Лабораторная работа №3

Изучение базового шаблона программы для Windows. Принцип отображения информации в окнах.

Работа выполняется в системе программирования Borland Delphi 2.0 в среде Windows 95. Использование средств визуального программирования и классов VCL не допускается, то есть Delphi используется просто как 32-разрядный компилятор с языка Паскаль для операционной системы Windows.

Теоретические положения

Любая программа Windows при своем запуске получает от операционной системы ряд параметров. Эти параметры могут быть проанализированы внутри программы. Для доступа к ним в Паскаль-программе следует использовать следующие идентификаторы:

hInstance: Longint — Хэндл экземпляра приложения. Это число идентифицирует процесс, который представляет собой запущенная программа, в операционной системе. Хэндл экземпляра требуется некоторым функциям API.

CmdLine: PChar — Командная строка, с помощью которой запущено приложение. Содержит полное имя исполняемого модуля, а также параметры командной строки.

CmdShow: integer — Числовая константа, определяющее, каким должно быть показано главное окно программы (см. описание функции ShowWindow). Этот параметр может игнорироваться приложением.

Как правило, программы, написанные для Windows, используют окна для организации пользовательского интерфейса. Из этого правила могут быть исключения, но в подавляющем большинстве случаев программисту приходится работать с окнами.

Основой оконного графического пользовательского интерфейса Windows является механизм сообщений. Когда с окном должно быть произведено некоторое действие, операционная система посылает окну сообщение соответствующего содержания, которое это окно должно обработать. Например, в качестве реакции на сообщение WM_HIDE окно должно исчезнуть с экрана (сделаться невидимым). На самом деле посылка сообщения приводит к вызову некоторой процедуры ("оконной процедуры"), в качестве параметра которой передается небольшой блок данных, который и называется сообщением.

Сообщения могут передаваться оконной процедуре немедленно, а могут помещаться в очередь сообщений, откуда программа должна выбирать их по мере обработки предыдущих сообщений. Хотя сообщения адресуются ОКНАМ, Windows создает очередь сообщений для каждого ПРОЦЕССА (точнее — для каждого потока выполнения, их в рамках одной задачи, т.е. процесса, может быть несколько). Когда в каком-либо потоке программы порождается окно, Windows фиксирует его принадлежность к этому потоку и помещает сообщения для этого окна в очередь сообщений породившего его потока. В простейшем случае, когда в программе не организуются несколько потоков выполнения, очередь сообщений — одна на всю программу, и все сообщения, адресуемые окнам программы, помещаются в эту очередь, откуда программа должна уметь их забирать.

Из очереди сообщения извлекаются в виде блоков данных, структура которых такова:

```
TMsg = packed record
    hwnd: HWND; // хэндл окна-адресата
```

```
    message: UINT; // код (тип) сообщения
    wParam: WPARAM; // параметр сообщения
    lParam: LPARAM; // параметр сообщения
    time: DWORD; // момент посылки сообщения
    pt: TPoint; // положение курсора мыши в момент посылки сообщения
  end;
```

Сообщения содержит в себе информацию о том, какому окну оно адресовано (хэндл окна в поле hwnd), идентификатор сообщения, определяющий тип требуемого действия (поле message), и два параметра, назначение которых специфично для каждого типа сообщений. В ранних версиях Windows поле wParam было 16-разрядным, поле lParam - 32-разрядным, однако в Windows 95 (Win32) оба эти параметра 32-разрядные.

Тип сообщения задается целым числом в поле Message. В Windows определены более 1000 стандартных сообщений; кроме того, программист может вводить дополнительные сообщения. Все сообщения Windows перечислены в файле MESSAGES.PAS в качестве числовых констант. В любом трансляторе для Windows используется устоявшаяся система идентификаторов для обозначения типов сообщений, например: WM_CLOSE, WM_PAINT и т.д. При этом нет необходимости помнить, что WM_CLOSE=10h, WM_PAINT=0Fh и т.д.

В тело сообщения включается также дополнительная информация: о времени посылки сообщения и положении курсора мыши в этот момент. Нетрудно видеть, что размер сообщения фиксирован и вся "полезная информация" должна укладываться в 8 байт (поля wParam и lParam). Если сообщение подразумевает необходимость передачи большего объема информации, то передается указатель на блок памяти, содержащий эту информацию, и откуда процедура, обрабатывающая сообщение, должна эту информацию прочитать.

Практически в любой программе, работающей с окнами Windows, можно логически выделить следующие разделы:

1. Порождение окон.
2. Цикл выборки сообщений из очереди.
3. Оконные процедуры, реализующие логику обработки сообщений в окнах программы.

Вся смысловая нагрузка программы, все ее прикладное значение обычно сосредоточено в третьем из перечисленных разделов, точнее — в процедурах, которые вызываются внутри оконной процедуры. Почти все, что делает программа в Windows, она делает в ответ на сообщение, свидетельствующее о каком-либо внешнем событии. Пользователь щелкнул мышью по кнопке — окно получило сообщение WM_COMMAND — начали выполняться какие-то действия. Пользователь закрыл окно — все окна, которые были под только что закрытым окном на экране получили сообщение WM_PAINT и должны перерисоваться.

Общая структура шаблона программы

В данной работе предлагается составить программу, соответствующую следующему шаблону:

```
uses Messages, Windows;

procedure WndProc(...); stdcall; //Оконная процедура
begin
  case
    //..................... обработка WM_PAINT
```

```
      else
        result:=DefWindowProc(...);
    end; //case
  end;

  begin
    {регистрация оконного класса}
    {порождение окна}
    {вывод окна на экран}
    {цикл обработки сообщений}
  end.
```

Оконные процедуры

Оконная процедура в Delphi описывается как функция, возвращающая 32-битовое целое значение, с использованием директивы **STDCALL.** STDCALL указывает компилятору, что вызов такой функции происходит по правилам вызова процедур системой Windows. Набор параметров оконной процедуры следующий:

```
function WndProc1(hWnd: THandle; Msg: integer;
                  wParam: longint; lParam: longint
): longint; stdcall;
```

Оконная процедура реализует логику обработки сообщений окном и вызывается операционной системой всякий раз, когда окну передается сообщение для обработки. Нетрудно видеть, что параметры оконной процедуры соответствуют полям структуры сообщения, определяющим суть сообщения. Это хэндл окна-получателя, типа сообщения и параметры wParam и lParam. Результат, возвращаемый оконной процедурой, зависит от типа обрабатываемого сообщения. Для некоторых сообщений необходимо вернуть определенный результат, например, чтобы сообщить отправившему сообщение процессу, что оно успешно обработано. Для других сообщений результат не важен, в этом случае рекомендуется возвращать 0.

Должна ли оконная процедура уметь обрабатывать все 1000 с лишним возможных сообщений? С одной стороны — да, ведь все сообщения, адресованные окну, проходят через его оконную процедуру. Но с другой стороны — в Windows определена реакция по умолчанию на все возможные сообщения, поэтому подавляющее большинство сообщений оконная процедура, которую пишет программист, может поручить обработать Windows, вызвав внутри себя

```
DefWindowProc(hwnd,msg,wparam,lparam);
```

Поэтому тело оконной процедуры обычно представляет собой один большой оператор **case**, завершаемый командой "а иначе выполнить обработку по умолчанию". Например так:

```
case Message of
  WM_PAINT: doPaint;     // вызов процедуры реакции на сообщение
  WM_DESTROY: PostQuitMessage(0); // приведет к завершению программы
  else
    result:=DefWindowProc(hwnd,msg,wparam,lparam); // обработка по умолчанию
end; // case
```

Регистрация оконных классов

Для порождения окна прежде всего должен быть описан класс окна. Класс окна — это некоторый блок памяти, существующий в недрах Windows, в котором содержится описание различных параметров, определяющих внешний вид и поведение окон (экземпляров данного

класса), и в частности — указатель на оконную процедуру, обрабатывающую сообщения любого окна данного класса. Как уже понятно из вышесказанного, одновременно может существовать множество окон - экземпляров одного и того же класса, со сходными свойствами и поведением, и окно в Windows всегда порождается как экземпляр некоторого зарегистрированного в системе класса. В Windows имеется несколько предописанных оконных классов, однако обычно программисту приходится описывать и регистрировать свои собственные классы окон.

Для регистрации оконного класса программа должна вызвать функцию

```
function RegisterClassEx(const WndClass: TWndClassEx): ATOM;
```

В случае успешной регистрации класса функция возвращает ненулевое целое значение, которое можно использовать для идентификации оконного класса в других процедурах API; нулевой результат свидетельствует о невозможности создания класса. Параметр WndClass описывает необходимые для создания класса параметры и имеет следующую структуру:

```
TWndClassEx = packed record
    cbSize: UINT; // размер структуры TWndClassEx
    style: UINT;  // битовые флаги, определяющие стиль окна
    lpfnWndProc: TFNWndProc; // адрес оконной процедуры
    cbClsExtra: Integer; // Размер дополнительной памяти класса
    cbWndExtra: Integer; // Размер дополнительной памяти окна
    hInstance: HINST;  // Хэндл экземпляра приложения, которому принадлежит
                       // оконная процедура
    hIcon: HICON;  // Хэндл значка окна (32x32 пиксела)
    hCursor: HCURSOR; //  Хэндл курсора окна по умолчанию
    hbrBackground: HBRUSH; // Хэндл кисти окна по умолчанию или цвет фона
    lpszMenuName: PAnsiChar; //  Оконное меню (имя ресурса)
    lpszClassName: PAnsiChar; // Строка - имя класса, заканчивается #0
    hIconSm: HICON; // Хэндл маленького значка окна (16x16 пикселов)
  end;
```

cbSize должно содержать размер передаваемого блока данных, т.е. SizeOf(TWndClassEx).

style является комбинацией битовых флагов, получаемых при логическом сложении (операция OR) следующих констант (приведены не все возможные стили):

CS_CLASSDC	Один контекст устройства выделяется для всех окон класса и используется при операциях рисования в них. При этом программист должен заботиться о том, чтобы операции рисования в окнах этого класса не происходили в разных потоках одновременно.
CS_DBLCLKS	Окна при двойном щелчке мышью внутри окна получают соответствующие сообщения.
CS_GLOBALCLASS	Позволяет объявить глобальный класс окна (актуально для DLL).
CS_HREDRAW	Перерисовывает все окно, если изменяется горизонтальный размер окна.
CS_NOCLOSE	Убирает команду "Закрыть" из системного меню окна и запрещает кнопку закрытия окна в его правом верхнем углу.
CS_OWNDC	Заранее выделяет уникальный контекст устройства для каждого окна класса..
CS_PARENTDC	Дочернее окно использует контекст устройства окна-родителя. В ряде случаев позволяет ускорить отрисовку дочерних окон.
CS_SAVEBITS	Окно при своем появлении сохраняет образ закрываемой им части экрана в буфере, а при скрытии восстанавливает этот образ из буфера; при этом тем окнам, что были под ним, не посылается сообщение WM_PAINT и они не перерисовываются. Этот стиль обычно используется для небольших по размеру окон, которые появляются на экране на короткое время (диалоги, меню).
CS_VREDRAW	Перерисовывает все окно, если изменяется вертикальный размер окна.

cbClsExtra и cbWndExtra указывают размер дополнительной памяти, резервируемой Windows для всего класса и каждого окна соответственно. Программа может обращаться к этой памяти и использовать ее для своих целей с помощью функций GetClassLong, GetClassWord, GetWindowLong, GetWindowWord, SetClassLong, SetClassWord, SetWindowLong, SetWindowWord (см. Win32 API Help).

hInstance должно содержать хэндл экземпляра приложения, в котором расположена оконная процедура. Инициализируется как

```
wndClass.hInstance:=hInstance;
```

В поле lpszClassname должен быть указатель на Z-строку (оканчивающуюся нулем), в которой содержится имя класса. Помните, что в Delphi тип String может означать как длинные строки, которые можно передавать в качестве Z-строк (ANSIString), так и короткие строки в стиле языка Паскаль предыдущих версий (ShortString), когда первый байт строки содержит ее длину, а нулевого символа в конце нет. Трактовка типа String определяется опцией компилятора {$H+/-}.

В оставшихся нерассмотренными полях указываются хэндлы ресурсов (значки, курсор), которые используются окнами класса по умолчанию. Если эти параметры не указывать (сделать нулевыми), то программа должна сама заботиться об изменении вида курсора при вводе указателя мыши в область окна, о рисовании значка при минимизации окна и т.д.

В системе Windows имеется небольшой набор предопределенных ресурсов и объектов GDI, которые можно использовать при описании классов окон. Их хэндлы можно получить с помощью функций

```
function LoadIcon(hInstance: HINST; lpIconName: PChar): HICON;
function LoadCursor(hInstance: HINST; lpCursorName: PChar): HCURSOR;
function GetStockObject(Index: Integer): HGDIOBJ;
```

При этом при загрузке предопределенного ресурса в функции LoadIcon и LoadCursor передается нулевое значение hInstance, а вместо имени ресурса передается индекс из числа предопределенных констант, например

```
wndClass.hCursor:=loadCursor(0, idc_Arrow);
```

(использовать в качестве курсора стандартный системный курсор в виде стрелки).

Классы окон, созданные приложением, уничтожаются после его завершения автоматически.

Создание окон

Зарегистрировав таким образом класс окна, программа может приступить к созданию самих окон. Создание окна производится функцией

```
function CreateWindow(
  lpClassName: PChar; // Имя зарегистрированного класса окна
  lpWindowName: PChar; // Заголовок окна
  dwStyle: DWORD;  // Стиль окна (комбинация флагов)
  X, Y, nWidth, nHeight: Integer;  // Начальное положение и размеры окна
  hWndParent: HWND; // Хэндл окна-родителя
  hMenu: HMENU;  // Хэндл меню окна
  hInstance: HINST;  // Экземпляр приложения, создающий окно
  lpParam: Pointer // Указатель на параметры для WM_CREATE
): HWND;
```

Существует также функция CreateWindowEx, среди параметров которой присутствует дополнительное поле флагов dwExStyle. остальные параметры и действия функции идентичны рассматриваемой.

Функция CreateWindow или CreateWindowEx создает окно и возвращает в случае успеха операции ненулевой хэндл созданного окна. В противном случае возвращается 0. Хэндл окна используется для идентификации окна при всех последующих операциях с ним. Рассмотрим подробнее некоторые ее параметры.

В качестве X, Y, nWidth и nHeight может быть передано значение CW_USEDEFAULT, при этом начальное положение окон и размеры будут определяться Windows.

Указание ненулевого хэндла родительского окна hWndParent заставляет порождаемое окно автоматически выполнять некоторые дополнительные действия, например: когда родительское окно минимизируется, подчиненные окна скрываются, подчиненные окна всегда располагаются на экране поверх родительского и т.д.

В поле dwStyle может указываться логическая сумма (OR) следующих флагов (указаны не все флаги):

WS_BORDER	Окно имеет границу в виде тонкой линии.
WS_CAPTION	Окно имеет границу и заголовок.
WS_CHILD	Окно является дочерним, т.е. отображается в клиентской области родительского окна.
WS_CLIPCHILDREN	Области, занятые дочерними окнами, не перерисовываются в родительском окне. Используется при создании родительских окон.
WS_CLIPSIBLINGS	Области, занятые перекрывающими данное дочернее дочерними окнами не перерисовываются. Используется при создании дочерних окон.
WS_DISABLED	Создает окно, в запрещенном состоянии, т.е. не воспринимающим событий от мыши и клавиатуры.
WS_DLGFRAME	Обрамление окна в стиле диалога. Окно не может иметь флага заголовка.
WS_HSCROLL	Окно имеет горизонтальную полосу прокрутки.
WS_ICONIC, WS_MINIMIZE	Окно создается минимизированным.
WS_MAXIMIZE	Окно создается максимизированным.
WS_MAXIMIZEBOX	Окно имеет кнопку максимизации.
WS_MINIMIZEBOX	Окно имеет кнопку минимизации.
WS_OVERLAPPED, WS_TILED	Создает экранное окно, имеющее заголовок и рамку.
WS_OVERLAPPEDWINDOW, WS_TILEDWINDOW	Комбинация WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX, WS_MAXIMIZEBOX.
WS_POPUP	Создает всплывающее окно. Несовместимо с WS_CHILD.
WS_POPUPWINDOW	Комбинация WS_BORDER, WS_POPUP, WS_SYSMENU. Чтобы системное меню было видимым, должны быть указаны стили WS_CAPTION и WS_POPUPWINDOW.
WS_SIZEBOX, WS_THICKFRAME	Окно с рамкой для изменения размеров.
WS_SYSMENU	Окно имеет системное меню в левом верхнем углу. Должен присутствовать флаг WS_CAPTION (заголовок).
WS_VISIBLE	Окно изначально видимо на экране.
WS_VSCROLL	Окно имеет вертикальную полосу прокрутки.

Замечание. Стиль WS_OVERLAPPED принимается по умолчанию (значение этой константы равно нулю). При этом управление видом рамки и наличием заголовка не работает (они есть всегда, и рамка всегда "толстая"). Для создания независимых (не-дочерних) окон с полностью управляемым внешним видом используйте WS_POPUP.

В параметре dwExStyle функции CreateWindowEx может указываться логическая сумма (OR) следующих флагов (указаны не все флаги), определяющих дополнительный стиль окна:

WS_EX_ACCEPTFILES Окно поддерживает перенос файлов drag-and-drop.

WS_EX_APPWINDOW Windows 95: Значок окна отображается в панели задач.

WS_EX_CLIENTEDGE Рамка такова, что окно выглядит "утопленным".

WS_EX_CONTROLPARENT Windows 95: Поддержка автоматического переключения между дочерними окнами управления по клавише TAB.

WS_EX_DLGMODALFRAME Окно с широкой рамкой диалога. dwStyle может содержать WS_CAPTION.

WS_EX_MDICHILD Дочернее окно многооконного интерфейса MDI.

WS_EX_OVERLAPPEDWINDOW Windows 95: Комбинация WS_EX_CLIENTEDGE и WS_EX_WINDOWEDGE.

WS_EX_PALETTEWINDOW Windows 95: Окно палитры. Комбинация WS_EX_WINDOWEDGE, WS_EX_TOOLWINDOW и WS_EX_TOPMOST.

WS_EX_STATICEDGE Windows 95: Трехмерная рамка для окон, не предназначенных для ввода информации пользователем.

WS_EX_TOOLWINDOW Windows 95: Инструментальное окно с уменьшенным заголовком, не появляется в панели задач или в списке, появляющемся по ALT+TAB.

WS_EX_TOPMOST Окно переднего плана. Изображается поверх всех окон, не имеющих этого атрибута.

WS_EX_TRANSPARENT Прозрачное окно: все окна находящиеся под ним, получают сообщение WM_PAINT и прорисовываются, после чего сообщение WM_PAINT получает данное окно.

WS_EX_WINDOWEDGE Windows 95: Рамка такова, что окно выглядит выпуклым.

Окно показывается или убирается с экрана при помощи функции

```
function ShowWindow(hWnd: HWND; nCmdShow: Integer): BOOL;
```

В параметре nCmdShow указывается действие, которое необходимо произвести с окном, хэндл которого hWnd. В частности, SW_SHOW означает "показать окно".

Организация обработки очереди сообщений

Сообщения, которые посылаются окнам с буферизацией (post), в отличие от сообщений, посылаемых без буферизации (send), передаются операционной системой не напрямую в оконную процедуру окна-получателя, а помещаются в очередь сообщений. Эта очередь организуется для каждого потока, в котором порождались окна. Если в программе не организовано множество потоков, то можно говорить о единой очереди сообщений программы. Выбор сообщений из очереди и передача их на обработку в соответствующие оконные процедуры возложены на само приложение.

В простейшем случае выбор сообщений организуется в виде следующего цикла:

```
while GetMessage(msg,0,0,0) do begin {получить очередное сообщение}
  TranslateMessage(msg);   {Windows транслирует сообщения от клавиатуры}
  DispatchMessage(msg);    {Windows вызовет оконную процедуру}
end; {выход по wm_quit, на которое GetMessage вернет FALSE}
```

Функция GetMessage с указанными параметрами извлекает очередное сообщение из очереди и помещает его в структуру msg типа TMsg. Если сообщений в очереди нет, то

текущий поток переводится в состояние ожидания и досрочно отдает управление другим потокам. Функция GetMessage возвращает истину при получении любого сообщения, кроме WM_QUIT.

Сообщение WM_QUIT свидетельствует о том, что либо пользователь, либо операционная система подали приложению команду завершиться. При получении сообщения WM_QUIT цикл выборки сообщений завершается и программа, возможно, выполнив какие-то завершающие действия, также завершается.

Любое другое сообщение, отличное от WM_QUIT, приводит к результату TRUE в функции GetMessage и должно быть передано в оконную процедуру. Это выполняется функцией DispatchMessage, при выполнении которой Windows анализирует данные сообщения и вызывает нужную оконную процедуру.

Обычно перед передачей сообщения оконной процедуре выполняются дополнительные действия, самое распространенное из которых — обработка сообщений от клавиатуры функцией TranslateMessage. Эта функция при получении низкоуровневого сообщения о нажатии клавиши WM_KEYDOWN в случае нажатия алфавитно-цифровой клавиши помещает в очередь сообщений сообщение WM_CHAR, содержащее вместо кода клавиши код соответствующего символа в зависимости от установленного в системе языкового драйвера. В дальнейшем это сообщение выбирается из очереди и обрабатывается на общих основаниях.

Посылка сообщений внутри программы

Посылка сообщений окнам не есть прерогатива операционной системы. Любой пользовательский процесс также может посылать сообщения. Для этого есть две основные функции Windows API, реализующие буферизованную и небуферизованную посылку сообщений.

Под буферизованной посылкой сообщений понимается помещение их в очередь того потока, в котором создавалось окно-получатель. Эта операция реализуется функцией

```
function PostMessage(hWnd: HWND; Msg: UINT;
                     wParam: WPARAM; lParam: LPARAM): BOOL;
```

Сообщение типа Msg с параметрами wParam и lParam отправляется окну с хэндлом hWnd. Функция возвращает TRUE, если сообщение отправлено успешно, и FALSE в противном случае. Важно, что после выполнения функции PostMessage, сообщение помещено в очередь, но еще не обработано оконной процедурой окна-получателя.

Посылка сообщения без буферизации выполняется функцией

```
function SendMessage(hWnd: HWND; Msg: UINT;
                     wParam: WPARAM; lParam: LPARAM): LRESULT;
```

При этом сообщение не помещается в очередь, а вместо этого непосредственно вызывается оконная процедура окна-приемника, которая обрабатывает сообщение. Значение, возвращаемое оконной функцией, возвращается в качестве результата функции SendMessage. Таким образом, после возврата из функции SendMessage посланное сообщение уже обработано и известен результат его обработки.

Организация отображения информации в окнах

Для операционной системы DOS были разработаны приложения и оконные библиотеки для текстового режима, в которых при открытии окна закрываемая им область

экрана сохранялась в особом буфере в памяти, а после закрытия окна восстанавливалась из этого буфера. Таким образом, окно обязано было заботиться о восстановлении экрана после своего исчезновения.

Метод сохранения и восстановления части экрана, закрываемой окном, в буфере неудобен для построения графического интерфейса, так как слишком велик объем информации, который требовалось бы сохранять: в полноцветном режиме, когда в видеопамяти отводится 24 или 32 бита на каждый пиксел, буфер для окна, занимающего весь экран, требовал бы более мегабайта оперативной памяти, что нереально. Поэтому в основе графического интерфейса Windows лежит другой принцип: каждое окно в любой момент времени должно уметь отображать себя само. Операционная система при этом отслеживает, какие участки каких окон требуют перерисовки и посылает этим окнам сообщения WM_PAINT, обрабатывая которые, оконные процедуры этих окон должны обеспечить отображение информации в окне.

Это накладывает на программиста некоторые дополнительные ограничения. Так, в программах для DOS программист мог один раз вывести на экран изображение и быть уверенным, что оно останется в целости и сохранности, пока сама программа его не сотрет. В то время как в Windows изображение в окне может быть стерто в силу внешних причин, например, когда другое окно закрыло окно программы, а затем было убрано с экрана. Поэтому данные, необходимые для прорисовки изображения в окне (редактируемый текст или массив статистических данных для диаграммы) должны быть наготове всегда, пока окно видимо на экране.

Обработка сообщения WM_PAINT

Когда операционная система считает, что окно или его часть должны быть перерисованы, оно посылает ему сообщение WM_PAINT, не имеющее параметров. Обработка сообщения WM_PAINT собственно и должна состоять в перерисовке изображения в окне. Оконная процедура при обработке WM_PAINT должна возвращать 0.

Для рисования в программе для Windows используются функции GDI (графического интерфейса устройства), являющиеся унифицированным интерфейсом драйверов дисплея, а также принтеров, графопостроителей и прочих устройств вывода графической информации. Чтобы воспользоваться большинством из этих функций, необходимо получить так называемый "контекст устройства". Фактически, это внутренняя структура данных GDI, содержащая такие сведения, как возможная глубина цвета, разрешающая способность устройства, способ пересчета логических координат в физические, а также об активных инструментах рисования, таких как перо, кисть и шрифт.

Часто требуется перерисовать не все окно целиком, а некоторую его часть, допустим, когда два окна частично перекрывались, и затем одно из них было закрыто. Поэтому при получении сообщения WM_PAINT определен так называемый недействительный регион, в пределах которого требуется производить рисование. Чаще всего недействительный регион является совокупностью прямоугольных областей. Также существует понятие региона отсечения, за пределами которого рисование не производится. Чаще всего регион отсечения по координатам совпадает с недействительным регионом, но это разные понятия.

Обработка сообщения WM_PAINT в оконной процедуре должна строиться примерно следующим образом:

```
case message of
  .......
  WM_PAINT: begin
      hdc:=BeginPaint(hwnd, paintStruct);
```

```
        {вызовы функций GDI}
        EndPaint(hwnd, paintStruct);
      end;
    ..............
end; //case
```

Функция BeginPaint возвращает хэндл контекста устройства для окна с хэндлом hWnd, кроме того, она заполняет структуру paintStruct некоторой полезной информацией. Помимо этих действий, функция BeginPaint устанавливает регион отсечения в соответствии с недействительным регионом, а сам недействительный регион делает действительным. Вызов BeginPaint должен производиться только при обработке сообщения WM_PAINT и обязательно должен сопровождаться вызовом функции EndPaint. Параметр paintStruct должен быть записью типа

```
TPaintStruct = packed record
    hdc: HDC;
    fErase: BOOL;
    rcPaint: TRect;
    fRestore: BOOL;
    fIncUpdate: BOOL;
    rgbReserved: array[0..31] of Byte;
  end;
```

Важными для программиста являются первые три поля этой структуры. В поле hdc дублируется хэндл полученного контекста устройства. В поле fErase содержится признак того, была ли произведена операция стирания фона окна в пределах региона отсечения, это стирание производится с использованием кисти, указанной при создании класса окна, и выполняется Windows автоматически при вызове BeginPaint. Наконец, rcPaint содержит координаты минимальной прямоугольной области, описанной вокруг региона отсечения.

Смысл параметра rcPaint в том, что за пределами этого прямоугольника рисование гарантированно не производится. В то же время, так как регион отсечения может иметь сложную форму, операции рисования могут физически не производиться и в некоторых областях внутри этого прямоугольника. Так как выполняется отсечение, обработчик WM_PAINT может просто перерисовывать окно, ни о чем не заботясь, так как все равно будет прорисована только часть изображения внутри региона отсечения. Тем не менее, использование знаний о размерах региона отсечения позволяет оптимизировать процесс рисования за счет того, что части изображения, лежащие заведомо вне rcPaint, можно просто не рисовать.

Дисциплина ожидания в очереди для сообщения WM_PAINT является особой: это сообщение имеет низший приоритет и может быть получено только в том случае, когда в очереди нет других сообщений. Кроме того, если в очереди уже присутствует сообщение WM_PAINT и приходит новое сообщение WM_PAINT для того же окна, то это приводит просто к добавлению нового региона обновления к региону обновления, соответствующему старому , еще не обработанному, сообщению WM_PAINT. При этом в очереди по-прежнему останется одно сообщение WM_PAINT. Наконец, если в очереди есть сообщение WM_PAINT, но соответствующий регион обновления каким-то образом стал корректным (например, приложение вызвало функцию ValidateRect), то сообщение WM_PAINT будет удалено из очереди, так как рисование больше не требуется. Эту логику прохождения сообщения WM_PAINT обеспечивает операционная система.

Функции рисования GDI

(Конкретные особенности использования функций см. в Help.)

Функция	**Примечание**

GetClientRect	Позволяет получить размеры клиентской области окна
MoveToEx	Перемещение текущей позиции пера
LineTo	Рисование отрезка из текущей позиции и перемещение пера
Ellipse	Рисование закрашенного эллипса, вписанного в указанный прямоугольник
Arc	Рисование дуги эллипса, вписанного в указанный прямоугольник
Rectangle	Рисование закрашенного прямоугольника
RoundRect	Рисование закрашенного прямоугольника со скругленными краями
PolyLine	Рисование ломаной, соединяющей указанные точки
Polygon	Рисование заполненного замкнутого многоугольника по заданным вершинам
TextOut	Выводит строку символов
DrawText	Выводит строку символов с расширенными возможностями форматирования
SetBkMode	Управляет "прозрачностью" текста
SetTextAlign	Устанавливает режим вывода текста
GetPixel	Прочитать цвет пиксела
SetPixel	Нарисовать пиксел

Задание

Выполняется при самостоятельной подготовке:

Изучить теоретическую часть описания ЛР и материал соответствующих лекций.

Выполняется в лаборатории:

1. Включить компьютер. Запустить систему Delphi 2.0.

2. Загрузить исходный текст примера LAB3.PAS, а также модули WINDOWS.PAS и MESSAGES.PAS, изучить логику построения программы для Windows.

3. Откомпилировать и запустить пример. Изучить поведение созданного окна.

4. Написать и отладить программу по индивидуальному заданию (см. ниже). Продемонстрировать результаты работы преподавателю.

5. Завершить работу в Delphi. Оставить компьютер включенным.

Варианты индивидуальных заданий

№	Задание
1.	Стандартное масштабируемое окно, в котором по центру нарисованы Декартовы оси координат с метками 0 и названием осей X и Y, размеры осей должны изменяться при изменении размеров окна.
2.	Масштабируемое окно без заголовка, в котором по центру нарисованы Декартовы оси координат с метками 0 и названием осей X и Y, размеры осей должны изменяться при изменении размеров окна.
3.	Два окна без заголовков. Программа должна завершаться после закрытия всех окон.
4.	"Главное окно" с заголовком и масштабируемое, а также два "вспомогательных" окна стиля Toolbar

(плавающая панель инструментов). Программа должна завершаться после закрытия главного окна.

5. Стандартное масштабируемое окно, в левом верхнем углу которого нарисованы Декартовы оси координат с метками 0 и названием осей X и Y, размеры осей должны составлять 200 пикселов по горизонтали и вертикали.
6. Масштабируемое окно без заголовка, в правом нижнем углу которого нарисованы Декартовы оси координат с метками 0 и названием осей X и Y, размеры осей должны составлять 200 пикселов по горизонтали и вертикали.
7. Два стандартных окна. Программа завершается при закрытии любого окна. При минимизации одного окна другое должно максимизироваться.
8. Два стандартных окна. Программа завершается при закрытии любого окна. При минимизации одного окна другое должно изменять цвет клиентской области на инверсный (с белого на черный и наоборот).
9. Родительское окно и два дочерних, окна стандартного вида, масштабируемые. В дочерних окнах по центру отображаются Декартовы оси координат с метками осей протяженностью 200 пикселов. Программа завершается при закрытии родительского окна.
10. Родительское окно и два дочерних, окна стандартного вида, масштабируемые. В дочерних окнах по центру отображаются Декартовы оси координат с метками осей, протяженность осей изменяется при изменении размеров окна. Программа завершается при закрытии родительского окна.