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 И. В. Бобко

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 6](#_Toc146752070)

[Заключение 8](#_Toc146752071)

[Список использованных источников 9](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 10](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Целью выполнения лабораторной работы является создание оконного приложения на Win32 API, обладающее минимальным функционалом, позволяющим отработать базовые навыки написания программы на Win32 API, использования виджетов и обработки оконных сообщений (как базовых, так и пользовательских). Реализовать вышеупомянутые требования на примере приложения для вычисления корней квадратного уравнения , которое обрабатывает основные оконные сообщения через функцию WindowProc. Основные сообщения, такие как WM\_PAINT, WM\_CREATE, WM\_SIZE, WM\_COMMAND, и WM\_CLOSE, обрабатываются для управления отрисовкой приложения, обработки основных событий (открытие, сохранение файла), закрытия окна.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Очевидно, что окна являются центральными для Windows. Элементы управления пользовательского интерфейса, такие как кнопки и поля редактирования, сами являются окнами. Основное различие между элементом управления пользовательского интерфейса и окном приложения заключается в том, что элемент управления не существует сам по себе. Вместо этого элемент управления располагается относительно окна приложения. При перетаскивании окна приложения элемент управления перемещается вместе с ним, как и ожидалось. Кроме того, элемент управления и окно приложения могут взаимодействовать друг с другом.

Поэтому, необходимо думать об окне как о программной конструкции, которая занимает определенную часть экрана, может и не отображаться в данный момент, знает, как рисовать себя, и отвечает на события пользователя или операционной системы.

Для создания окна необходимо зарегистрировать класс окна с помощью функции RegisterClassEx и создать окно с помощью функции CreateWindowEx. Окно может иметь различные свойства, такие как заголовок, размеры, стиль и обработчики сообщений.

Во время выполнения приложение получит тысячи сообщений. Кроме того, приложение может иметь несколько окон, каждое из которых имеет собственную процедуру окна. Как программа получает все эти сообщения и доставляет их в правильную процедуру окна? Приложению требуется цикл для получения сообщений и их отправки в правильные окна.

Для каждого потока, создающего окно, операционная система создает очередь для оконных сообщений. Эта очередь содержит сообщения для всех окон, созданных в этом потоке. Сама очередь скрыта от программы. Однако можно извлечь сообщение из очереди, вызвав функцию GetMessage. Эта функция удаляет первое сообщение из заголовка очереди. Если очередь пуста, функция блокируется до тех пор, пока в очередь не будет помещено другое сообщение.

Для обработки оконных сообщений необходимо определить функцию оконной процедуры (WndProc), которая будет вызываться системой при возникновении сообщения. В функции WndProc нужно обрабатывать различные типы сообщений с помощью условных операторов и выполнять соответствующие действия.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В ходе выполнения лабораторной работы было разработано оконное приложение вычисляющее корни квадратного уравнения. (рисунок 1).

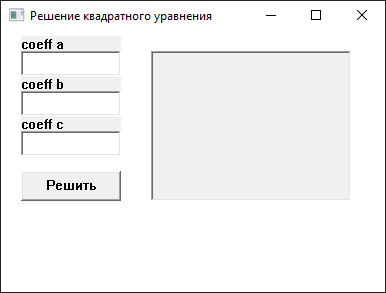
****

Рисунок 1 – Окно приложения

Для вычисления корней, пользователь должен вписать коэффициенты a, b, c и нажать кнопку решить. (рисунок 2).

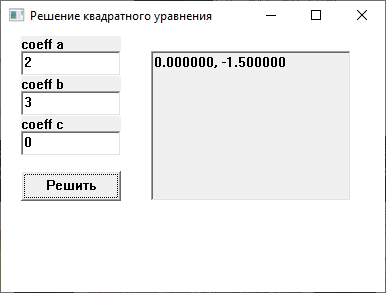


Рисунок 2– Выбор системы счисления

Приложение работает только числами. В случае ввода некорректного значения в поле, которое предназначено для исходного числа, и нажатия кнопки *«Решить»* происходит появление окна-сообщения с ошибкой (рисунки 3).

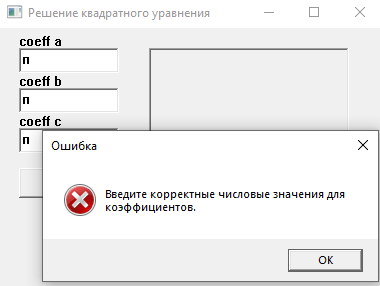


Рисунок 3– Введено некорректное значение

# 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 оконное приложение с минимальной функциональной достаточностью – приложение для перевода чисел между системами счисления с возможностью выбора исходной и конечной системы, а также обработкой некорректного ввода: слишком большие числа, некорректный символы в поле для ввода числа.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Что такое окно? [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/what-is-a-window.
2. Начало работы с классическими приложениями для Windows, которые используют API Win32 [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/desktop-programming.
3. Сообщения окна [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/window-messages.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл Source.cpp

#include <windows.h>

#include <cmath>

#include <string>

HWND hAEdit, hBEdit, hCEdit, hResultEdit, hButtonSolve, hButtonChangeColor, hButton;

HINSTANCE hInstance;

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

LRESULT CALLBACK KeyboardHook(int nCode, WPARAM wParam, LPARAM lParam);

HHOOK hKeyboardHook = NULL;

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {

WNDCLASSEX wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszMenuName = NULL;

wcex.lpszClassName = L"QuadraticEquationSolver";

wcex.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);

if (!RegisterClassEx(&wcex)) {

MessageBox(NULL, L"Не удалось зарегистрировать класс окна.", L"Ошибка", MB\_ICONERROR);

return 1;

}

HWND hWnd = CreateWindow(L"QuadraticEquationSolver", L"Решение квадратного уравнения",

WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT, 400, 300, NULL, NULL, hInstance, NULL);

if (!hWnd) {

MessageBox(NULL, L"Не удалось создать окно.", L"Ошибка", MB\_ICONERROR);

return 1;

}

CreateWindow(L"static", L"coeff a", WS\_VISIBLE | WS\_CHILD, 20, 5, 100, 25, hWnd, NULL, NULL, NULL);

CreateWindow(L"static", L"coeff b", WS\_VISIBLE | WS\_CHILD, 20, 45, 100, 25, hWnd, NULL, NULL, NULL);

CreateWindow(L"static", L"coeff c", WS\_VISIBLE | WS\_CHILD, 20, 85, 100, 25, hWnd, NULL, NULL, NULL);

hAEdit = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"EDIT", L"", WS\_CHILD | WS\_VISIBLE | ES\_NUMBER,

20, 20, 100, 25, hWnd, NULL, hInstance, NULL);

hBEdit = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"EDIT", L"", WS\_CHILD | WS\_VISIBLE | ES\_NUMBER,

20, 60, 100, 25, hWnd, NULL, hInstance, NULL);

hCEdit = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"EDIT", L"", WS\_CHILD | WS\_VISIBLE | ES\_NUMBER,

20, 100, 100, 25, hWnd, NULL, hInstance, NULL);

hButton = CreateWindow(L"BUTTON", L"Решить", WS\_CHILD | WS\_VISIBLE,

20, 140, 100, 30, hWnd, (HMENU)1, hInstance, NULL);

hResultEdit = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"EDIT", L"", WS\_CHILD | WS\_VISIBLE | ES\_READONLY,

150, 20, 200, 150, hWnd, NULL, hInstance, NULL);

hButtonSolve = CreateWindow(L"BUTTON", L"Решить", WS\_CHILD | WS\_VISIBLE,

20, 140, 100, 30, hWnd, (HMENU)1, hInstance, NULL);

hButtonChangeColor = CreateWindow(L"BUTTON", L"Изменить цвет фона", WS\_CHILD | WS\_VISIBLE,

180, 180, 150, 30, hWnd, (HMENU)2, hInstance, NULL);

hInstance = hInst;

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

hKeyboardHook = SetWindowsHookEx(WH\_KEYBOARD\_LL, KeyboardHook, hInstance, 0);

if (!hKeyboardHook) {

MessageBox(NULL, L"Не удалось установить глобальный хук клавиш.", L"Ошибка", MB\_ICONERROR);

return 1;

}

MSG msg;

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

UnhookWindowsHookEx(hKeyboardHook);

return (int)msg.wParam;

}

void ChangeWindowBackgroundColor(HWND hWnd) {

static COLORREF customColor = RGB(255, 255, 255);

CHOOSECOLOR cc = { sizeof(CHOOSECOLOR) };

cc.hwndOwner = hWnd;

cc.lpCustColors = &customColor;

cc.Flags = CC\_FULLOPEN | CC\_RGBINIT;

if (ChooseColor(&cc)) {

customColor = cc.rgbResult;

HBRUSH hBrush = CreateSolidBrush(customColor);

SetClassLongPtr(hWnd, GCLP\_HBRBACKGROUND, (LONG\_PTR)hBrush);

InvalidateRect(hWnd, NULL, TRUE);

RedrawWindow(hWnd, NULL, NULL, RDW\_ERASE | RDW\_FRAME | RDW\_INVALIDATE);

}

}

LRESULT CALLBACK KeyboardHook(int nCode, WPARAM wParam, LPARAM lParam) {

if (nCode >= 0 && wParam == WM\_KEYDOWN) {

KBDLLHOOKSTRUCT\* kbStruct = (KBDLLHOOKSTRUCT\*)lParam;

if (kbStruct->vkCode == 'B') {

ChangeWindowBackgroundColor(GetForegroundWindow());

}

}

return CallNextHookEx(hKeyboardHook, nCode, wParam, lParam);

}

void SolveQuadraticEquation(HWND hWnd) {

wchar\_t aBuffer[100], bBuffer[100], cBuffer[100];

GetWindowText(hAEdit, aBuffer, sizeof(aBuffer));

GetWindowText(hBEdit, bBuffer, sizeof(bBuffer));

GetWindowText(hCEdit, cBuffer, sizeof(cBuffer));

double a = \_wtof(aBuffer);

double b = \_wtof(bBuffer);

double c = \_wtof(cBuffer);

double discriminant = b \* b - 4 \* a \* c;

if (discriminant < 0) {

SetWindowText(hResultEdit, L"Нет действительных корней");

}

else if (discriminant == 0) {

double root = -b / (2 \* a);

SetWindowText(hResultEdit, std::to\_wstring(root).c\_str());

}

else {

double root1 = (-b + sqrt(discriminant)) / (2 \* a);

double root2 = (-b - sqrt(discriminant)) / (2 \* a);

SetWindowText(hResultEdit, (std::to\_wstring(root1) + L", " + std::to\_wstring(root2)).c\_str());

}

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) {

switch (message) {

case WM\_COMMAND:

if (LOWORD(wParam) == 1) {

SolveQuadraticEquation(hWnd);

}

else if (LOWORD(wParam) == 2) {

ChangeWindowBackgroundColor(hWnd);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

break;

}

return 0;

}