Лабораторная работа №5

Основы GDI

А-13а-19 Кутдусов Р.К.

Подготовка к лабораторной работе

1. Дескриптор контекста устройства и функции для получения и освобождения контекста:
    a. GetWindowDC

    Функция `GetWindowDC` получает контекст устройства для всего окна, учитывая панель заголовка, меню, и полосы прокрутки. Контекст устройства окна разрешает рисование где-либо в окне потому, что начало координат контекста устройства – это верхний левый угол окна, а не клиентской области.
    `GetWindowDC` определяет атрибуты по умолчанию для контекста устройства окна каждый раз, когда получается контекст устройства. Прежние атрибуты теряются.

```
HDC GetWindowDC(
    HWND hWnd  // дескриптор окна
   );
```

    При успешном завершении функции возвращается дескриптор контекста устройства для указанного окна.

    b. GetDC

    `GetDC` возвращает контекст устройства (DC) окна или другого объекта, имеющего дескриптор. Захватив контекст устройства, нужно вернуть его системе через функцию `ReleaseDC`. Если попытаться получить контекст чего-то, не являющимся устройством, функция возвратит 0.

```
HDC GetDC(
  HWND hWnd   // дескриптор окна
);
```

    c. ReleaseDC

    `ReleaseDC` освобождает ресурсы, которые были заняты при использовании `GetDC` для получения контекста устройства. Функция вызывается при завершении работы с контекстом устройства.

```
int ReleaseDC(
  HWND hWnd,
  HDC  hDC
);
```

2. Выбор и создание инструментов рисования:
    a. GetStockObject

    Функция `GetStockObject` извлекает дескриптор одного из предопределенных (стандартных) перьев, кистей, шрифтов или палитр.

```
HGDIOBJ GetStockObject(
  int fnObject   // тип предопределенного объекта
);
```

    b. SelectObject

    Функция `SelectObject` выбирает объект в заданный контекст устройства (**DC**). Новый объект заменяет предыдущий объект того же самого типа.

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

c. CreateSolidBrush

Функция `CreateSolidBrush` создает кисть, которая имеет заданный сплошной тон.

```
HBRUSH CreateSolidBrush(
  COLORREF crColor   // код цвета кисти
);
```

d. CreatePen

Функция `CreatePen` создает перо, которое имеет заданные стиль, ширину и цвет. Перо может быть впоследствии выбрано в контекст устройства и использовано, чтобы рисовать линии и кривые.

```
HPEN CreatePen(
  int fnPenStyle,    // стиль пера
  int nWidth,        // ширина пера
  COLORREF crColor   // цвет пера
);
```

e. DeleteObject

Функция `DeleteObject` удаляет перо, кисть, шрифт, точечную картинку, регион или палитру, освобождая все системные ресурсы, связанные с объектом. После того, как объект удаляется, его дескриптор становится недоступным.

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

f. SetBkColor

Функция `SetBkColor` устанавливает текущий цвет фона в заданном коде цвета или в самом близком физическом цвете, если устройство не может предоставить указанный код цвета.

```
COLORREF SetBkColor(
  HDC hdc,           // дескриптор DC
  COLORREF crColor   // значение цвета фона
);
```

3. Функции рисования.

```
BOOL LineTo(HDC hDC, int x, int y);
BOOL Line(HDC hdc, int x1, int y1, int x2, int y2)
{
MoveToEx(hdc, x1, y1, NULL); //сделать текущими координаты x1, y1
return LineTo(hdc, x2, y2);
}

BOOL Arc(HDC hDC, int left, int top, int right, int bottom, int x1, int y1, int x2, int
y2);

BOOL Rectangle(HDC hDC, int left, int top, int right, int bottom); // аргументы – это
координанты левого верхнего и правого нижнего углов

BOOL Ellipse(HDC hDC, int x1, int y1, int x2, int y2);

BOOL Chord(HDC hDC, int left, int top, int right, int bottom, int x1, int y1, int x2,
int y2); // Функция соединяет хордой точки начала и конца дуги эллипса и закрашивает
выделенный сегмент текущей кистью.

BOOL Pie(HDC hDC, int left, int top, int right, int bottom, int x1, int y1, int x2, int
y2);
```

4. Функции заливки `ExtFloodFill` и `FillRect`.
   `int FillRect(HDC hDC, CONST RECT *lprc, HBRUSH hbr);`
   `lprc` – закрашиваемый прямоугольник типа RECT.
   `hbr` – кисть
   Функция `ExtFloodFill` закрашивает область поверхности изображения текущей кистью.

```
BOOL ExtFloodFill(
  HDC hdc,           // дескриптор DC
  int nXStart,       // начальная x-координата
  int nYStart,       // начальная y-координата
  COLORREF crColor,  // цвет заливки
  UINT fuFillType    // тип заливки
);
```

5. Функции для работы с пикселами: `GetPixel` и `SetPixel`.
   `SetPixel` устанавливает заданный цвет в точке с указанными координатами:

```
COLORREF SetPixel(HDC hDC, int x, int y, COLORREF crColor);
```

   Пример:

```
SetPixel(hDC, 10,10, RGB(0,0,0));
```

   Функция `GetPixel`, соответственно, возвращает цвет в заданных координатах.

```
COLORREF GetPixel(HDC hDC, int x, int y);
```

6. Режимы рисования: (функция `SetROP2`) `R2_XORPEN`, `R2_NOTXORPEN` и т.д.
   Функция `SetROP2` устанавливает текущий высокоприоритетный режим смешивания. **GDI** использует высокоприоритетный режим смешивания, чтобы объединять перья и внутренние области закрашенных объектов с цветом уже на экране. Высокоприоритетный режим смешивания определяет, как должны комбинироваться цвета кисти или пера и цвета в существующем изображении.

```
int SetROP2(
  HDC hdc,          // дескриптор DC
  int fnDrawMode    // режим рисования
);
```

R2_COPYPEN	Пиксель - цвет пера.
R2_NOP	Пиксель остается неизменным.
R2_NOT	Пиксель - инверсия цвета экрана.
R2_NOTXORPEN	Пиксель - инверсия цвета **R2_XORPEN**.
R2_WHITE	Пиксель всегда **1**.
R2_XORPEN	Пиксель - комбинация цветов в пере и в экране, но не обоих.

   Режимы смешивания определяют, как **GDI** объединяет источник и цвета места назначения, рисуя текущим пером. Режимы смешивания - коды операции бинарного растра, представляя все возможные булевы функции двух переменных, используя бинарные операции **AND**, **OR** и **XOR** (исключающее **OR**), и унарную операцию **NOT**. Режим смешивания только для растровых устройств; он не доступен для векторных устройств.

7. COLORREF
   При определении чистого цвета RGB, значение COLORREF имеет нижеследующую шестнадцатеричную форму:

   0x00bbggrr

   Младший байт содержит величину относительной яркости красного цвета; второй байт содержит величину для зеленого; и третий байт содержит величину для синего. Старший байт должен начинаться с нуля. Максимальное значение для отдельно взятого байта - 0xFF.

8. Методы

1) `InvalidateRect`

Функция `InvalidateRect` добавляет прямоугольник к обновляемому региону заданного окна. Обновляемый регион представляет часть рабочей области окна, которая должна быть перерисована.

```
BOOL InvalidateRect(
  HWND hWnd,            // дескриптор окна
  CONST RECT* lpRect,   // координаты прямоугольника
  BOOL bErase           // состояние очистки
);
```

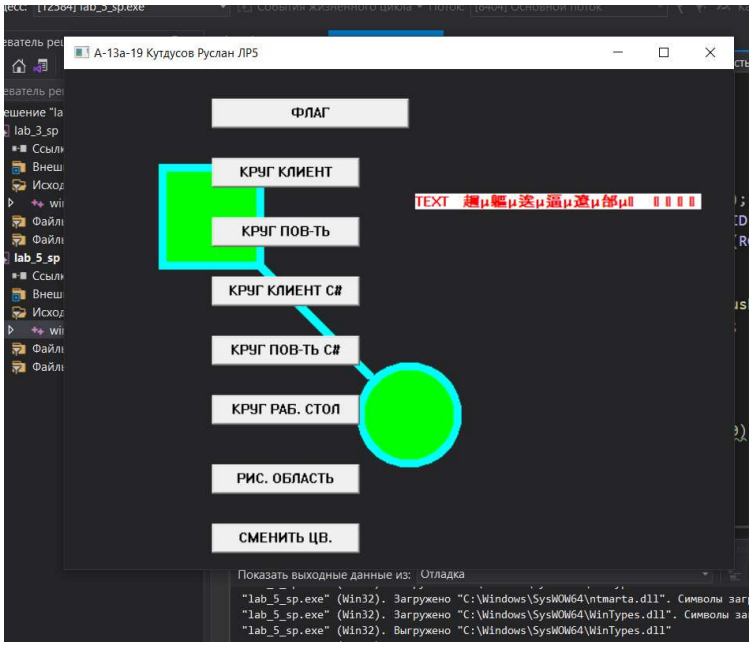
2) `UpdateWindow`

Функция `UpdateWindow` обновляет рабочую область заданного окна, отправляя сообщение `WM_PAINT` окну, если регион обновления окна не пуст. Функция отправляет сообщение `WM_PAINT` непосредственно оконной процедуре указанного окна, обходя очередь приложения. Если регион обновления пуст, никакое сообщение не отправляется.

```
BOOL UpdateWindow(
  HWND hWnd    // дескриптор окна
);
```

Ход работы

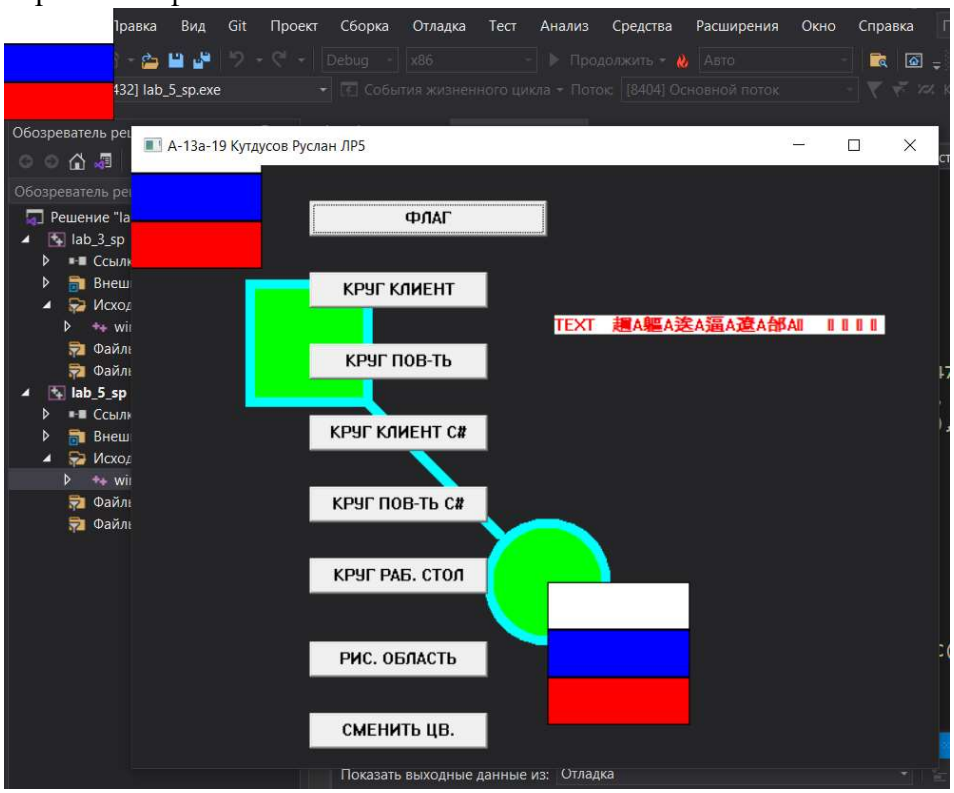
1. Закрасить окно приложения основным (преобладающим) цветом рабочего стола и нарисовать на нем геометрические фигуры и надписи (по событию WM_PAINT).
Окно закрасилось в цвет не самого окна рабочего стола, а в цвет окна программы Visual Studio.




```
case WM_PAINT:
        PAINTSTRUCT ps;
        hDC = BeginPaint(hWnd, &ps);
        if (dsktp_wnd) {
                dsktp_dc = GetDC(dsktp_wnd);
                col_1 = GetPixel(dsktp_dc, 50, 800);
                SelectObject(hDC, CreatePen(PS_SOLID, 8, RGB(0, 255, 255)));
                SelectObject(hDC, CreateSolidBrush(RGB(0, 255, 0)));
                GetWindowRect(hWnd, &rect);
                rect.top = rect.left = 0;
                FillRect(hDC, &rect, CreateSolidBrush(col_1));
                Rectangle(hDC, 100, 100, 200, 200);
                MoveToEx(hDC, 310, 310, NULL);
                LineTo(hDC, 200, 200);
                Ellipse(hDC, 300, 300, 400, 400);
```

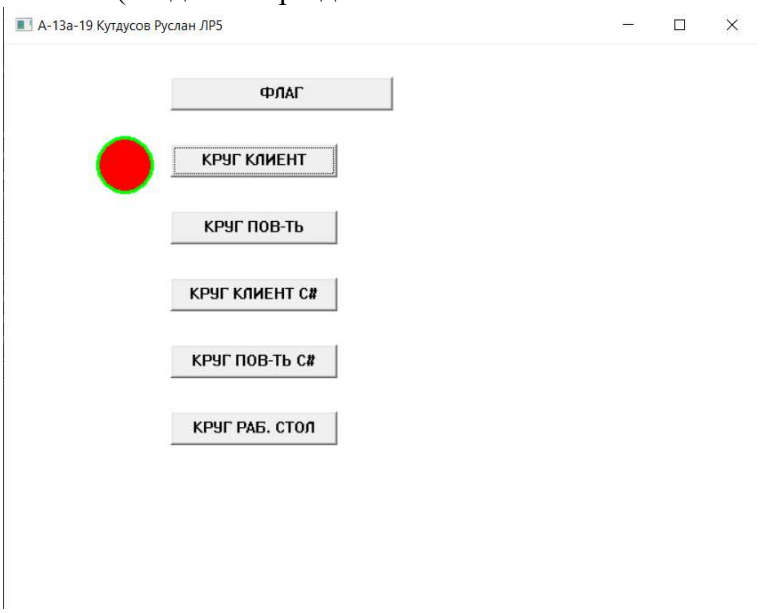
```
                SetTextColor(hDC, RGB(255, 0, 0));
                TextOutW(hDC, 356, 126, L"TEXT", 30);
                ReleaseDC(dsktp_wnd, dsktp_dc);}
        EndPaint(hWnd, &ps);
        break;
```

2. На рабочем столе, в рабочей области окна приложения и на поверхности окна приложения нарисовать флаг России.



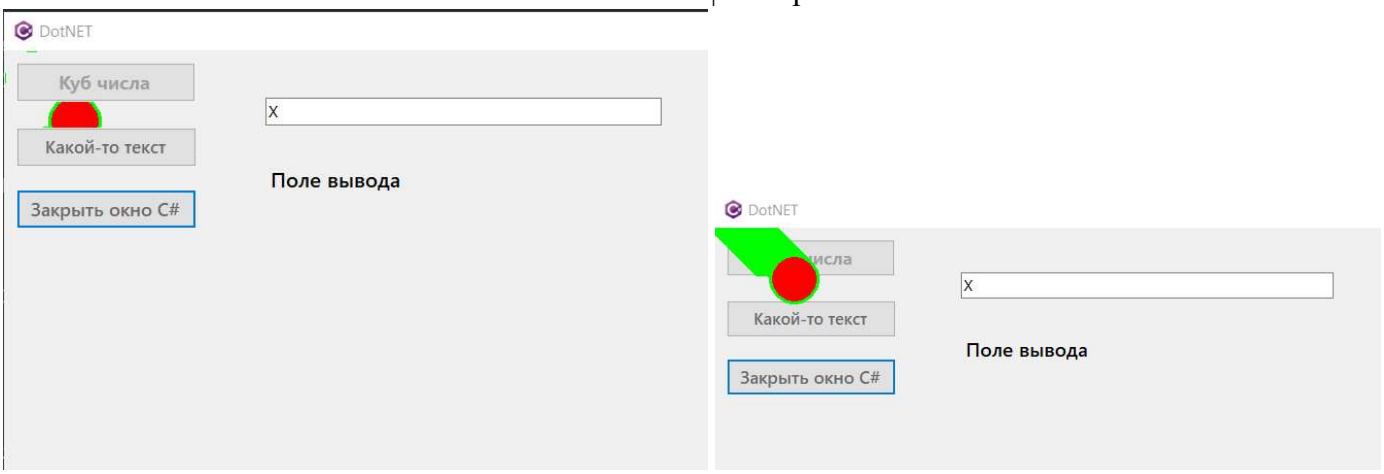

```
case ID_RUSSIA:
        if (dsktp_wnd) {
                RUS(hWnd, GetDC(hWnd), 350, 350, 470, 470);
                RUS(hWnd, GetWindowDC(hWnd), 0, 0, 120, 120);
                RUS(dsktp_wnd, GetDC(dsktp_wnd), 0, 0, 120, 120); }
        break;
void RUS(HWND hWnd, HDC hDC, int x1, int y1, int x2, int y2) {
        SelectObject(hDC, CreatePen(PS_SOLID, 1, RGB(0, 0, 0)));
        SelectObject(hDC, CreateSolidBrush(RGB(255, 255, 255)));
        Rectangle(hDC, x1, y1, x2, y2 - int(2 * (y2 - y1) / 3));
        SelectObject(hDC, CreateSolidBrush(RGB(0, 0, 255)));
        Rectangle(hDC, x1, y1 + int((y2 - y1) / 3), x2, y2 - int((y2 - y1) / 3));
        SelectObject(hDC, CreateSolidBrush(RGB(255, 0, 0)));
        Rectangle(hDC, x1, y1 + int( 2 * (y2 - y1) / 3), x2, y2);
        ReleaseDC(hWnd, hDC); }
```

3. В своем окне, в окне чужой программы (эту программу можно создать на C#, разместив в ее окне управляющие компоненты) и на рабочем столе нарисовать круг, движущийся по окну. Сравнить работу программы при рисовании в рабочей области окна, на поверхности окна и на рабочем столе.
   Окно главного приложения (следа нет при движении в клиентской области и по поверхности):



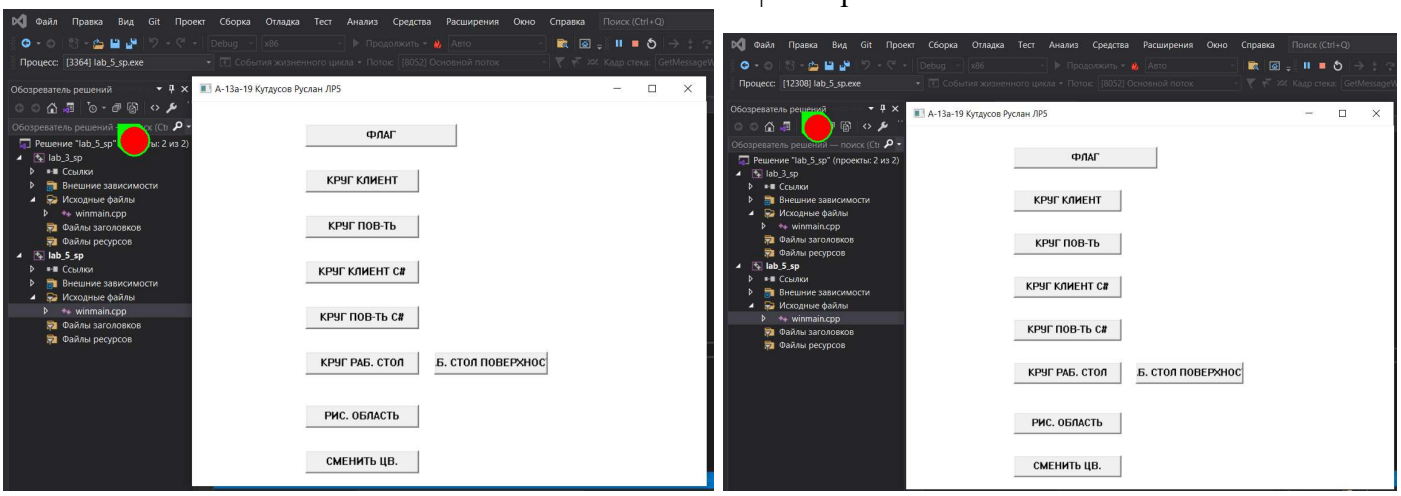
   След после движения круга в окне C#. Если движение в клиентской области, то след пропадает, а если движение на поверхности окна – то след отчётливо виден.

Клиентская область | Поверхность окна



   И рабочий стол.

Клиентская область | Поверхность окна

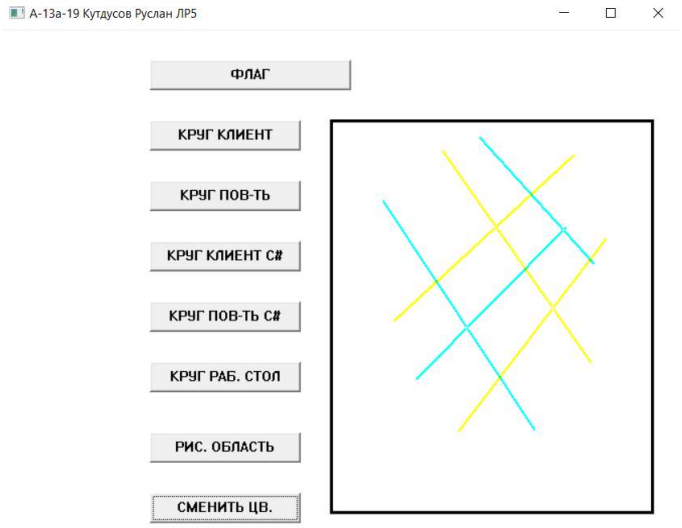


Добавлено движение по таймеру.

```
case ID_WND_CLIENT: // клиентская область главного окна
                    if (!timer_1) {
                           timer_1 = true;
                           r.left = r.top = 0;
                           r.right = r.bottom = 50;
                           SetTimer(hWnd, TIMER_1, 200, NULL);
                    }
                    else {
                           timer_1 = false;
                           KillTimer(hWnd, TIMER_1);
                           InvalidateRect(hWnd, &r, true);
                           UpdateWindow(hWnd);
                    }
                    break;
             case ID_WND: // поверхность главного окна
                    if (!timer_2) {
                           timer_2 = true;
                           r.left = r.top = 0;
                           r.right = r.bottom = 50;
                           SetTimer(hWnd, TIMER_2, 200, NULL);
                    }
                    else {
                           timer_2 = false;
                           KillTimer(hWnd, TIMER_2);
                    }
                    break;
             case ID_CSHARP_CLIENT: // клиентская область окна C#
                    if (!dotnet) {
                           dotnet = FindWindow(NULL, _T("DotNET"));
                           if (dotnet)
                                  ShowWindow(dotnet, SW_SHOWDEFAULT);
                    }
                    else {
                           if (!timer_3) {
                                  timer_3 = true;
                                  r.left = r.top = 0;
                                  r.right = r.bottom = 50;
                                  SetTimer(hWnd, TIMER_3, 200, NULL);
                           }
                           else {
                                  timer_3 = false;
                                  KillTimer(hWnd, TIMER_3);
                           }
                    }
                    break;
             case ID_CSHARP: // поверхность C#
                    if (!dotnet) {
                           dotnet = FindWindow(NULL, _T("DotNET"));
                           if (dotnet)
                                  ShowWindow(dotnet, SW_SHOWDEFAULT);
                    }
                    else {
                           if (!timer_4) {
                                  timer_4 = true;
                                  r.left = r.top = 0;
                                  r.right = r.bottom = 50;
                                  SetTimer(hWnd, TIMER_4, 200, NULL);
                           }
                           else {
                                  timer_4 = false;
                                  KillTimer(hWnd, TIMER_4);
                           }
                    }
                    break;
             case ID_DSKTP_CLIENT: // клиентская область окна рабочего стола
                    if (!timer_5) {
                           timer_5 = true;
                           r.left = r.top = 0;
                           r.right = r.bottom = 50;
```

```
                        SetTimer(hWnd, TIMER_5, 200, NULL);
                }
                else {
                        timer_5 = false;
                        KillTimer(hWnd, TIMER_5);
                }
                break;
        case ID_DSKTP: // поверхность окна рабочего стола
                if (!timer_6) {
                        timer_6 = true;
                        r.left = r.top = 0;
                        r.right = r.bottom = 50;
                        SetTimer(hWnd, TIMER_6, 200, NULL);
                }
                else {
                        timer_6 = false;
                        KillTimer(hWnd, TIMER_6);
                }
                break;
case WM_TIMER:
        switch (vmId) {
        case TIMER_1:
                draw_move_ellipse(hWnd, GetDC(hWnd));
                break;
        case TIMER_2:
                draw_move_ellipse(hWnd, GetWindowDC(hWnd));
                break;
        case TIMER_3:
                draw_move_ellipse(dotnet, GetDC(dotnet));
                break;
        case TIMER_4:
                draw_move_ellipse(dotnet, GetWindowDC(dotnet));
                break;
        case TIMER_5:
                draw_move_ellipse(dsktp_wnd, GetDC(dsktp_wnd));
                break;
        case TIMER_6:
                draw_move_ellipse(dsktp_wnd, GetWindowDC(dsktp_wnd));
                break;
        }
void draw_move_ellipse(HWND hWnd, HDC hDC)
{
        InvalidateRect(hWnd, &r, true);
        UpdateWindow(hWnd);
        SelectObject(hDC, CreatePen(PS_SOLID, 3, RGB(0, 255, 0)));
        SelectObject(hDC, CreateSolidBrush(RGB(255, 0, 0)));
        r.left += 2;
        r.right += 2;
        r.bottom += 2;
        r.top += 2;
        Ellipse(hDC, r.left, r.top, r.right, r.bottom);
        ReleaseDC(hWnd, hDC);
}
```

4. Приложение должно реализовать функции простейшего графического редактора, в котором пользователь может рисовать (линиями двух заданных цветов) в пределах заданного прямоугольного поля (поле закрашивается заданным цветом).




```
case WM_LBUTTONDOWN:
                if (hDC) {
                        point.x = LOWORD(lParam);
                        point.y = HIWORD(lParam);
                        if (point.x >= 330 && point.x <= 650 && point.y <= 480 && point.y >= 90) {
                                clicked = true;
                                MoveToEx(hDC, point.x, point.y, NULL);
                                SelectObject(hDC, CreatePen(PS_SOLID, 2, main_color));
                                SetROP2(hDC, R2_NOTXORPEN);
                        }
                old.x = point.x;
                old.y = point.y;
        }
        break;
case WM_MOUSEMOVE:
                if (clicked) {
                        cur.x = LOWORD(lParam);
                        cur.y = HIWORD(lParam);
                if (cur.x >= 330 && cur.x <= 650 && cur.y <= 480 && cur.y >= 90) {
                        LineTo(hDC, old.x, old.y);
                        MoveToEx(hDC, point.x, point.y, NULL);
                        LineTo(hDC, cur.x, cur.y);
                        MoveToEx(hDC, point.x, point.y, NULL);
                }
                old.x = cur.x;
                old.y = cur.y;
        }
        break;
case WM_LBUTTONUP:
        clicked = false;
        break;
```