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

ОТЧЕТ
к лабораторной работе №1
на тему

ОСНОВЫ ПРОГРАММИРОВАНИЯ В WIN 32 API

Студент Н. С. Шмидт
Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

1 Цель работы..... 3
2 Теоретические сведения..... 4
3 Результат выполнения..... 6
Заключение..... 8
Список использованных источников..... 9
Приложение А (обязательное) Листинг кода..... 10

1 ЦЕЛЬ РАБОТЫ

1 Изучение основных принципов работы с Win32 API.

2 Обработка основных оконных сообщений (создание и удаление окна, сообщения управляющих элементов).

3 Разработка оконного приложения с минимальной функциональной достаточностью – приложение, которое позволяет пользователю рисовать и редактировать графические фигуры (круги, прямоугольники, треугольники, ромбы) с помощью мыши и клавиш клавиатуры с возможностью изменения цвета последней нарисованной фигуры.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

В операционной системе Windows реализована объектноориентированная идеология. Базовый объект системы – окно, поведение которого определяется методом, называемым *функцией окна*. Графический образ окна на экране дисплея – прямоугольная рабочая область.

Независимо от своего типа любой объект Windows идентифицируется описателем или дескриптором (handle). *Дескриптор* – это ссылка на объект. Все взаимоотношения программного кода с объектом осуществляются только через его дескриптор.

Интерфейс прикладного программирования (API – Application Programming Interface) представляет собой совокупность 32-битных функций (Win32 API), которые предназначены для создания приложений (программ), работающих под управлением Microsoft Windows. Функции объявлены в заголовочных файлах. Главный из них − файл *windows.h*, в котором содержатся ссылки на другие заголовочные файлы.

Окно – это прямоугольная область экрана, в котором приложение отображает информацию и получает реакцию от пользователя. Одновременно на экране может отображаться несколько окон, в том числе, окон других приложений, однако лишь одно из них может получать реакцию от пользователя – *активное окно*. Пользователь использует клавиатуру, мышь и прочие устройства ввода для взаимодействия с приложением, которому принадлежит активное окно.

Каждое 32-битное приложение создает, по крайней мере, одно окно, называемое *главным окном*, которое обеспечивает пользователя основным интерфейсом взаимодействия с приложением. Кроме главного окна, приложение может использовать еще и другие типы окон: управляющие элементы, диалоговые окна, окна-сообщения.

Управляющий элемент – окно, непосредственно обеспечивающее тот или иной способ ввода информации пользователем. К управляющим элементам относятся: кнопки, поля ввода, списки, полосы прокрутки и т.п. Управляющие элементы обычно находятся в каком-либо диалоговом окне.

Диалоговое окно – это временное окно, содержащее управляющие элементы, обычно использующееся для получения дополнительной информации от пользователя. Диалоговые окна бывают модальные и немодальные. *Модальное* диалоговое окно требует, чтобы пользователь обязательно ввел обозначенную в окне информацию и закрыл окно прежде, чем приложение продолжит работу. *Немодальное* диалоговое окно позволяет пользователю, не закрывая диалогового окна, переключаться на другие окна этого приложения.

Окно-сообщение – это диалоговое окно предопределенного системой формата, предназначенное для вывода небольшого текстового сообщения с одной или несколькими кнопками.

В отличие от традиционного программирования на основе линейных алгоритмов, программы для Windows строятся по принципам событийно-управляемого программирования – стиля программирования, при котором поведение компонента системы определяется набором возможных внешних событий и ответных реакций компонента на них. Такими компонентами в Windows являются окна. С каждым окном в Windows связана определенная функция обработки событий. События для окон называются *сообщениями*. Сообщение относится к тому или иному типу, идентифицируемому 32-битным целым числом (например, *WM_COMMAND, WM_CREATE* и *WM_DESTROY*), и сопровождается парой 32-битных параметров (*WPARAM* и *LPARAM*), интерпретация которых зависит от типа сообщения.

Каждое окно принадлежит определенному *классу окон*. Окна одного класса имеют схожий вид, обслуживаются общей процедурой обработки событий, имеют одинаковые иконки и меню. Обычно каждое приложение создает для главного окна программы свой класс. Если приложению требуются дополнительные нестандартные окна, оно регистрирует другие классы. Стандартные диалоги и управляющие элементы принадлежат к предопределенным классам окон, для них не надо регистрировать новые классы.

Управляющие элементы, как и другие окна, принадлежат тому или иному классу окон. Windows предоставляет несколько предопределенных классов управляющих элементов. Программа может создавать управляющие элементы поштучно при помощи функции *CreateWindow* или оптом, загружая их вместе с шаблоном диалога из своих ресурсов. Управляющие элементы – это всегда дочерние окна. Управляющие элементы при возникновении некоторых событий, связанных с реакцией пользователя, посылают своему родительскому окну *сообщения-оповещения*.

3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ

Было создано приложение позволяющее рисовать фигуры(круги, прямоугольники, треугольники, ромбы). Для изменения типа фигуры используются клавиши numpad: numpad1 - прямоугольник, numpad2 - элипс, numpad3 - треугольник, numpad4 - ромб (рисунок 1).

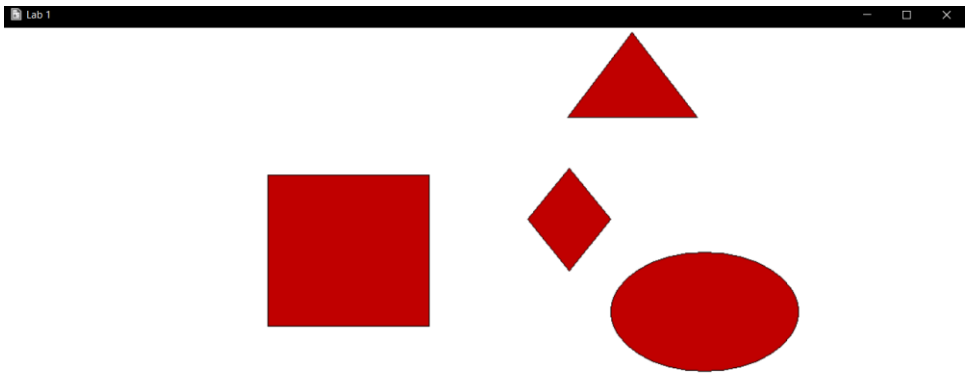

Рисунок 1 – Окно приложения

Для перехода в режим редактирования фигур используется клавиша *VK_SHIFT* и затем требуется выбрать нужную фигуру с помощью ЛКМ. Для изменения положения фигур используется обработка нажатия клавиш *VK_LEFT*, *VK_RIGHT*, *VK_UP* и *VK_DOWN* (рисунок 2).

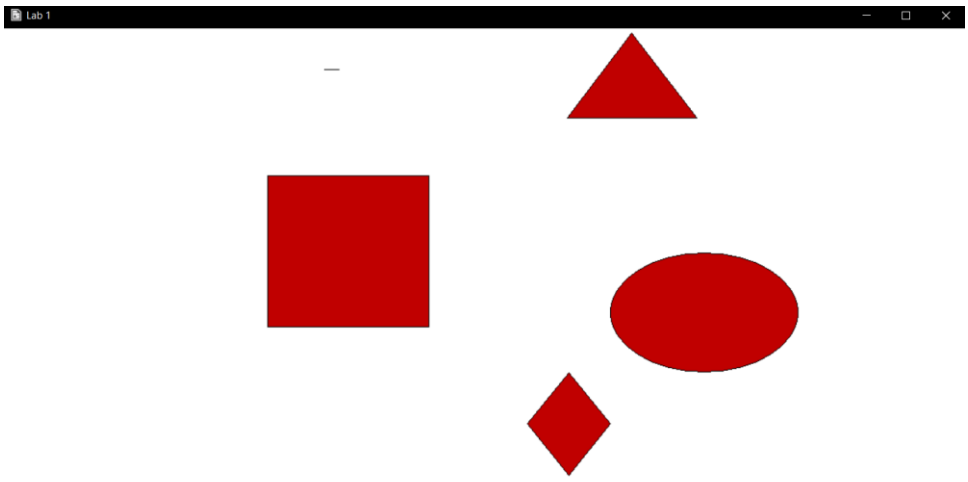

Рисунок 2 – Изменения положения ромба с помощью клавиш

Так же в этот момент появляется возможность изменить размер фигуры. Для этого необходимо нажать кнопки плюса или минуса на верхней панели клавиатуры (рисунок 3).

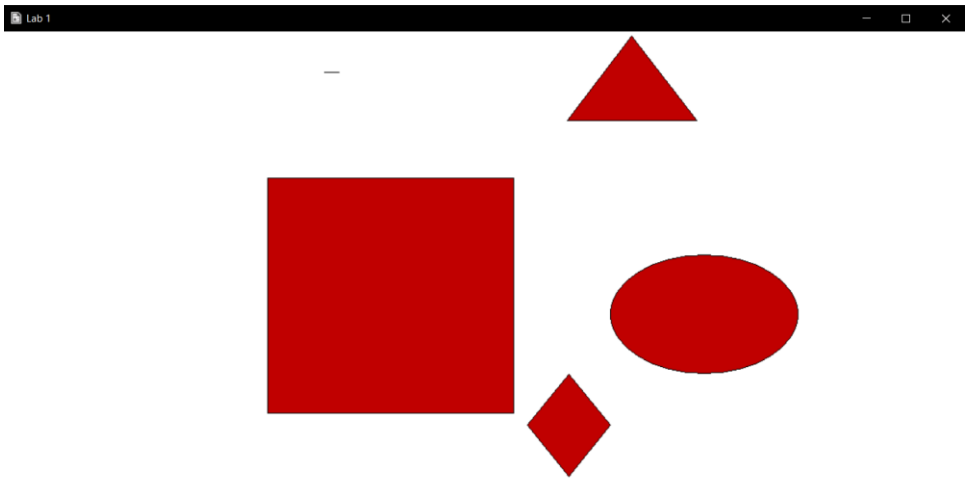

Рисунок 3 – Изменения размера прямоугольника с помощью клавиш

ЗАКЛЮЧЕНИЕ

В результате лабораторной работы были изучены основные принципы работы с Win32 API: виды окон, классы окон и их регистрация, обработка сообщений разных типов. Было создано приложение, которое позволяет пользователю рисовать и редактировать графические фигуры (круги, прямоугольники, треугольники, ромбы) с помощью мыши и клавиш клавиатуры.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Справочник по программированию для API Win32 [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/.

[2] КАК рисовать в Win32 API? [Электронный ресурс]. – Электронные данные. – Режим доступа: https://radiofront.narod.ru/htm/prog/htm/winda/api/paint.html#s

[3] Основные сообщения ОС Windows (Win32 API). Программирование в ОС Windows. Лекция 1. – Электронные данные. – Режим доступа: https://www.youtube.com/watch?v=wTArIolxch0

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

Листинг 1 – Файл main.cpp

```
#ifndef UNICODE
#define UNICODE
#endif

#include <windows.h>
#include <tchar.h>
#include <vector>
#include "resource.h"

#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;

int lastShapeId = 0;

class Shape {
public:
int shapeId;
int x1, y1, x2, y2;
int shapeType;
COLORREF color;

Shape() {}

Shape(int _shapeType)
{
shapeId = ++lastShapeId;
shapeType = _shapeType;
}

void checkCoord()
{
if (this->shapeType != 2) {
if (this->x1 > this->x2)
{
int buff = this->x1;
this->x1 = this->x2;
this->x2 = buff;
}
if (this->y1 > this->y2)
{
int buff = this->y1;
this->y1 = this->y2;
this->y2 = buff;
}
}
}
};
```

```
int choosenShapeType = 0;
std::vector<Shape*> shapes;
Shape* currentShape = nullptr;
bool isDrawing = false;
bool isEditing = false;

int selectedShapeIndex = -1;
HBRUSH brush = CreateSolidBrush(RGB(192, 0, 0));
HBRUSH original = CreateSolidBrush(RGB(192, 0, 0));
COLORREF red = RGB(255, 0, 0);
COLORREF white = RGB(255, 255, 255);
COLORREF black = RGB(0, 0, 0);
COLORREF green = RGB(0, 255, 0);
COLORREF blue = RGB(0, 0, 255);
LOGBRUSH br = { 0 };


void DrawShape(HDC hdc, int shapeType, int x1, int y1, int x2, int y2)
{
if (shapeType == 0)
{
SelectObject(hdc, brush);
Rectangle(hdc, x1, y1, x2, y2);
}
else if (shapeType == 1)
{
SelectObject(hdc, brush);
Ellipse(hdc, x1, y1, x2, y2);
}
else if (shapeType == 2)
{
SelectObject(hdc, brush);
POINT vertices[] = { {x1, y1},  {x2, y1}, {(x2 - x1) / 2 + x1, y2} };
Polygon(hdc, vertices, sizeof(vertices) / sizeof(vertices[0]));
}
else
{
SelectObject(hdc, brush);
POINT vertices[] = { {(x2 - x1) / 2 + x1, y1},  {x2, (y2 - y1) /2 + y1},
(x2 - x1) / 2 + x1, y2, { x1, (y2 - y1) / 2 + y1}};
Polygon(hdc, vertices, sizeof(vertices) / sizeof(vertices[0]));
}

}

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

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR
pCmdLine, int nCmdShow)
{
const wchar_t CLASS_NAME[] = L"Sample Window Class";

WNDCLASS wc = { };

HICON hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_ICON1));

wc.lpfnWndProc = WindowProc;
wc.hInstance = hInstance;
wc.lpszClassName = CLASS_NAME;
wc.style = CS_HREDRAW | CS_VREDRAW;
wc.hIcon = hIcon;
```

```
	RegisterClass(&wc);


	HWND hwnd = CreateWindowEx(
	0,
	CLASS_NAME,
	L"Lab 1",
	WS_OVERLAPPEDWINDOW,


	CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,

	NULL,
	NULL,
	hInstance,
	NULL
	);

	if (hwnd == NULL)
	{
	return 0;
	}

	ShowWindow(hwnd, nCmdShow);



	MSG msg = { };
	while (GetMessage(&msg, NULL, 0, 0) > 0)
	{
	TranslateMessage(&msg);
	DispatchMessage(&msg);
	}

	return 0;
	}

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

	case WM_DESTROY:
	shapes.clear();
	PostQuitMessage(0);
	return 0;

	case WM_PAINT:
	{
	PAINTSTRUCT ps;
	HDC hdc = BeginPaint(hwnd, &ps);

	FillRect(hdc, &ps.rcPaint, (HBRUSH)(COLOR_WINDOW + 1));

	for (int i = 0; i < shapes.size(); i++)
	{
	Shape shape = *shapes[i];
	brush = CreateSolidBrush(shape.color);
	DrawShape(hdc, shape.shapeType, shape.x1, shape.y1, shape.x2, shape.y2);
	}
```

```
      brush = original;
      if (currentShape != nullptr)
      DrawShape(hdc, currentShape->shapeType, currentShape->x1, currentShape-
>y1, currentShape->x2, currentShape->y2);

      EndPaint(hwnd, &ps);
      }
      return 0;

      case WM_LBUTTONDOWN:
      if (!isEditing)
      {
      isDrawing = true;
      currentShape = new Shape(choosenShapeType);
      currentShape->x1 = currentShape->x2 = LOWORD(lParam);
      currentShape->y1 = currentShape->y2 = HIWORD(lParam);
      }
      else if (isEditing)
      {
      if (selectedShapeIndex == -1)
      {
      for (int i = shapes.size() - 1; i >= 0; i--)
      {
      Shape shape = *shapes[i];
      int x = LOWORD(lParam);
      int y = HIWORD(lParam);
      if (shape.x1 <= x && shape.x2 >= x && shape.y1 <= y && shape.y2 >= y)
      {
      selectedShapeIndex = i;
      break;
      }
      }
      }
      }
      break;

      case WM_MOUSEMOVE:
      if (isDrawing)
      {
      currentShape->x2 = LOWORD(lParam);
      currentShape->y2 = HIWORD(lParam);
      InvalidateRect(hwnd, 0, TRUE);
      }
      break;

      case WM_LBUTTONUP:
      if (isDrawing)
      {
      isDrawing = false;
      GetObject(brush, sizeof(br), &br);
      currentShape->color = br.lbColor;
      currentShape->checkCoord();
      shapes.push_back(currentShape);
      }

      case WM_SIZE:
      rc.right = LOWORD(lParam);
      rc.bottom = HIWORD(lParam);
      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;

      case WM_KEYDOWN:
      if (wParam == VK_ESCAPE)
      PostMessage(hwnd, WM_DESTROY, 0, 0);
      else if (wParam == VK_OEM_PLUS) {
      if (isEditing && selectedShapeIndex != -1) {
      shapes[selectedShapeIndex]->x2 += 20;
      shapes[selectedShapeIndex]->y2 += 20;
      }
      InvalidateRect(hwnd, 0, TRUE);
      }
      else if (wParam == VK_OEM_MINUS) {
      if (isEditing && selectedShapeIndex != -1) {
      shapes[selectedShapeIndex]->x2 -= 20;
      shapes[selectedShapeIndex]->y2 -= 20;
      }
      InvalidateRect(hwnd, 0, TRUE);
      }
      else if (wParam == VK_NUMPAD1)
      {
      choosenShapeType = 0;
      }
      else if (wParam == VK_NUMPAD2)
      {
      choosenShapeType = 1;
      }
      else if (wParam == VK_NUMPAD3)
      {
      choosenShapeType = 2;
      }
      else if (wParam == VK_NUMPAD4)
      {
      choosenShapeType = 3;
      }
      else if (wParam == VK_SHIFT)
      {
      if (!isEditing)
      {
      if (isDrawing)
      {
      isDrawing = false;
      shapes.push_back(currentShape);
      }
      isEditing = true;
      }
      else
      {
      isEditing = false;
      selectedShapeIndex = -1;
      }
      }
      else if (wParam == VK_LEFT)
      {
      if (isEditing && selectedShapeIndex != -1) {
      shapes[selectedShapeIndex]->x1 -= 5;
      shapes[selectedShapeIndex]->x2 -= 5;
      }
      InvalidateRect(hwnd, 0, TRUE);
```

```
}
else if (wParam == VK_RIGHT)
{
if (isEditing && selectedShapeIndex != -1) {
shapes[selectedShapeIndex]->x1 += 5;
shapes[selectedShapeIndex]->x2 += 5;
}
InvalidateRect(hwnd, 0, TRUE);
}
else if (wParam == VK_UP)
{
if (isEditing && selectedShapeIndex != -1)
if (isEditing && selectedShapeIndex != -1) {
shapes[selectedShapeIndex]->y1 -= 5;
shapes[selectedShapeIndex]->y2 -= 5;
}
InvalidateRect(hwnd, 0, TRUE);
}
else if (wParam == VK_DOWN)
{
if (isEditing && selectedShapeIndex != -1) {
shapes[selectedShapeIndex]->y1 += 5;
shapes[selectedShapeIndex]->y2 += 5;
}
InvalidateRect(hwnd, 0, TRUE);
}
break;

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