя**;
* при создании дочернего окна **начало системы координат расположено в левом верхнем углу внутренней поверхности окна-родителя** (но не в верхнем углу экрана, как для перекрывающихся и временных окон);
* так как дочернее окно как бы “прилипает” к поверхности окна-родителя, то **при щелчке мышью над поверхностью дочернего окна сообщение от мыши попадет в функцию дочернего, а не родительского окна**;
* **при создании дочернего окна** в качестве девятого параметра (вместо идентификатора меню, которого не может быть у дочернего окна) функции **CreateWindow** **необходимо указать идентификатор дочернего окна** (переменная типа int, любое целое число);
* если для приложения создается несколько дочерних окон**, для каждого окна необходимо указать свой идентификатор,** эти идентификаторы будут использоваться дочерними окнами **при отправлении сообщений родительскому окну**;
* дочернее окно “**прилипает” к поверхности родительского окна и перемещается вместе с ним,** оно никогда **не может выйти за пределы родительского окна**;
* все **дочерние окна скрываются при сворачивании окна-родителя** в пиктограмму **и появляются вновь при восстановлении** родительского окна;
* **при изменении размеров родительского окна дочерние окна получают сообщение WM\_PAINT**, но не получают сообщения **WM\_SIZE**, это сообщение попадает только в родительское окно.

Пример создания окон различных стилей

Фрагмент функции WinMain, создающей **окна разных стилей** на основе двух классов окон:

// регистрация классов окон с именами

// "MainWindow”, "Window1", "Window2"

. . .

// создание главного перекрывающегося окна

HWND hWndMain =

CreateWindow("MainWindow","OVERLAPPEDWINDOW",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT,CW\_USEDEFAULT,

CW\_USEDEFAULT,CW\_USEDEFAULT,

NULL,NULL,hInstance,NULL);

if(hWndMain == 0) return FALSE;

ShowWindow(hWndMain, nCmdShow);

UpdateWindow(hWndMain);

// создание временного окна

HWND hWndPopup =

CreateWindow("Window1","POPUPWINDOW",

WS\_POPUPWINDOW|WS\_CAPTION|WS\_MINIMIZEBOX|

WS\_MAXIMIZEBOX|WS\_VISIBLE,100,100,300,300, hWndMain,

NULL,hInstance,NULL);

if(hWndPopup == 0)

{ DestroyWindow(hWndMain); return FALSE; }

ShowWindow(hWndPopup, nCmdShow);

UpdateWindow(hWndPopup);

// создание дочернего окна

HWND hWndChild =

CreateWindow("Window2","CHILDWINDOW",

WS\_CHILDWINDOW|WS\_CAPTION|WS\_MINIMIZEBOX|

WS\_MAXIMIZEBOX,150,150,250,250,hWndMain,NULL,

hInstance,NULL);

if(hWndChild== 0)

{ DestroyWindow(hWndMain);

DestroyWindow(hWndPopup); return FALSE;

}

ShowWindow(hWndChild, nCmdShow);

UpdateWindow(hWndChild);

// цикл обработки очереди сообщений

. . .

*Окна с полосами прокрутки*

Полосы прокрутки являются одними из самых лучших возможностей, которые дает графический интерфейс и манипулятор мышь. Они просты в использовании и обеспечивают **удобный просмотр информации** как в вертикальном, так и в горизонтальном направлениях.

**Добавить в окно приложения вертикальную или горизонтальную полосу прокрутки очень просто**. Все, что необходимо сделать, это включить стиль **WS\_VSCROLL** и **WS\_HSCROLL** или оба сразу в описание стиля создаваемого функцией **CreateWindow** окна.

Следует заметить, что:

* **рабочая область окна не включает в себя пространство, занятое полосами прокрутки**. Ширина вертикальной и высота горизонтальной полос прокрутки **постоянны для конкретного дисплейного драйвера** (эти значения можно получить с помощью функции **GetSystemMetrics**);
* **Windows обеспечивает всю логику работы мыши с полосами прокрутки**. Однако **у полос прокрутки нет интерфейса клавиатуры**. Если приложение желает дублировать клавишами управления курсором некоторые функции полос прокрутки, оно должно **самостоятельно реализовать эту логику при обработке клавиатурных сообщений**.

*Диапазон и положение полос прокрутки*

**Каждая полоса прокрутки имеет соответствующий диапазон** (range) – два целых числа, отражающих минимальное и максимальное значения, **и положение** (position) – местоположение бегунка внутри диапазона. **По умолчанию устанавливается следующий диапазон прокрутки – минимум 0 и максимум 100**, но диапазон легко **изменить на любое другое значение** с помощью функции **SetScrollRange** (для полос прокрутки окна hWnd)

SetScrollRange(hWnd,iBar,iNewMin,iNewMax,bRedraw);

параметр iBar равен либо **SB\_VERT**, либо **SB\_HORZ**, iNewMin и iNewMax являются минимальной и максимальной границами диапазона, а bRedraw устанавливается в TRUE, если необходимо, чтобы Windows перерисовала полосы в соответствии с новыми значениями.

**Положение бегунка всегда дискретно**. Например, полоса прокрутки с диапазоном от 0 до 4 имеет пять положений бегунка. **Для установки нового положения бегунка можно использовать функцию SetScrollPos**

SetScrollPos(hWnd,iBar,iNewPos,bRedraw);

**Для определения текущего диапазона полосы прокрутки и текущего положения бегунка используются функции GetScrollRange и GetScrollPos**.

Если приложение использует полосы прокрутки, то **оно совместно с Windows обеспечивает** поддержку полос прокрутки и обновление положения бегунка.

**Cферы ответственности Windows при поддержке полос прокрутки**:

* управляет логикой работы мыши с полосой прокрутки;
* обеспечивает временную “инверсию цвета” при нажатии на кнопку мыши в полосе прокрутки;
* перемещает бегунок в соответствии с тем, как внутри полосы прокрутки его перемещает пользователь;
* отправляет сообщения полосы прокрутки в оконную процедуру для окна, содержащего полосу прокрутки.

**Сферы ответственности приложения** по поддержке полос прокрутки:

* инициализация диапазона полосы прокрутки;
* обработка сообщений полосы прокрутки;
* обновление положения бегунка полосы прокрутки.

*Сообщения полос прокрутки*

**Windows посылает оконной процедуре сообщения WM\_VSCROLL и WM\_HSCROLL, когда на полосе прокрутки щелкают мышью или перетаскивают бегунок**.

**При работе с оконными полосами прокрутки следует игнорировать параметр lParam,** он используется только для полос прокрутки – элементов управления. **Младшее слово параметра wParam этих сообщений – это число, показывающее, что мышь осуществляет какие-то действия на полосе прокрутки.** Его значение соответствует определенным идентификаторам, которые начинаются на **SB\_**:

int nScrollCode=(int)LOWORD(wParam); // произведенное действие

short int nPos=(short int)HIWORD(wParam); // текущая позиция

**Оконная процедура может получить сообщения с кодами типа SB\_LINEUP, SB\_PAGEUP, SB\_LINEDOWN и SB\_PAGEDOWN, если пользователь изменяет текущее положение на 1 положение или на 1 “страницу”** (эти коды приходят с сообщениями как от вертикальной, так и от горизонтальной полос прокрутки).

**Сообщения с кодом SB\_ENDSCROLL приходят, когда кнопка мыши отпущена**. Как правило, приложение игнорирует такие сообщения.

Если **пользователь перемещает бегунок при помощи мыши**, то сообщение от полосы прокрутки несет с собой в младшем слове параметра wParam код **SB\_THUMBTRACK** (таких сообщений может быть слишком много!). Если **затем пользователь отпускает клавишу мыши**, то в оконную процедуру поступает сообщение с кодом **SB\_THUMBPOSITION.**

Если младшее слово параметра wParam равно **SB\_THUMBTRACK** или **SB\_THUMBPOSITION**, то в этих случаях старшее слово wParam определяет **текущее положение полосы прокрутки**. Во всех остальных случаях работы с полосой прокрутки старшее слово wParam **можно игнорировать**.

**Замечание.** Согласно документации, сообщения с кодами **SB\_TOP** и **SB\_BOTTOM** показывают, что полоса прокрутки переведена в свое максимальное или минимальное положение. Однако, **если приложение самостоятельно не реализовывает интерфейс для работы с полосой прокрутки** при помощи клавиатуры **и не посылает само себе такие сообщения** по мере необходимости, то оно **никогда его не получит** от Windows.

*Пример обработки сообщений от полосы прокрутки окна*

Пусть **создано окно hWnd с вертикальной линейкой прокрутки и затем для нее установлены диапазон и текущее положение** бегунка:

static int min\_sb=1,max\_sb=100,pos\_sb=20;

SetScrollRange(hWnd,SB\_VERT,min\_sb,max\_sb,TRUE);

SetScrollPos(hWnd,SB\_VERT,pos\_sb,TRUE);

**Фрагмент оконной функции созданного окна, демонстрирующий возможную обработку действий пользователя с вертикальной линейкой прокрутки**:

. . .

case WM\_VSCROLL:

{

// запоминаем предыдущую позицию бегунка

int old\_pos\_sb=pos\_sb;

// lParam для оконных полос просмотра всегда равен NULL

int nScrollCode=(int)LOWORD(wParam);

// произведенное действие

short int nPos=(short int)HIWORD(wParam);

// текущая позиция

// изменеяем текущую позицию бегунка

switch(nScrollCode)

{

case SB\_PAGEDOWN: pos\_sb+=10; break;

case SB\_PAGEUP: pos\_sb-=10; break;

case SB\_LINEDOWN: pos\_sb+=1; break;

case SB\_LINEUP: pos\_sb-=1; break;

case SB\_THUMBPOSITION:pos\_sb=nPos; break;

case SB\_THUMBTRACK: pos\_sb=nPos; break;

default: return 0l;

}

if(pos\_sb<min\_sb) pos\_sb=min\_sb;

else if(pos\_sb>max\_sb) pos\_sb=max\_sb;

if(old\_pos\_sb!=pos\_sb)

SetScrollPos(hWnd,SB\_VERT,pos\_sb,TRUE);

}; return 0l;

. . .

*Различные метрики Windows*

В программном интерфейсе Windows имеются **функции, позволяющие получить информацию о размерах отдельных компонентов Windows, таких, как ширина рамки окна, ширина и высота экрана и т.п.** Эта информация нужна для определения габаритов и расположения создаваемых окон и других изображений.

*Системные метрики*

**Метрики системных компонентов Windows можно определить при помощи функции GetSystemMetrics.** Единственный аргумент этой функции задает параметр, значение которого необходимо определить

int GetSystemMetrics(int nIndex);

**Для определения того или иного компонента** в заголовочных файлах Windows имеются константы с префиксом **SM\_**. Некоторые из них:

* **SM\_CXCURSOR** - ширина курсора.
* **SM\_CXICON** - ширина пиктограммы.
* **SM\_CXSCREEN** - ширина экрана.
* **SM\_CYCAPTION** - высота заголовка окна.
* **SM\_CYCURSOR** - высота курсора.
* **SM\_CYICO**N - высота пиктограммы.
* **SM\_CYMENU** - высота одной строки в полосе меню.
* **SM\_CYSCREEN** - высота экрана.
* **SM\_CYHSCROLL** – высота горизонтальной полосы прокрутки.
* **SM\_CXVSCROLL** – ширина вертикальной полосы прокрутки.

**Пример определения высоты графического экрана**

int h=GetSystemMetrics(SM\_CYSCREEN);

*Определение возможностей устройств ввода/вывода*

В программном интерфейсе Windows имеется **функция, позволяющая по контексту устройства определить возможности и параметры драйверов, обслуживающих устройства ввода/вывода**:

int **GetDeviceCaps**(HDC hdc, int iCapability);

Первый параметр функции задает **контекст устройства, для которого необходимо получить информацию о его возможностях** (получить контекст устройства можно от функций **BeginPaint** при обработке сообщения **WM\_PAINT** или **GetDC** при обработке сообщения, отличного от **WM\_PAINT**). Второй параметр функции **GetDeviceCaps** определяет параметр устройства, значение которого необходимо получить.

**Замечание.** Остановимся на некоторых из **возможных значений параметров устройства.** Значения **ASPECTX, ASPECTY, ASPECTXY определяют размеры пикселя.** Дело в том, что пикселы не всегда квадратные. Поэтому масштаб изображения по оси X может отличаться от масштаба по оси Y. Размеры пикселя позволяют вычислить отношение сторон пикселя и выполнить правильное масштабирование. В этом случае отображаемые окружности будут круглыми, а квадраты - квадратными.

**Пример получения информации о соотношении размеров пикселя по осям X и Y** при обработке сообщения WM\_PAINT в некоторой оконной функции

case WM\_PAINT:

{ PAINTSTRUCT ps;

HDC hDC=BeginPaint(hWnd,&ps);

// получение контекста отображения окна hWnd

int asp\_xy=GetDeviceCaps(hDC,ASPECTXY);

. . . // использование asp\_xy

EndPaint(hWnd,&ps);

}; return 0;

*Определение размера окна*

**Приложение Windows, как правило, не знает заранее размеров отведенных ему окон.** К тому же в любой момент эти размеры можно изменить при помощи мыши. **Поэтому приложение Windows должно уметь определять размеры своих окон.**

Существует два разных метода определения размера окна.

**Первый метод основан на том факте, что при изменении размера окна функция окна получает сообщение** **WM\_SIZE**, параметры которого несут с собой информацию о новых размерах. Параметры сообщения **WM\_SIZE** описывают новый размер окна и способ, которым оно изменило размер:

wParam - способ, при помощи которого окно изменило свой размер;

LOWORD(lParam) - новая ширина клиентской части окна;

HIWORD(lParam) - новая высота клиентской части окна.

Параметр wParam может принимать одно из нескольких значений, символические имена которых начинаются с префикса **SIZE\_** (например, **SIZE\_MAXIMIZED** – окно было максимизировано).

**Обрабатывая сообщение WM\_SIZE, функция окна должна сохранить параметры сообщения и послать сама себе сообщение WM\_PAINT**. Для этого при помощи функции **InvalidateRect** нужно объявить все окно как недействительное и затем вызвать функцию **UpdateWindow**. При обработке сообщения **WM\_PAINT** необходимо пользоваться сохраненными значениями новых размеров клиентской части окна.

**Второй метод определения размеров окна заключается в вызове функций, возвращающих размеры окна по его идентификатору.** Например, функция **GetClientRect** предназначена для определения координат внутренней области окна (эти координаты вычисляются относительно левого верхнего угла внутренней области окна)

void GetClientRect(HWND hwnd, RECT FAR\* lprc);

**Фрагменты оконной функции, определяющей размеры клиентской части окна двумя способами** и осуществляющей вывод в зависимости от этих размеров:

static h, w;

. . .

case WM\_SIZE:

{ w=LOWORD(lParam); h=HIWORD(lParam);

InvalidateRect(hWnd,NULL);

UpdateWindow(hWnd);

}; return 0;

. . .

case WM\_PAINT:

{ PAINTSTRUCT ps;

HDC hDC=BeginPaint(hWnd,&ps);

// 1-й способ вывода в соответствии

// с текущими размерами окна

// вывод с использованием значений w и h

EndPaint(hWnd,&ps);

}; return 0;

или

. . .

case WM\_PAINT:

{ PAINTSTRUCT ps;

HDC hDC=BeginPaint(hWnd,&ps);

// 2-й способ вывода в соответствии с

// текущими размерами окна

Rect r; GetClientRect(hWnd,&r);

. . .

// вывод с использованием значений полей структуры r

EndPaint(hWnd,&ps);

}; return 0;

. . .

*Определение расположения окна*

В предыдущем разделе показано, как определять размеры окна. Другая важная задача - **определение расположения окна на экране**.

**При перемещении окна функция окна получает сообщение WM\_MOVE**, вместе с ним она получает новые координаты внутренней области окна:

wParam - не используется;

LOWORD(lParam) - X-координата верхнего левого угла клиентской части окна;

HIWORD(lParam) - Y-координата верхнего левого угла клиентской части окна.

**Замечание.** Для окон, имеющих стили **WS\_OVERLAPPED** и **WS\_POPUP**, координаты отсчитываются **от верхнего левого угла экрана**. Для окон стиля **WS\_CHILD** эти координаты отсчитываются **от верхнего левого угла внутренней области родительского окна**.

В любой момент времени приложение может также **определить расположение и размер окна**, вызвав функцию:

void GetWindowRect(HWND hwnd, RECT FAR\* lprc);

Эта функция выдает информацию о расположении и размере прямоугольной области, ограничивающей окно, с учетом заголовка, рамки и полос просмотра. Все координаты отсчитываются от верхнего левого угла экрана.

*Метрики текста*

Приложения Windows могут выводить текст с использованием различных шрифтов. Прежде всего, следует отметить, что **все шрифты можно разделить на две группы относительно ширины букв:**

* к первой относятся **шрифты с фиксированной шириной букв** (fixed-pitch font). Все буквы такого шрифта имеют одинаковую ширину;
* вторая группа шрифтов - **шрифты с переменной шириной букв** (variable-pitch font). Для таких шрифтов каждая буква имеет свою ширину.

Кроме того, **шрифты также можно разделить на следующие группы относительно начертания**:

* **растровые шрифты** (raster font) состоят из отдельных пикселов. Если выполнить масштабирование растрового шрифта в сторону увеличения размера букв, наклонные линии и закругления будут изображаться в виде “лестницы”;
* **контурные шрифты** (stroke font) больше подходят для плоттеров. При масштабировании таких шрифтов можно достигнуть лучших результатов, чем при масштабировании растровых. Однако при большом размере букв результат все равно получается неудовлетворительный;
* **масштабируемые шрифты TrueType** сохраняют начертание символов при любом изменении размеров, поэтому они используются чаще всего.

**В контекст отображения по умолчанию выбран так называемый системный шрифт SYSTEM\_FONT**. который используется в Windows, например, для текста в заголовках окон, меню и диалоговых панелях**.**

**Системный шрифт относится к растровым шрифтам с переменной шириной букв.**

Переменная ширина букв усложняет задачу вывода текста, но в составе программного интерфейса Windows имеется **функция, предназначенная для подсчета длины текстовой строки** lpszString длиной cbString

DWORD GetTextExtent(HDC hdc, LPSTR lpszString,int cbString);

Для получения **информации о шрифте, выбранном в контекст устройства,** предназначена функция:

BOOL GetTextMetrics(HDC hdc, TEXTMETRIC FAR\* lptm);

**Хорошо спроектированное приложение позволяет пользователю выбирать для отображения текста произвольные шрифты**. Поэтому приложение **никогда не должно ориентироваться на конкретные размеры шрифта**. Вместо этого следует **определять эти размеры динамически** с помощью специально предназначенных для этого средств.

*Управление окнами Windows*

Окно в Windows можно определить как прямоугольную область на экране. Однако это определение в своей простоте скрывает множество функциональных возможностей под абстрактной идеей окна как основной единицы взаимодействия пользователя и Windows-приложения.

*Окна Windows*

**Окно – это не только область на экране,** посредством которой приложение может представить свой вывод, это еще и **адресат событий и сообщений в среде Windows.**

**Окно идентифицируется по дескриптору окна.** Этот дескриптор (переменная типа HWND) однозначно определяет каждое окно в системе. **Список окон содержит** очевидные **окна приложений и диалоговых панелей**, а также менее очевидные, такие как **рабочий стол, пиктограммы или кнопки**.

**События пользовательского интерфейса объединяются с дескриптором соответствующего окна, образуя сообщение Windows**, и посылаются или помещаются **в очередь приложения** (точнее, в очередь потока), который владеет этим окном.

*Иерархия окон*

**Windows организует свои окна в иерархическую структуру:**

* **Каждое окно имеет родителя, корнем дерева всех окон является окно рабочего стола**, создаваемого Windows при загрузке.
* **Для всех окон верхнего уровня** (для главных окон приложений и других перекрывающихся и всплывающих окон приложений) **родительским окном является рабочий стол**.
* **Родитель дочернего окна – окно верхнего уровня или другое дочернее окно**, более высокого уровня по иерархии.

Между **окнами верхнего уровня** (перекрывающиеся и всплывающие окна) существует еще одна иерархическая связь. **Владельцем окна верхнего уровня может быть другое окно того же уровня**. Окно, имеющее владельца, всегда отображается поверх своего владельца и исчезает при минимизации окна-владельца.

**Типичным случаем владения одного окна верхнего уровня другим является приложение, отображающее диалоговое окно.** Диалоговое окно не является дочерним окном (оно является всплывающим окном), но его владельцем остается окно приложения.

**Наиболее часто употребляемые функции, позволяющие приложению исследовать иерархию окон и находить определенные окна**:

* **GetDesktopWindow** – позволяет приложению получить дескриптор окна рабочего стола.
* **FindWindow** – используется для поиска окна верхнего уровня по имени его класса окна или по заголовку окна;
* **GetParent** – идентифицирует родительское окно указанного окна;
* **GetWindow** – предоставляет наиболее гибкий способ с иерархией окон. В зависимости от значения второго параметра эту функцию можно использовать для получения идентификатора родительского окна, владельца, окон одного уровня или дочерних окон.

*Сообщения управления окнами*

**Типичное окно отвечает на множество сообщений.** **Наиболее часто обрабатываемые** сообщения:

* **WM\_CREATE** - это первое сообщение, которое получает оконная процедура созданного класса. Оно посылается перед тем, как окно станет видимым, и перед завершением функций **CreateWindow** или **CreatWindowEx**. В ответ на это сообщение приложение может выполнить необходимые функции инициализации перед тем, как окно станет видимым.
* **WM\_DESTROY** - посылается оконной процедуре окна, которое уже удалено с экрана и должно быть разрушено.
* **WM\_CLOSE** - указывает, что окно должно быть закрыто. Обработчик по умолчанию в функции **DefWindowProc** при получении этого сообщения вызывает **DestroyWindow**. Приложение может, например, вывести диалоговое окно подтверждения и вызвать **DestroyWindow** только в случае подтверждения пользователем закрытия окна.
* **WM\_QUIT** - это обычно последнее сообщение, которое получает основное окно приложения. Получение этого сообщения приводит к возврату нулевого значения функцией **GetMessage**, что, в свою очередь, приводит к завершению цикла сообщений большинства приложений. Это сообщение требует завершить приложение. Оно генерируется в ответ на вызов функции **PostQuitMessage**.
* **WM\_QUERYENDSESSION** - уведомляет приложение о намерении Windows закончить сеанс. Приложение может возвратить значение FALSE в ответе на это сообщение, предотвратив этим выключение Windows. После обработки сообщения **WM\_QUERYENDSESSION** Windows посылает всем приложениям сообщение **WM\_ENDSESSION** с результатами этой обработки.
* **WM\_ENDSESSION** - посылается приложениям после обработки сообщения **WM\_QUERYENDSESSION**. Оно указывает, должна ли Windows выключиться, или выключение отложено. При указании выключения сеанс Windows может закончиться в любое время после обработки сообщения **WM\_ENDSESSION** всеми приложениями. Поэтому важно, чтобы приложения выполнили все задачи по обеспечению безопасного завершения работы.
* **WM\_ACTIVATE** - указывает, что окно верхнего уровня будет активизировано или деактивизировано. Сообщение сначала посылается окну, которое должно быть деактивизировано, а потом окну, которое должно быть активизировано.
* **WM\_SHOWWINDOW** - указывает, что окно должно быть скрыто или отображено. Окно может быть скрыто в результате вызова функции **ShowWindow** или в результате максимизации другого окна.
* **WM\_ENABLE** – посылается окну, когда оно становится доступным или недоступным. Это может произойти после вызова функции **EnableWindow**. Недоступное окно не может принимать вводимые данные от мыши или клавиатуры.
* **WM\_MOVE** – указывает, что расположение окна изменилось.
* **WM\_SIZE** – указывает, что размер окна был изменен.
* **WM\_SETFOCUS** – указывает, что окно получило фокус клавиатуры.
* **WM\_KILLFOCUS** – указывает, что окно должно потерять фокус клавиатуры.
* **WM\_GETTEXT** – посылается окну, запрашивая копирование текста окна в буфер. Для большинства окон текстом является их заголовок. Для элементов управления типа кнопок и т.д. текстом окна является текст, отображаемый в этих элементах. Обычно это сообщение обрабатывается функцией **DefWindowProc**.
* **WM\_SETTEXT** – запрашивает установку текста окна из содержимого буфера. Функция **DefWindowProc** в ответ на это сообщение устанавливает текст окна и соответственно его отображает.

*Оконная процедура, стандартные оконные процедуры*

**Каждое окно связано с классом окна. Класс окна – это класс, или предоставляемый Windows** (заранее определенный класс), **или определенный пользователем** и зарегистрированный с помощью функции **RegisterClass**.

**Задача класса окна – определение характеристик и функциональных возможностей множества связанных с ним окон.**

Наиболее примечательным, но, разумеется, не единственным свойством класса окна является оконная процедура. **Именно через оконную процедуру реализуется поведение окна.** Отвечая на различные сообщения, **оконная процедура определяет реакцию окна** на события мыши и клавиатуры и изменение его внешнего вида в соответствии с этими событиями.

**Оконная процедура вызывается каждый раз, когда сообщение посылается окну** с помощью функции **SendMessage**, и **каждый раз, когда сообщение отправляется окну** посредством функции **DispatchMessage**. Роль оконной процедуры заключается в обработке посланных и отправленных этому окну сообщений, при **этом она может пользоваться оконной процедурой по умолчанию для обработки не интересующих ее сообщений**.

**Windows поддерживает две оконные процедуры по умолчанию: DefWindowProc и DefDlgProc**.

Функция **DefWindowProc** реализует **поведение по умолчанию типичного окна верхнего уровня**.

Функция **DefDlgProc** используется диалоговыми окнами, и **кроме поведения по умолчанию окон верхнего уровня,** она управляет **переходом фокуса ввода** между элементами управления диалогового окна.

**Кроме оконных процедур по умолчанию, Windows также поддерживает множество классов окон.** Они реализуют такие элементы управления диалоговых окон, как кнопки, поля ввода и другие**. Такие классы называются системными (предопределенными) классами.**

Является ли класс поддерживаемым Windows или определенным приложением, **приложение может использовать любой существующий класс (системный или определенный приложением) для порождения нового класса и реализации его нового или измененного поведения.** Для этого используются механизмы, называемые разбиением на субклассы (подклассы) или объединение в суперклассы.

**Замечание.** Следует только заметить, что

* **Приложение не должно пытаться применить операцию разбиения на субклассы или объединения в суперклассы к окну, принадлежащему другому процессу**.
* **В Win32 изменения в системных классах влияют только на окна этого класса внутри одного приложения**. Эти изменения никак не влияют на окна в других приложениях, потому что приложения Win32 запускаются в разных адресных пространствах; таким образом, они защищены друг от друга.

*Разбиение на субклассы - замещение оконной процедуры*

**Разбиение на суперклассы (subclassing) подразумевает замещение оконной процедуры класса окна другой процедурой**. Это выполняется с помощью вызова функции **SetWindowLong** или **SetClassLong**.

**Замечание.**

* **Вызов SetWindowLong со значением индекса GWL\_WNDPROC замещает оконную процедуру для определенного окна**.
* В противоположность этому, **вызов SetClassLong со значением индекса GCL\_WNDPROC замещает оконную процедуру для всех окон этого класса**, которые будут созданы после вызова **SetClassLong**.

**Пример, в котором системный класс “button” разбивается на субклассы, обеспечивая замещение оконной процедуры.** Замещающая процедура реализует специальную реакцию на получение сообщения **WM\_LBUTTONUP.**

. . .

// глоб. переменная - для хранения адреса старой оконной функции

FARPROC OldWndProc;

// объявление новой оконной функции

LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);

. . .

. . .

// <--- начало фрагмента кода некоторой функции ---

// создание субкласса класса "button"

HWND **hWnd**=CreateWindow**("button"**,"subclass",WS\_VISIBLE|BS\_CENTER,

100,100,200,50,NULL,NULL,hInstance,NULL);

**OldWndProc=(FARPROC)SetWindowLong(hWnd,**

**GWL\_WNDPROC,(LONG)WndProc);**

// --- конец фрагмента кода некоторой функции --->

. . .

// определение новой оконной функции

LRESULT CALLBACK WndProc(HWND hWnd,UINT msg,

WPARAM wParam,LPARAM lParam)

{

switch(msg)

{

case WM\_LBUTTONUP: MessageBeep(-1); break;

**default:**

**return CallWindowProc**(**OldWndProc**,hWnd,msg,wParam,lParam);

}

return 0l;

}

**Стоит обратить внимание на механизм, использованный в новой оконной процедуре при ссылке на прежнюю оконную процедуру**. Прежняя процедура вызывается через функцию **CallWindowProc** из Win32 API.

**Замечание.** В приведенном примере разбиение на субклассы выполняется с помощью функции **SetWindowLong, подразумевая, что это повлияет только на отдельное окно кнопки, для которого вызывается SetWindowLong.** Если вместо этого вызвать **SetClassLong**, то **изменится поведение всех кнопок, созданных приложением впоследствии.**

*Объединение в суперклассы - использование информации о классе*

**Объединение в суперклассы предполагает создание нового класса на основе поведения существующего**. Чтобы применить эту операцию к существующему классу, приложение может использовать функцию **GetClassInfo** для получения описывающей этот класс структуры **WNDCLASS**. **После соответствующего изменения этой структуры ее можно использовать в вызове функции RegisterClass для регистрации нового класса**.

**Пример, который демонстрирует этот механизм объединения в суперклассы.** В этом примере создается новый класс “beepbutton”, функциональные возможности которого основаны на классе по умолчанию “button”. Отметим, что **поведение окон стандартного класса "button" при этом не изменяется**.

. . .

// глоб. переменная - для хранения адреса старой оконной функции

FARPROC OldWndProc;

// объявление новой оконной функции

LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);

. . .

. . .

// <--- начало фрагмента кода некоторой функции ---

// создание суперкласса "button"

**WNDCLASS wc;**

**GetClassInfo(hInstance,"button",&wc);**

**wc.lpszClassName="beepbutton";**

**wc.hInstance=hInstance;**

**OldWndProc=(FARPROC)wc.lpfnWndProc;**

**wc.lpfnWndProc=(WNDPROC)WndProc;**

**RegisterClass(&wc);**

HWND **hWnd**=CreateWindow(**"beepbutton"**,"superclassing",

WS\_VISIBLE|BS\_CENTER, 100,100,200,50,NULL,NULL,

hInstance,NULL);

// --- конец фрагмента кода некоторой функции --->

. . .

LRESULT CALLBACK WndProc(HWND hWnd,UINT msg,

WPARAM wParam,LPARAM lParam)

{

switch(msg)

{

case WM\_LBUTTONDOWN: MessageBeep(-1);

**default:**

**return CallWindowProc(OldWndProc,hWnd,msg,wParam,lParam);**

}

return 0l;

}

**Разница между двумя между двумя механизмами** – разбиением на субклассы и объединением в суперклассы – очевидна с точки зрения их реализации. **Разница в плане применения**:

* **разбиение на субклассы изменяет поведение существующего класса;**
* **объединение в суперклассы создает новый класс на основе поведения существующего класса,**

Другими словами, **при использовании разбиения на субклассы подразумевается изменение каждого окна приложения, принадлежащего этому классу.**

В противоположность этому **объединение в суперклассы действует только на окна, основанные на новом классе; это никак не действует на окна, основанные на исходном классе.**

**3. ЗАДАНИЕ**

На основе простейшего приложения создать приложение, которое демонстрирует основные стили окон (окно верхнего уровня, всплывающее окно с родителем, всплывающее окно без родителя, дочернее окно).

Замечания:

* Для окон различных стилей (главное, всплывающее и дочернее) зарегистрировать отдельные классы окон ("MainWindows", "PopupWindows" и "ChildWindows"), предусмотрев для каждого класса собственный цвет фона и собственную функцию окна.
* При создании окон название стиля окна вывести в заголовке окна, временные и дочерние окна расположить над поверхностью главного окна так, чтобы они не закрывали друг друга.
* Для определения размера экрана используется функция GetSystemMetrics, а для определения рабочей области окна - функция GetClientRect. Определение расположения и размеров окна - функция GetWindowRect. Все эти величины необходимо вывести в рабочие области окон и обновлять по мере изменения состояния окон.

Отчет должен содержать исходный текст приложения, снабженный комментариями и объяснениями поведения окон того или иного стиля.

**4. КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Что определяет (характеризует) стиль класса окна? Что определяет стиль окна?

**Стиль окна определяет внешний вид окна и его поведение.**

Для класса окна также определяется понятие стиля **- стиль класса определяет внешний вид и поведение всех окон, созданных на базе данного класса**.

1. Как задается стиль класса окна?

**Стиль класса окна определяется при регистрации класса окна**. Он задается в виде отдельных битов, для которых определены символические константы с префиксом **CS\_**. **Определенный в классе окна стиль класса окна используется при создании всех окон на базе этого класса.**

1. Для чего задаются стили класса CS\_HREDRAW, CS\_VREDRAW?

Как уже говорилось, если для класса заданы стили **CS\_HREDRAW** и **CS\_VREDRAW**, то **при изменении размеров окна функция окна может получить сообщение** **WM\_PAINT**. В этом случае функция окна должна перерисовать часть окна или все окно.

1. Для чего используется стиль класса CS\_DBLCLKS?

**Стиль CS\_DBLCLKS используется при необходимости отслеживать двойные щелчки мышью.** При этом в функцию окна посылаются сообщения **WM\_LBUTTONDBLCLK** и **WM\_RBUTTONDBLCLK**. Если этот стиль не будет задан, функция окна получит только идущие парами сообщения об одиночном нажатии клавиш мыши **WM\_LBUTTONDOWN** и **WM\_RBUTTONDOWN.**

1. Какая характеристика используется для уточнения внешнего вида и поведения окна, создаваемого на базе какого-либо класса?

Определенный в классе окна **стиль класса окна используется при создании** всех окон на базе этого класса. **Для дальнейшего уточнения внешнего вида и поведения окна используется другая характеристика - стиль окна**. Стиль окна указывается при создании окна функцией **CreateWindow**. Для определения стиля окна используются символические константы с префиксом **WS\_**.

1. Какой внешний вид и поведение обычно характерен для перекрывающихся окон (назвать стиль класса)?

**Для определения стиля перекрывающегося окна существует символическая константа WS\_OVERLAPPEDWINDOW**,

1. Что такое окно-владелец и подчиненное окно? Чем поведение подчиненного окна отличается от поведения обычного перекрывающегося окна?

* **если окно-хозяин сворачивается в пиктограмму,** все окна, которыми оно владеет, становятся **невидимыми;**
* если **сначала свернули в пиктограмму окно, которым владеет другое окно, а затем и окно-хозяина,** то пиктограмма подчиненного окна **исчезнет;**
* **при уничтожении окна-владельца** автоматически **уничтожаются** и все принадлежащие ему окна;
* **обычное перекрывающееся окно, не имеющее окна-владельца**, может располагаться **в любом месте экрана** и принимать любые размеры; **подчиненные окна** располагаются всегда **над поверхностью окна-владельца**, перекрывая его изображение;

1. Для чего чаще всего используются временные (всплывающие, popup) окна?

Другим базовым стилем является стиль **временных окон**, которые обычно используются **для вывода информационных сообщений и остаются на экране непродолжительное время**.

1. Какой внешний вид обычно имеют временные окна (назвать стиль класса)?

Часто для **создания выпадающих окон, имеющих рамку, используется стиль WS\_POPUPWINDOW**

1. Где располагается начало системы координат для перекрывающихся и временных окон?

* **координаты создаваемых функцией CreateWindow перекрывающихся окон указываются по отношению ко всему экрану,** т.е. при создании окна с координатами (0,0) оно будет расположено в верхнем левом углу экрана;
* начало системы координат, используемой при создании временных окон, находится в левом верхнем углу экрана, поэтому **при создании временных окон применяются экранные координаты, как и при создании перекрывающихся окон**;

1. Для чего обычно используются дочерние окна?

**Дочерние окна** чаще всего используются приложениями Windows. Эти окна нужны **для создания таких органов управления, как кнопки или списки**. Все органы управления - дочерние окна.

1. Где располагается начало системы координат для дочерних окон?

* при создании дочернего окна **начало системы координат расположено в левом верхнем углу внутренней поверхности окна-родителя** (но не в верхнем углу экрана, как для перекрывающихся и временных окон);

1. Какой внешний вид обычно имеют дочерние окна (назвать стиль класса)?

**Стиль дочернего окна определяется константой WS\_CHILDWINDOW.**

1. Каковы особенности поведения дочерних окон?

* **дочерние окна должны иметь окно-родителя**; только **дочерние окна могут иметь родителей, перекрывающиеся и временные окна могут иметь окно-хозяина, но не родителя**; у дочерних окон могут быть “братья” (или “сестры”);
* дочерние окна **всегда располагаются на поверхности окна-родителя**;
* при создании дочернего окна **начало системы координат расположено в левом верхнем углу внутренней поверхности окна-родителя** (но не в верхнем углу экрана, как для перекрывающихся и временных окон);
* так как дочернее окно как бы “прилипает” к поверхности окна-родителя, то **при щелчке мышью над поверхностью дочернего окна сообщение от мыши попадет в функцию дочернего, а не родительского окна**;
* **при создании дочернего окна** в качестве девятого параметра (вместо идентификатора меню, которого не может быть у дочернего окна) функции **CreateWindow** **необходимо указать идентификатор дочернего окна** (переменная типа int, любое целое число);
* если для приложения создается несколько дочерних окон**, для каждого окна необходимо указать свой идентификатор,** эти идентификаторы будут использоваться дочерними окнами **при отправлении сообщений родительскому окну**;
* дочернее окно “**прилипает” к поверхности родительского окна и перемещается вместе с ним,** оно никогда **не может выйти за пределы родительского окна**;
* все **дочерние окна скрываются при сворачивании окна-родителя** в пиктограмму **и появляются вновь при восстановлении** родительского окна;
* **при изменении размеров родительского окна дочерние окна получают сообщение WM\_PAINT**, но не получают сообщения **WM\_SIZE**, это сообщение попадает только в родительское окно.

1. Каким простым способом можно сообщить Windows о том, что окно должно иметь горизонтальную и/или вертикальную полосу прокрутки? В чем недостатки такого метода?

**Добавить в окно приложения вертикальную или горизонтальную полосу прокрутки очень просто**. Все, что необходимо сделать, это включить стиль **WS\_VSCROLL** и **WS\_HSCROLL** или оба сразу в описание стиля создаваемого функцией **CreateWindow** окна.

1. Какое значение имеют понятия диапазон полосы прокрутки и ее текущее положение? Как можно изменять эти характеристики?

**Каждая полоса прокрутки имеет соответствующий диапазон** (range) – два целых числа, отражающих минимальное и максимальное значения, **и положение** (position) – местоположение бегунка внутри диапазона. **По умолчанию устанавливается следующий диапазон прокрутки – минимум 0 и максимум 100**, но диапазон легко **изменить на любое другое значение** с помощью функции **SetScrollRange** (для полос прокрутки окна hWnd)

SetScrollRange(hWnd,iBar,iNewMin,iNewMax,bRedraw);

параметр iBar равен либо **SB\_VERT**, либо **SB\_HORZ**, iNewMin и iNewMax являются минимальной и максимальной границами диапазона, а bRedraw устанавливается в TRUE, если необходимо, чтобы Windows перерисовала полосы в соответствии с новыми значениями.

1. Какова сфера ответственности Windows в организации работы пользователя с полосой просмотра?

**Cферы ответственности Windows при поддержке полос прокрутки**:

* управляет логикой работы мыши с полосой прокрутки;
* обеспечивает временную “инверсию цвета” при нажатии на кнопку мыши в полосе прокрутки;
* перемещает бегунок в соответствии с тем, как внутри полосы прокрутки его перемещает пользователь;
* отправляет сообщения полосы прокрутки в оконную процедуру для окна, содержащего полосу прокрутки.

1. Какова сфера ответственности приложения в организации работы пользователя с полосой просмотра?

**Сферы ответственности приложения** по поддержке полос прокрутки:

* инициализация диапазона полосы прокрутки;
* обработка сообщений полосы прокрутки;
* обновление положения бегунка полосы прокрутки.

1. Какие сообщения приходят окну от полос просмотра?

**Windows посылает оконной процедуре сообщения WM\_VSCROLL и WM\_HSCROLL, когда на полосе прокрутки щелкают мышью или перетаскивают бегунок**.

1. Организован ли на системном уровне интерфейс клавиатуры для работы с полосами прокрутки?

* **Windows обеспечивает всю логику работы мыши с полосами прокрутки**. Однако **у полос прокрутки нет интерфейса клавиатуры**. Если приложение желает дублировать клавишами управления курсором некоторые функции полос прокрутки, оно должно **самостоятельно реализовать эту логику при обработке клавиатурных сообщений**.

1. При помощи какой функции можно определить системные метрики Windows?

**Метрики системных компонентов Windows можно определить при помощи функции GetSystemMetrics.**

1. Когда окно получает сообщение WM\_SIZE и какие дополнительные параметры передаются окну вместе с этим сообщением?

**Первый метод основан на том факте, что при изменении размера окна функция окна получает сообщение** **WM\_SIZE**, параметры которого несут с собой информацию о новых размерах. Параметры сообщения **WM\_SIZE** описывают новый размер окна и способ, которым оно изменило размер:

wParam - способ, при помощи которого окно изменило свой размер;

LOWORD(lParam) - новая ширина клиентской части окна;

HIWORD(lParam) - новая высота клиентской части окна.

Параметр wParam может принимать одно из нескольких значений, символические имена которых начинаются с префикса **SIZE\_** (например, **SIZE\_MAXIMIZED** – окно было максимизировано).

1. Для чего предназначена функция GetClientRect?

Функция **GetClientRect** предназначена для определения координат внутренней области окна (эти координаты вычисляются относительно левого верхнего угла внутренней области окна)

1. Когда окно получает сообщение WM\_MOVE и какие дополнительные параметры передаются окну вместе с этим сообщением?

**При перемещении окна функция окна получает сообщение WM\_MOVE**, вместе с ним она получает новые координаты внутренней области окна:

wParam - не используется;

LOWORD(lParam) - X-координата верхнего левого угла клиентской части окна;

HIWORD(lParam) - Y-координата верхнего левого угла клиентской части окна.

1. Для чего предназначена функция GetWindowRect?

В любой момент времени приложение может также **определить расположение и размер окна**, вызвав функцию:

void GetWindowRect(HWND hwnd, RECT FAR\* lprc);

1. Как определить метрики шрифта, установленного в контексте отображения окна?

Для получения **информации о шрифте, выбранном в контекст устройства,** предназначена функция:

BOOL GetTextMetrics(HDC hdc, TEXTMETRIC FAR\* lptm);

**5. СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ**

1. Рихтер, Д. Windows для профессионалов: создание эффективных Win32-приложений (с CD-ROM)./Д. Рихтер. – М.: Русская редакция, 2000. – 752с.
2. Шилдт, Г. "MFC основы программирования". /Г. Шилдт. - Киев: BHV, 1997. – 452с.
3. Румянцев, П.В. Азбука программирования в Win32 API. / П.В. Румянцев. – М.: Радио и связь, 1999. – 242с.
4. Соломон, Г. Внутреннее устройство Microsoft Windows 2000. / Г. Соломон, М. Руссинович. – 4-е изд. – Спб.: Питер, 2001. - 752с.
5. Роббинс, Д. Отладка Windows-приложений (+ CD-ROM)./ Д. Роббинс. - Спб.: BHV, 2001. - 512с.
6. Финогенов, К. Г. Win32. Основы пpогpаммиpования/К.Г. Финогенов. – М.:Диалог - МИФИ, 2002.-416 с.
7. MSDN. <http://msdn.microsoft.com>
8. RSDN. <http://www.rsdn.ru>

Программирование Windows-приложений. Разработка многооконного приложения Windows. [Электронный ресурс]: методические указания к выполнению лабораторной работы №3 для студентов очной формы обучения по направлению подготовки 02.03.03 «Математическое обеспечение и администрирование информационных систем», профиль «Технология программирования».

БЕЛОВ ЕВГЕНИЙ АЛЕКСАНДРОВИЧ

Научный редактор Д.И. Булатицкий

Компьютерный набор Е.А. Белов

Иллюстрации Е.А. Белов

Подписано в печать \_\_.\_\_.2020г. Формат 60х84 1/16 Бумага офсетная. Офсетная печать. Усл.печ.л. 1,63 Уч.-изд.л. 1,63 Тираж 1 экз.

Брянский государственный технический университет

Кафедра «Информатика и программное обеспечение», тел. 56-09-84

241035, Брянск, бульвар 50 лет Октября, 7 БГТУ