Федеральное агентство образования

Димитровградский институт технологий,

управления и дизайна

Лабораторная работа №1,2,3,4,5,6
по курсу: “Программирование для Windows NT на языке ‘C’ с использованием Windows API 32 NT”

Выполнил студент гр.ВТ-31:
Потеренко А.Г.
Проверил преподаватель:
Наскальнюк А.Н.

Димитровград 2006г.

Содержание

Задание к лабораторным работам

Лабораторная работа №1
Создать программу, выводящую на экран надпись в рабочем окне программы “Hello Windows NT” разными цветами.
Цель работы:
Приобрести практические навыки создание простых приложений под Windows NT. Освоить работу основных блоков программы таких, как очередь сообщений, оконная процедура, регистрация класса окна, цикл обработки сообщений и обработка сообщений программы. Научиться выводить текст на экран рабочего окна программы, задавать различный фон окна, цвет надписи (выводимой информации в окне) “Hello Windows NT”.

Лабораторная работа №2
Создать программу, выводящую на экран различными цветами и стилями элементы графики.
Цель работы:
Разобраться с работой с сообщением WM_PAINT, приобрести навыки работы с элементами графики.

Лабораторная работа №3
Добавить в программу созданной на первой, второй лабораторной работе горизонтальные и вертикальные линии прокрутки.
Цель работы:
Разобраться с работой сообщений WM_VSCROLL, WM_HVSCROLL.

Лабораторная работа №4
Добавить в программу созданной лабораторной работе № 3 меню. Предусмотреть возможность реакции на горячие клавиши. Меню должно предусматривать выбор для рисования различных элементов графики (окружность, эллипс, прямоугольник и т. п.), а также выбор цвета для рисования. Загрузка и вывода *.bmp файла в отдельном окне. Массив пикселов должен храниться в оперативной памяти.
Цель работы:
Научиться разрабатывать меню, работать с графическими файлами, выделять массивы в оперативной памяти. Обработка сообщения WM_COMMAND.

Лабораторная работа №5
Добавить в программу созданной лабораторной работе №4 окно диалога о программе (About). Окно диалога должно иметь две кнопки OK и Cancel, причем одна из кнопок должна быть нестандартной формы (эллипс, ромб). Для кнопки нестандартной формы реакция должна осуществляться только при попадании указателя мыши во внутреннюю часть.
Цель работы:
Изучить работу с диалогами и его элементами, курсорами, иконками, способы создания нестандартных объектов.

Лабораторная работа №6
Добавить в программу созданной в лабораторной работе № 5 процедуру изменения яркости и контрастности изображения. Процедуры должны выполняться в отдельном потоке.
Цель работы:
Изучить работу с потоками, изменение приоритета потока.

Алгоритм решения и теория к программам

Теория к ЛР №1

1.1. Функции API, использованные в программе.

1.1.1. RegisterClassEx

Регистрирует класс окна, атрибуты которого определены параметром WNDCLASSEX, для последующего использования.

Синтаксис:

```
ATOM RegisterClassEx(
    CONST WNDCLASSEX *lpwcx   //адрес структуры
   );
```

1.2. Структуры API, использованные в программе.

1.2.1. Структура WNDCLASSEX

Структура WNDCLASSEX содержит информацию класса окна. Она используется в Функции GetClassInfoEx и RegisterClassEx.

Синтаксис:

```
typedef struct _WNDCLASSEX {    // wc
    UINT    cbSize;
    UINT    style;
    WNDPROC lpfnWndProc;
    int     cbClsExtra;
    int     cbWndExtra;
    HANDLE  hInstance;
    HICON   hIcon;
    HCURSOR hCursor;
    HBRUSH  hbrBackground;
    LPCTSTR lpszMenuName;
    LPCTSTR lpszClassName;
    HICON   hIconSm;
} WNDCLASSEX;
```

1.3. Процедуры пользователя, использованные в программе.

1.3.1. Процедура WinMain

Точка входа в программу.

```
int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
       flag=false; //Нельзя писать в окне
       MSG msg;    //Структура для сообщений
       ///Инициализация глобальных строк//////////////////////////
       LoadString(hInstance,IDS_LAB1_TITLE,szTitle,MAX_LOADSTRING);
       LoadString(hInstance,IDC_LAB1_CL,szWindowClass,MAX_LOADSTRING);
       LAB1_RegisterClass(hInstance); //Регистрируем класс окна
       if (!InitInstance (hInstance, nCmdShow))
       {
              return FALSE;
       }
       ////////Главный цикл обработки сообщений//////////////////////
```

```
        while (GetMessage(&msg, NULL, 0, 0))
        {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
        }
        return msg.wParam;
}
```

1.3.2. Процедура InitInstance

Функция создания окна.

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   HWND hWnd;
   hInst = hInstance;  //hInst - глобальная переменная
   hWnd = CreateWindow(
                           szWindowClass,
                                        szTitle,
                                        WS_OVERLAPPEDWINDOW,
                                        CW_USEDEFAULT,
                                        0,
                                        CW_USEDEFAULT,
                                        0,
                                        NULL,
                                        NULL,
                                        hInstance,
                                        NULL
                                       );
   if (!hWnd)
   {
          return FALSE;
   }
   ShowWindow(hWnd, nCmdShow);
   return TRUE;
}
```

1.3.3. Процедура LAB1_RegisterClass

Регистрирует окно в системе.

```
ATOM LAB1_RegisterClass(HINSTANCE hInstance)
{
        WNDCLASSEX wcex; //Объявляем переменную типа WNDCLASSEX
        wcex.cbSize = sizeof(WNDCLASSEX);
        wcex.style     = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc = (WNDPROC)WndProc;
        wcex.cbClsExtra       = 0;
        wcex.cbWndExtra       = 0;
        wcex.hInstance = hInstance;
        wcex.hIcon = LoadIcon(hInstance, (LPCTSTR)IDI_MY100);
        wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground = (HBRUSH)(COLOR_3DDKSHADOW+1);
        wcex.lpszMenuName = (LPCSTR)IDC_LAB1_MENU;
        wcex.lpszClassName = szWindowClass;
        wcex.hIconSm = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);
        return RegisterClassEx(&wcex);
}
```

1.3.4. Процедура PRINT_HELLO

Выводит текст в окно.

```
void PRINT_HELLO(HDC hdc,short int R,short int G,short int B)
{
  SetBkColor(hdc,GetSysColor(COLOR_3DDKSHADOW)); //Устанавливаем цвет фона
  char * S;
  S="Hello Windows NT!";
  for (int i=0;i<10;i++)
  {
    SetTextColor(hdc,RGB(R-i*20,G-i*7,B+i)); //Устанавливаем цвет текста
    TextOut(hdc,10,10+20*i,S,lstrlen(S));
  };
  S=NULL;
}
```

1.3.5. Процедура WndProc

Функция обработки сообщения для окна.

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
       int wmId_LAB1;
       wmId_LAB1=LOWORD(wParam); //Идентификатор дочернего окна
       switch (message)
       {
              case WM_COMMAND:
                     switch (wmId_LAB1)
                     {
                            case IDM_ABOUT:
                               DialogBox(hInst,(LPCTSTR)IDD_ABOUTBOX,hWnd,(DLGPROC)About);
                               break;
                            case IDM_EXIT:
                               DestroyWindow(hWnd);
                                break;
                            case IDM_PASTE:
                               flag=true; //Разрешить постоянно выводить надпись
                               InvalidateRect(hWnd,NULL,true);
                                UpdateWindow(hWnd);
                               break;
                            default:
                               return DefWindowProc(hWnd, message, wParam, lParam);
                     }
                     break;
              case WM_PAINT:
                 PAINTSTRUCT ps;
                 HDC hdc;
                  hdc = BeginPaint(hWnd, &ps);
                 if (flag==true)
                        PRINT_HELLO(hdc,200,100,50);
                 EndPaint(hWnd, &ps);
                break;
              case WM_DESTROY:
                 PostQuitMessage(0);
                 break;
              default:
                 return DefWindowProc(hWnd, message, wParam, lParam);
      }
      return 0;
}
```

Теория к ЛР №2

2.1. Функции API, использованные в программе.

2.1.1. CreateBrushIndirect

Создает логическую кисть.

Синтаксис:

```
HBRUSH CreateBrushIndirect(
    CONST LOGBRUSH *lplb      //Указывает на структуру LOGBRUSH, которая содержит
                              //информацию об кисти
);
```

2.1.2. CreatePen

Создает логическое перо.

Синтаксис:

```
HPEN CreatePen(
    int fnPenStyle,     //Стиль пера
    int nWidth,         //Толщина пера
    COLORREF crColor    //Цвет пера
   );
```

2.1.3. DeleteObject

Удаляет hObject из памяти и освобождает связанные с ним ресурсы системы.

Синтаксис

```
BOOL DeleteObject(
    HGDIOBJ hObject     //Дескриптор графического объекта
   );
```

2.2. Структуры API, использованные в программе.

2.2.1. Структура LOGBRUSH

Структура BITMAP определяет тип, ширину, высоту, формат цвета, и значение bitmap.

Синтаксис:

```
typedef struct tagBITMAP {  // bm
   LONG   bmType;        //Определяет тип карты бит. Должен быть ноль.
   LONG   bmWidth;       //Определяет ширину в пикселях.
   LONG   bmHeight;      //Определяет высоту в пикселях.
   LONG   bmWidthBytes;  //Число байт к каждой строке растра и должно быть четным
   WORD   bmPlanes;      //Число цветных плоскостей
   WORD   bmBitsPixel;   //Число смежных бит цвета на каждой плоскости
   LPVOID bmBits;        //Указатель на фактические биты, составляющие карту бит
} BITMAP;
```

2.3. Процедуры пользователя, использованные в программе.

2.3.1. Процедура PRINT_HELLO

Выводит фигуры на экран.

Входные параметры:

1. HDC – контекст окна

```
void PRINT_HELLO(HDC hdc)
{
  int p=0;
  HPEN hPen;
  HBRUSH hBr;
```

```
   LOGBRUSH lb;
   ////////////////Фон закраски//////////////////////////
   lb.lbColor=RGB(200,100,10);
   lb.lbStyle=BS_SOLID;
   lb.lbHatch=NULL;
   hBr = CreateBrushIndirect(&lb);
   SelectObject(hdc, hBr);
   ////////////////////////////////////////////////////////
   hPen = CreatePen(NULL,3,RGB(10,78,90));
   SelectObject(hdc, hPen);
   Arc(hdc,10,10,50,150,p,p,p,p);
   ////////////////////////////////////////////////////////
   hPen = CreatePen(NULL,8,RGB(100,178,90));
   SelectObject(hdc, hPen);
   Arc(hdc,100,100,150,150,p,p,p,p);
   ////////////////////////////////////////////////////////
   hPen = CreatePen(NULL,2,RGB(1,240,90));
   SelectObject(hdc, hPen);
   Arc(hdc,100,300,150,350,170,170,10,600);
   ////////////////////////////////////////////////////////
   hPen = CreatePen(NULL,3,RGB(200,178,90));
   SelectObject(hdc, hPen);
   Rectangle(hdc, 300, 100, 350, 150);
   ////////////////////////////////////////////////////////
   hPen = CreatePen(PS_DASHDOT,1,RGB(255,1,255));
   SelectObject(hdc, hPen);
   Rectangle(hdc, 400, 200, 450, 250);
   ////////////////////////////////////////////////////////
   hPen = CreatePen(PS_DASHDOTDOT,1,RGB(25,1,255));
   SelectObject(hdc, hPen);
   MoveToEx(hdc,0,0,NULL);
   LineTo(hdc,800,200);
   ////////////////////////////////////////////////////////
   DeleteObject(hPen);
   DeleteObject(hBr);
}
```

Теория к ЛР №3

4.1. Функции API, использованные в программе.

4.1.1. SetScrollRange

Устанавливает минимальное и максимальное положения указателя прокрутки.
Синтаксис:

```
BOOL SetScrollRange(
    HWND hWnd,    //Идентификатор окна
    int nBar,     //Одна из констант SB_CTL,SB_HORZ,SB_VERT
    int nMinPos,  //Минимальное положение прокрутки
    int nMaxPos,  //Максимальное положение прокрутки
    BOOL bRedraw  //Не нуль, если полоса перерисовывается
   );
```

4.1.2. SetScrollPos

Устанавливает указатель прокрутки в позицию nPos.
Синтаксис:

```
int SetScrollPos(
    HWND hWnd,    //Идентификатор окна
    int nBar,     //Одна из констант SB_CTL,SB_HORZ,SB_VERT
    int nPos,     //Новая позиция ползунка
    BOOL bRedraw  //Не нуль, если полоса перерисовывается
   );
```

4.2. Сообщения API, использованные в программе.

Для того чтобы в программе работал скроллинг, необходимо создать окно соответствующего стиля. Необходимо также обработать сообщения:

```
/////////////вертикальный скролллинг//////////////////////
case WM_VSCROLL:
   switch(LOWORD(wParam))
       {
              case SB_LINEUP:
                 if (pos_y>min)
                   {
                      pos_y=pos_y-1;
                      SetScrollPos (hWnd,SB_VERT,pos_y,TRUE);
                      y_BUF=ceil(pos_y*(b/max_y));
                   }
                 break;
              case SB_LINEDOWN:
                 if (pos_y<max_y)
                   {
                     pos_y=pos_y+1;
                     SetScrollPos (hWnd,SB_VERT,pos_y,TRUE);
                     y_BUF=ceil(pos_y*(b/max_y));
                   }
                 break;
              case SB_THUMBTRACK:
                pos_y=HIWORD(wParam);
                SetScrollPos (hWnd,SB_VERT,pos_y,TRUE);
                y_BUF=ceil(pos_y*(b/max_y));
              default:
                 break;
       };
   InvalidateRect(hWnd,NULL,true);
   UpdateWindow(hWnd);
```

```
   break;
```

```
//////////////горизонтальный скроллинг//////////////////
case WM_HSCROLL:
  switch(LOWORD(wParam))
       {
              case SB_LINEUP:
                 if (pos_x>min)
                   {
                    pos_x=pos_x-1;
                    SetScrollPos (hWnd,SB_HORZ,pos_x,TRUE);
                    x_BUF=ceil(pos_x*(a/max_x));
                   }
                 break;
             case SB_LINEDOWN:
                 if (pos_x<max_x)
                   {
                    pos_x=pos_x+1;
                    SetScrollPos (hWnd,SB_HORZ,pos_x,TRUE);
                    x_BUF=ceil(pos_x*(a/max_x));
                   }
                 break;
             case SB_THUMBTRACK:
                 pos_x=HIWORD(wParam);
                 SetScrollPos(hWnd,SB_HORZ,pos_x,TRUE);
                 x_BUF=ceil(pos_x*(a/max_x));
             default:
                 break;
       };
  InvalidateRect(hWnd,NULL,true);
  UpdateWindow(hWnd);
  break;
```

При различных действиях пользователя с ползунком, вычисляется положение ползунка и глобальная нулевая точка, относительно которой выводится текст в клиентском окне.

Теория к ЛР №4

4.1. Функции API, использованные в программе.

4.1.1. CreateCompatibleDC

Создает контекст устройства в памяти (DC), совместимый с заданным устройством.

Синтаксис:

```
HDC CreateCompatibleDC(
    HDC hdc       //контекст устройства памяти
   );
```

4.1.2. SelectObject

Выбирает объект в заданный контекст устройства (DC). Новый объект заменяет предыдущий объект того же самого типа.

Синтаксис:

```
HGDIOBJ SelectObject(
    HDC hdc,            //дескриптор контекста устройства
    HGDIOBJ hgdiobj     //дескриптор объекта
   );
```

4.1.3. SetMapMode

Устанавливает режим отображения контекста устройства, который определяет преобразования логических единиц в единицы устройства для GDI и координат осей X и Y.

Синтаксис:

```
int SetMapMode(
    HDC hdc,            //дескриптор контекста устройства
    int fnMapMode       //новый режим карты
   );
```

4.1.4. GetMapMode

Определяет текущий режим отображения.

Синтаксис:

```
int GetMapMode(
    HDC hdc       //дескриптор контекста устройства
   );
```

4.1.5. GetObject

Заполняет буфер данными, которые определяют логический объект. Возвращает только число элементов для логических палитр.

Синтаксис:

```
int GetObject(
    HGDIOBJ hgdiobj,    //дескриптор графического объекта
    int cbBuffer,       //размер буфера для объекта
    LPVOID lpvObject    //указатель на буфер объекта
   );
```

4.1.6. DPtoLP

Преобразует точки устройства в логические точки.

Синтаксис:

```
BOOL DPtoLP(
    HDC hdc,           //дескриптор контекста устройства
    LPPOINT lpPoints,  //указатель на массив точек
    int nCount         //число точек
   );
```

4.1.7. BitBlt

Выполняет передачу битовых блоков данных о цвете, соответствующих прямоугольнику пикселей из заданного исходного контекста устройства в целевой контекст устройства.

Синтаксис:

```
BOOL BitBlt(
    HDC hdcDest,  //дескриптор целевого DC
    int nXDest,   //x-координата левого верхнего угла целевого прямоугольника
    int nYDest,   //y-координата левого верхнего угла целевого прямоугольника
    int nWidth,   //ширина целевого прямоугольника
    int nHeight,  //высота целевого прямоугольника
    HDC hdcSrc,   //дескриптор исходного DC
    int nXSrc,    //x-координата левого верхнего угла исходного прямоугольника
    int nYSrc,    //y-координата левого верхнего угла исходного прямоугольника
    DWORD dwRop   //код растровой операции
   );
```

4.1.8. LoadImage

Загружает изображение, курсор, или bitmap.

Синтаксис:

```
HANDLE LoadImage(
    HINSTANCE hinst,    //дескриптор приложения instance
    LPCTSTR lpszName,   //имя или путь к файлу
    UINT uType,         //тип изображения
    int cxDesired,      //желаемая ширина
    int cyDesired,      //желаемая высота
    UINT fuLoad         //флаг загрузки
   );
```

4.1.9. RegisterHotKey

Определяет комбинацию “горячая” клавиша для всего пространства системы.

Синтаксис:

```
BOOL RegisterHotKey(
    HWND hWnd,          //дескриптор окна, которое примет сообщение WM_HOTKEY
    int id,             //идентификатор горячей клавиши
    UINT fsModifiers,   //определяет клавиши, которые должны быть нажаты в комбинации
    UINT vk             //определяет код виртуальной клавиши
   );
```

4.2. Структуры API, использованные в программе.

4.2.1. Структура BITMAP

Структура BITMAP определяет тип, ширину, высоту, формат цвета, и значение bitmap.

Синтаксис:

```
typedef struct tagBITMAP {  // bm
   LONG   bmType;        //Определяет тип карты бит. Должен быть ноль.
   LONG   bmWidth;       //Определяет ширину в пикселях.
   LONG   bmHeight;      //Определяет высоту в пикселях.
   LONG   bmWidthBytes;  //Число байт к каждой строке растра и должно быть четным
   WORD   bmPlanes;      //Число цветных плоскостей
   WORD   bmBitsPixel;   //Число смежных бит цвета на каждой плоскости
   LPVOID bmBits;        //Указатель на фактические биты, составляющие карту бит
} BITMAP;
```

4.3. Процедуры, использованные в программе.

4.3.1. Процедура DRAW_BMP

Данная процедура выводит в окно из файла изображение в формате BMP.

Входные параметры:

1. HDC – контекст окна
2. int – положение верхнего левого угла относительно начала окна по оси OX
3. int – положение верхнего левого угла относительно начала окна по оси OY

```
void DRAW_BMP(HDC hdc, int xStart, int yStart)
{
  BITMAP bm;
  HDC hdcMem;
  POINT ptSize, ptOrg;
  ////////////////////////////////////////////////////////////////////
  hdcMem = CreateCompatibleDC(hdc);
  SelectObject(hdcMem, hBitmap);
  SetMapMode(hdcMem,GetMapMode(hdc));
  GetObject(hBitmap,sizeof(BITMAP),&bm);
  ////////////////////////////////////////////////////////////////////
  ptOrg.x = 0;
  ptOrg.y = 0;
  ptSize.x = bm.bmWidth;
  ptSize.y = bm.bmHeight;
  ////////////////////////////////////////////////////////////////////
  DPtoLP(hdc,&ptSize,1);
  DPtoLP(hdcMem,&ptOrg,1);
  ////////////////////////////////////////////////////////////////////
  BitBlt(hdc,xStart,yStart,ptSize.x,ptSize.y,hdcMem,ptOrg.x,ptOrg.y,SRCCOPY);
  DeleteDC(hdcMem);
}
```

4.3.2. Процедура DRAW_BRUSH

Данная процедура выводит в окно фигуру установленным цветом.

Входные параметры:

1. HDC – контекст окна
2. int – 1 (эллипс), 2 (окружность), 3 (прямоугольник)
3. int – положение верхнего левого угла относительно начала окна по оси OX
4. int – положение верхнего левого угла относительно начала окна по оси OY
5. COLORREF – цвет выводимой фигуры

```
void DRAW_BRUSH(HDC hdc, int NUM, int xo, int yo, COLORREF CR)
{
    HPEN hPen;
    HBRUSH hBr;
    LOGBRUSH lb;
    ////////////////Фон закраски////////////////////////////
    lb.lbColor=RGB(200,100,10);
    lb.lbStyle=BS_SOLID;
    lb.lbHatch=NULL;
    hBr = CreateBrushIndirect(&lb);
    SelectObject(hdc, hBr);
       ////////////////PEN/////////////////////////////////////
       hPen = CreatePen(NULL,1,CR); //CR - выбирает пользователь
       SelectObject(hdc, hPen);
       switch (NUM)
```

```
       {
              case 1:
           Arc(hdc,xo+10,yo+10,xo+150,yo+250,0,0,0,0);
                    break;
              case 2:
           Arc(hdc,xo+10,yo+10,xo+150,yo+150,0,0,0,0);
                    break;
              case 3:
           Rectangle(hdc,xo+10,yo+10,xo+450,yo+350);
                    break;
       }
       DeleteObject(hPen);
    DeleteObject(hBr);
}
```

4.3.3. Процедура RGB_USER

Данная процедура выводит диалог выбора цвета для фигур.

Выходные параметры:

1. Помещает в переменную CR типа COLORREF цвет, который был выбран пользователем.

```
void RGB_USER()
{
   ////////////////////////////////////////////////////////////////////
   HBRUSH hbrush;
   CHOOSECOLOR cc;
   static COLORREF acrCustClr[16];
   ////////////////////////////////////////////////////////////////////
   ZeroMemory(&cc, sizeof(CHOOSECOLOR));
   cc.lStructSize = sizeof(CHOOSECOLOR);
   cc.hwndOwner = hWnd;
   cc.lpCustColors = (LPDWORD) acrCustClr;
   cc.rgbResult = CR;
   cc.Flags = CC_FULLOPEN | CC_RGBINIT;
   ////////////////////////////////////////////////////////////////////
   if (ChooseColor(&cc)==TRUE)
   {
    hbrush = CreateSolidBrush(cc.rgbResult);
    CR = cc.rgbResult;
   }
}
```

4.3.4. Процедура IZOBR

Данная процедура принимает сообщения от окна диалога выбора фигуры для вывода в окно.

```
LRESULT CALLBACK IZOBR(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
       switch (message)
       {
              case WM_INITDIALOG:
                 CheckRadioButton(hDlg, IDC_RADIO1, IDC_RADIO3, IDC_RADIO1);
                 return TRUE;
              case WM_COMMAND:
                 if (LOWORD(wParam) == IDCANCEL)
                 {
                    EndDialog(hDlg, LOWORD(wParam));
                    return TRUE;
                 };
                 if (LOWORD(wParam) == IDOK)
                 {
                       if (IsDlgButtonChecked(hDlg,IDC_RADIO1)==1)
                            FIGURA=1;
                       if (IsDlgButtonChecked(hDlg,IDC_RADIO2)==1)
                            FIGURA=2;
                       if (IsDlgButtonChecked(hDlg,IDC_RADIO3)==1)
                            FIGURA=3;
                       EndDialog(hDlg, LOWORD(wParam));
                      return TRUE;
                 };
       }
    return FALSE;
}
```

4.3.5. Процедура IZOBR

Данная процедура принимает сообщения от дочернего окна вывода BMP. При инициализации окна загружается изображение в переменную hBitmap. При перерисовки окна изображение копируется в контекст окна.

```
LRESULT CALLBACK ChildProc(HWND hwnd, UINT Message, WPARAM wparam,LPARAM
lparam)
{
       switch (Message)
       {
           case WM_CREATE:
               hBitmap = (HBITMAP)
               LoadImage(hInst,PCHAR(S),IMAGE_BITMAP,0,0,LR_LOADFROMFILE);
           case WM_PAINT:
                  PAINTSTRUCT ps;
                  HDC hdc;
                  hdc = BeginPaint(hwnd, &ps);
                  DRAW_BMP(GetDC(hwnd),0,0);
                  UpdateWindow(hwnd);
                  EndPaint(hwnd, &ps);
                 break;
           case WM_DESTROY:
                  FLAG_OKNO=false; //Теперь можно открыть окно с другим BMP
          default:
                return DefWindowProc(hwnd, Message, wparam, lparam);
   }
   return 0;
}
```

Теория к ЛР №5

5.1. Функции API, использованные в программе.

5.1.1. CreateEllipticRgn

Создает эллиптическую область.

Синтаксис:

```
HRGN CreateEllipticRgn(
 int nLeftRect,   // x -координата верхнего левого угла
 int nTopRect,    // y - координата верхнего левого угла
 int nRightRect,  // x -координата нижнего правого угла
 int nBottomRect  // y - координата нижнего правого угла
);
```

5.1.2. SetWindowRgn

Устанавливает регион окна Windows.

Синтаксис:

```
int SetWindowRgn(
    HWND hWnd,    //идентификатор окна, чей регион окна должен быть установлен
    HRGN hRgn,    //дескриптор региона
    BOOL bRedraw  //флаг перерисовки окна
   );
```

5.2. Процедуры, использованные в программе.

5.2.1. Процедура BUTTON_DRAW

Данная процедура рисует саму кнопку.

Входные параметры:

1. HWND – дескриптор окна
2. int – переменная 0..255 для цвета RGB

```
void BUTTON_DRAW(HWND hw,int k)
{
  HPEN hPen;
  HBRUSH hBr;
  LOGBRUSH lb;
  HDC hdcc=GetDC(hw);
  ////////////////////Фон кнопки//////////////////////////////////////////
  lb.lbColor=RGB(100,60,k);
  lb.lbStyle=BS_SOLID;
  lb.lbHatch=NULL;
  hBr = CreateBrushIndirect(&lb);
  SelectObject(hdcc, hBr);
  ////////////////////Контур кнопки в виде эллипса//////////////////////
  hPen = CreatePen(NULL,0,RGB(1,1,1));
  SelectObject(hdcc, hPen);
  Ellipse(hdcc,0,0,130,50);
  ////////////////////////Текст в кнопке//////////////////////////////
  char * S="Cancel";
  SetBkColor(hdcc,lb.lbColor);
  TextOut(hdcc,42,14,S,strlen(S));
}
```

5.2.2. Процедура About

Обрабатывает сообщения от элементов управления диалогом.

Входные параметры:

1. HWND – дескриптор окна
2. int – переменная 0..255 для цвета RGB

При инициализации диалога, в нем создается дочернее окно стиля BS_OWNERDRAW. Этот стиль говорит о том, что либо мы будем загружать в кнопку BMP файл, либо будем рисовать кнопку сами на его поверхности. При этом нужно теперь обработать сообщение WM_DRAWITEM,

возникающее всякий раз, когда надо перерисовать кнопку. Мы также создаем регион для кнопки: овальной формы и регистрируем его.

```
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
       LPDRAWITEMSTRUCT lpdis;
       HRGN hp;
       switch (message)
       {
              case WM_INITDIALOG:
                //////////////Создаем кнопку в диалоге//////////////////
                hwndSmaller = CreateWindow("button", "",
                                           WS_CHILD | WS_VISIBLE | BS_OWNERDRAW,
                                           315,45,130,50,hDlg,NULL, hInst, NULL);
                ////////Придаем ей вид эллипса//////////////////////////
                hp=CreateEllipticRgn(0,0,130,50);
                SetWindowRgn(hwndSmaller,hp,true);
                return TRUE;
              case WM_COMMAND:
                     if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                     {
                            EndDialog(hDlg, LOWORD(wParam));
                            return TRUE;
                     }
                     //////////Реакция на нажатие овальной кнопки/////////////////
                     if (HIWORD(wParam) == BN_CLICKED)
                     {
                        EndDialog(hDlg, LOWORD(wParam));
                            return TRUE;
                     }
                     break;
              /////////////////Рисование овальной кнопки//////////////////////
              case WM_DRAWITEM:
                     lpdis = (LPDRAWITEMSTRUCT) lParam;
                     if (lpdis->itemState & ODS_SELECTED)
                     {
                         BUTTON_DRAW(hwndSmaller,56);
                            return TRUE;
                     }
                     else
                        BUTTON_DRAW(hwndSmaller,156);
               break;
       }
    return FALSE;
}
```

Теория к ЛР №6

6.1. Функции API, использованные в программе.

6.1.1. TerminateThread

Завершает работу потока.

Синтаксис:

```
BOOL TerminateThread(
    HANDLE hThread,     //дескриптор потока
    DWORD dwExitCode    //код завершения для потока
   );
```

6.1.2. CreateThread

Создает поток, который выполняется в пределах виртуального адресного пространства вызывающего процесса.

Синтаксис:

```
HANDLE CreateThread(
    LPSECURITY_ATTRIBUTES lpThreadAttributes,  //дескриптор защиты
    DWORD dwStackSize,                         //начальный размер стека
    LPTHREAD_START_ROUTINE lpStartAddress,     //функция потока
    LPVOID lpParameter,                        //параметр потока
    DWORD dwCreationFlags,                     //опции создания
    LPDWORD lpThreadId                         //идентификатор потока
   );
```

6.1.3. SetThreadPriority

Устанавливает значение приоритета для заданного потока, обуславливает базовый уровень приоритета потока.

Синтаксис:

```
BOOL SetThreadPriority(
    HANDLE hThread,     //дескриптор потока
    int nPriority       //уровень приоритета потока
   );
```

6.1.4. ResumeThread

Уменьшает счет времени приостановки работы потока. Когда счет времени приостановки работы уменьшается до нуля, выполнение потока продолжается.

Синтаксис:

```
DWORD ResumeThread(
    HANDLE hThread      //дескриптор потока
   );
```

6.1.5. InitializeCriticalSection

Инициализирует критическую секцию.

Синтаксис:

```
VOID InitializeCriticalSection(
    LPCRITICAL_SECTION lpCriticalSection //адрес критической секции объекта
   );
```

6.1.6. DeleteCriticalSection

Функция DeleteCriticalSection удаляет все ресурсы критического объекта секции.

Синтаксис:

```
VOID DeleteCriticalSection(
    LPCRITICAL_SECTION lpCriticalSection //указатель на объект критической секции
   );
```

6.1.7. EnterCriticalSection

Вход в критическую секцию и инициализация соответствующей структуры.

Синтаксис:

```
VOID EnterCriticalSection(
    LPCRITICAL_SECTION lpCriticalSection //указатель на объект критической секции
);
```

6.1.8. LeaveCriticalSection

Позволяет занимать критическую секцию другим потокам.

Синтаксис:

```
VOID LeaveCriticalSection(
    LPCRITICAL_SECTION lpCriticalSection //адрес критической секции объекта
   );
```

6.2. Процедуры, использованные в программе.

6.2.1. Процедура END_THREAD

Данная процедура завершает работу потока hThrd.

```
int END_THREAD()
{
  DWORD IDThread;
  TerminateThread(hThrd,IDThread);
  return 0;
}
```

6.2.2. Процедура BEGIN_THREAD

Данная процедура создает поток hThrd с приоритетом THREAD_PRIORITY_NORMAL. Затем она запускает его.

```
int BEGIN_THREAD()
{
  DWORD IDThread;
  hThrd = CreateThread(NULL,
                       0,
                       (LPTHREAD_START_ROUTINE) ThreadFunc,
                       (LPVOID)hWnd,
                       CREATE_SUSPENDED,
                       &IDThread);
  SetThreadPriority(hThrd,THREAD_PRIORITY_NORMAL);
  ResumeThread(hThrd);
  return 0;
}
```

6.2.3. Процедура WndProc

Данная процедура обрабатывает сообщения от главного окна. Здесь инициализируется критическая секция:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
       switch (message)
       {
         ...
         case WM_DESTROY:
            DeleteCriticalSection(&csCriticalSection_hBitmap);
            ...
         case WM_CREATE:
```

```
            InitializeCriticalSection(&csCriticalSection_hBitmap);
            ...
}
```

Критическая секция нам нужна для hBitmap:

```
HBITMAP hBitmap;
CRITICAL_SECTION csCriticalSection_hBitmap;
```

Критическая секция (critical section) — это небольшой участок кода, требующий монопольного доступа к каким-то общим данным. Она позволяет сделать так, чтобы единовременно только один поток получал доступ к определенному ресурсу. Естественно, система может в любой момент вытеснить ваш поток и подключить к процессору другой, но ни один из потоков, которым нужен занятый вами ресурс, не получит процессорное время до тех пор, пока ваш поток не выйдет за границы критической секции.

6.2.4. Процедура ChildProc

Данная процедура обрабатывает сообщения от дочернего окна, к которое загружается изображение BMP. Здесь реализуется часть логики потоков.

```
LRESULT CALLBACK ChildProc(HWND hwnd, UINT Message, WPARAM wparam,LPARAM lparam)
{
       switch (Message)
       {
           case WM_CREATE:
                 FLAG_CHILD=true;
                 hBitmap = (HBITMAP) LoadImage(hInst,PCHAR(S),IMAGE_BITMAP,0,0,LR_LOADFROMFILE);
                 if (hBitmap!=0) //Если с диска загрузили в память BMP
                    BEGIN_THREAD(); //Инициализируем и запускаем поток
                 break;
           case WM_PAINT:
                 PAINTSTRUCT ps;
                 HDC hdc;
                 hdc = BeginPaint(hwnd, &ps);
                 if (hBitmap!=0)
                 DRAW_BMP(GetDC(hwnd),0,0);  //Выводим в окне BMP
                 EndPaint(hwnd, &ps);
                break;
          case WM_DESTROY:
                 FLAG_CHILD=false;
                 FLAG_END_THREAD=true;     //Разрешаем потоку завершиться
                 END_KONTRAST_USER=false;  //Выходим из нижнего цикла
                 //Ждем ответ от потока - должен дойти до конца в ThreadFunc
                 //и установить флаг true
                 while (FLAG_THREAD_ES_END!=true) {}
                 END_THREAD(); //Останавливаем поток
                 KONTR=0;      //Теперь новое окно откроется с контрастом 0
           default:
                return DefWindowProc(hwnd, Message, wparam, lparam);
   }
   return 0;
}
```

6.2.5. Процедура ThreadFunc

Данная процедура считывает данные, введенные пользователем о контрасте изображения, копирует в память массив пикселов изображения, выполняет алгоритм преобразования пикселов и заносит изменения в hBitmap.

```
DWORD ThreadFunc(HWND hwnd)
{
 ///////////Поток начал работу/////////////////////////////////////////
 FLAG_END_THREAD=false;   //Пока false - поток завершить нельзя
                          //возможна несогласованность в критических секциях
 FLAG_THREAD_ES_END=false; //Сообщение потока о том что он готов завершиться
 ////////////////////////////////////////////////////////////////////////
 BITMAP bm;
 long int CO;
 int * lpvBits;
 long int f;
 int Value;
 while (FLAG_END_THREAD==false)
 {
 ////////////////////////////////////////////////////////////////////////
    EnterCriticalSection(&csCriticalSection_hBitmap);
```

```
    hBitmap=(HBITMAP) LoadImage(hInst,PCHAR(S),IMAGE_BITMAP,0,0,LR_LOADFROMFILE);
    GetObject(hBitmap,sizeof(BITMAP),&bm);
    CO=bm.bmWidthBytes*bm.bmHeight*bm.bmPlanes;
    lpvBits = new int[CO];
    f=GetBitmapBits(hBitmap,CO,lpvBits);
    Value = KONTR; //Загружаем контраст
    LeaveCriticalSection(&csCriticalSection_hBitmap);
    //////////Изменяем контраст/////////////////////////////////////////
    short int mB= 128;
    double vd;
    if (Value > 0)
       {
         vd = 1 + (Value / 10);
       }
    else
       {
         vd = 1 - (sqrt(-Value) / 10);
       }
    for (int i=0;i<CO;i++)
       {
         short int b,g,r;
         b=GetBValue(lpvBits[i]);
         g=GetGValue(lpvBits[i]);
         r=GetRValue(lpvBits[i]);
         b = GET_RGB(mB + ceil((b - mB) * vd));
         g = GET_RGB(mB + ceil((g - mB) * vd));
         r = GET_RGB(mB + ceil((r - mB) * vd));
         lpvBits[i]=RGB(r,g,b);
       }
    ////////////////////////////////////////////////////////////////////
    EnterCriticalSection(&csCriticalSection_hBitmap);
    SetBitmapBits(hBitmap,CO,lpvBits);
    LeaveCriticalSection(&csCriticalSection_hBitmap);
    ////////////////////////////////////////////////////////////////////
    delete [] lpvBits;
    lpvBits=NULL;
    UPDATE_USER(child);
    /////////////Проверка - стоит ли изменить контраст//////////////////
       END_KONTRAST_USER=true;  //Флаг - дальше идти нельзя
       while (END_KONTRAST_USER==true)
       {
              Sleep(100);
       }
 ////////////////////////////////////////////////////////////////////
 }
 //////Сюда добираемся - если пользователь завершает работу///////////
 //////и значение флага FLAG_END_THREAD==true//////////////////////////
 //////В свою очередь мы сообщаем о том, что поток готов завершиться//////
 FLAG_THREAD_ES_END=true;  //После этого можно вызывать END_THREAD()//////
 return 0;
}
```

6.2.6. Процедура DRAW_BMP

Данная процедура выводит изображение на экран. Она использует глобальную переменную hBitmap, поэтому здесь необходима работа с критической секцией.

```
void DRAW_BMP(HDC hdc, int xStart, int yStart)
{
  ...
  EnterCriticalSection(&csCriticalSection_hBitmap);
  GetObject(hBitmap,sizeof(BITMAP),&bm);
  SelectObject(hdcMem, hBitmap);
  LeaveCriticalSection(&csCriticalSection_hBitmap);
  ...
}
```