On converting from C to C++

main was cleaned from this

```
#include <Windows.h>#include <string>

HWND hwnd;
std::string className = "myWindowClass";

LRESULT CALLBACK WndProc(HWND  hwnd, UINT msg, WPARAM wParam, L
PARAM lParam);
int initWindow(HINSTANCE hInstance, int nCmdShow);
int messageLoop(HINSTANCE hInstance, int nCmdShow);
int run(HINSTANCE hInstance, int nCmdShow);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE PrevInstance, LPSTR
LpCmdLine, int nCmdShow) {

    int runValue = run(hInstance, nCmdShow);

    return runValue;

}

int run(HINSTANCE hInstance, int nCmdShow) {
    if (initWindow(hInstance, nCmdShow) != 0)
        return 1;
    int wParam = messageLoop(hInstance, nCmdShow);

    return     wParam;
}

int messageLoop(HINSTANCE hInstance, int nCmdShow) {
    MSG msg{};
```

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

    return (int)msg.wParam;
}

int initWindow(HINSTANCE hInstance, int nCmdShow) {
    WNDCLASSEX wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = (WNDPROC)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 = className.c_str();
    wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

    // Registering the window
    if (!RegisterClassEx(&wcex)) {
        MessageBox(NULL, "Window Registration Failed!", "Error!", MB_ICONEXC
LAMATION | MB_OK);
        return 1;
    }

    // Creating the window
    hwnd = CreateWindow(
        className.c_str(),
        className.c_str(),
```

```
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        640,
        480,
        NULL,
        NULL,
        hInstance,
        NULL
    );

    if (!hwnd) {
        MessageBox(NULL, "Window Creation Failed!", "Error!", MB_ICONEXCLA
MATION | MB_OK);
        return 1;
    }

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    return 0;
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, L
PARAM lParam) {
    switch (msg) {

    case WM_LBUTTONDOWN:
    {
        char FileNameC[MAX_PATH];
        HINSTANCE hInstance = GetModuleHandle(NULL);
        GetModuleFileName(hInstance, FileNameC, MAX_PATH);
        MessageBox(hwnd, FileNameC, "This program is:", MB_OK | MB_ICONIN
FORMATION);
        break;
    }
```

```
    case WM_CLOSE:
    {
        DestroyWindow(hwnd);
        break;
    }
    case WM_DESTROY:
    {
        PostQuitMessage(0);
        break;
    }
    default:
        return  DefWindowProc(hwnd, msg, wParam, lParam);
    }
    return 0;
}
```

to this

```
#include <Windows.h>
#include "App.h"

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE PrevInstance, LPSTR
LpCmdLine, int nCmdShow) {

    App app{hInstance, nCmdShow};
    return app.run();

}
```

Also made a nice class interface

```
#pragma once
#include <Windows.h>
```

```
#include <string>


class App {

public:

    App(HINSTANCE hInstance, int nCmdShow);
    ~App();
    HWND windowHandle_;

    static LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wP
aram, LPARAM lParam);
    int run();

private:
    const std::string className_;
    HINSTANCE instanceHandle_;
    INT initialWindowState_;

    int initWindow();
    int messageLoop();
};
```

And implementation file

```
#include "App.h"

App::App(HINSTANCE hInstance, int nCmdShow)
    : windowHandle_(nullptr),
    className_("myWindowClass"),
    instanceHandle_(hInstance),
    initialWindowState_(nCmdShow)
{
```

```
    initWindow();
}

App::~App() {

    if (windowHandle_) {
        DestroyWindow(windowHandle_);
        windowHandle_ = nullptr;
    }
    UnregisterClass(className_.c_str(), instanceHandle_);
}

int App::run() {
    return  messageLoop();
}

int App::messageLoop() {
    MSG msg{};
    while (GetMessage(&msg, nullptr, 0, 0) > 0)
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return (int)msg.wParam;
}

int App::initWindow() {
    WNDCLASSEX wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = (WNDPROC)WndProc;
    wcex.cbClsExtra = 0;
    wcex.cbWndExtra = 0;
    wcex.hInstance = this→instanceHandle_;
```

```
    wcex.hIcon = LoadIcon(nullptr, IDI_APPLICATION);
    wcex.hCursor = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wcex.lpszMenuName = nullptr;
    wcex.lpszClassName = this->className_.c_str();
    wcex.hIconSm = LoadIcon(nullptr, IDI_APPLICATION);

    // Registering the window
    try {
        if (!RegisterClassEx(&wcex))
            throw "Window Registration Failed!";
    }
    catch (const char* expression)
    {
        MessageBox(nullptr, expression, "Error!", MB_ICONEXCLAMATION | MB_
OK);
        return 1;
    }


    // Creating the window
    windowHandle_ = CreateWindow(
        this->className_.c_str(),
        this->className_.c_str(),
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        640,
        480,
        nullptr,
        nullptr,
        this->instanceHandle_,
        nullptr
    );

    try
```

```
    {
        if (!windowHandle_)
            throw "Window Creation Failed!";
    }
    catch(const char* expression)
    {
        MessageBox(nullptr, expression, "Error!", MB_ICONEXCLAMATION | MB_
OK);
        return 1;
    }

    ShowWindow(windowHandle_, this->initialWindowState_);
    UpdateWindow(windowHandle_);

    return 0;
}

LRESULT CALLBACK App::WndProc(HWND hwnd, UINT msg, WPARAM wPara
m, LPARAM lParam) {
    switch (msg) {

    case WM_LBUTTONDOWN:
    {
        char FileNameC[MAX_PATH];
        HINSTANCE instanceHandle = GetModuleHandle(nullptr);
        GetModuleFileName(instanceHandle, FileNameC, MAX_PATH);
        MessageBox(hwnd, FileNameC, "This program is:", MB_OK | MB_ICONIN
FORMATION);
        break;
    }
    case WM_CLOSE:
    {
        DestroyWindow(hwnd);
        break;
    }
    case WM_DESTROY:
```

```
    {
        PostQuitMessage(0);
        break;
    }
    default:
        return  DefWindowProc(hwnd, msg, wParam, lParam);
    }
    return 0;
}
```