Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №2

на тему

РАСШИРЕННОЕ ИСПОЛЬЗОВАНИЕ ОКОННОГО ИНТЕРФЕЙСА WIN 32 И GDI. ФОРМИРОВАНИЕ СЛОЖНЫХ ИЗОБРАЖЕНИЙ, СОЗДАНИЕ И ИСПОЛЬЗОВАНИЕ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ, ОБРАБОТКА РАЗЛИЧНЫХ СООБЩЕНИЙ, МЕХАНИЗМ ПЕРЕХВАТА СООБЩЕНИЙ (WINHOOK)

Выполнил студент гр.153502 Леоненко А.О.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Теоретические сведения 3](#_Toc147699657)

[2 Формулировка задачи 4](#_Toc147699658)

[3 Описание функций программы 5](#_Toc147699659)

[3.1 Изменение режима 5](#_Toc147699660)

[3.2 Режим создания фигур 5](#_Toc147699661)

[3.3 Рисование на различных слоях 5](#_Toc147699662)

[3.4 Режим редактирования фигур 6](#_Toc147699663)

[3.5 Изменение цвета заливки фигуры 7](#_Toc147699664)

[Список использованных источников 8](#_Toc147699665)

[Приложение А 9](#_Toc147699666)

1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

*Windows GDI+* — это часть операционной системы *Windows XP* или *Windows Server 2003*, которая предоставляет двумерную векторную графику, визуализацию и оформление. *GDI+* улучшает интерфейс графических устройств *Windows (GDI)* (интерфейс графического устройства, включенный в более ранние версии *Windows*) за счет добавления новых функций и оптимизации существующих функций [1].

Контекст устройства — это структура, используемая *Windows* для хранения сведений о возможностях конкретного устройства отображения и атрибутов, определяющих способ рисования элементов на этом устройстве. Контекст устройства для видеотрансляции также связан с определенным окном на дисплее. Сначала получается дескриптор для контекста устройства (*HDC*), а затем этот дескриптор передается в качестве аргумента функциям *GDI*, которые фактически выполняют рисование. Дескриптор также передается в качестве аргумента в функции *GDI*, которые получают или задают атрибуты контекста устройства.

При использовании *GDI+*, не нужно беспокоиться о дескрипторах и контекстах устройств, как при использовании GDI. Просто создается объект *Graphics*, а затем вызываются его методы в знакомом объектно-ориентированном стиле — *myGraphicsObject.DrawLine(parameters*). Объект *Graphics* находится в основе *GDI+* так же, как контекст устройства находится в основе *GDI*. Контекст устройства и объект *Graphics* играют схожие роли, но существуют некоторые фундаментальные различия между моделью программирования на основе дескрипторов, используемой с контекстами устройства (*GDI*), и объектно-ориентированной моделью, используемой с графическими объектами (*GDI+*).

Объект *Graphics*, как и контекст устройства, связан с определенным окном на экране и содержит атрибуты (например, режим сглаживания и указание отрисовки текста), указывающие способ рисования элементов. Однако объект *Graphics* не привязан к перу, кисти, пути, изображению или шрифту в контексте устройства. В *GDI* перед использованием контекста устройства для рисования линии необходимо вызвать *Метод SelectObject*, чтобы связать объект пера с контекстом устройства. Это называется выделением пера в контексте устройства. Все линии, нарисованные в контексте устройства, будут использовать это перо, пока вы не выберете другое перо. При использовании *GDI+* объект *Pen* передается в качестве аргумента методу *DrawLine* класса *Graphics*. Вы можете использовать другой объект *Pen* в каждом из серий вызовов *DrawLine* без необходимости связывать данный объект *Pen* с объектом *Graphics* [2].

2 ФОРМУЛИРОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является создание оконного приложения на *Win32 API*, обладающее минимальным функционалом, позволяющим отработать базовые навыки написания программы на *Win32* *API*, таких как обработка оконных сообщений, а также расширенное использование оконного интерфейса *Win32* и *GDI*. Также необходимо изучить формирование сложных изображений, создание и использование элементов управления, обработку различных сообщений, механизм перехвата сообщений (*winhook*).

В качестве задачи необходимо разработать графический редактор, позволяющий создавать и редактировать сложные многослойные изображения с использованием *GDI*+.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

1 Изменение выбранного цвета;

2 Создание фигур (кругов, прямоугольников);

3 Редактирование фигур;

4 Рисование на различных слоях.

3.1 Изменение режима

Для изменения режима (создание фигур, редактирования) необходимо нажать клавишу Shift.

3.2 Режим создания фигур

Режим создания фигур (рисунок 1) доступен изначально при запуске программы. Для создания фигуры необходимо зажать ЛКМ и провести желаемую фигуру, после отжать ЛКМ. Для изменения фигуры, необходимо нажать пробел.

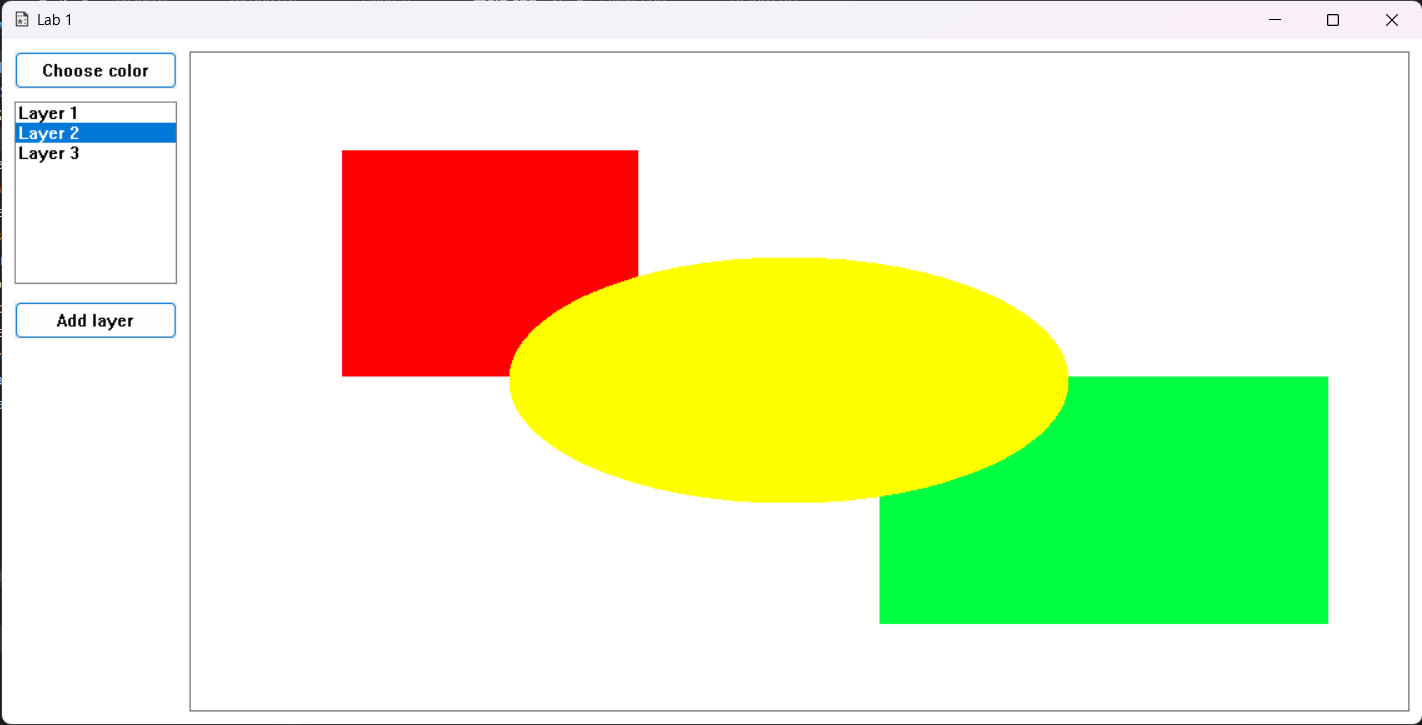


Рисунок 1 – Режим создания фигур

3.3 Рисование на различных слоях

Для рисования на различных слоях необходимо создать нужное количество слоев с помощью кнопки “*Add layer*”. После чего для рисования необходимо выбирать нужный слой нажатием ЛКМ на название этого слоя. Пример рисования изображен на рисунке 2 и рисунке 3.

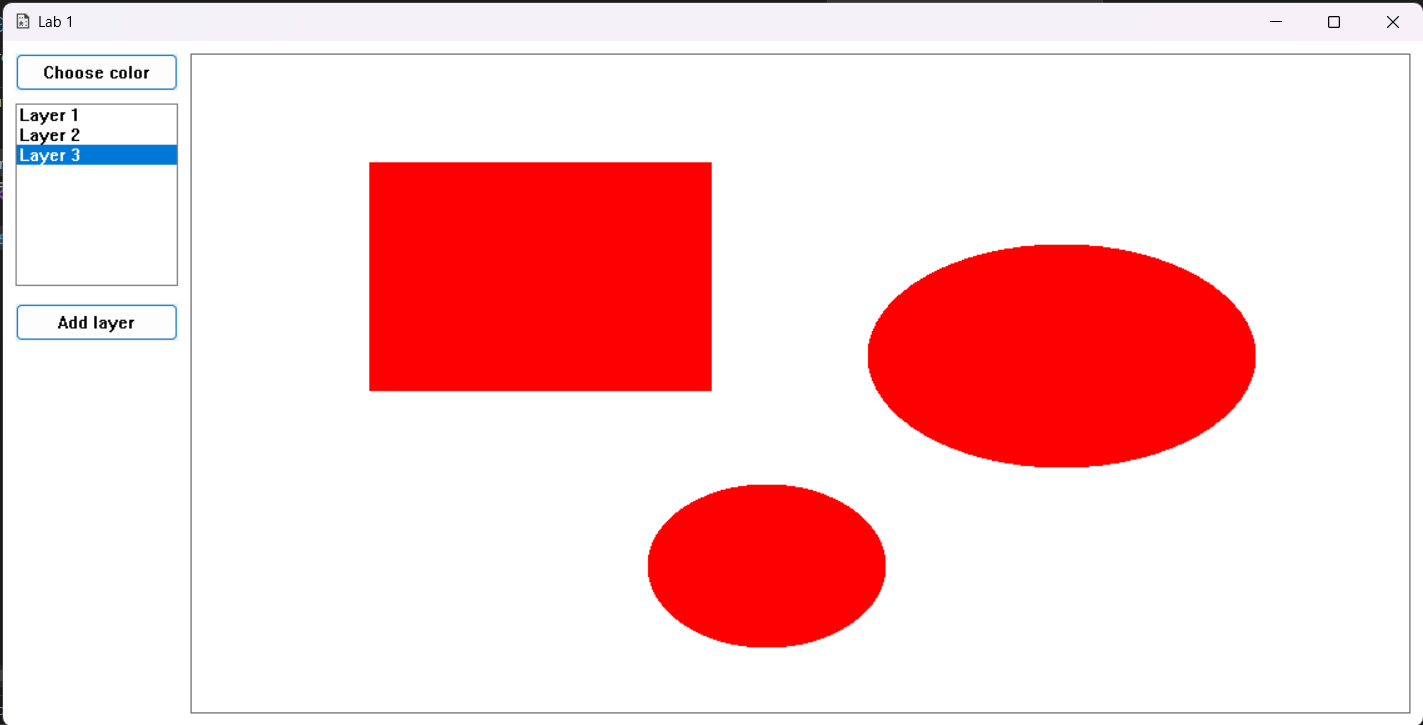


Рисунок 2 – Холст, после рисования на слое 3

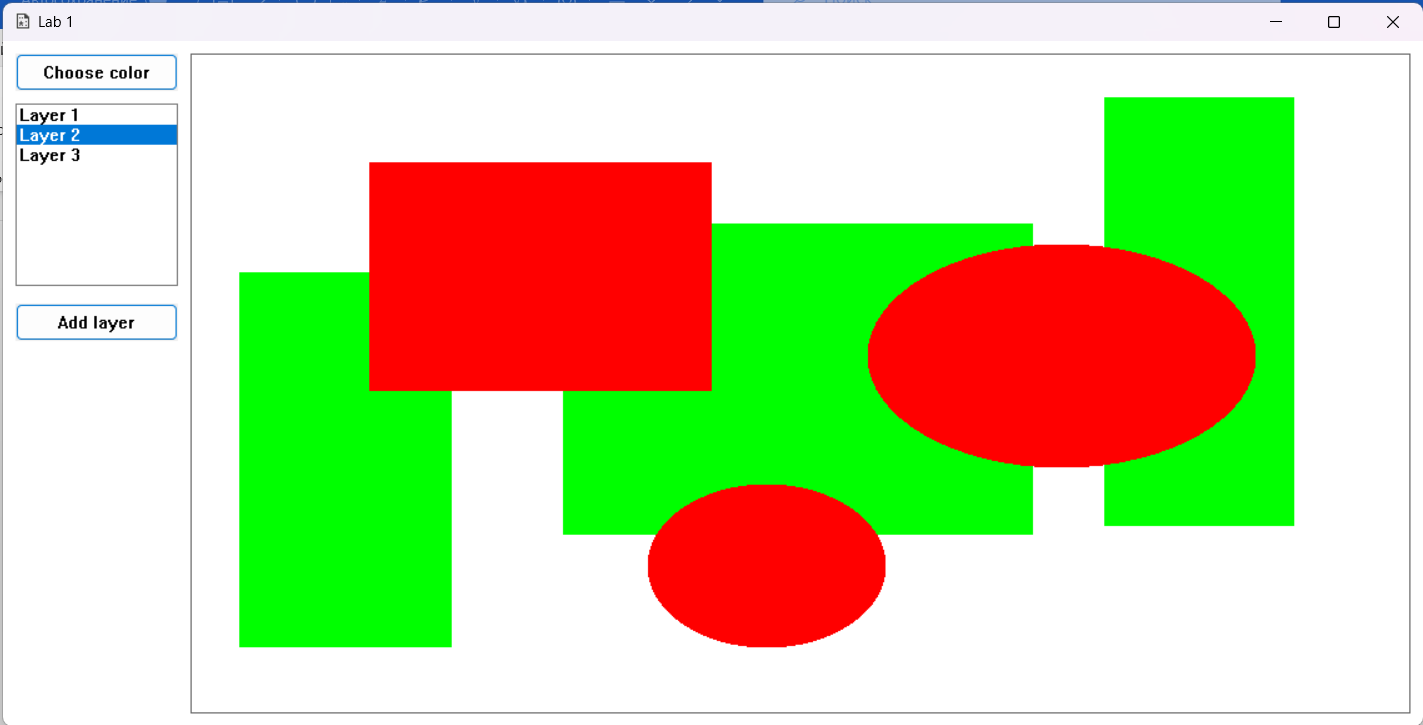


Рисунок 3 – Холст, после рисования фигур на слое 2

3.4 Режим редактирования фигур

Для изменения фигуры необходимо перейти в режим редактирования. Далее необходимо навести курсор на желаемую геометрическую форму, после чего можно изменить положение фигуры в окне с помощью клавишей влево, вправо, вверх и вниз (рисунок 4). Так же в этот момент появляется возможность изменить саму фигуру. Для этого необходимо навести курсор на правый нижний либо левый верхний угол, нажать ЛКМ, после чего навести курсор в желаемое положение и нажать ЛКМ (рисунок 5).

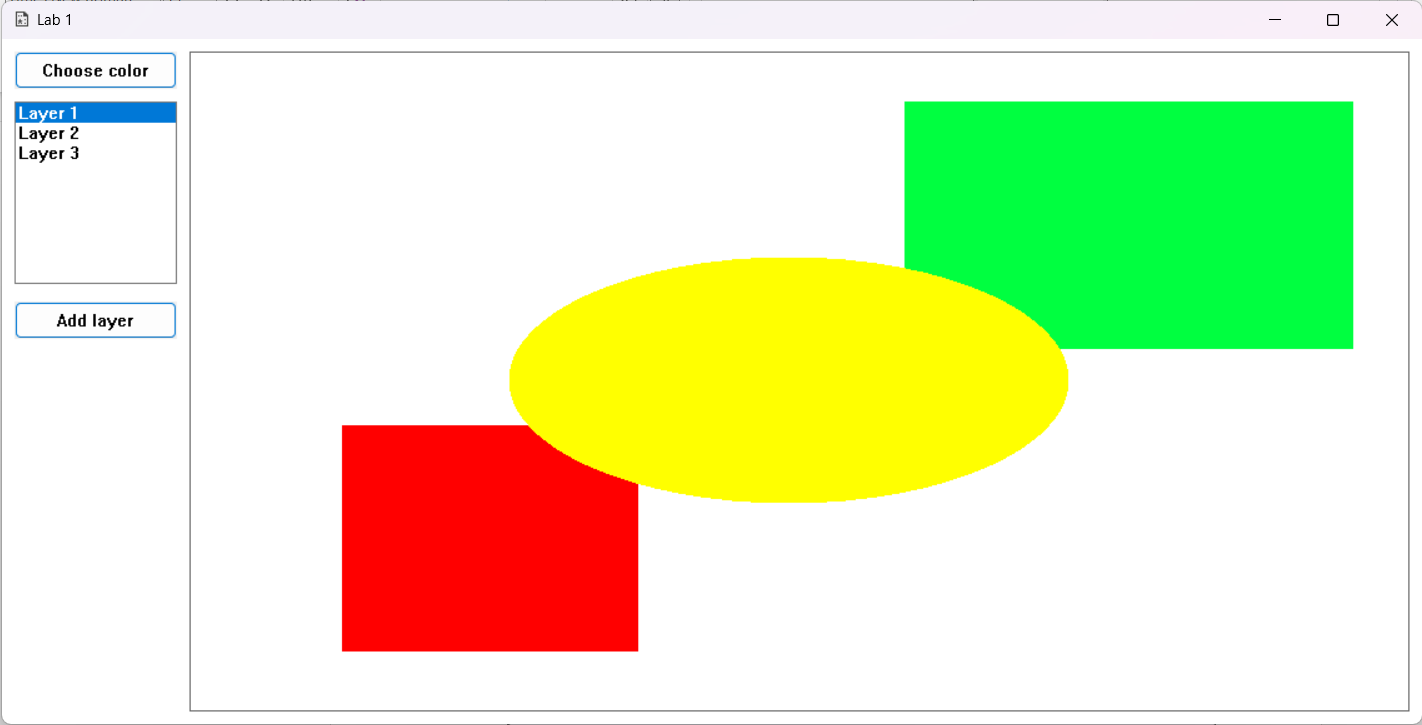


Рисунок 4 – Результат изменения положения фигур в режиме редактирования

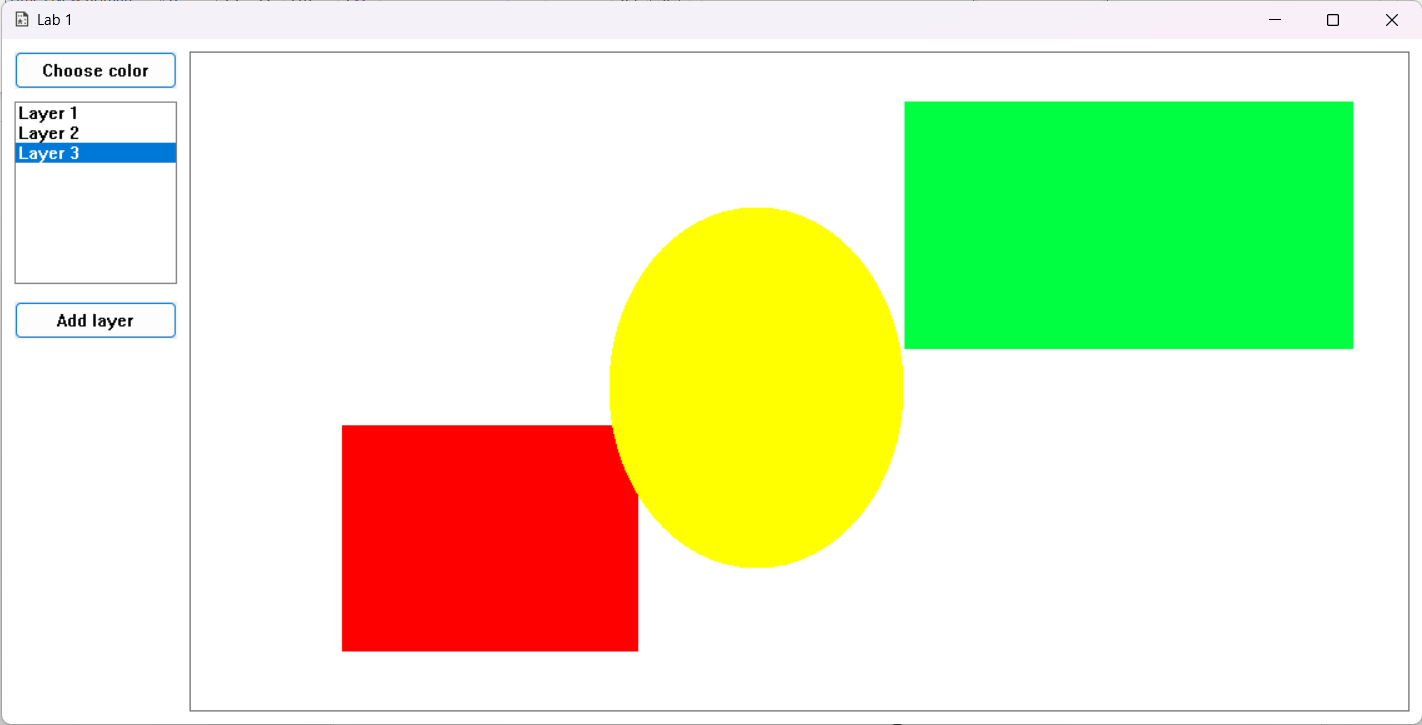


Рисунок 5 – Результат изменения размера фигуры в режиме редактирования

3.5 Изменение цвета заливки фигуры

Для изменения цвета заливки фигуры предусмотрена кнопка “*Choose color*”. После нажатия на эту кнопку откроется окно, где необходимо выбрать новый цвет. После выбора цвета необходимо нажать на кнопку “*Ok*”, после чего цвет будет успешно изменен.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Сведения о GDI+ [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/gdiplus/-gdiplus-about-gdi--about

[2] Изменения в модели программирования [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/gdiplus/-gdiplus-changes-in-the-programming-model-about

ПРИЛОЖЕНИЕ А

Листинг кода

Файл main.cpp

#ifndef UNICODE

#define UNICODE

#endif

#define ID\_BUTTON 1001

#define ID\_LISTBOX 1002

#define ID\_LAYER\_BUTTON 1003

#include <windows.h>

#include <tchar.h>

#include <vector>

#include "resource.h"

#include "shape.cpp"

//#include "layers.cpp"

#include <gdiplus.h>

using namespace Gdiplus;

#pragma comment (lib,"Gdiplus.lib")

#pragma comment(linker,"\"/manifestdependency:type='win32' \

name='Microsoft.Windows.Common-Controls' version='6.0.0.0' \

processorArchitecture='\*' publicKeyToken='6595b64144ccf1df' language='\*'\"") // updated styles

RECT rc = { 0 };

int WindowPosX = 0;

int WindowPosY = 0;

HWND hwndChild;

HWND hwndListBox;

HWND AddLayerButton;

HDC hdc;

HDC hdcBuffer;

HBITMAP hBitmap;

HGDIOBJ oldBitmap;

Graphics\* graphics;

int toolsWidth = 150;

int margin = 10;

int drawWindowX = toolsWidth;

int drawWindowY = margin;

int drawWindowWidth;

int drawWindowHeight;

int buttonHeight = 30;

int listHeight = 150;

Shape::ShapeType choosenShapeType = Shape::RECTANGLE;

Shape\* currentShape = nullptr;

bool isDrawing = false; // while LMB clicked

bool isEditing = false;

// Editing staff

Shape\* selectedShape = nullptr;

bool isCornerSelected = false;

bool isLtCornerSelected = false;

int deltaWay = 20;

COLORREF dColors[16];

COLORREF currentColor = RGB(255, 0, 0);

class Layers

{

private:

std::vector<std::vector<Shape\*>\*>\* layers;

int layersCount = 0;

std::vector<Shape\*>\* currentLayer = nullptr;

public:

Layers()

{

layers = new std::vector<std::vector<Shape\*>\*>;

addLayer();

}

int getLayersCount()

{

return layersCount;

}

std::vector<Shape\*>\* getCurrentLayer()

{

return currentLayer;

}

std::vector<Shape\*>\* addLayer()

{

layers->push\_back(new std::vector<Shape\*>);

currentLayer = (\*layers)[layersCount++];

return currentLayer;

}

std::vector<Shape\*>\* operator[](int index)

{

return (\*layers)[index];

}

std::vector<Shape\*>\* setCurrentLayer(int index)

{

if (getLayersCount() > index && index >= 0)

currentLayer = (\*layers)[index];

return currentLayer;

}

~Layers()

{

for (int i = 0; i < layers->size(); i++)

{

for (int j = 0; j < layers[i].size(); j++)

{

delete(layers[i][j]);

}

layers[i].clear();

}

layers->clear();

}

};

Layers\* layers;

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

LRESULT CALLBACK DrawWindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

// buffer functions

void CreateBuffer(int width, int height) {

hdcBuffer = CreateCompatibleDC(NULL);

hBitmap = CreateCompatibleBitmap(hdcBuffer, width, height);

oldBitmap = SelectObject(hdcBuffer, hBitmap);

}

void DeleteBuffer() {

SelectObject(hdcBuffer, oldBitmap);

DeleteObject(hBitmap);

DeleteDC(hdcBuffer);

}

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow)

{

// Register the window class.

const wchar\_t CLASS\_NAME[] = L"Sample Window Class";

WNDCLASS wc = { };

GdiplusStartupInput gdiplusStartupInput;

ULONG\_PTR gdiplusToken;

// Initialize GDI+.

GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

HICON hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_ICON1));

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

wc.style = CS\_HREDRAW | CS\_VREDRAW; // Window redraw after sizing

wc.hIcon = hIcon;

RegisterClass(&wc);

WNDCLASS DrawWndClass = { 0 };

DrawWndClass.lpfnWndProc = DrawWindowProc;

DrawWndClass.hInstance = hInstance;

DrawWndClass.lpszClassName = L"DrawWindowClass";

RegisterClass(&DrawWndClass);

// Create the window.

HWND hwnd = CreateWindowEx(

0, // Optional window styles.

CLASS\_NAME, // Window class

L"Lab 1", // Window text

WS\_OVERLAPPEDWINDOW, // Window style

// Size and position

CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT,

NULL, // Parent window

NULL, // Menu

hInstance, // Instance handle

NULL // Additional application data

);

if (hwnd == NULL)

{

return 0;

}

RECT clientRect;

GetClientRect(hwnd, &clientRect);

drawWindowHeight = clientRect.bottom - clientRect.top - 2 \* margin;

drawWindowWidth = clientRect.right - clientRect.left - toolsWidth - margin;

hwndChild = CreateWindowEx(0, L"DrawWindowClass", L"Дочернее Окно", WS\_CHILD | WS\_VISIBLE | WS\_BORDER,

drawWindowX, drawWindowY, drawWindowWidth, drawWindowHeight, hwnd, (HMENU)NULL, hInstance, NULL);

if (hwndChild == NULL)

{

return 0;

}

HWND hwndButton = CreateWindow(

L"BUTTON", // Имя класса для кнопки

L"Choose color", // Текст на кнопке

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON, // Стили кнопки

margin, margin, toolsWidth - 2 \* margin, buttonHeight, // Позиция и размеры кнопки

hwnd, // Родительское окно

(HMENU)ID\_BUTTON, // Идентификатор кнопки

hInstance, // Дескриптор экземпляра приложения

NULL // Дополнительные параметры

);

ShowWindow(hwnd, nCmdShow);

// Run the message loop.

MSG msg = { };

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

GdiplusShutdown(gdiplusToken);

return 0;

}

// Draw help functions

void DrawShape(Graphics\* graphics, Color color, int shapeType, int x1, int y1, int x2, int y2)

{

SolidBrush brush(color);

//SolidBrush brush(Color(255, 0, 0, 255));

int start\_x, start\_y;

int width, height;

if (x1 < x2)

{

start\_x = x1;

width = x2 - x1;

}

else

{

start\_x = x2;

width = x1 - x2;

}

if (y1 < y2)

{

start\_y = y1;

height = y2 - y1;

}

else

{

start\_y = y2;

height = y1 - y2;

}

if (shapeType == Shape::RECTANGLE)

{

RectF rect(start\_x, start\_y, width, height);

graphics->FillRectangle(&brush, rect);

}

else if (shapeType == Shape::CIRCLE)

{

RectF ellipseRect(start\_x, start\_y, width, height);

graphics->FillEllipse(&brush, ellipseRect);

}

}

void StartDraw(LPARAM lParam)

{

isDrawing = true;

currentShape = new Shape(choosenShapeType);

currentShape->x1 = currentShape->x2 = LOWORD(lParam);

currentShape->y1 = currentShape->y2 = HIWORD(lParam);

currentShape->color = Color(GetRValue(currentColor), GetGValue(currentColor), GetBValue(currentColor));

}

void SelectShape(LPARAM lParam)

{

std::vector<Shape\*>\* currentLayer = layers->getCurrentLayer();

for (int i = currentLayer->size() - 1; i >= 0; i--)

{

Shape\* shape = (\*currentLayer)[i];

int x = LOWORD(lParam);

int y = HIWORD(lParam);

if (shape->x1 <= x && shape->x2 >= x && shape->y1 <= y && shape->y2 >= y)

{

selectedShape = shape;

break;

}

}

}

void SelectCorner(LPARAM lParam)

{

int x1 = selectedShape->x1;

int y1 = selectedShape->y1;

int x2 = selectedShape->x2;

int y2 = selectedShape->y2;

int x = LOWORD(lParam);

int y = HIWORD(lParam);

double lt\_corner\_way = sqrt(pow((x1 - x), 2) + pow((y1 - y), 2));

double rd\_corner\_way = sqrt(pow((x2 - x), 2) + pow((y2 - y), 2));

isCornerSelected = true;

if (lt\_corner\_way < rd\_corner\_way)

isLtCornerSelected = true;

else

isLtCornerSelected = false;

}

void ChangeCorner(LPARAM lParam, HWND hwnd)

{

if (isLtCornerSelected)

{

selectedShape->x1 = LOWORD(lParam);

selectedShape->y1 = HIWORD(lParam);

}

else

{

selectedShape->x2 = LOWORD(lParam);

selectedShape->y2 = HIWORD(lParam);

}

selectedShape->checkCoord();

isCornerSelected = false;

selectedShape = nullptr;

InvalidateRect(hwnd, NULL, TRUE);

}

void ShapeDrawingCoord(LPARAM lParam, HWND hwnd)

{

currentShape->x2 = LOWORD(lParam);

currentShape->y2 = HIWORD(lParam);

InvalidateRect(hwnd, NULL, TRUE);

}

void EndDrawing()

{

isDrawing = false;

currentShape->checkCoord();

layers->getCurrentLayer()->push\_back(currentShape);

currentShape = nullptr;

}

// Color picker

void OpenColorPicker(HWND hwnd) {

CHOOSECOLOR cc;

COLORREF acrCustColors[16]; // Массив для пользовательских цветов

ZeroMemory(&cc, sizeof(cc));

cc.lStructSize = sizeof(cc);

cc.hwndOwner = hwnd; // Родительское окно для диалогового окна выбора цвета

cc.lpCustColors = (LPDWORD)acrCustColors;

cc.rgbResult = RGB(255, 0, 0); // Начальный выбранный цвет

cc.Flags = CC\_FULLOPEN | CC\_RGBINIT; // Разрешает полный выбор цвета и инициализацию RGB

if (ChooseColor(&cc) == TRUE) {

currentColor = cc.rgbResult;

if (currentShape != nullptr)

currentShape->color = Color(GetRValue(currentColor), GetGValue(currentColor), GetBValue(currentColor));

}

}

// Procedure

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case WM\_COMMAND:

if (LOWORD(wParam) == ID\_BUTTON) { // ID\_BUTTON - это идентификатор вашей кнопки

OpenColorPicker(hwnd);

SetFocus(hwndChild);

}

if (LOWORD(wParam) == ID\_LAYER\_BUTTON) {

layers->addLayer();

wchar\_t layerText[20];

swprintf\_s(layerText, sizeof(layerText) / sizeof(layerText[0]), L"Layer %d", layers->getLayersCount());

SendMessage(hwndListBox, LB\_ADDSTRING, 0, (LPARAM)layerText);

SetFocus(hwndChild);

}

if (LOWORD(wParam) == ID\_LISTBOX && HIWORD(wParam) == LBN\_SELCHANGE) {

int selectedIndex = SendMessage(hwndListBox, LB\_GETCURSEL, 0, 0);

layers->setCurrentLayer(selectedIndex);

selectedShape = nullptr;

isCornerSelected = false;

SetFocus(hwndChild);

}

break;

case WM\_CREATE:

hwndListBox = CreateWindow(L"LISTBOX", L"",

WS\_VISIBLE | WS\_CHILD | LBS\_STANDARD,

margin, margin \* 2 + buttonHeight, toolsWidth - 2 \* margin, listHeight, hwnd, (HMENU)ID\_LISTBOX, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

layers = new Layers();

SendMessage(hwndListBox, LB\_ADDSTRING, 0, (LPARAM)L"Layer 1");

SendMessage(hwndListBox, LB\_SETCURSEL, 0, 0);

AddLayerButton = CreateWindow(

L"BUTTON", // Имя класса для кнопки

L"Add layer", // Текст на кнопке

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON, // Стили кнопки

margin, margin \* 3 + buttonHeight + listHeight, toolsWidth - 2 \* margin, buttonHeight, // Позиция и размеры кнопки

hwnd, // Родительское окно

(HMENU)ID\_LAYER\_BUTTON, // Идентификатор кнопки

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), // Дескриптор экземпляра приложения

NULL // Дополнительные параметры

);

break;

case WM\_DESTROY:

delete(layers);

delete(currentShape);

PostQuitMessage(0);

return 0;

case WM\_PAINT:

{

PAINTSTRUCT ps;

hdc = BeginPaint(hwnd, &ps);

// All painting occurs here, between BeginPaint and EndPaint.

FillRect(hdc, &ps.rcPaint, (HBRUSH)(COLOR\_WINDOW + 1));

EndPaint(hwnd, &ps);

}

case WM\_SIZE:

rc.right = LOWORD(lParam);

rc.bottom = HIWORD(lParam);

RECT clientRect;

GetClientRect(hwnd, &clientRect);

drawWindowHeight = clientRect.bottom - clientRect.top - 2 \* margin;

drawWindowWidth = clientRect.right - clientRect.left - toolsWidth - margin;

SetWindowPos(hwndChild, NULL, drawWindowX, drawWindowY, drawWindowWidth, drawWindowHeight, SWP\_NOZORDER);

break;

case WM\_MOVE:

WindowPosX = (int)(short)LOWORD(lParam); // horizontal position

WindowPosY = (int)(short)HIWORD(lParam); // vertical position

InvalidateRect(hwnd, 0, TRUE); // update window after moving

break;

}

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

}

LRESULT CALLBACK DrawWindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {

switch (uMsg)

{

case WM\_CREATE:

hdc = GetDC(hwnd);

RECT clientRect;

GetClientRect(hwnd, &clientRect);

hdcBuffer = CreateCompatibleDC(hdc);

hBitmap = CreateCompatibleBitmap(hdc, clientRect.right - clientRect.left, clientRect.bottom - clientRect.top);

SelectObject(hdcBuffer, hBitmap);

graphics = Graphics::FromHDC(hdcBuffer);

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

hdc = BeginPaint(hwnd, &ps);

// All painting occurs here, between BeginPaint and EndPaint.

FillRect(hdcBuffer, &ps.rcPaint, (HBRUSH)(COLOR\_WINDOW + 1));

for (int i = 0; i < layers->getLayersCount(); i++)

{

for (int j = 0; j < (\*layers)[i]->size(); j++)

{

Shape\* shape = (\*(\*layers)[i])[j];

DrawShape(graphics, shape->color, shape->shapeType, shape->x1, shape->y1, shape->x2, shape->y2);

}

if (layers->getCurrentLayer() == (\*layers)[i] && currentShape != nullptr)

{

DrawShape(graphics, currentShape->color, currentShape->shapeType, currentShape->x1, currentShape->y1, currentShape->x2, currentShape->y2);

}

}

BitBlt(hdc, ps.rcPaint.left, ps.rcPaint.top, ps.rcPaint.right - ps.rcPaint.left, ps.rcPaint.bottom - ps.rcPaint.top, hdcBuffer, 0, 0, SRCCOPY);

EndPaint(hwnd, &ps);

}

return 0;

case WM\_LBUTTONDOWN:

if (!isEditing)

{

StartDraw(lParam);

}

else if (isEditing && !isCornerSelected)

{

if (selectedShape == nullptr)

{

SelectShape(lParam);

}

else

{

SelectCorner(lParam);

}

}

else // corner selected

{

ChangeCorner(lParam, hwnd);

}

SetFocus(hwndChild);

break;

case WM\_MOUSEMOVE:

if (isDrawing)

{

ShapeDrawingCoord(lParam, hwnd);

}

break;

case WM\_LBUTTONUP:

if (isDrawing)

{

EndDrawing();

}

case WM\_SIZE:

rc.right = LOWORD(lParam);

rc.bottom = HIWORD(lParam);

GetClientRect(hwnd, &clientRect);

drawWindowHeight = clientRect.bottom - clientRect.top - 2 \* margin;

drawWindowWidth = clientRect.right - clientRect.left - toolsWidth - margin;

break;

case WM\_KEYDOWN:

if (wParam == VK\_ESCAPE)

PostMessage(hwnd, WM\_DESTROY, 0, 0);

else if (wParam == VK\_SPACE)

{

if (choosenShapeType == Shape::RECTANGLE)

choosenShapeType = Shape::CIRCLE;

else

choosenShapeType = Shape::RECTANGLE;

}

else if (wParam == VK\_SHIFT)

{

if (!isEditing)

{

if (isDrawing)

{

EndDrawing();

}

isEditing = true;

}

else

{

isEditing = false;

selectedShape = nullptr;

isCornerSelected = false;

}

}

else if (wParam == VK\_LEFT)

{

if (isEditing && selectedShape != nullptr)

selectedShape->moveFigure(-deltaWay, 0);

InvalidateRect(hwnd, 0, TRUE);

}

else if (wParam == VK\_RIGHT)

{

if (isEditing && selectedShape != nullptr)

selectedShape->moveFigure(deltaWay, 0);

InvalidateRect(hwnd, 0, TRUE);

}

else if (wParam == VK\_UP)

{

if (isEditing && selectedShape != nullptr)

selectedShape->moveFigure(0, -deltaWay);

InvalidateRect(hwnd, 0, TRUE);

}

else if (wParam == VK\_DOWN)

{

if (isEditing && selectedShape != nullptr)

selectedShape->moveFigure(0, deltaWay);

InvalidateRect(hwnd, 0, TRUE);

}

break;

}

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

}