```cpp
#include <iostream>
#include <vector>
using namespace std;

#include <GL/glew.h>
#include <GLFW/glfw3.h>
#include <glm/glm.hpp>

#include "Graphics.h"

Graphics::Graphics() {

};

Graphics::~Graphics() {

};

int Graphics::Init() {
    if (!glfwInit()) {                          // Checking for GLFW
        cout << "Could not initialise GLFW...";
        return 1;
    }

    glfwSetErrorCallback(ErrorCallbackGLFW);    // Setup a function to catch and display all GLFW errors.

    hintsGLFW();                                // Setup glfw with various hints.

                                                // Start a window using GLFW
    string title = "My OpenGL Application";
    window = glfwCreateWindow(windowWidth, windowHeight, title.c_str(), NULL, NULL);
    if (!window) {                              // Window or OpenGL context creation failed
        cout << "Could not initialise GLFW...";
        endProgram();
        return 1;
    }

    glfwMakeContextCurrent(window);             // making the OpenGL context current

                                                // Start GLEW (note: always initialise GLEW after creating your window context.)
    glewExperimental = GL_TRUE;                 // hack: catching them all - forcing newest debug callback (glDebugMessageCallback)
    GLenum errGLEW = glewInit();
    if (GLEW_OK != errGLEW) {                   // Problems starting GLEW?
        cout << "Could not initialise GLEW...";
        endProgram();
        return 1;
    }

    SetupRender();

    return 0;
}

void Graphics::hintsGLFW() {
    glfwWindowHint(GLFW_OPENGL_DEBUG_CONTEXT, GL_TRUE);         // Create
      context in debug mode - for debug message callback
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
}

void ErrorCallbackGLFW(int error, const char* description) {
    cout << "Error GLFW: " << description << "\n";
}


void Graphics::endProgram() {
    glfwMakeContextCurrent(window);     // destroys window handler
    glfwTerminate();    // destroys all windows and releases resources.
}

void Graphics::SetupRender() {
    glfwSwapInterval(1);    // Ony render when synced (V SYNC)

    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
    glfwWindowHint(GLFW_SAMPLES, 0);
    glfwWindowHint(GLFW_STEREO, GL_FALSE);
}

void Graphics::SetOptimisations() {
    glEnable(GL_CULL_FACE);
    glFrontFace(GL_CCW);

    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
}

void Graphics::ClearViewport() {
    glViewport(0, 0, windowWidth, windowHeight);
    static const GLfloat silver[] = { 0.9f, 0.9f, 0.9f, 1.0f };
    glClearBufferfv(GL_COLOR, 0, silver);
    static const GLfloat one = 1.0f;
    glClearBufferfv(GL_DEPTH, 0, &one);
}



```