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 №1

на тему

Основы программирования в Win32 API. Оконное приложение Win32 с минимальной достаточной функциональностью. Обработка основных оконных приложений

Выполнил:

студент гр. 153503

Звягинцева В.А.

Проверил:

Гриценко Н. Ю.

Минск 2023

СОДЕРЖАНИЕ

1 Постановка задачи5

2 Платформа программного обеспечения6

3 Результаты выполнения лабораторной работы8

Вывод10

Список использованных источников11

Приложение А12

**1 ПОСТАНОВКА ЗАДАЧИ**

Целью выполнения лабораторной работы является изучить основы программирования Win32 и разработать оконное приложение с минимальной функциональностью, обрабатывающее основные оконные сообщения. Разработать калькулятор с графическим интерфейсом, способный выполнять сложные математические операции, такие как вычисление функций синуса, косинуса, тангенса и котангенса.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Win32 API (также называемый Windows API) — это исходная платформа для собственных Windows-приложений на языке C/C++, которым требуется прямой доступ к Windows и оборудованию. Он обеспечивает первоклассный опыт разработки без зависимости от управляемой среды выполнения, такой как .NET и WinRT (для приложений UWP для Windows 10). Благодаря этому API Win32 стает оптимальной платформой для приложений, которым требуется самый высокий уровень производительности и прямой доступ к системному оборудованию.

Win32 API состоит из набора функций и подпрограмм, предоставляющих программный доступ к возможностям операционной системы, т.е. программные интерфейсы приложений представляют собой наборы функций (в этот обобщённый термин мы включаем и подпрограммы), которые обеспечивают сервисы данного приложения. Win32 API содержит более 3000 функции для реализации всех видов сервисов операционной системы.

API-функции Windows входят в состав динамически подключаемых библиотек. Динамически подключаемая библиотека (Dynamic Link Library – DLL) является исполняемым файлом, который содержит несколько экспортируемых функции (exportable functions), то есть функций, к которым могут обращаться другие исполняемые приложения (ЕХЕ или DLL). Файлы DLL намного проще файлов ЕХЕ, например, в них нет кода, который управлял бы графическим интерфейсом или обрабатывал сообщения Windows.

В ходе выполнения лабораторной работы были использованы следующие технологии:

* События и обработчик событий: События - это событийные действия, которые происходят внутри приложения в ответ на различные действия пользователя или другие события операционной системы. Для обработки этих событий используются обработчики событий. В данном приложении определяется функция WndProc (которая является обработчиком событий для главного окна приложения) и в этой функции обрабатывает различные сообщения, связанные с событиями.
* Создание пользовательского интерфейса: для создания графического интерфейса приложения были использованы стандартные элементы управления Windows, такие как окна, кнопки и текстовые поля. Эти элементы были созданы и настроены с использованием Win32 API функций.
* Создание дочерних элементов управления: с помощью функции CreateWindow создаются дочерние элементы управления, такие как кнопки и поле ввода, которые добавляются в главное окно приложения. Каждому элементу присваивается уникальный идентификатор (ID), который используется для их идентификации и обработки событий.
* Создание окна: CreateWindow используется для создания главного окна приложения. Оно принимает параметры, такие как имя класса окна, заголовок окна, стиль окна и др.

**3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ**

В ходе выполнения лабораторной работы был разработан калькулятор. При запуске пользователь видит интерфейс приложения, который представлен на рисунке 1.

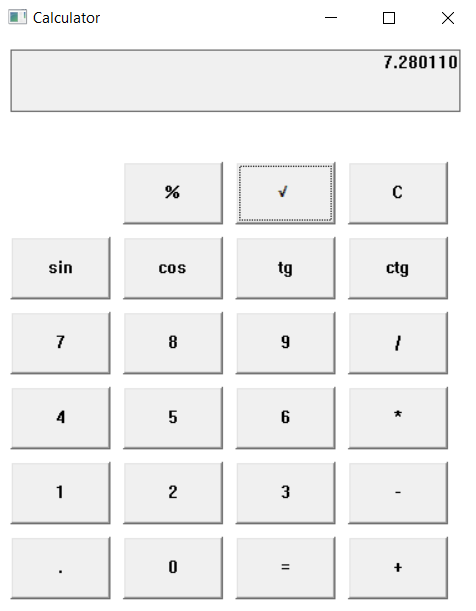


Рисунок 1. – Результат работы программы

При попытке поделить на 0 или вычислить корень из отрицательного числа, пользователь увидит сообщение об ошибке, которые представлены на рисунках 2 и 3 соответственно:

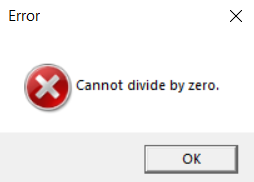


Рисунок 2. – Сообщение об ошибке при делении на 0

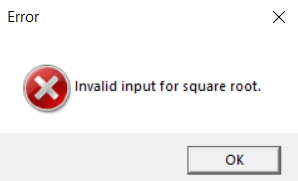


Рисунок 3. – Сообщение об ошибке при попытке вычислить корень из отрицательного числа

**ВЫВОД**

В результате выполнения лабораторной работы были изучены и освоены основы программирования Win32. Было разработано оконное приложение с минимальной функциональностью, обрабатывающее основные оконные сообщения, представляющее собой калькулятор с графическим интерфейсом, способный выполнять сложные математические операции, такие как вычисление функций синуса, косинуса, тангенса и котангенса.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

[1] [Электронный ресурс]. – Режим доступа: [https://learn.microsoft.com/ru-ru/windows/win32/api – Дата доступа 28.09.2023](https://learn.microsoft.com/ru-ru/windows/win32/api%20–%20Дата%20доступа%2028.09.2023)

[2] [Электронный ресурс]. – Режим доступа: <https://prog-cpp.ru/winelements/> – Дата доступа 28.09.2023

[3] [Электронный ресурс]. – Режим доступа: [https://learn.microsoft.com/en-us/windows/win32/controls/button-messages – Дата доступа 28.09.2023](https://learn.microsoft.com/en-us/windows/win32/controls/button-messages%20–%20Дата%20доступа%2028.09.2023)

[4] Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.: ип.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг кода**

**main.cpp**

#include <Windows.h>

#include <string>

#include <cmath>

#define PI 3.14159265358979323846

#define ID\_EDIT 1000

#define ID\_BUTTON\_0 1001

#define ID\_BUTTON\_1 1002

#define ID\_BUTTON\_2 1003

#define ID\_BUTTON\_3 1004

#define ID\_BUTTON\_4 1005

#define ID\_BUTTON\_5 1006

#define ID\_BUTTON\_6 1007

#define ID\_BUTTON\_7 1008

#define ID\_BUTTON\_8 1009

#define ID\_BUTTON\_9 1010

#define ID\_BUTTON\_ADD 1011

#define ID\_BUTTON\_SUBTRACT 1012

#define ID\_BUTTON\_MULTIPLY 1013

#define ID\_BUTTON\_DIVIDE 1014

#define ID\_BUTTON\_EQUALS 1015

#define ID\_BUTTON\_CLEAR 1016

#define ID\_BUTTON\_SQUARE\_ROOT 1017

#define ID\_BUTTON\_DECIMAL 1018

#define ID\_BUTTON\_SIN 1019

#define ID\_BUTTON\_COS 1020

#define ID\_BUTTON\_TAN 1021

#define ID\_BUTTON\_COT 1022

#define ID\_BUTTON\_DEGREES 1023

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

double Trigonometric(const std::wstring& functionName, double value);

HWND g\_hEdit;

std::wstring g\_inputText;

double g\_currentValue = 0.0;

bool g\_isNewInput = true;

bool g\_decimalEntered = false;

std::wstring g\_operator; // Хранение текущего оператора.

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)

{

WNDCLASSEX wcex;

memset(&wcex, 0, sizeof(WNDCLASSEX));

wcex.cbSize = sizeof(WNDCLASSEX);

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

wcex.lpfnWndProc = WndProc;

wcex.hInstance = hInstance;

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

wcex.lpszClassName = L"Calculator";

RegisterClassEx(&wcex);

HWND hWnd = CreateWindow(L"Calculator", L"Calculator", WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, 400, 500, nullptr, nullptr, hInstance, nullptr);

if (!hWnd)

{

return false;

}

g\_hEdit = CreateWindow(L"EDIT", L"", WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_RIGHT | ES\_READONLY,

10, 10, 360, 50, hWnd, (HMENU)ID\_EDIT, hInstance, nullptr);

CreateWindow(L"BUTTON", L"0", WS\_CHILD | WS\_VISIBLE, 100, 400, 80, 50, hWnd, (HMENU)ID\_BUTTON\_0, hInstance, nullptr);

CreateWindow(L"BUTTON", L"1", WS\_CHILD | WS\_VISIBLE, 10, 340, 80, 50, hWnd, (HMENU)ID\_BUTTON\_1, hInstance, nullptr);

CreateWindow(L"BUTTON", L"2", WS\_CHILD | WS\_VISIBLE, 100, 340, 80, 50, hWnd, (HMENU)ID\_BUTTON\_2, hInstance, nullptr);

CreateWindow(L"BUTTON", L"3", WS\_CHILD | WS\_VISIBLE, 190, 340, 80, 50, hWnd, (HMENU)ID\_BUTTON\_3, hInstance, nullptr);

CreateWindow(L"BUTTON", L"4", WS\_CHILD | WS\_VISIBLE, 10, 280, 80, 50, hWnd, (HMENU)ID\_BUTTON\_4, hInstance, nullptr);

CreateWindow(L"BUTTON", L"5", WS\_CHILD | WS\_VISIBLE, 100, 280, 80, 50, hWnd, (HMENU)ID\_BUTTON\_5, hInstance, nullptr);

CreateWindow(L"BUTTON", L"6", WS\_CHILD | WS\_VISIBLE, 190, 280, 80, 50, hWnd, (HMENU)ID\_BUTTON\_6, hInstance, nullptr);

CreateWindow(L"BUTTON", L"7", WS\_CHILD | WS\_VISIBLE, 10, 220, 80, 50, hWnd, (HMENU)ID\_BUTTON\_7, hInstance, nullptr);

CreateWindow(L"BUTTON", L"8", WS\_CHILD | WS\_VISIBLE, 100, 220, 80, 50, hWnd, (HMENU)ID\_BUTTON\_8, hInstance, nullptr);

CreateWindow(L"BUTTON", L"9", WS\_CHILD | WS\_VISIBLE, 190, 220, 80, 50, hWnd, (HMENU)ID\_BUTTON\_9, hInstance, nullptr);

CreateWindow(L"BUTTON", L".", WS\_CHILD | WS\_VISIBLE, 10, 400, 80, 50, hWnd, (HMENU)ID\_BUTTON\_DECIMAL, hInstance, nullptr);

CreateWindow(L"BUTTON", L"+", WS\_CHILD | WS\_VISIBLE, 280, 400, 80, 50, hWnd, (HMENU)ID\_BUTTON\_ADD, hInstance, nullptr);

CreateWindow(L"BUTTON", L"-", WS\_CHILD | WS\_VISIBLE, 280, 340, 80, 50, hWnd, (HMENU)ID\_BUTTON\_SUBTRACT, hInstance, nullptr);

CreateWindow(L"BUTTON", L"\*", WS\_CHILD | WS\_VISIBLE, 280, 280, 80, 50, hWnd, (HMENU)ID\_BUTTON\_MULTIPLY, hInstance, nullptr);

CreateWindow(L"BUTTON", L"/", WS\_CHILD | WS\_VISIBLE, 280, 220, 80, 50, hWnd, (HMENU)ID\_BUTTON\_DIVIDE, hInstance, nullptr);

CreateWindow(L"BUTTON", L"=", WS\_CHILD | WS\_VISIBLE, 190, 400, 80, 50, hWnd, (HMENU)ID\_BUTTON\_EQUALS, hInstance, nullptr);

CreateWindow(L"BUTTON", L"C", WS\_CHILD | WS\_VISIBLE, 280, 100, 80, 50, hWnd, (HMENU)ID\_BUTTON\_CLEAR, hInstance, nullptr);

CreateWindow(L"BUTTON", L"√", WS\_CHILD | WS\_VISIBLE, 190, 100, 80, 50, hWnd, (HMENU)ID\_BUTTON\_SQUARE\_ROOT, hInstance, nullptr);

CreateWindow(L"BUTTON", L"%", WS\_CHILD | WS\_VISIBLE, 100, 100, 80, 50, hWnd, (HMENU)ID\_BUTTON\_DEGREES, hInstance, nullptr);

CreateWindow(L"BUTTON", L"sin", WS\_CHILD | WS\_VISIBLE, 10, 160, 80, 50, hWnd, (HMENU)ID\_BUTTON\_SIN, hInstance, nullptr);

CreateWindow(L"BUTTON", L"cos", WS\_CHILD | WS\_VISIBLE, 100, 160, 80, 50, hWnd, (HMENU)ID\_BUTTON\_COS, hInstance, nullptr);

CreateWindow(L"BUTTON", L"tg", WS\_CHILD | WS\_VISIBLE, 190, 160, 80, 50, hWnd, (HMENU)ID\_BUTTON\_TAN, hInstance, nullptr);

CreateWindow(L"BUTTON", L"ctg", WS\_CHILD | WS\_VISIBLE, 280, 160, 80, 50, hWnd, (HMENU)ID\_BUTTON\_COT, hInstance, nullptr);

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

MSG msg;

while (GetMessage(&msg, nullptr, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return (int)msg.wParam;

}

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

{

switch (message)

{

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case ID\_BUTTON\_0:

case ID\_BUTTON\_1:

case ID\_BUTTON\_2:

case ID\_BUTTON\_3:

case ID\_BUTTON\_4:

case ID\_BUTTON\_5:

case ID\_BUTTON\_6:

case ID\_BUTTON\_7:

case ID\_BUTTON\_8:

case ID\_BUTTON\_9:

if (g\_isNewInput)

{

g\_inputText.clear();

g\_isNewInput = false;

}

g\_inputText += wchar\_t('0' + LOWORD(wParam) - ID\_BUTTON\_0);

SetWindowText(g\_hEdit, g\_inputText.c\_str());

break;

case ID\_BUTTON\_ADD:

case ID\_BUTTON\_SUBTRACT:

case ID\_BUTTON\_MULTIPLY:

case ID\_BUTTON\_DIVIDE:

if (!g\_isNewInput)

{

// Если предыдущий ввод был числом, сбросьте флаг десятичной точки.

if (g\_inputText.find(L".") != std::wstring::npos)

{

g\_decimalEntered = false;

}

if (!g\_inputText.empty()) // Если ввод начинается с оператора, добавляем 0.

{

g\_currentValue = std::wcstod(g\_inputText.c\_str(), nullptr);

g\_inputText.clear();

g\_isNewInput = true;

}

}

switch (LOWORD(wParam))

{

case ID\_BUTTON\_ADD:

g\_operator = L"+";

break;

case ID\_BUTTON\_SUBTRACT:

g\_operator = L"-";

break;

case ID\_BUTTON\_MULTIPLY:

g\_operator = L"\*";

break;

case ID\_BUTTON\_DIVIDE:

g\_operator = L"/";

break;

}

break;

case ID\_BUTTON\_EQUALS:

if (!g\_isNewInput)

{

if (!g\_inputText.empty()) // Учитываем второе число.

{

double newValue = std::wcstod(g\_inputText.c\_str(), nullptr);

if (g\_operator == L"+")

g\_currentValue += newValue;

else if (g\_operator == L"-")

g\_currentValue -= newValue;

else if (g\_operator == L"\*")

g\_currentValue \*= newValue;

else if (g\_operator == L"/")

{

if (newValue != 0.0)

g\_currentValue /= newValue;

else

{

MessageBox(hWnd, L"Cannot divide by zero.", L"Error", MB\_OK | MB\_ICONERROR);

g\_currentValue = 0.0;

}

}

g\_inputText = std::to\_wstring(g\_currentValue);

g\_isNewInput = true;

SetWindowText(g\_hEdit, g\_inputText.c\_str());

}

}

break;

case ID\_BUTTON\_SIN:

case ID\_BUTTON\_COS:

case ID\_BUTTON\_TAN:

case ID\_BUTTON\_COT:

if (!g\_isNewInput)

{

double newValue = std::wcstod(g\_inputText.c\_str(), nullptr);

switch (LOWORD(wParam))

{

case ID\_BUTTON\_SIN:

g\_operator = L"sin";

break;

case ID\_BUTTON\_COS:

g\_operator = L"cos";

break;

case ID\_BUTTON\_TAN:

g\_operator = L"tg";

break;

case ID\_BUTTON\_COT:

g\_operator = L"ctg";

break;

}

double result = Trigonometric(g\_operator, newValue);

g\_currentValue = result;

g\_inputText = std::to\_wstring(result);

g\_isNewInput = true;

SetWindowText(g\_hEdit, g\_inputText.c\_str());

}

break;

case ID\_BUTTON\_DEGREES:

if (!g\_isNewInput)

{

double newValue = std::wcstod(g\_inputText.c\_str(), nullptr);

g\_currentValue = newValue / 100;

g\_inputText = std::to\_wstring(g\_currentValue);

g\_isNewInput = true;

SetWindowText(g\_hEdit, g\_inputText.c\_str());

}

break;

case ID\_BUTTON\_DECIMAL:

// Проверяем, что десятичная точка ещё не была введена.

if (!g\_decimalEntered)

{

g\_inputText += L"."; // Добавляем десятичную точку к введённому тексту.

g\_decimalEntered = true; // Устанавливаем флаг, что десятичная точка введена.

SetWindowText(g\_hEdit, g\_inputText.c\_str());

}

break;

case ID\_BUTTON\_CLEAR:

g\_inputText.clear();

g\_currentValue = 0.0;

g\_isNewInput = true;

g\_decimalEntered = false;

g\_operator.clear(); // Очистка текущего оператора.

SetWindowText(g\_hEdit, L"");

break;

case ID\_BUTTON\_SQUARE\_ROOT:

double newValue = std::wcstod(g\_inputText.c\_str(), nullptr);

if (newValue >= 0.0)

{

g\_currentValue = sqrt(newValue);

g\_inputText = std::to\_wstring(g\_currentValue);

g\_isNewInput = true;

SetWindowText(g\_hEdit, g\_inputText.c\_str());

}

else

{

MessageBox(hWnd, L"Invalid input for square root.", L"Error", MB\_OK | MB\_ICONERROR);

g\_currentValue = 0.0;

g\_inputText.clear();

SetWindowText(g\_hEdit, L"");

}

break;

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

double Trigonometric(const std::wstring& functionName, double value)

{

if (functionName == L"sin")

{

return sin(value \* PI / 180);

}

else if (functionName == L"cos")

{

return cos(value \* PI / 180);

}

else if (functionName == L"tan")

{

return tan(value \* PI / 180);

}

else if (functionName == L"ctg")

{

return 1.0 / tan(value \* PI / 180);

}

else return 0.0;

}