МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет: Информационных технологий

Кафедра: Программной инженерии

Выполнила: студентка 1 курса 5 группы

специальности ПОИТ Дзивнель М.А.

Проверил: преподаватель

Белодед Николай Иванович

**РЕФЕРАТ**

По дисциплине “Основы алгоритмизации и программирования”

На тему “Основы создания оконных приложений”

Минск

2023

Оглавление

[**Введение** 3](#_Toc135000864)

[**Классическое Windows приложение** 3](#_Toc135000865)

[**Использование WinForms** 7](#_Toc135000866)

[**Использование MFC** 9](#_Toc135000867)

[**Вывод** 12](#_Toc135000868)

# **Введение**

Создание оконных приложений на C++ - важная и интересная тема, которая имеет широкое применение в современной разработке программного обеспечения. Оконные приложения являются неотъемлемой частью операционных систем и обеспечивают удобный и интуитивно понятный интерфейс взаимодействия пользователя с программой. В данном реферате будет рассмотрено создание оконных приложений на языке С++ тремя способами: с помощью классического Windows приложения, с помощью WinForms, а также MFC.

# **Классическое Windows приложение**

Для создания оконного приложения на основе классического виндоус приложения на C++ необходимо выполнить следующие шаги:

1. Создать новый проект в Visual Studio, выбрав тип приложения "Windows Desktop Application".
2. Определить функцию "WinMain", которая является точкой входа в приложение. В ней необходимо зарегистрировать класс окна, создать окно и запустить цикл обработки сообщений.

Пример кода главной функции:

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

{

HWND hwnd;

MSG msg;

WNDCLASSEX wc={ };

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

wc.cbSize = sizeof(WNDCLASSEX);

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

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

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

wc.hbrBackground = (HBRUSH)GetStockObject(WHITE\_BRUSH);

wc.hInstance = hInstance;

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.lpfnWndProc = WndProc;

wc.lpszMenuName = NULL;

wc.lpszClassName = L"WinMainClass";

RegisterClassEx(&wc);

hwnd = CreateWindow(

wc.lpszClassName,

L"Calculator",

WS\_OVERLAPPEDWINDOW,

400, 200, 340, 400,

NULL, NULL, hInstance, NULL);

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}

1. Определить процедуру обработки сообщений "WndProc", которая будет вызываться при получении приложением сообщений от операционной системы. В этой функции необходимо обработать все необходимые сообщения и произвести соответствующие действия.

Обьявление процедуры WndProc:

LRESULT WINAPI WndProc(HWND hwnd, UINT msg, WPARAM wparam, LPARAM lparam)

1. Создать окно с помощью функции "CreateWindow". В этой функции необходимо задать параметры окна, такие как его класс, заголовок, размер и положение на экране.

Регистрация класса и создание окна:

RegisterClassEx(&wc);

hwnd = CreateWindow(

wc.lpszClassName,

L"Calculator",

WS\_OVERLAPPEDWINDOW,

400, 200, 340, 400,

NULL, NULL, hInstance, NULL);

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

1. В цикле обработки сообщений вызывать функцию "TranslateMessage", которая преобразует сообщения клавиатуры в символы и функцию "DispatchMessage", которая передает сообщения процедуре обработки сообщений "WndProc".

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

1. Обработать все необходимые сообщения в процедуре "WndProc". Например, при получении сообщения WM\_DESTROY необходимо закрыть окно с помощью функции PostQuitMessage(0).

Реализация WndProc:

LRESULT WINAPI WndProc(HWND hwnd, UINT msg, WPARAM wparam, LPARAM lparam)

{

static HWND hStatic;

static HWND hEdit1, hEdit2;

static HWND hBtn;

int num1 = 0, num2 = 1, result=0;

float fresult;

char cnum1[100], cnum2[100];

GetWindowTextA(hEdit1, cnum1, sizeof(cnum1));

GetWindowTextA(hEdit2, cnum2, sizeof(cnum2));

switch (msg)

{

case WM\_CREATE: {

//buttons

hBtn = CreateWindow(

L"BUTTON",

L"+",

WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON,

20, 65, 40, 30,

hwnd, reinterpret\_cast<HMENU>(1), NULL, NULL

);

hBtn = CreateWindow(

L"BUTTON",

L"-",

WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON,

80, 65, 40, 30,

hwnd, reinterpret\_cast<HMENU>(2), NULL, NULL

);

hBtn = CreateWindow(

L"BUTTON",

L"\*",

WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON,

140, 65, 40, 30,

hwnd, reinterpret\_cast<HMENU>(3), NULL, NULL

);

hBtn = CreateWindow(

L"BUTTON",

L"/",

WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON,

200, 65, 40, 30,

hwnd, reinterpret\_cast<HMENU>(4), NULL, NULL

);

hBtn = CreateWindow(

L"BUTTON",

L"C",

WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON | BS\_OWNERDRAW,

260, 65, 40, 30,

hwnd, reinterpret\_cast<HMENU>(5), NULL, NULL

);

//text input areas

hEdit1 = CreateWindow(

L"eDit",

L"",

WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_LEFT | ES\_AUTOHSCROLL,

20, 10, 135, 35,

hwnd, NULL, NULL, NULL

);

hEdit2 = CreateWindow(

L"eDit",

L"",

WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_LEFT | ES\_AUTOHSCROLL,

165, 10, 135, 35,

hwnd, NULL, NULL, NULL

);

//text result

hStatic = CreateWindow(

L"static",

L"0",

WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_AUTOHSCROLL,

20, 120, 280, 30,

hwnd, NULL, NULL, NULL

);

}

case WM\_DRAWITEM:

{

static HFONT hfontButton = CreateFont(28, 0, 0, 0, FW\_NORMAL, 0, 0, 0, DEFAULT\_CHARSET,

OUT\_DEFAULT\_PRECIS, CLIP\_DEFAULT\_PRECIS,

DEFAULT\_QUALITY, DEFAULT\_PITCH, L"Arial Black");

LPDRAWITEMSTRUCT Item = (LPDRAWITEMSTRUCT)lparam;

/\* установить отображение текста \*/

SetBkMode(Item->hDC, TRANSPARENT);

SetTextColor(Item->hDC, 0x000000);

SelectObject(Item->hDC, hfontButton);

HBRUSH brush;

brush = CreateSolidBrush(RGB(255, 59, 0));

/\* белый фон при нажатии \*/

if (Item->itemState & ODS\_SELECTED)

FillRect(Item->hDC, &Item->rcItem, (HBRUSH)GetStockObject(WHITE\_BRUSH));

else

FillRect(Item->hDC, &Item->rcItem, brush);

/\* нарисовать текст \*/

int len = GetWindowTextLength(Item->hwndItem);

char\* buf = new char[len + 1];

GetWindowTextA(Item->hwndItem, buf, len + 1);

DrawTextA(Item->hDC, buf, len, &Item->rcItem, DT\_CENTER);

return true;

}

case WM\_COMMAND:

switch (LOWORD(wparam))

{

case PLUS\_NUM:

{

result = atoi(cnum1) + atoi(cnum2);

SetWindowTextA(hStatic, (std::to\_string(result)).c\_str());

break;

}

case MINUS\_NUM:

{

result = atoi(cnum1) - atoi(cnum2);

SetWindowTextA(hStatic, (std::to\_string(result)).c\_str());

break;

}

case MUL\_NUM:

{

result = atoi(cnum1) \* atoi(cnum2);

SetWindowTextA(hStatic, (std::to\_string(result)).c\_str());

break;

}

case DIV\_NUM:

{

if (atoi(cnum2) == 0)

{

MessageBox(GetActiveWindow(), L"Второе число не может быть 0",L"Error", MB\_ICONERROR);

SetWindowTextA(hEdit2, "");

break;

}

fresult = atof(cnum1) / atof(cnum2);

SetWindowTextA(hStatic, (std::to\_string(fresult)).c\_str());

break;

}

case ERROR\_CALC:

{

num1 = 0;

num2 = 0;

SetWindowTextA(hEdit1, "");

SetWindowTextA(hEdit2, "");

SetWindowTextA(hStatic, "0");

break;

}

default:

break;

}

break;

case WM\_DESTROY: {

PostQuitMessage(0);

break;

}

default:

return DefWindowProc(hwnd, msg, wparam, lparam);

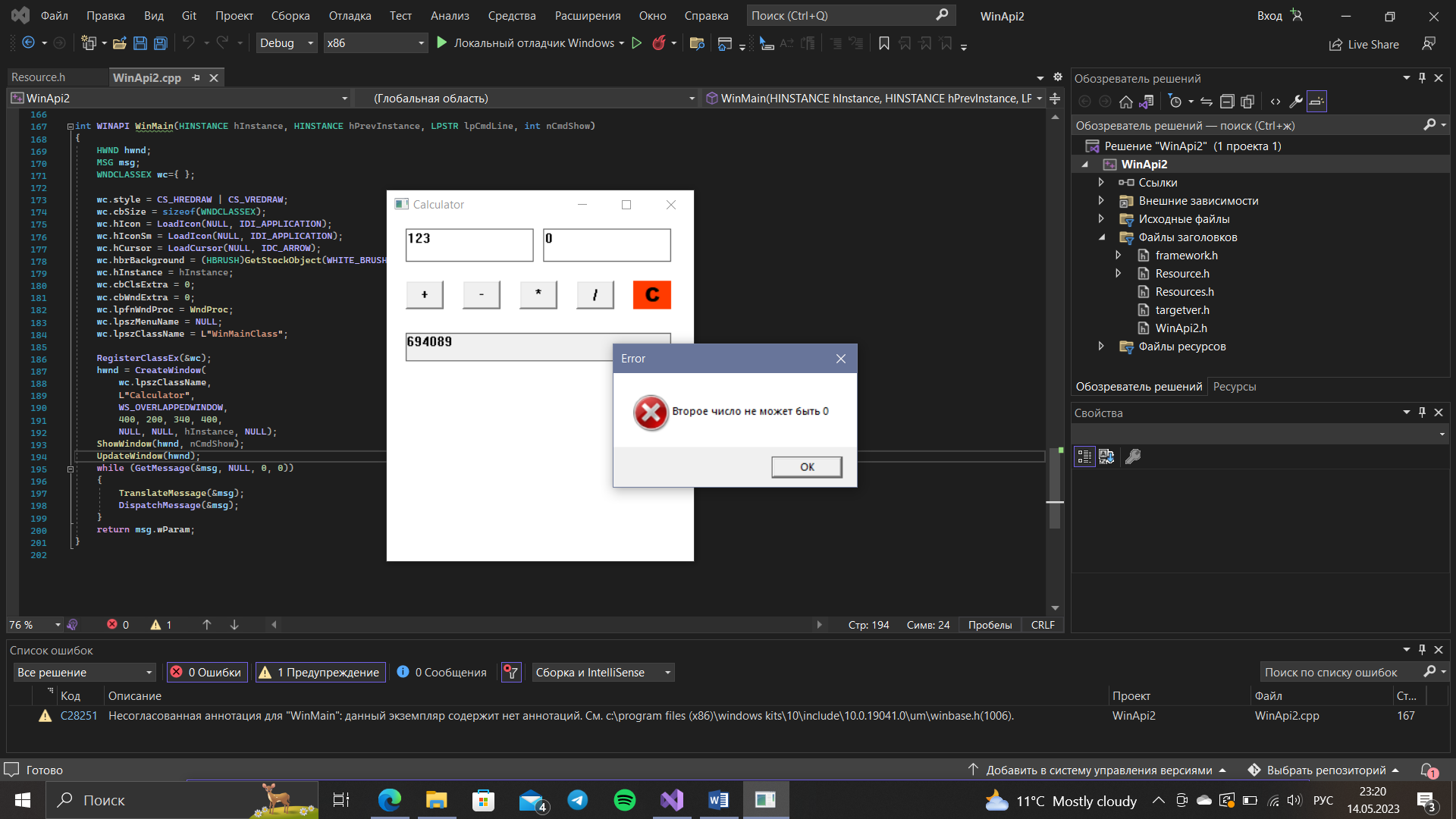
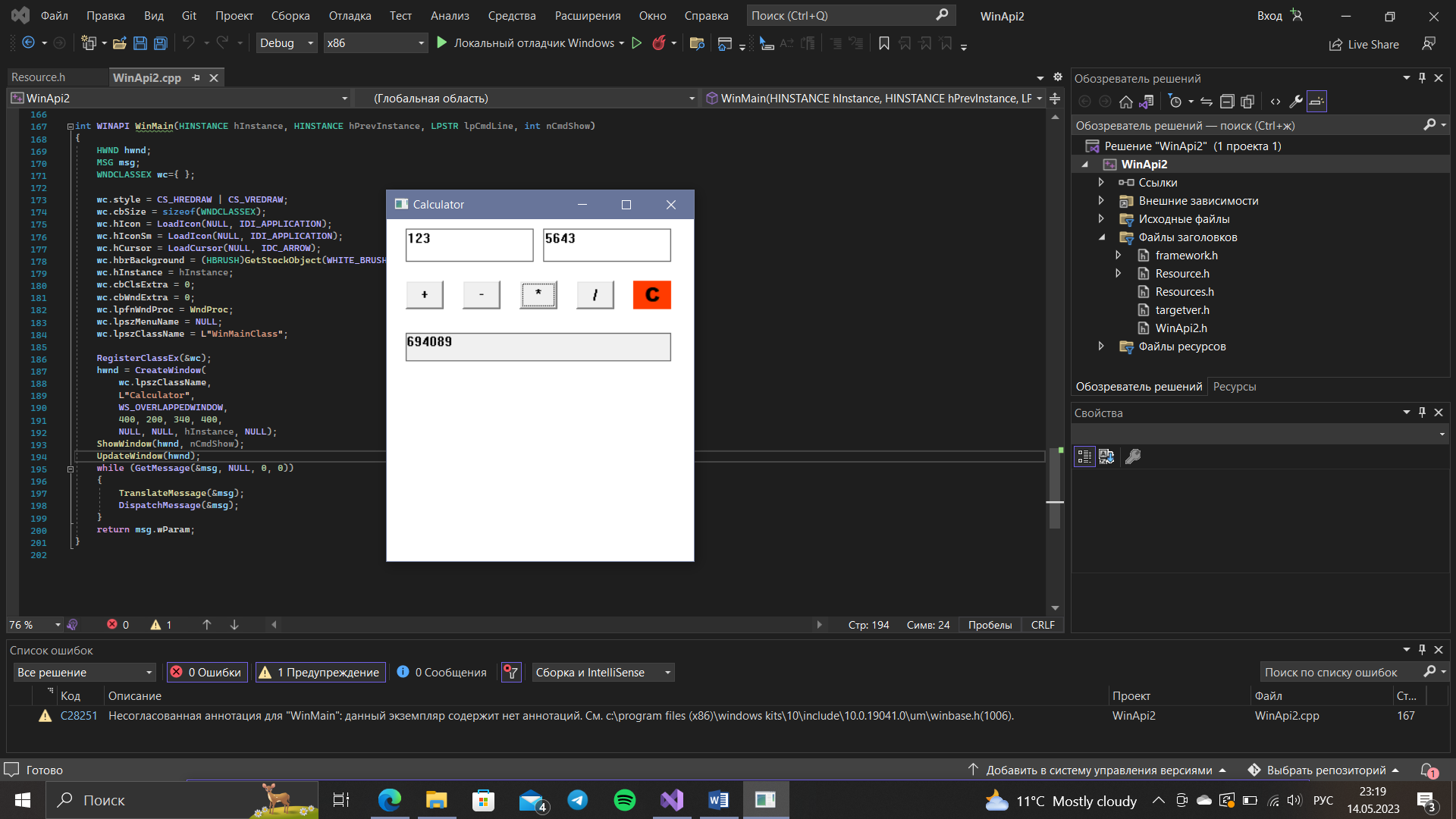
}

}

1. После закрытия окна завершить приложение.

Для вывода сообщения об ошибке использовалось всплывающее окно MessageBox(GetActiveWindow(), L"Второе число не может быть 0",L"Error", MB\_ICONERROR);

Результат выполнения кода:



# **Использование WinForms**

Для создания оконного приложения с помощью WinForms на C++ необходимо выполнить следующие шаги:

1. Создать новый проект в Visual Studio, выбрав тип приложения "Windows Forms Application". Программа состоит из заголовочного файла, где происходит обработка событий, и cpp-файла.
2. Добавить элементы управления на форму приложения с помощью редактора форм. Элементы управления могут быть добавлены из панели элементов. Редактироваться элементы могут в окне свойства, где прописаны как визуальные, так и программные свойства.
3. Обработать события элементов управления, такие как клики на кнопки или изменение текста в текстовых полях, используя событийные методы. Эти методы автоматически генерируются Visual Studio для каждого элемента управления и могут быть изменены пользователем. Добавить событие можно двойным кликом на необходимый объект.
4. Добавить код в событийные методы для выполнения необходимых действий. Например, для обработки клика на кнопку необходимо добавить код для выполнения соответствующих действий.

Код добавленных функций обработки событий:

private: System::Void MyForm\_Load(System::Object^ sender, System::EventArgs^ e) {

}

private: System::Void AC\_Click(System::Object^ sender, System::EventArgs^ e) {

this->result\_label->Text = "0";

this->num1->Text = "0";

this->num2->Text = "0";

user\_action = ' ';

}

private: System::Void minus\_Click(System::Object^ sender, System::EventArgs^ e) {

math\_action('-');

result = num\_one - num\_two;

this->result\_label->Text = System::Convert::ToString(result);

}

private: System::Void divis\_Click(System::Object^ sender, System::EventArgs^ e) {

math\_action('/');

if (num\_two == 0)

{

MessageBox::Show(this, "Деление на ноль невозможно", "Error", MessageBoxButtons::OK, MessageBoxIcon::Error);

result = 0;

}

else {

result = (float)num\_one / (float)num\_two;

}

this->result\_label->Text = System::Convert::ToString(result);

}

private: System::Void mul\_Click(System::Object^ sender, System::EventArgs^ e) {

math\_action('\*');

result = num\_one \* num\_two;

this->result\_label->Text = System::Convert::ToString(result);

}

private: System::Void plus\_Click(System::Object^ sender, System::EventArgs^ e) {

math\_action('+');

result = num\_one + num\_two;

this->result\_label->Text = System::Convert::ToString(result);

}

private: System::Void math\_action(char action) {

num\_one = System::Convert::ToInt32(this->num1->Text);

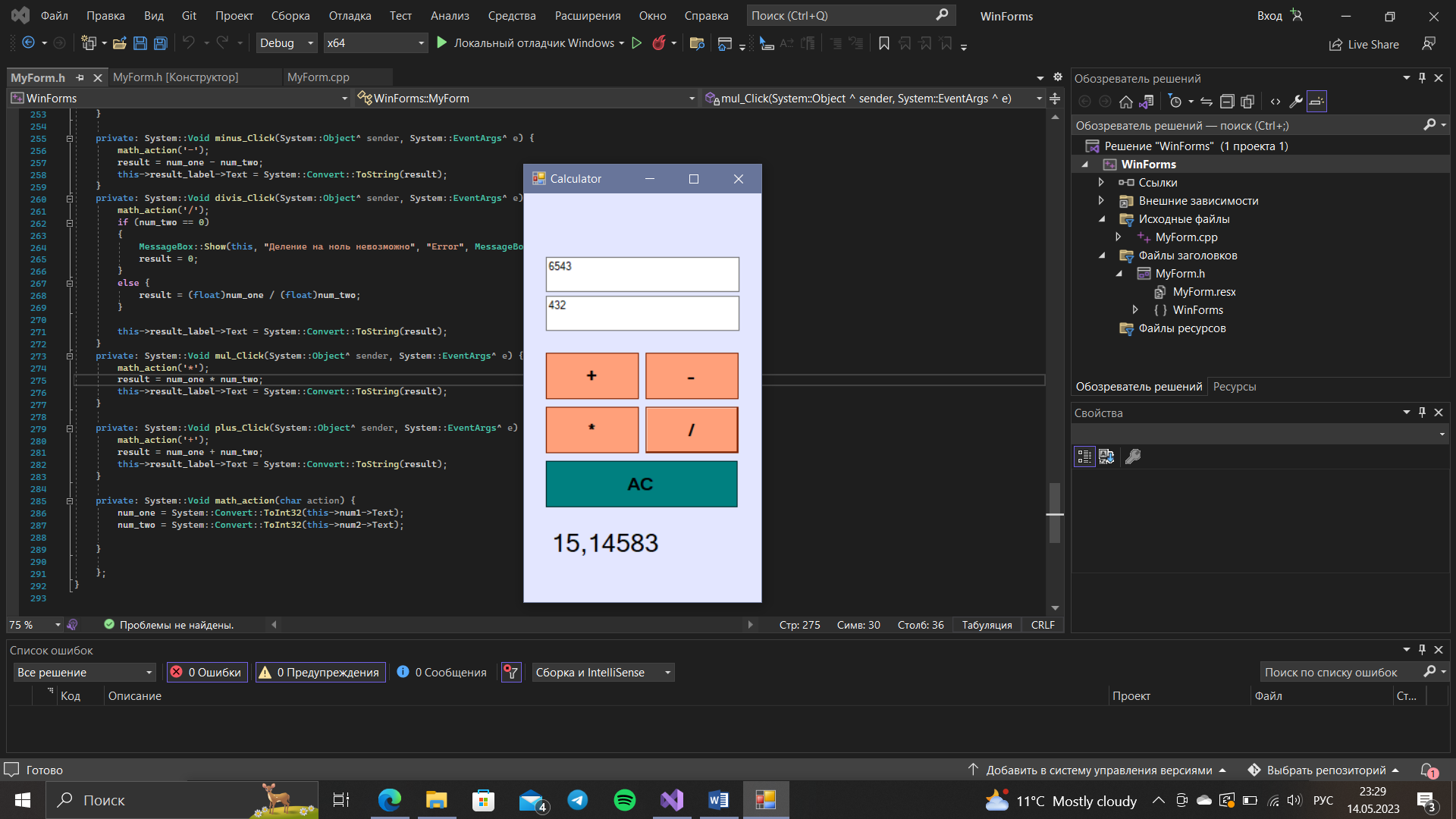
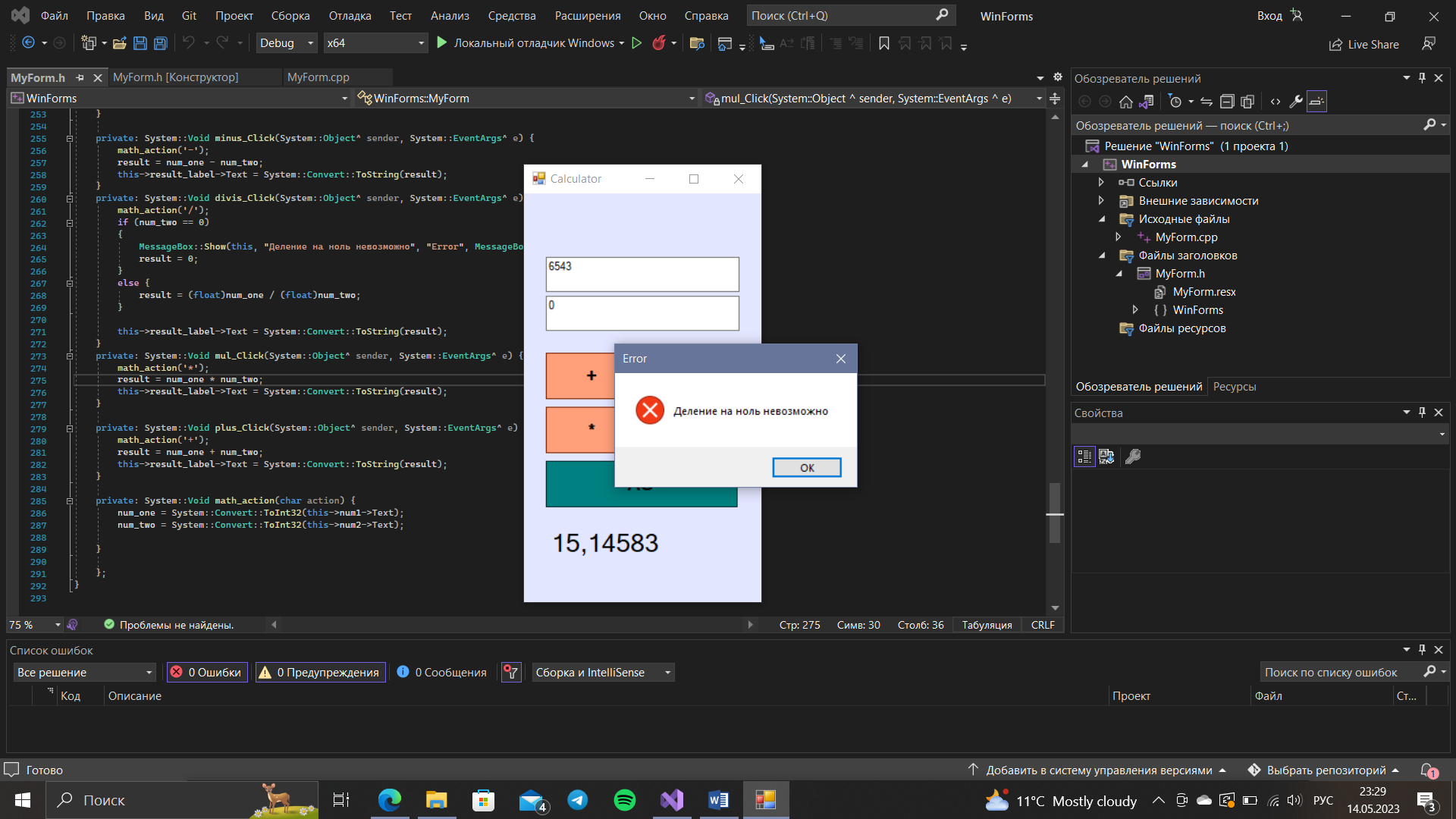
num\_two = System::Convert::ToInt32(this->num2->Text);

}

};

Для вывода сообщения об ошибке использовалось всплывающее окно MessageBox::Show(this, "Деление на ноль невозможно", "Error", MessageBoxButtons::OK, MessageBoxIcon::Error);

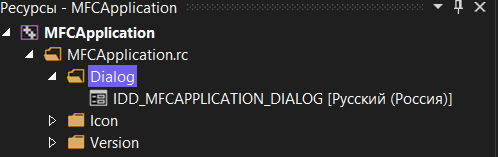
Результат выполнения:



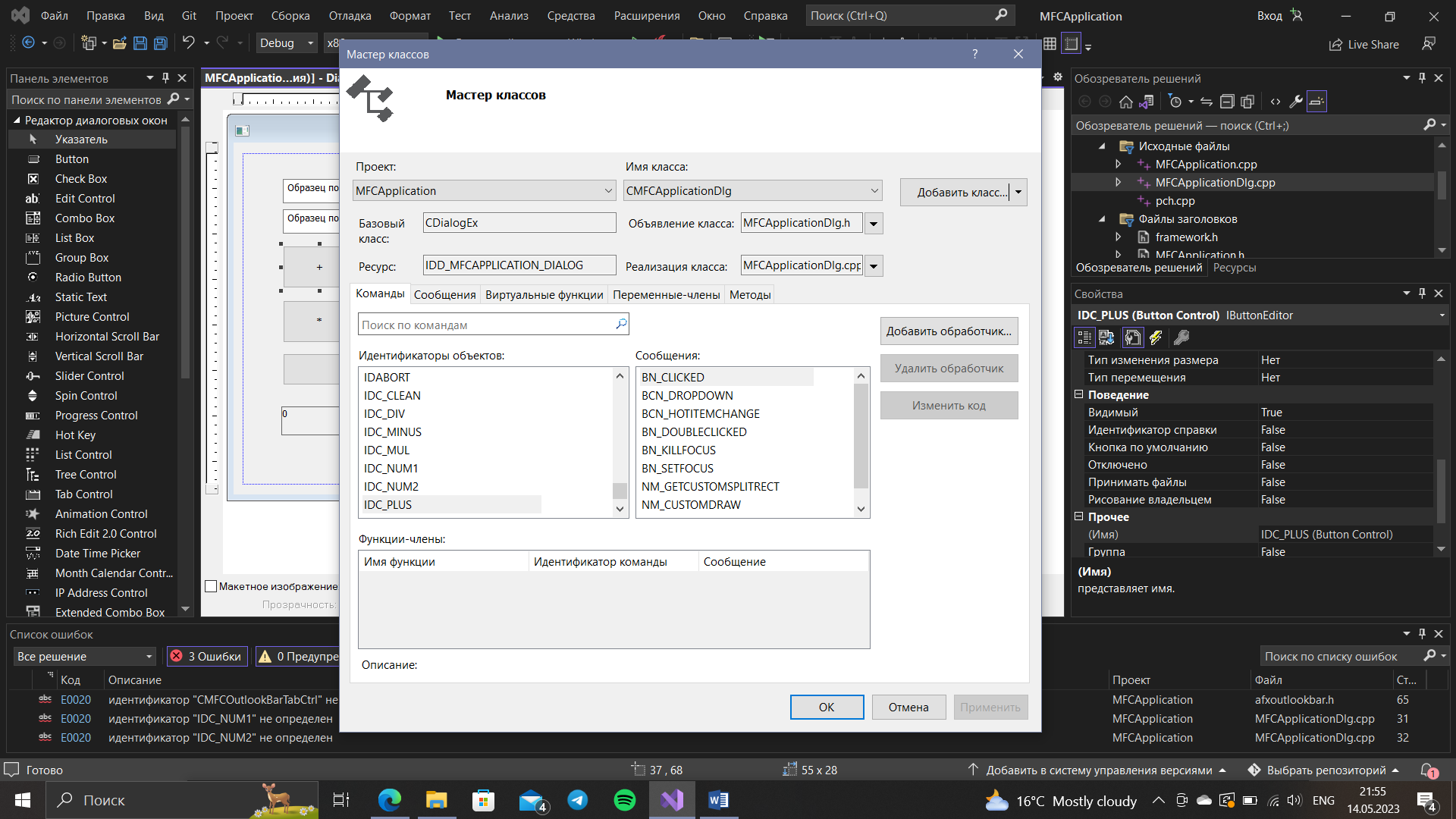
# **Использование MFC**

Для создания оконного приложения с помощью MFC на C++ необходимо выполнить следующие шаги:

1. Создать новый проект в Visual Studio, выбрав тип приложения "MFC Application".
2. В мастере создания проекта выбрать «Тип приложения»-> «На основе диалоговых окон», а также «Использование MFC в статической библиотеке». Будет создан проект с заголовочными файлами, исполняемыми файлами и файлами ресурсов. Через окно ресурсы мы можем открыть графический интерфейс для редактирования окна.



1. Добавить элементы управления на форму приложения с помощью редактора форм. Элементы управления могут быть добавлены также из панели элементов.
2. После добавления элементов необходимо добавить их как переменные. Для этого в контекстном меню необходимо выбрать «Добавить перемнную» и поставить необходимые параметры.
3. Обработать события элементов управления, такие как клики на кнопки или изменение текста в текстовых полях, используя событийные методы. Эти методы автоматически генерируются Visual Studio для каждого элемента управления и могут быть изменены пользователем. Для добавления обработчика событий нужно открыть мастер классов и для класса Диалогового окна выбрать нужные элементы и их условия обработки.



1. Добавить код в событийные методы для выполнения необходимых действий.

Код добавленных функций:

void CMFCApplicationDlg::OnClickedPlus()

{

UpdateData(true); //метод для считывания данных из контролов окна

str1 = num1;

str2 = num2;

result = atoi(\_bstr\_t(LPCTSTR(str1))) + atoi(\_bstr\_t(LPCTSTR(str2)));

wchar\_t buffer[256];

wsprintfW(buffer, L"%d", result);

res\_m.SetWindowText(buffer);

}

void CMFCApplicationDlg::OnClickedMinus()

{

UpdateData(true); //метод для считывания данных из контролов окна

str1 = num1;

str2 = num2;

result = atoi(\_bstr\_t(LPCTSTR(str1))) - atoi(\_bstr\_t(LPCTSTR(str2)));

wchar\_t buffer[256];

wsprintfW(buffer, L"%d", result);

res\_m.SetWindowText(buffer);

}

void CMFCApplicationDlg::OnClickedMul()

{

UpdateData(true); //метод для считывания данных из контролов окна

str1 = num1;

str2 = num2;

result = atoi(\_bstr\_t(LPCTSTR(str1))) \* atoi(\_bstr\_t(LPCTSTR(str2)));

wchar\_t buffer[256];

wsprintfW(buffer, L"%d", result);

res\_m.SetWindowText(buffer);

}

void CMFCApplicationDlg::OnClickedDiv()

{

UpdateData(true); //метод для считывания данных из контролов окна

str1 = num1;

str2 = num2;

if (num2 == '0')

{

MessageBox(\_T("На ноль делить нельзя"), NULL, MB\_OK);

UpdateData(TRUE);

num2.Format(\_T(""));

res\_m.SetWindowText(\_T(""));

UpdateData(FALSE);

}

else {

result = atoi(\_bstr\_t(LPCTSTR(str1))) / atoi(\_bstr\_t(LPCTSTR(str2)));

wchar\_t buffer[256];

wsprintfW(buffer, L"%d", result);

res\_m.SetWindowText(buffer);

}

}

void CMFCApplicationDlg::OnClickedClean()

{

res\_m.SetWindowText(\_T(""));

UpdateData(TRUE);

num1.Format(\_T(""));

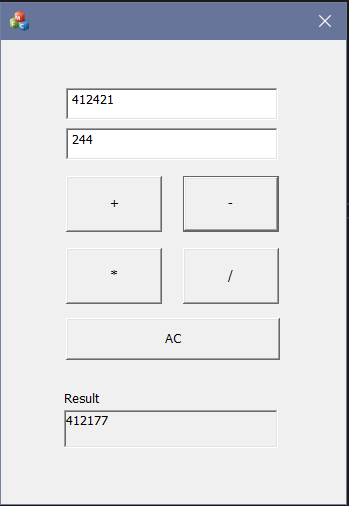
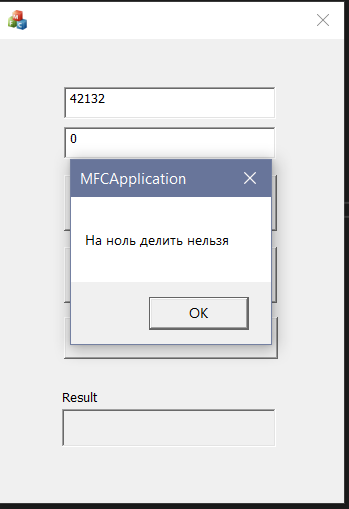
num2.Format(\_T(""));

UpdateData(FALSE);

}

Для вывода сообщения об ошибке использовалось всплывающее окно MessageBox(\_T("На ноль делить нельзя"), NULL, MB\_OK);

Результат выполнения:

# **Вывод**

Создание оконных приложений на C++ может быть выполнено с помощью различных инструментов, таких как классическое Windows-приложение, WinForms или MFC. Каждый инструмент имеет свои особенности и преимущества, которые могут быть удобны для различных типов проектов. В целом, создание оконных приложений на C++ может быть сложным процессом, который требует от программиста наличия глубоких знаний языка и технологий, используемых для создания оконных приложений. Однако, при правильном подходе и использовании соответствующих инструментов, создание оконных приложений на C++ может быть достаточно простым и эффективным процессом.