**Задание 1**

Цель работы: написать функцию на создание родительского и дочернего окна, которое будет рисоваться внутри родительского окна, с рамками и заголовками

При создании окна, в конструкторе будущего окна мы делаем примерно следующее:

CMainWin::CMainWin( )

{

Create(NULL, "My Programm", WS\_OVERLAPPEDWINDOW);

}

На самом деле за приставкой WS\_ скрывается стиль будущего окна. Ниже идет набор стилей. При задании набора стилей нужно использовать оператор побитового сложения OR ( | );

WS\_BORDER - Создание окна с рамкой.

WS\_CAPTION - Создание окна с заголовком (невозможно использовать одновременно со стилем WS\_DLGFRAME).

WS\_CHILD

WS\_CHILDWINDOW - Создание дочернего окна (невозможно использовать одновременно со стилем WS\_POPUP).

WS\_CLIPCHILDREN - Исключает область, занятую дочерним окном, при выводе в родительское окно.

WS\_CLIPSIBLINGS - Используется совместно со стилем WS\_CHILD для отрисовки в дочернем окне областей клипа, перекрываемых другими окнами.

WS\_DISABLED - Создает окно, которое недоступно.

WS\_DLGFRAME - Создает окно с двойной рамкой, без заголовка.

WS\_GROUP - Позволяет объединять элементы управления в группы.

WS\_HSCROLL - Создает окно с горизонтальной полосой прокрутки.

WS\_MAXIMIZE - Создает окно максимального размера.

WS\_MAXIMIZEBOX - Создает окно с кнопкой развертывания окна.

WS\_MINIMIZE

WS\_ICONIC - Создает первоначально свернутое окно (используется только со стилем WS\_OWERLAPPED).

WS\_MINIMIZEBOX - Создает окно с кнопкой свертывания.

WS\_OVERLAPPED - Создает перекрывающееся окно (которое, как правило, имеет заголовок и WS\_TILED рамку).

WS\_OVERLAPPEDWINDOW - Создает перекрывающееся окно, имеющее стили WS\_OVERLAPPED, WS\_CAPTION, WS\_SYSMENU, WS\_THICKFRAME, WS\_MINIMIZEBOX, WS\_MAXIMIZEBOX.

WS\_POPUP - Создает popup-окно (невозможно использовать совместно со стилем WS\_CHILD.

WS\_POPUPWINDOW - Создает popup-окно, имеющее стили WS\_BORDER, WS\_POPUP, WS\_SYSMENU.

WS\_SYSMENU - Создает окно с кнопкой системного меню (можно использовать только с окнами имеющими строку заголовка).

WS\_TABSTOP - Определяет элементы управления, переход к которым может быть выполнен по клавише TAB.

WS\_THICKFRAME - Создает окно с рамкой, используемой для изменения

WS\_SIZEBOX - размера окна.

WS\_VISIBLE - Создает первоначально неотображаемое окно.

WS\_VSCROLL - Создает окно с вертикальной полосой прокрутки.

Ниже идет программа реализующая окно Windows. При помощи неё поэкспериментируйте со стилями, названными выше.

// My.h

class CMainWin : public CFrameWnd// Класс создания окна

{

public:

CMainWin();// Конструктор

DECLARE\_MESSAGE\_MAP()// Объявление очереди сообщений

};

class CMainApp : public CWinApp// Класс создания приложения

{

public:

BOOL InitInstance();// Точка начала работы приложения

};

// My.cpp

#include <afxwin.h>// Подключение библиотеки MFC

#include "skelet.h"// Содержит определения классов

CMainWin::CMainWin()// Реализация конструктора

{

CRect r;// Отвечает за размер и позицию окна

r.top = r.left = 10;// Расположение окна на экране (сверху и слева)

r.bottom = r.right = 200;// Размеры окна

Create(NULL, "Скелет окна", WS\_OVERLAPPEDWINDOW, r);// Создание окна

}

BOOL CMainApp::InitInstance()// Вместо функции WinMain() (Только для MFC)

{

m\_pMainWnd = new CMainWin;// Создание объекта окна

m\_pMainWnd -> ShowWindow(m\_nCmdShow);// Вывод окна

m\_pMainWnd -> UpdateWindow();// Обновление окна

return TRUE;// TRUE, если окно создано

}

// Реализация очереди сообщений. (пока она пустая)

BEGIN\_MESSAGE\_MAP(CMainWin, CFrameWnd)

END\_MESSAGE\_MAP()

// Создание экземпляра приложения

CMainApp App;

**Задание 2**

Цель: Добавить изображение в дочернее окно по нажатию кнопки в родительском окне

Простая программа, использующая средства GDI

Рассмотрим программу 2-1, которая выводит в главное окно приложения строки текста и простые геометрические фигуры. Результат запуска программы изображен на рис. 2.1.

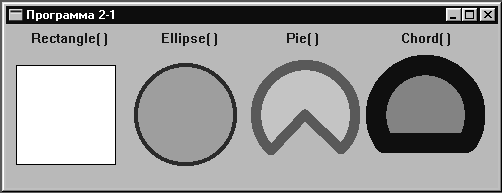


Рис. 2.1. Вывод программы 2-1

/\*Пример 2-1. Вывод текста и простых геометрических фигур\*/

#include <windows.h>

#include <windowsx.h>

/\*Прототипы используемых в программе функций пользователя\*/

BOOL OnCreate(HWND,LPCREATESTRUCT);

void OnPaint(HWND);

void OnDestroy(HWND);

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

/\*Глобальные переменные, доступные всем функциям\*/

HPEN hRedPen,hGreenPen,hBluePen;//Дескрипторы новых перьев

HBRUSH hRedBrush,hGreenBrush, hBlueBrush;//и новых кистей

/\*Главная функция WinMain\*/

int WINAPI WinMain(HINSTANCE hInst,HINSTANCE,LPSTR,int){

char szClassName[]="MainWindow";

char szTitle[]="Программа 2-1";

MSG msg;

WNDCLASS wc;

/\*Зарегистрируем класс главного окна\*/

ZeroMemory(&wc,sizeof(wc));

wc.lpfnWndProc=WndProc;

wc.hInstance=hInst;

wc.hIcon=LoadIcon(NULL,IDI\_APPLICATION);

wc.hCursor=LoadCursor(NULL,IDC\_ARROW);

wc.hbrBackground=CreateSolidBrush(RGB(200,200,100));

wc.lpszClassName=szClassName;

RegisterClass(&wc);

/\*Создадим главное окно и сделаем его видимым\*/

HWND hwnd=CreateWindow(szClassName,szTitle,

WS\_OVERLAPPEDWINDOW,20,20,500,190,

HWND\_DESKTOP,NULL,hInst,NULL);

ShowWindow(hwnd,SW\_SHOWNORMAL);

/\*Организуем цикл обработки сообщений\*/

while(GetMessage(&msg,NULL,0,0))

DispatchMessage(&msg);

return 0;

}

/\*Оконная функция WndProc главного окна\*/

LRESULT CALLBACK WndProc(HWND hwnd,UINT msg,

WPARAM wParam,LPARAM lParam){

switch(msg) {

HANDLE\_MSG(hwnd,WM\_CREATE,OnCreate);

HANDLE\_MSG(hwnd,WM\_PAINT,OnPaint);

HANDLE\_MSG(hwnd,WM\_DESTROY,OnDestroy);

default:

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

}

}

/\*Функия OnCreate обработки сообщений WM\_CREATE\*/

BOOL OnCreate(HWND,LPCREATESTRUCT){

/\*Создадим новые перья\*/

hRedPen=CreatePen(PS\_SOLID,4,RGB(150,0,0));

hGreenPen=CreatePen(PS\_SOLID,10,RGB(0,150,0));

hBluePen=CreatePen(PS\_SOLID,20,RGB(0,0,150));

/\*Создадим новые кисти\*/

hRedBrush=CreateSolidBrush(RGB(255,120,120));

hGreenBrush=CreateSolidBrush(RGB(120,255,120));

hBlueBrush=CreateSolidBrush(RGB(120,120,255));

return TRUE;

}

/\*Функия OnPaint обработки сообщений WM\_PAINT\*/

void OnPaint(HWND hwnd){

PAINTSTRUCT ps;//Структура для функции BeginPaint()

char szText1[]="Rectangle( )";//Просто

char szText2[]="Ellipse( )"; //надписи

char szText3[]="Pie( )"; //над

char szText4[]="Chord( )"; //фигурами

HDC hdc=BeginPaint(hwnd,&ps);//Получим контекст устройства

SetBkColor(hdc,(RGB(200,200,100));//Фон под текстом

SetTextColor(hdc,RGB(0,0,200));//Синий цвет текста

/\*Выведем четыре строки текста\*/

TextOut(hdc,25,5,szText1,strlen(szText1));

TextOut(hdc,155,5,szText2,strlen(szText2));

TextOut(hdc,280,5,szText3,strlen(szText3));

TextOut(hdc,395,5,szText4,strlen(szText4));

/\*Выведем четыре фигуры\*/

Rectangle(hdc,10,40,110,140);//Квадрат

SelectPen(hdc,hRedPen);//Выберем в контекст новое перо

SelectBrush(hdc,hRedBrush);//и новую кисть

Ellipse(hdc,130,40,230,140);//Круг

SelectPen(hdc,hGreenPen);//Выберем новое перо

SelectBrush(hdc,hGreenBrush);//и новую кисть

Pie(hdc,250,40,350,140,350,140,250,140);//Сектор

SelectPen(hdc,hBluePen);//Выберем новое перо

SelectBrush(hdc,hBlueBrush);//и новую кисть

Chord(hdc,370,40,470,140,470,125,370,125);//Сегмент

EndPaint(hwnd,&ps);//Освободим контекст устройства

}

/\*Функция OnDestroy обработки сообщения WM\_DESTROY\*/

void OnDestroy(HWND){

PostQuitMessage(0);//Завершим программу

}

Начальная часть программы (до функции WndProc()) осталась фактически без изменений, за исключением того, что в секцию прототипов добавлены прототипы трех используемых в программе функций OnCreate(), OnPaint() и OnDestroy(), а в секцию глобальных переменных – дескрипторы перьев и кистей, которые будут создаваться в программе; кроме этого, у главного окна увеличен размер и для наглядности ему назначен светло-коричневый цвет.

Оконная процедура состоит всего лишь из трех макросов HANDLE\_MSG по числу обрабатываемых в программе сообщений Windows. В сущности, вся программа, кроме инициализирующей части, заключена в функциях OnCreate(), OnPaint() и OnDest­roy() обработки этих сообщений.

В функцию OnCreate(), которая вызывается в процессе создания окна (в процессе выполнения функции Windows CreateWindow()) обычно выносятся разного рода инициализирующие действия: открытие файлов, создание таймеров, начальные вычисления и пр. В данной программе в этой функции создаются шесть используемых в дальнейшем графических инструментов – три пера разной толщины и разного цвета и три кисти таких же цветов, но более бледных оттенков.

Вся содержательная часть программы – вывод на экран текста и фигур – заключена в прикладной функции OnPaint(). Само рисование осуществляется довольно просто – вызовом соответствующих функций Windows. Для вывода текста предусмотрена функция TextOut(), в числе параметров которой фигурируют координаты начала строки, ее адрес и длина; прямоугольник (в частности, квадрат) рисуется функцией Rectangle() с указанием двух пар координат левого верхнего и правого нижнего углов; эллипс (в частности, круг) отображается функцией Ellipse(), в параметры которой входят те же две пары координат описанного вокруг эллипса прямоугольника. Несколько сложнее выглядит функция Pie() для вывода сектора и функция Chord(), рисующая сегмент. Эти функции, кроме двух пар координат описанных прямоугольников, требуют еще по две пары координат для концов лучей, формирующих вырез в фигуре.

Поскольку вывод любых изображений в окно приложения почти всегда осуществляется в функции OnPaint(), вызываемой в ответ на каждое сообщение WM\_PAINT, следует рассмотреть роль этого сообщения более внимательно.

Обработка сообщений WM\_PAINT

Сообщения WM\_PAINT выделяются среди всех остальных тем, что их обработка включается, практически, в любое приложение Windows, если в нем хоть что-нибудь рисуется на экране. Общее правило рисования заключается в том, что вывод в окно приложения любых графических объектов – текстовых строк, геометрических фигур, отдельных точек, растровых изображений – должен выполняться исключительно в процедуре обработки сообщения WM\_PAINT. Только в этом случае графическое содержимое окна не будет теряться при загораживании данного окна окнами других приложений. Рассматривая реакцию Windows на манипуляции с окном, следует выделить несколько типичных ситуаций.

Если мы с помощью мыши или клавиатуры перемещаем окно приложения по экрану, то этот процесс не затрагивает само приложение. Копирование содержимого окна по мере его перемещения на новые места экрана обеспечивают системные программы Windows. Следовательно, о сохранении содержимого окна в этом случае можно не думать.

Если в приложении имеется линейка меню, то при разворачивании пунктов меню они перекрывают часть окна и, следовательно, при их сворачивании заслоненную ранее область окна надо перерисовать. Эту задачу Windows также берет на себя, сохраняя в своей памяти заслоняемое изображение и выводя его на экран при сворачивании меню.

По-другому обстоит дело, если перерисовка окна потребовалась в результате разворачивания окна, свернутого ранее в пиктограмму, а также при растягивании ранее сжатого окна или при перемещении по пространству главного окна внутренних окон. В этих случаях, когда поврежденной может оказаться значительная часть окна или даже окно целиком (как при разворачивании пиктограммы приложения), Windows уже не берет на себя задачу сохранения и восстановления изображения в окне, а вместо этого посылает в приложение сообщение WM\_PAINT. Программа в ответ на сообщение WM\_PA­INT должна сама восстановить все, что должно изображаться в окне. При этом Windows сообщает в приложение, какая часть окна требует перерисовки, и приложение в принципе может перерисовывать только поврежденную часть окна, что заметно сократило бы временные издержки на вывод изображения. Практически, однако, такой алгоритм рисования составить слишком сложно, если вообще возможно, и в ответ на сообщение WM\_PAINT программа вынуждена заново рисовать все, что должно изображаться в окне, даже если перед этим затертой оказалась лишь малая часть окна.

Главное окно приложения обычно имеет заголовок с управляющими кнопками и толстую рамку, а также во многих случаях линейку меню. Все эти элементы окна образуют нерабочую область окна (nonclient area, “неклиентская” область) и обычно недоступны программе. Остальная часть окна, куда программа может выводить что угодно, называется рабочей областью (client area, область клиента) (рис. 2.2). Восстановление нерабочей области при любых манипуляциях с окном Windows берет на себя, программа обязана восстанавливать только рабочую область.

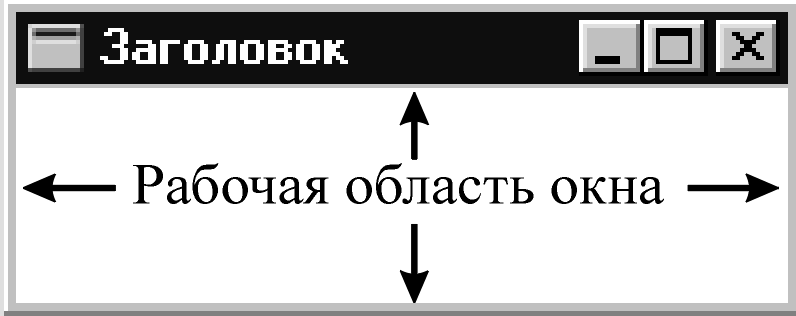


Рис. 2.2. Рабочая область окна

Контекст устройства

Обработка сообщения WM\_PAINT связана с использованием важнейшего поля данных Windows, называемого контекстом устройства.

Контекст устройства представляет собой системную область памяти, закрепляемую за конкретным окном, в которой, в частности, хранятся текущие значения режимов, связанных с рисованием, а также дескрипторы инструментов рисования – кисти, пера, шрифта и пр. Все графические функции GDI используют контекст устройства для определения режима рисования и характеристик применяемых ими инструментов. Например, функция вывода линии получает из контекста устройства (через дескриптор пера) толщину и цвет пера, которым должна быть нарисована линия; функции вывода геометрических фигур, в добавление к характеристикам пера, получают (через дескриптор кисти) цвет и фактуру кисти для закрашивания (заливки) рисуемых фигур; функции вывода текста получают (через дескриптор шрифта) все необходимые характеристики шрифта – гарнитуру, размер, цвет, насыщенность (жирность) и пр. Сам же контекст устройства становится известен графическим функциям GDI через дескриптор контекста, который для всех этих функций служит первым параметром.

Контекст устройства относится к числу системных ресурсов, количество которых в системе может быть ограничено; работа с такого рода ресурсами всегда протекает одинаково: сначала надо получить у системы требуемый ресурс, а закончив работу с ним, вернуть его системе.

Таким образом, для того, чтобы вывести в окно некоторое изображение, необходимо выполнить следующие действия, последовательность которых, в сущности, определяет алгоритм обработки сообщения WM\_PAINT:

с помощью функции Windows BeginPaint() получить у системы контекст устройства для данного окна;

изменить при необходимости режимы рисования (например, режим изображения фона под текстом функцией SetBkMode()) или загрузить (“выбрать”) в контекст инструменты с требуемыми характеристиками, используя для этого функцию Windows SelectObject() или макросы Select­Pen(), SelectBrush() и др.;

сформировать с помощью графических функций GDI (например, Rectangle(), Ellipse(), Pie() и др.) требуемое изображение;

с помощью функции Windows EndPaint() вернуть Win­dows занятый у нее контекст устройства, приведя его предварительно в исходное состояние.

Все перечисленные действия выполняются в прикладной функции OnPaint(), которая всегда начинается с вызова функции Be­gin­Paint(), а завершается вызовом функции EndPaint(). Внутри этих “функциональных скобок” располагаются все предложения, связанные с рисованием.

При вызове функции BeginPaint() в качестве первого параметра указывается дескриптор того окна, в котором предполагается рисовать, а в качестве второго параметра – адрес структурной переменной типа PAINTSTRUCT, которую функция BeginPaint() заполняет некоторыми данными. В случае своего успешного выполнения функция BeginPaint() возвращает значение дескриптора контекста устройства, который имеет тип HDC и в нашей программе поступает в переменную hdc.

Возврат контекста в Windows осуществляется функцией Windows End­Paint(), использующей те же аргументы, что и функция Begin­Paint().

Функции BeginPaint() и EndPaint() используют структурную переменную (в нашем случае ps) типа PAINTSTRUCT. В настоящей программе она никак не используется, однако иногда к ней приходится обращаться, чтобы получить координаты так называемой области вырезки. Эта область описывается в структуре PAINTSTRUCT элементом rcPaint, имеющим тип RECT. Переменные этого типа чрезвычайно широко используются в программах для Windows, поскольку области экрана, с которыми имеет дело Windows, всегда имеют прямоугольную форму. В данном случае переменная ps.rcPaint описывает ту область окна, которая в процессе манипуляций с окном была повреждена и требует перерисовки. В частности, если повреждено было все окно (как это происходит, например, при развертывании приложения, свернутого ранее в значок), область вырезки совпадает с рабочей областью окна.

Функция EndPaint(), завершая процедуру рисования, возвращает Windows занятый у нее контекст устройства. Считается (как это уже упоминалось выше), что возвращать контекст следует в том виде, в каком мы его получили, т. е. не с нашими инструментами, а с инструментами по умолчанию. Поэтому, строго говоря, процедура использования новых инструментов должна выглядеть следующим образом:

/\*Сохраним исходное перо, выберем в контекст новое \*/

HPEN hOldPen=SelectPen(hdc,hRedPen);

/\*Сохраним исходную кисть, выберем в контекст новую\*/

HBRUSH hOldBrush=SelectBrush(hdc,hRedBrush);

...//Рисуем новыми инструментами

/\*Восстановим в контексте исходные перо и кисть\*/

SelectPen(hdc,hOldPen)

SelectPen(hdc,hOldBrush)

/\*Освободим контекст устройства\*/

EndPaint(hwnd,&ps);

Реально, однако, современные системы Windows сами восстанавливают состояние своих объектов после их использования в программе, и пренебрежение описанной выше процедурой в большинстве случаев не приведет к неприятностям.

Такое же замечание можно сделать относительно создания новых инструментов. Вообще говоря, все созданные пользователем объекты (кисти, перья, шрифты, таймеры и пр.) необходимо перед завершением программы удалить, чтобы они не загромождали память, что обычно выполняется в функции OnDestroy() обработки сообщения WM\_DESTROY. Однако система Windows при завершении приложения сама освобождает всю занятую ею память, и эти действия во многих случаях оказываются избыточными.

Использование графических инструментов

Как уже отмечалось, вывод текста осуществляется с помощью функции TextOut(). Функция позволяет задать графические координаты выводимой строки, причем координаты, указываемые в качестве параметров функции, относятся к верхнему левому углу первого знакоместа строки. Поскольку при выводе строк указываются графические координаты, возникает некоторая сложность при выводе на экран нескольких строк; действительно, смещение каждой следующей строки должно быть не меньше высоты предыдущей, а высота строки зависит от вида используемого шрифта. Поэтому в отличие от программ DOS, где можно указывать текстовые координаты строк, в программах для Windows при выводе текста приходится определять характеристики действующего в настоящий момент шрифта.

Для определения характеристик (или, как говорят, метрик) текущего шрифта (дескриптор которого находится в контексте устройства) используется структура TEXTMETRIC. В нее входят 20 элементов, определяющих различные характеристики шрифта; полная высота знакоместа хранится в элементе tmHeight.

Для получения характеристик текущего шрифта в GDI предусмотрена функция GetTextMetrics(), в качестве первого параметра которой передается дескриптор контекста устройства, а в качестве второго – адрес структурной переменной типа TEXTMET­RIC, которую необходимо объявить в программе. Таким образом, вывод двух строк текста непосредственно друг под другом потребует следующих операций:

TEXTMETRIC tm;//Объявляем переменную типа TEXTMETRIC

GetTextMetrics(hdc,&tm);//Получили характеристики шрифта

TextOut(hdc,0,0,s1,strlen(s1);//Вывод строки s1

TextOut(hdc,0,tm.tmHeight,s2,strlen(s2);//Вывод строки s2

Следует иметь в виду, что структура TEXTMETRIC носит чисто информационный характер и используется только для определения, но не для задания характеристик шрифта. Операция придания шрифту требуемых характеристик (гарнитуры, размера и др.) называется созданием шрифта и выполняется с помощью другой структуры – LOGFONT, которая допускает изменение ее элементов.

При использовании шрифта, загруженного или хранящегося по умолчанию в контексте устройства (т. е. не прибегая к процедуре создания шрифта), можно изменить только те его характеристики, которые непосредственно хранятся в контексте устройства. К ним относятся цвет символов и фона под ними, режим фона знакомест (прозрачный или непрозрачный), интервал между символами и атрибут выравнивания текста относительно заданных в функции TextOut() координат.

По умолчанию для символов в контексте устройства задан черный цвет, для фона – белый. Цвет символов изменяется с помощью функции SetTextColor(), цвет фона под символами – функцией SetBkColor(), а режим фона знакомест – функцией SetBk­Mode(). Поскольку эти характеристики хранятся в контексте устройства, настройка цвета будет действовать до следующего изменения или до закрытия контекста.

Перо для рисования линий и фигур создается функцией CreatePen(), в качестве параметров которой указывается тип пера (сплошное, пунктирное и пр.), его толщина в пикселах и цвет. Функция CreatePen() возвращает дескриптор созданного пера, который в нужный момент выбирается в контекст устройства. После загрузки в контекст нового пера все дальнейшее рисование будет осуществляться именно этим пером.

Перо, загружаемое в контекст, не обязательно создавать, его можно взять со склада Windows. Для получения пера со склада используется макрос GetStockPen(), которому в качестве параметра указывается одна из символических констант, присвоенных имеющимся на складе перьям. В частности, таким образом можно получить дескриптор прозрачного, невидимого пера, использование которого позволяет рисовать аккуратные цветные фигуры без черного контура (по умолчанию в контексте находится черное перо толщиной 1 пиксел).

Сплошная кисть создается функцией CreateSolidBrush() с указанием цвета кисти; как и в случае перьев, кисти серого, черного и белого цветов (а также прозрачную) можно получить со склада с помощью макроса GetStockBrush().

Для создания штрихованной кисти предусмотрена функция Cre­ateHatchBrush(), при вызове которой указывается род штриховки (горизонтальная, наклонная вправо, наклонная влево и др.), а также ее цвет. Пространство между штрихами по умолчанию закрашивается белым цветом; изменить цвет заливки можно с помощью той же функции SetBkColor(), которая используется для задания цвета фона под символами текста.

Как уже подчеркивалось, выбор инструментов в контекст устройства осуществляется в функции OnPaint() обработки сообщения WM\_PAINT; создание новых инструментов можно выполнить в любом месте программы, однако наиболее естественно это сделать при обработке сообщения WM\_CREATE для главного окна, т. е. в функции OnCreate().

Задание 3

Цель: научиться изменять размер и форму окна, сделать дочернее окно по нажатию на кнопку в родительском больше в 1,5 раза, с помощью задания координат сделать дочернее окно круглым.

Внешним видом любой формы управляют несколько свойств. Наиболее важным, естественно, является дескриптор окна (Handle). В данной заметке я хочу рассказать о том, как получить нестандартную овальную форму окна (вместо привычной прямоугольной, естественно).

Начну, пожалуй, с того, что существует определенный набор функций API, которые дают возможность изменять вид привычных окон. Разработчиками предусмотрена возможность создания прямоугольной области (CreateRectRgn), правда, у существующих компонентов и так прямоугольная форма. Именно поэтому, куда больший интерес представляют собой функции CreateEllipticRgn (она дает возможность создать область овальной формы) и CreatePlygonRgn (она дает возможность создать область произвольной формы).

Для того чтоб задействовать данные функции, призванные изменить внешний вид формы, необходимо сначала задать область, а после передать ее в функцию SetWindowRgn. В таком случае вид окна будет изменен в соответствии с переданной областью.

Например, окно овальной формы может быть получено следующим образом:

|  |  |
| --- | --- |
| 1  2 | HRGN ellips = CreateEllipticRgn(start\_x, start\_y, width, height);  SetWindowRgn(Handle, ellips, TRUE); |

где параметры start\_x и start\_y – начальные точки «вырезания» овальной формы, а width и height ширина и высота формы соответственно.

Единственный момент: следует помнить о том, что окна, имеющие необычную форму, лишаются своих заголовков, а также стандартных кнопок управления их размерами.

Возможность перемещения формы при помощи указателя мыши может быть реализована так:

|  |  |
| --- | --- |
| 1  2  3 | long SC\_DRAGMOVE = 0xF012;  ReleaseCapture();  SendMessage(Handle,WM\_SYSCOMMAND,SC\_DRAGMOVE,0); |

Стандартные же кнопки могут быть заданы следующим образом:

|  |  |
| --- | --- |
| 1  2  3 | Form1->WindowState=wsMinimized; *// кнопка «свернуть»*  Form1->WindowState=wsMaximized; *// кнопка «развернуть»*  Close(); *// кнопка «закрыть»* |

В итоге, имеем вот такую форму.

