МИНЕСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ

Отчет к лабораторной работе № 5

Дисциплина: «Системное программирование»

Михаил Чуворкин
19.11.2021

Основы GDI

Подготовка к лабораторной работе

Для выполнения лабораторной работы необходимо ознакомиться (по литературе или в сети) и сделать краткое описание следующих функций и параметров, а также используемых событий.

1). Дескриптор контекста устройства и функции для получения и освобождения контекста:

- **GetWindowDC**;

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

Функция **GetWindowDC** извлекает контекст устройства (**DC**) для всего окна, включая области заголовка, меню и полосы прокрутки. Контекст устройства окна разрешает красить в любом месте окна, потому что начало координат контекста устройства - левый верхний угол окна вместо рабочей области.
GetWindowDC назначает значения атрибутов контекста устройства окна по умолчанию каждый раз, когда она извлекает этот контекст устройства. Предыдущие атрибуты теряются.
Если функция завершается успешно, возвращаемое значение - дескриптор контекста устройства заданного окна. Если функция завершается ошибкой, возвращаемое значение - NULL, что служит признаком ошибки или недопустимого параметра **hWnd**.

- **GetDC**;

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

Функция **GetDC** извлекает дескриптор дисплейного контекста устройства (**DC**) для рабочей области заданного окна или для всего экрана. Вы можете использовать возвращенный дескриптор в последующих функциях **GDI**, чтобы рисовать в контексте устройства.
Если функция завершается успешно, возвращаемое значение - дескриптор контекста устройства (**DC**) для рабочей области заданного окна. Если функция завершается ошибкой, возвращаемое значение - NULL.

- **ReleaseDC;**

```
int ReleaseDC(
  HWND hWnd,   // дескриптор окна
  HDC hDC      // дескриптор контекста устройства (DC)
);
```

Функция **ReleaseDC** освобождает контекст устройства (**DC**) для использования другими приложениями. Действие функции **ReleaseDC** зависит от типа контекста устройства (**DC**). Она освобождает только общий и контекст устройства (**DC**) окна. Она не имеет никакого действия на контексты устройства класса или частный **DC**.
Возвращаемое значение указывает, был ли контекст устройства (**DC**) освобожден. Если контекст устройства был освобожден, возвращаемое значение равно 1.
Если контекст устройства (**DC**) не был освобожден, величина возвращаемого значения - ноль.

2). Выбор и создание инструментов рисования:

- GetStockObject;

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

Функция **GetStockObject** извлекает дескриптор одного из предопределенных (стандартных) перьев, кистей, шрифтов или палитр. Если функция завершается успешно, возвращаемое значение - дескриптор затребованного логического объекта. Если функция завершается с ошибкой, возвращаемое значение - ПУСТО (NULL).

Значения для **fnObject:**

Значение	Предназначение
BLACK_BRUSH	Черная кисть.
DKGRAY_BRUSH	Темно-серая кисть.
DC_BRUSH	**Windows 2000/XP:** Кисть сплошного тона. Цвет по умолчанию является белым. Цвет может измениться при помощи использования функция SetDCBrushColor. Подробную информацию, см. в разделе Замечаний.
GRAY_BRUSH	Серая кисть.
HOLLOW_BRUSH	Пустая кисть (эквивалент NULL_BRUSH).
LTGRAY_BRUSH	Светло-серая кисть.
NULL_BRUSH	Нулевая (пустая) кисть (эквивалент HOLLOW_BRUSH).
WHITE_BRUSH	Белая кисть.
BLACK_PEN	Черное перо.
DC_PEN	**Windows 2000/XP:** Сплошной цвет пера. Цвет по умолчанию является белым. Цвет может измениться при помощи использования функция SetDCPenColor. Подробную информацию см. в разделе Замечаний.
WHITE_PEN	Белое перо.
ANSI_FIXED_FONT	Моноширинный системный шрифт (фиксированной ширины) Windows.
ANSI_VAR_FONT	Системный шрифт Windows с переменным шагом (разноширинный шрифт).
DEVICE_DEFAULT_FONT	**Windows NT/2000/XP:** Аппаратно-зависимый шрифт.
DEFAULT_GUI_FONT	Заданный по умолчанию (типичный) шрифт для объектов пользовательского интерфейса таких как меню и диалоговые окна. Им является **MS Sans Serif**. Сравните это с SYSTEM_FONT.
OEM_FIXED_FONT	Предусматриваемый поставщиком основного оборудования (**OEM**) моноширинный шрифт (фиксированной ширины).
SYSTEM_FONT	Системный шрифт. По умолчанию, система использует системный шрифт, чтобы выводить тексты меню, управляющие элементы диалогового окна и текст. **Windows 95/98 and Windows NT:** Системный шрифт - **MS Sans Serif**. **Windows 2000/XP:** Системный шрифт - **Tahoma.**
SYSTEM_FIXED_FONT	Моноширинный (фиксированной ширины) системный шрифт . Этот предопределенный (стандартный) объект предусматривается только для совместимости с 16-битовыми версиями Windows ранее чем 3.0.
DEFAULT_PALETTE	Заданная по умолчанию(Типичная) палитра. Эта палитра состоит из статических цветов в системной палитре.

- SelectObject;

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

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

Выбираемые объекты: bitmap, brush, font, pen, region

Если выбранный объект - не регион, и функция завершается успешно, возвращаемое значение - дескриптор заменяемого объекта. Если выбранный объект - регион, и функция завершается успешно, возвращаемое значение - одно из нижеперечисленных значений.

Значение	Предназначение
SIMPLEREGION	Регион состоит из одиночного прямоугольника.
COMPLEXREGION	Регион состоит из более чем одного прямоугольника.
NULLREGION	Регион пустой.

- CreateSolidBrush;

```
HBRUSH CreateSolidBrush(
  COLORREF crColor    // код цвета кисти (можно получить макросом RGB)
);
```

Функция **CreateSolidBrush** создает логическую кисть, которая имеет заданный сплошной тон. Если функция завершается успешно, возвращаемое значение идентифицирует логическую кисть. Если функция завершается ошибкой, возвращаемое значение - ПУСТО (NULL).

- CreatePen;

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

Функция **CreatePen** создает логическое перо, которое имеет заданные стиль, ширину и цвет. Перо может быть впоследствии выбрано в контекст устройства и использовано, чтобы рисовать линии и кривые.
Значения для **fnPenStyle:**

Значение	Предназначение
PS_SOLID	Перо является сплошным.
PS_DASH	Перо является штриховым. Этот стиль допустим только тогда, когда толщина пера равняется единице или меньше ее, в единицах измерения устройства (пикселях).
PS_DOT	Перо является пунктирным. Этот стиль допустим только тогда, когда толщина пера равняется единице или меньше ее, в единицах измерения устройства (пикселях).
PS_DASHDOT	Перо имеет чередующиеся штрихи и точки. Этот стиль допустим только тогда, когда толщина пера равняется единице или меньше ее, в единицах измерения устройства (пикселях).
PS_DASHDOTDOT	Перо имеет чередующиеся штрихи и двойные точки. Этот стиль допустим только тогда, когда толщина пера равняется единице или меньше ее, в единицах измерения устройства (пикселях).
PS_NULL	Перо невидимо.
PS_INSIDEFRAME	Перо является сплошным. Когда это перо используется в какой-либо рисующей функции **GDI**, которая забирает прямоугольник ограничения, размеры фигуры сжимаются так, чтобы она вместилась полностью в прямоугольнике ограничения, принимая во внимание толщину пера. Это применяется только к геометрическим перьям.

- DeleteObject;

```
BOOL DeleteObject(
  HGDIOBJ hObject    // дескриптор графического объекта (перо, кисть и т.п.)
);
```

Функция **DeleteObject** удаляет логическое перо, кисть, шрифт, точечную картинку, регион или палитру, освобождая все системные ресурсы, связанные с объектом. После того, как объект удаляется, его дескриптор более не допустим. Если функция завершается успешно, возвращаемое значение - не нуль. Если установленный дескриптор не допустим или является текущим выбранным в **DC**, возвращаемое значение - нуль.

- SetBkColor;

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

Функция **SetBkColor** устанавливает текущий цвет фона в заданном коде цвета или в самом близком физическом цвете, если устройство не может предоставить указанный код цвета. Если функция завершается успешно, возвращаемое значение определяет предыдущий цвет фона как значение **COLORREF**.
Если функция завершается ошибкой, возвращаемое значение - **CLR_INVALID**.

3). Функции рисования.

```
MoveToEx(hdc,x,y,LPPOINT lpPoint); //Определяет текущую позицию пера
LineTo(hdc,x,y); //Рисует прямую до заданной точки
Rectangle(hdc,x1,y1,x2,y2); //Рисует прямоугольник
Ellipse(hdc,x1,y1,x2,y2); //Рисует эллипс
Arc(hdc,x1,y1,x2,y2,xb,yb,xend,yend); //Рисует дугу
Chord(hdc,x1,y1,x2,y2,xb,yb,xend,yend); //Рисует сектор (хорда)
Pie(hdc,x1,y1,x2,y2,xb,yb,xend,yend); //Рисует сектор окружности/эллипса
Polyline(points,count); //Рисует ломаную линию
FrameRect(hdc,x1,y1,x2,y2); //Выводит рамку вокруг прямоугольника
GetCurrentPositionEx(hdc,&p); //Получение текущей позиции карандаша
RoundRect (hdc,x1,y1,x2,y2); //Рисует скругленный прямоугольник
```

4). Функции заливки ExtFloodFill и FillRect.

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

Функция **ExtFloodFill** закрашивает область поверхности изображения текущей кистью.
Типы заливки:

Значение	Предназначение
FLOODFILLBORDER	Закрашиваемая область ограничивается цветом, заданным параметром ***crColor***. Этот стиль идентичен заливке, выполненной функцией **FloodFill**.
FLOODFILLSURFACE	Закрашенная область определяется цветом, который задается ***crColor***. Заливка происходит снаружи во всех направлениях, пока цвет не натолкнется на контур. Этот стиль полезен для заливки областей с разноцветными границами.

```
int FillRect(
  HDC        hdc,      // дескриптор DC
  const RECT *lprc,    // Координаты прямоугольной области для заполнения
  HBRUSH     hbr       // Описатель кисти для заполнения прямоугольной области
);
```

FillRect заполняет прямоугольную область на устройстве, используя указанную кисть. Граница прямоугольной области не рисуется, и основание и правые грани данного прямоугольника не заполняются (они не являются частью внутренней области прямоугольника). В случае ошибки функция возвращает 0. В успешном случае возвращается ненулевое значение.

5). Функции для работы с пикселами: GetPixel и SetPixel.

```
COLORREF GetPixel(
  HDC hdc,    // дескриптор DC
  int nXPos,  // x-координата пикселя
  int nYPos   // y-координата пикселя
);
```

Функция **GetPixel** извлекает значения красного, зеленого, синего (**RGB**) цвета пикселя в заданных координатах. Возвращаемое значение - значение **RGB** пикселя. Если пиксель - вне текущего региона отсечения, возвращаемое значение - **CLR_INVALID**.

```
COLORREF SetPixel(
  HDC hdc,            // дескриптор DC
  int X,              // x-координата пикселя
  int Y,              // y-координата пикселя
  COLORREF crColor    // цвет пикселя
);
```

Функция **SetPixel** устанавливает пиксель в заданных координатах в заданном цвете. Если функция завершается успешно, возвращаемое значение - значение **RGB**, в которое функция устанавливает пиксель. Это значение может отличаться от цвета, заданного параметром **crColor**; это происходит тогда, когда точное соответствие для заданного цвета не может быть найдено. Если функция завершается с ошибкой, возвращаемое значение равно -(минус)1.

6). Режимы рисования: (функция SetROP2) R2_XORPEN, R2_NOTXORPEN и т.д.

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

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.

Режимы смешивания:

Режим смешивания	Описание
R2_BLACK	Пиксель всегда черный.
R2_WHITE	Пиксель всегда белый.
R2_NOP	Пиксель остается неизменяемым.
R2_NOT	Пиксель - инверсия экранного цвета.
R2_COPYPEN	Пиксель - перьевой цвет.
R2_NOTCOPYPEN	Пиксель - инверсия перьевого цвета.
R2_MERGEPENNOT	Пиксель - комбинация перьевого цвета и инверсии цвета экрана (заключительный пиксель = (NOT экранный пиксель) OR перо).
R2_MASKPENNOT	Пиксель - комбинация цветов, общих, и к перу и инверсии экрана (заключительный пиксель = (NOT экранный пиксель) AND перо).
R2_MERGENOTPEN	Пиксель - комбинация экранного цвета и инверсии цвета пера (заключительный пиксель = (NOT перо) AND экранный пиксель).
R2_MASKNOTPEN	Пиксель - комбинация цветов, общих, и к экрану и инверсии пера (заключительный = (NOT перо) AND экранный пиксель).
R2_MERGEPEN	Пиксель - комбинация перьевого цвета и цвета экрана (заключительный = перьевой OR экранный пиксель).
R2_NOTMERGEPEN	Пиксель - инверсия R2_MERGEPEN цвета (заключительный пиксель = NOT (перьевой OR экранный пиксель)).
R2_MASKPEN	Пиксель - комбинация цветов, общих, и к перу и экрану (заключительный пиксель = перьевой AND экранный пиксель).
R2_NOTMASKPEN	Пиксель - инверсия R2_MASKPEN цвета (заключительный пиксель = NOT (перьевой AND экранный пиксель)).
R2_XORPEN	Пиксель - комбинация цветов, которые находятся в пере или в экране (заключительный пиксель = перьевой пиксель XOR экранный пиксель).
R2_NOTXORPEN	Пиксель - инверсия R2_XORPEN цвета (заключительный пиксель = NOT (перьевой пиксель экрана XOR экранный пиксель)).

7). COLORREF

Значение COLORREF используется, чтобы определить цвет RGB.

```
typedef DWORD COLORREF;
typedef DWORD *LPCOLORREF;
```

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**.

Чтобы создать код цвета **COLORREF**, используют макрос **RGB**. Чтобы извлечь отдельные значения компонентов красного, зеленого и синие кода цвета, используют соответственно макроопределения **GetRValue**, **GetGValue** и **GetBValue**.

8). Методы

- InvalidateRect

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

Функция **InvalidateRect** добавляет прямоугольник к обновляемому региону заданного окна. Обновляемый регион представляет часть рабочей области окна, которая должна быть перерисована.
hWnd - Дескриптор окна, обновляемый регион которого изменился. Если этот параметр - NULL, система делает недействительными и перерисовывает все окна, и отправляет сообщения `WM_ERASEBKGND` и `WM_NCPAINT` оконной процедуре перед тем, как возвращает значения функцией.
lpRect - Указатель на структуру `RECT`, содержащую в себе координаты рабочей области прямоугольника, который будет добавлен к обновляемому региону. Если этот параметр - NULL, вся рабочая область добавляется к обновляемому региону.
bErase - Устанавливает, должен ли фон внутри обновляемого региона быть стерт, когда обновляемый регион обрабатывается. Если этот параметр - TRUE, то фон стирается, когда вызывается функция `BeginPaint`. Если этот параметр - FALSE, фон остается неизменным.

- UpdateWindow.

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

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.

Задание

Для выполнения работы создать приложение Win32 Project (в Microsoft Visual Studio).

1. Закрасить окно приложения основным (преобладающим) цветом рабочего стола и нарисовать на нем геометрические фигуры и надписи (по событию WM_PAINT).

   Объявим нужные глобальные переменные (дескриптор для кнопки, ее id и флаг задания, а также переменную, в которую будем записывать значение усредненного цвета рабочего стола):

```
// для задания 1
HWND btnTask1;
const int idBtnTask1 = 1;
bool flagTask1 = false;
COLORREF desktopMeanColor; //значение цвета рабочего стола
```

Создадим кнопку и напишем обработчик нажатия на нее:

```
btnTask1 = CreateWindow(L"BUTTON", L"Desktop color", WS_CHILD | BS_PUSHBUTTON | WS_VISIBLE,
20, 40, 100, 60, hWnd, (HMENU)idBtnTask1, hInstance, 0);

case idBtnTask1: {
       flagTask2 = flagTask3 = flagTask4 = false;
       flagTask1 = !flagTask1;
       desktopMeanColor = GetDesktopMeanColor();
       InvalidateRect(hWnd, NULL, true);
       UpdateWindow(hWnd);
}
break;
```

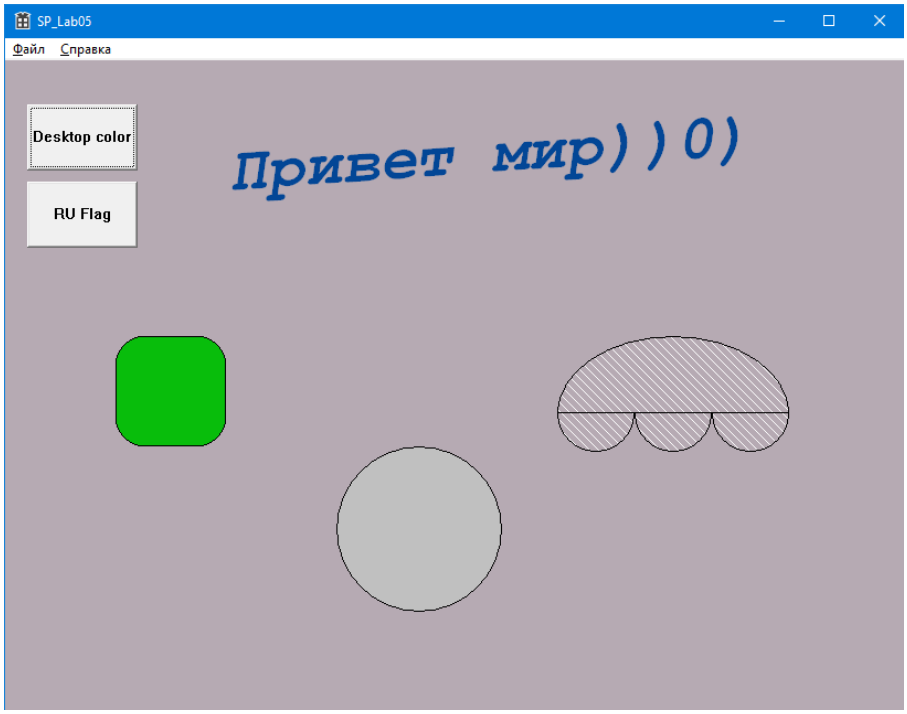
И функцию `GetDesktopMeanColor()`:

```
COLORREF GetDesktopMeanColor() {
    HWND hwndDesktop = GetDesktopWindow();
    HDC hdcDesktop = GetDC(hwndDesktop);
    RECT rec;
    int red = 0, green = 0, blue = 0, count = 0;
    GetWindowRect(hwndDesktop, &rec);
    // Используем метод Монте-Карло для уменьшения количества вычислений
    for (int i = 0; i < 30; ++i){
        int x = (double)rand() / (RAND_MAX + 1) * rec.right;
        int y = (double)rand() / (RAND_MAX + 1) * rec.bottom;
              count++;
              COLORREF color = GetPixel(hdcDesktop, x, y);
              red += GetRValue(color);
              green += GetGValue(color);
              blue += GetBValue(color);
      }
    red = int(red / count);
    green = int(green / count);
    blue = int(blue / count);
    ReleaseDC(hwndDesktop, hdcDesktop);
    return RGB(red, green, blue);
}
```

В обработчике события `WM_PAINT` будем вызывать в соответствии со значением флага функцию:

```
WM_PAINT:
if (flagTask1) {
       Task1OnPaint(hWnd, hdc);
}

void Task1OnPaint(HWND hWnd, HDC hdc) {
    RECT hWnd_rect;
    GetWindowRect(hWnd, &hWnd_rect);
    hWnd_rect.top = 0;
    hWnd_rect.left = 0;
    FillRect(hdc, &hWnd_rect, (HBRUSH)CreateSolidBrush(desktopMeanColor));
    SetBkColor(hdc, desktopMeanColor);

    // Пишем курсивным жирным шрифтом с поворотом в 40 градусов
    HGDIOBJ font = CreateFont(64, 0, 40, 0, FW_BOLD, true, false, false, RUSSIAN_CHARSET,
OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, ANTIALIASED_QUALITY, FF_DONTCARE, L"Courier
new");
    SetTextColor(hdc, RGB(2, 70, 150));
    SelectObject(hdc, font);
    TextOut(hdc, 200, 60, L"Привет мир))0)", 15);
```

```
    // Рисуем скругленный прямоугольник случайным цветом
    SelectObject(hdc, CreateSolidBrush(RGB(rand() % 256, rand() % 256, rand() % 256)));
    int xr = 200, yr = 250;
    RoundRect(hdc, xr, yr, xr + 100, yr + 100, 50, 50);

    // Рисуем круг серой кистью
    SelectObject(hdc, GetStockObject(LTGRAY_BRUSH));
    Ellipse(hdc, xr + 100, yr + 100, xr + 350, yr + 250);

    // Рисуем облако из четырех полукругов с диагональной штриховкой
    SelectObject(hdc, CreateHatchBrush(HS_FDIAGONAL, RGB(255, 255, 255)));
    int _size = 70, d1 = 3 * _size, d2 = 2 * _size;
    int x = 500, y = 250;
    Chord(hdc, x, y, x + d1, y + d2, x + d1, y + d2 / 2, x, y + d2 / 2);
    int dx = _size;
    Chord(hdc, x, y + d2 / 4, x + dx, y + 3 * d2 / 4, x, y + d2 / 2, x + d1, y + d2 / 2);
    Chord(hdc, x + dx, y + d2 / 4, x + 2 * dx, y + 3 * d2 / 4, x, y + d2 / 2, x + d1, y +
d2 / 2);
    Chord(hdc, x + 2 * dx, y + d2 / 4, x + 3 * dx, y + 3 * d2 / 4, x, y + d2 / 2, x + d1,
y + d2 / 2);

    ReleaseDC(hWnd, hdc);

    flagTask1 = false;
}
```



Добавим обработчик изменения размеров окна, чтобы не пропадало изображение:

```
case WM_SIZING:
{
	UpdateWindow(hWnd);
}
break;
```

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

Объявим нужные глобальные переменные (дескриптор для кнопки, ее id и флаг задания):

```
// для задания 2
HWND btnTask2;
const int idBtnTask2 = 2;
bool flagTask2 = false;
```

Создадим кнопку и напишем обработчик нажатия на нее:

```
btnTask2 = CreateWindow(L"BUTTON", L"RU Flag", WS_CHILD | BS_PUSHBUTTON | WS_VISIBLE, 20, 110, 100, 60, hWnd, (HMENU)idBtnTask2, hInstance, 0);
```

```
case idBtnTask2: {
	flagTask1 = flagTask3 = flagTask4 = false;
	flagTask2 = !flagTask2;
	InvalidateRect(hWnd, NULL, true);
	UpdateWindow(hWnd);
}
break;
```

В обработчике события `WM_PAINT` будем вызывать в соответствии со значением флага функцию:

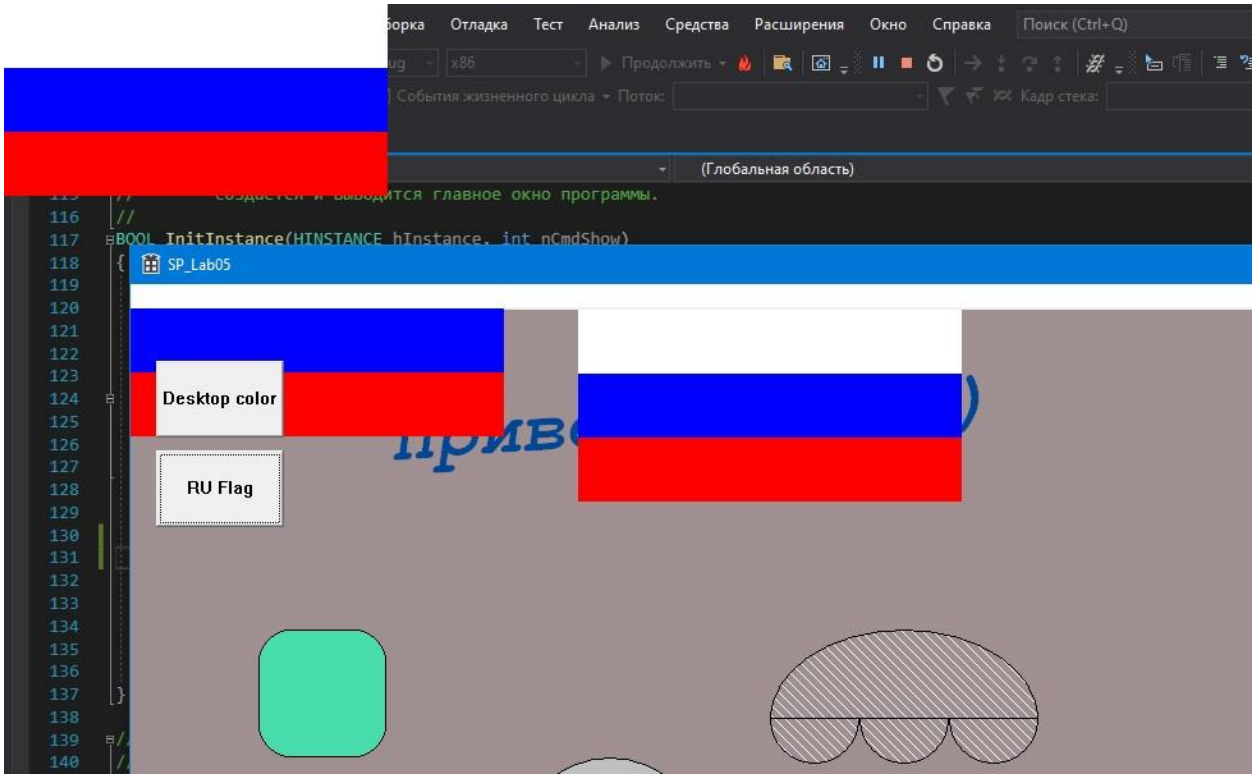
```
WM_PAINT:
if (flagTask2) {
	Task2OnPaint(hWnd);
}
```

```
void Task2OnPaint(HWND hWnd) {
    HDC hdc;

    // рисование на рабочем столе
    HWND hwndDesktop = GetDesktopWindow();
    hdc = GetWindowDC(hwndDesktop);
    DrawFlag(hdc, 0, 0, 300, 50);
    ReleaseDC(hwndDesktop, hdc);

    // рисование в рабочей области окна
    hdc = GetDC(hWnd);
    DrawFlag(hdc, 350, 0, 300, 50);
    ReleaseDC(hWnd, hdc);

    // рисование на всей области окна
    hdc = GetWindowDC(hWnd);
    DrawFlag(hdc, 0, 0, 300, 50);
    ReleaseDC(hWnd, hdc);

    flagTask2 = false;
}
```

Функция рисования флага:

```
void DrawFlag(HDC hdc, int x0, int y0, int width, int height) {

    HBRUSH whiteBrush = CreateSolidBrush(RGB(255, 255, 255));
    HBRUSH blueBrush = CreateSolidBrush(RGB(0, 0, 255));
    HBRUSH redBrush = CreateSolidBrush(RGB(255, 0, 0));
```

```
    RECT flagRect = { x0, y0, x0 + width, y0 + height };
    FillRect(hdc, &flagRect, whiteBrush);
    flagRect = { x0, y0 + height, x0 + width, y0 + 2 * height };
    FillRect(hdc, &flagRect, blueBrush);
    flagRect = { x0, y0 + 2 * height, x0 + width, y0 + 3 * height };
    FillRect(hdc, &flagRect, redBrush);
    // очищаем ресурсы
    DeleteObject(whiteBrush);
    DeleteObject(blueBrush);
    DeleteObject(redBrush);
}
```



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.

Объявим нужные глобальные переменные (дескриптор для кнопки, ее id и флаг задания, дескрипторы окон и контекста устройства для этих окон, а также таймер и ограничивающий квадрат для круга):

```
// для задания 3
HWND btnTask3;
const int idBtnTask3 = 3;
bool flagTask3 = false;
RECT circleRect = { 0, 0, 50, 50 }; // начальный bbox для круга
int dx = 20, dy = 20; // скорость движения круга
HWND hwndDesktop;
HWND hwndCSharp;
```

```
HDC hdcDesktop;
HDC hdcClient;
HDC hdcCSharp;
const static int idTimer = 5; // идентификатор таймера
static int timerTick = 0; // счетчик тиков
```

Создадим кнопку и напишем обработчик нажатия на нее:

```
btnTask3 = CreateWindow(L"BUTTON", L"Circle", WS_CHILD | BS_PUSHBUTTON | WS_VISIBLE, 20,
180, 100, 60, hWnd, (HMENU)idBtnTask3, hInstance, 0);

if (flagTask3){
	// получаем контекст устройства для каждого окна
	hdcClient = GetDC(hWnd);
	hwndDesktop = GetDesktopWindow();
	hdcDesktop = GetDC(hwndDesktop);
	hwndCSharp = FindWindow(NULL, L"Form1");
	if (hwndCSharp) hdcCSharp = GetDC(hwndCSharp);
	timerTick = 0; // обнулить таймер
	SetTimer(hWnd, idTimer, 50, (TIMERPROC)NULL); // таймер с периодом 50мс
}
else {
	// очистка областей
	InvalidateRect(hwndDesktop, &circleRect, true);
	InvalidateRect(hWnd, &circleRect, true);
	if (hwndCSharp) InvalidateRect(hwndCSharp, &circleRect, true);
	// освобождение контекста
	ReleaseDC(hwndDesktop, hdcDesktop);
	ReleaseDC(hWnd, hdcClient);
	if (hdcCSharp) ReleaseDC(hwndCSharp, hdcCSharp);
	KillTimer(hWnd, idTimer); // остановить таймер
}
```

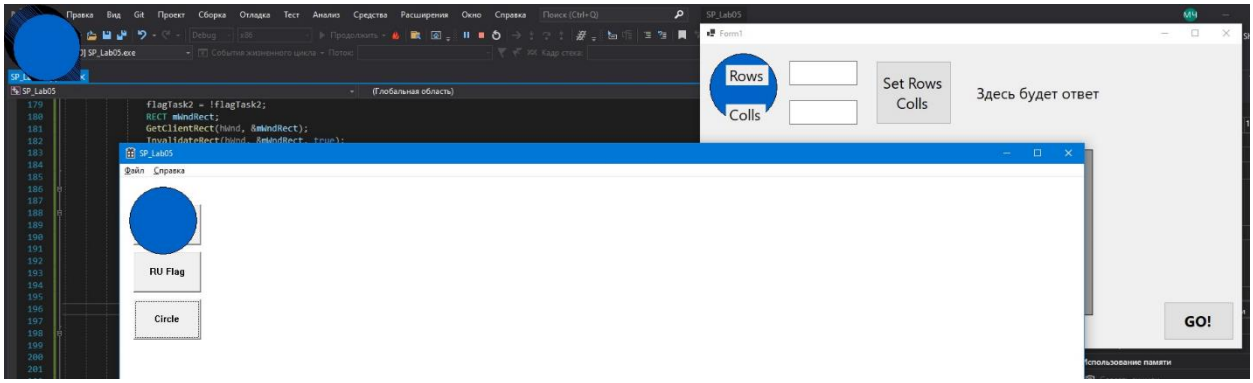
В обработчике события WM_TIMER:

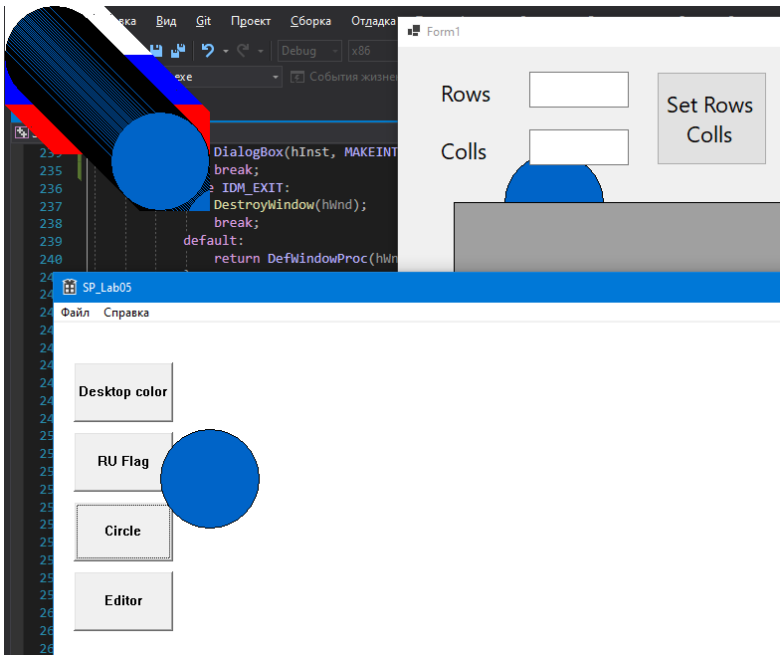
```
if (flagTask3) {
	++timerTick;
	Circles(hWnd);
}
else {
	KillTimer(hWnd, idTimer); // остановить таймер
}
```

Функции рисования одного круга в контексте для заданного окна и функция для рисования трех кругов:

```
void DrawCircle(HDC hdc){
    HBRUSH brush = CreateSolidBrush(RGB(0, 100, 200));
    SelectObject(hdc, brush);
    Ellipse(hdc, circleRect.left, circleRect.top, circleRect.right, circleRect.bottom);
    DeleteObject(brush);
}


void Circles(HWND hWnd) {
    InvalidateRect(hwndDesktop, &circleRect, true);
    UpdateWindow(hwndDesktop);

    InvalidateRect(hWnd, &circleRect, true);
    UpdateWindow(hWnd);

    if (hdcCSharp) {
```

```
        InvalidateRect(hwndCSharp, &circleRect, true);
        UpdateWindow(hwndCSharp);
    }

    int shift = 1;
    circleRect = { timerTick * shift, timerTick * shift, timerTick * shift + 100,
timerTick * shift + 100 };

    DrawCircle(hdcDesktop);
    DrawCircle(hdcClient);
    if (hdcCSharp) DrawCircle(hdcCSharp);
}
```





4. Приложение Win API должно реализовать функции простейшего графического редактора, в котором пользователь может рисовать (линиями двух заданных цветов) в пределах заданного прямоугольного поля (поле закрашивается заданным цветом). Рисунок создается с помощью линий, левая кнопка мыши определяет начало линии, при перемещении мыши линия тоже перемещается, так что пользователь может оценить правильность ее расположения, при отпускании мыши линия фиксируется.

Объявим нужные глобальные переменные (дескриптор для кнопки, ее id и флаг задания, дескриптор пера, его ширину, поле для редактора, начальную и конечную точки):

```
// для задания 4
HWND btnTask4;
const int idBtnTask4 = 4;
bool flagTask4 = false;
bool isDrawingState = false; // флаг состояния, когда зажата ЛКМ
POINT PBegin, PEnd; // начало и конец линии
HPEN pen;
int penWidth = 5;
RECT editorRect = { 200, 30, 800, 450 }; // прямоугольник редактора
```

Создадим кнопку и напишем обработчик нажатия на нее:

```
btnTask4 = CreateWindow(L"BUTTON", L"Editor", WS_CHILD | BS_PUSHBUTTON | WS_VISIBLE, 20,
250, 100, 60, hWnd, (HMENU)idBtnTask4, hInstance, 0);

case idBtnTask4: {
       flagTask1 = flagTask2 = flagTask3 = false;
       flagTask4 = !flagTask4;
       InvalidateRect(hWnd, NULL, true);
       UpdateWindow(hWnd);
}
break;
```

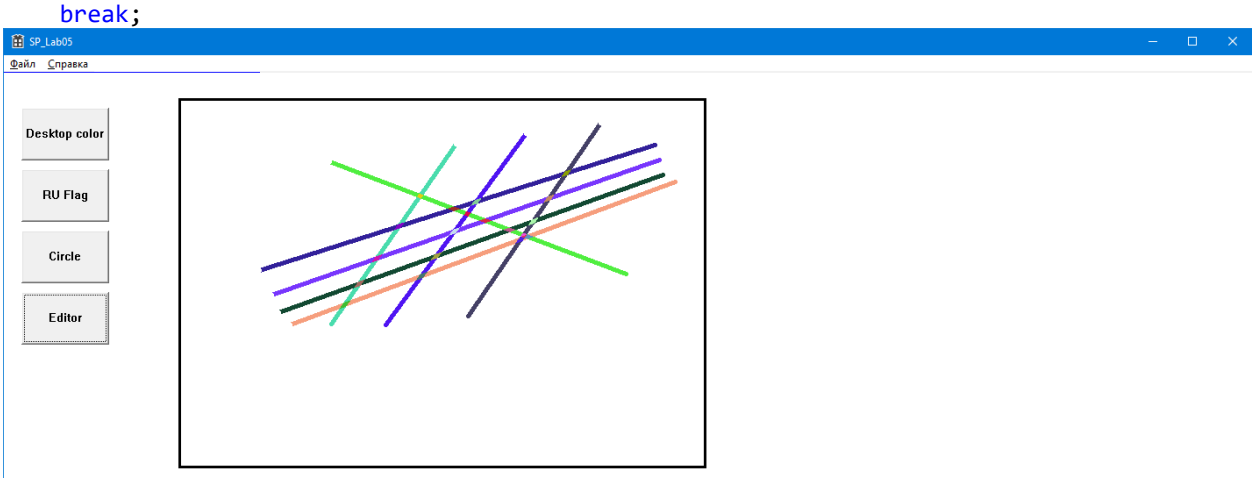
Функция определения попадания мыши в область редактора:

```
bool isInEditor(DWORD lParam) {
    return  LOWORD(lParam) > editorRect.left + penWidth &&
            LOWORD(lParam) < editorRect.right - penWidth &&
            HIWORD(lParam) > editorRect.top + penWidth &&
            HIWORD(lParam) < editorRect.bottom - penWidth;
}
```

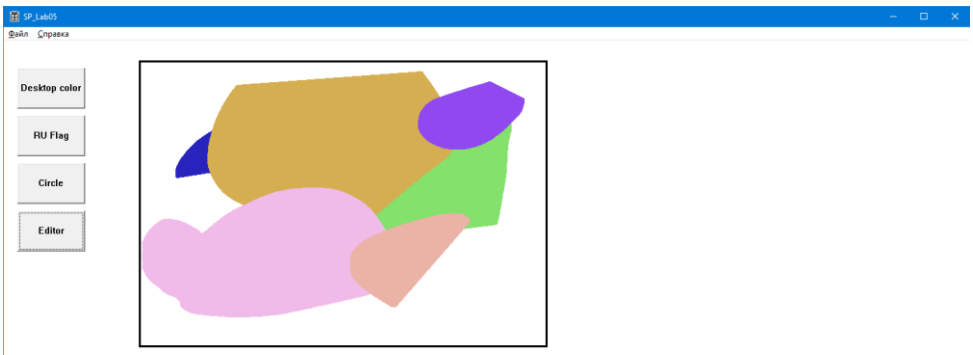
В обработчики событий мыши напишем фиксацию начальной точки и обновление конечной, а также рисование линии с соответствующим режимом смешивания:

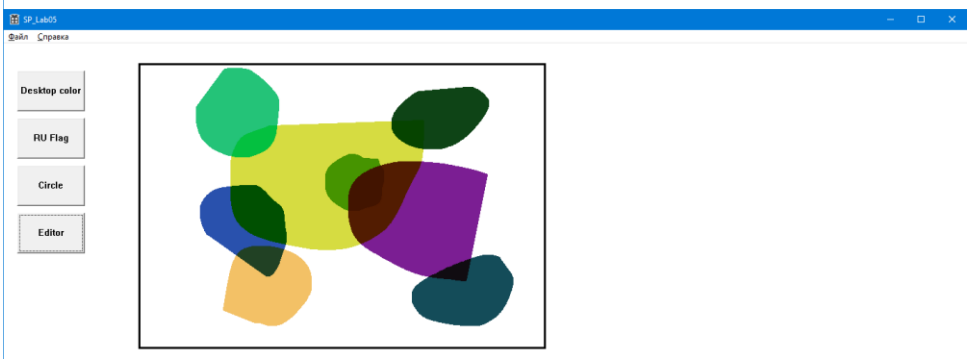
```
    case WM_LBUTTONDOWN:
    {
        if (flagTask4 && isInEditor(lParam))
        {
            isDrawingState = true;
            PEnd = PBegin = { LOWORD(lParam) , HIWORD(lParam) };
            pen = CreatePen(PS_SOLID, penWidth, RGB(rand() % 256, rand() % 256, rand() %
256));
        }
    }
    break;
    case WM_LBUTTONUP:
    {
        // если в состоянии рисования линии отпускаем ЛКМ
        if (isDrawingState && isInEditor(lParam))
        {
            DeleteObject(pen);
            isDrawingState = false;
        }
    }
    break;
    case WM_MOUSEMOVE:
    {
        // если в состоянии рисования линии двигаем мышь
```

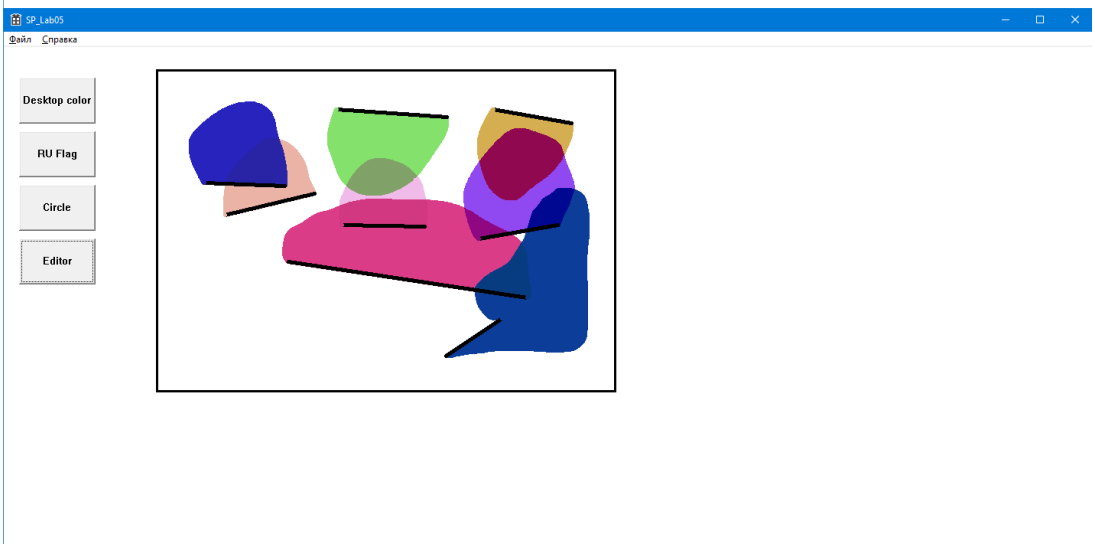
```
        if (isDrawingState && isInEditor(lParam))
        {
            HDC hdc = GetDC(hWnd);
            SelectObject(hdc, pen);
            // чтобы оставались только линии R2_NOTMASKPEN, R2_XORPEN, R2_NOTXORPEN
            SetROP2(hdc, R2_XORPEN);
            MoveToEx(hdc, PBegin.x, PBegin.y, nullptr);
            LineTo(hdc, PEnd.x, PEnd.y);
            MoveToEx(hdc, PBegin.x, PBegin.y, nullptr);
            LineTo(hdc, LOWORD(lParam), HIWORD(lParam));
            PEnd = { LOWORD(lParam) , HIWORD(lParam) };
            ReleaseDC(hWnd, hdc);
        }
    }
    break;
```



При других режимах смешивания можно рисовать области:







Полный код программы доступен здесь: https://pastebin.com/Bi4xGyNw