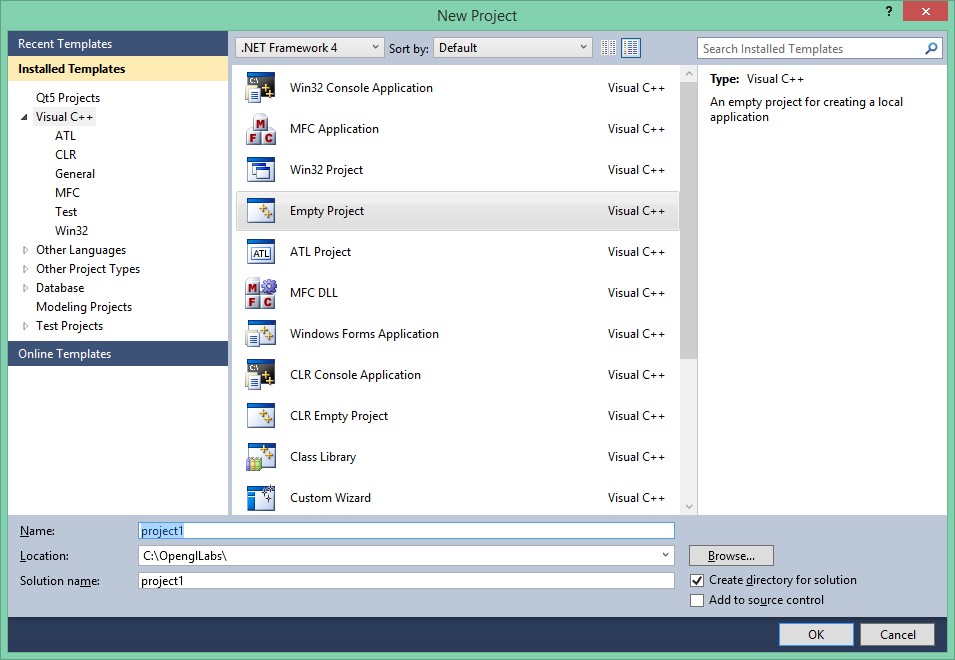
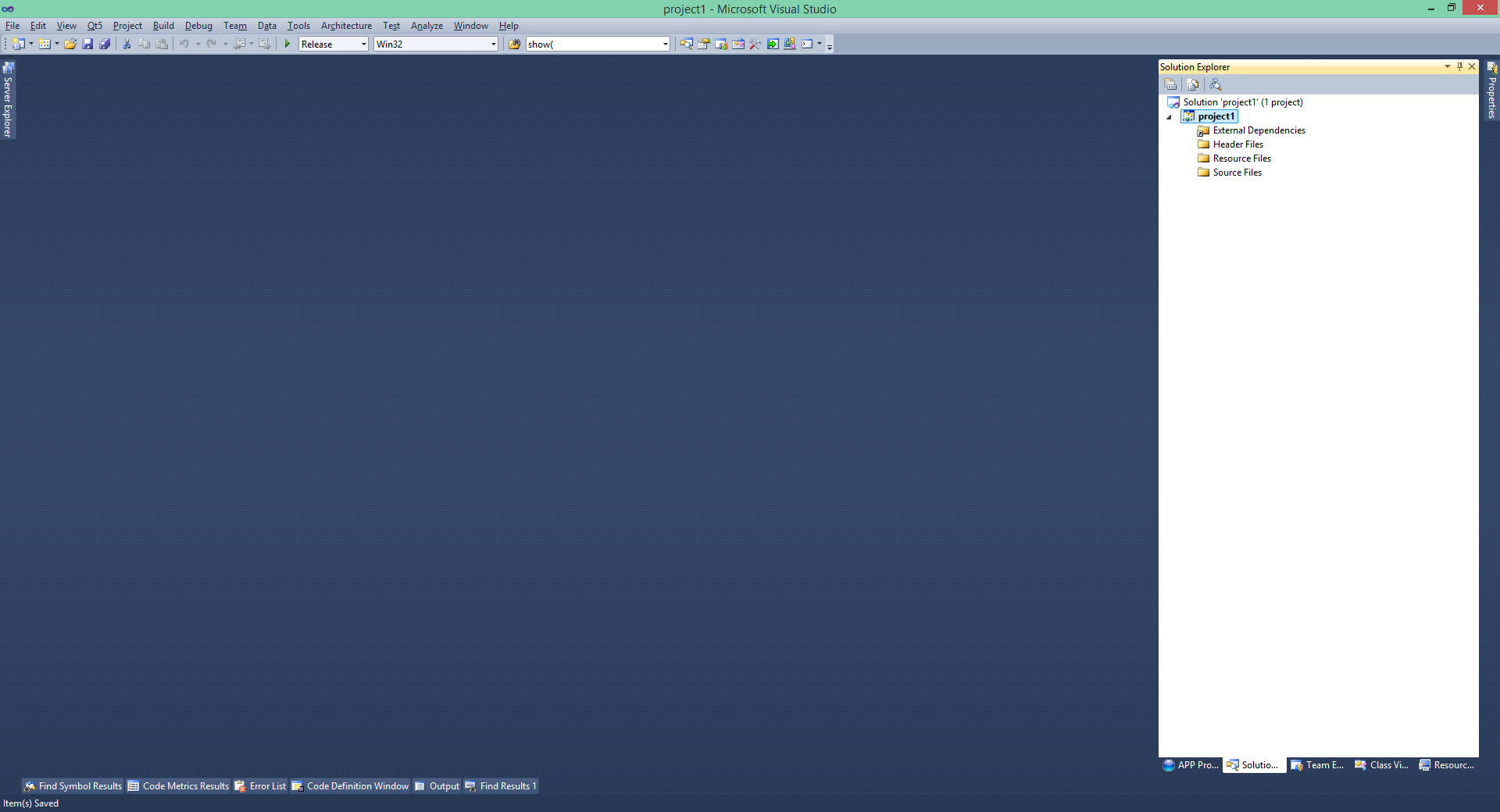
**Цель лабораторной работы:**

1. Собрать простейшие приложение OpenGL под Visual Studio 10/12 используя библиотеку GLFW.
2. Приложение должно содержать обработчики событий на клавиатуру и мышь.
3. Приложение должно отрисовывать вращающийся разноцветный треугольник.

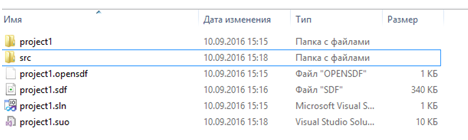
**Порядок исполнения:**

1. Создать пустой проект в Visual Studio с названием project1:

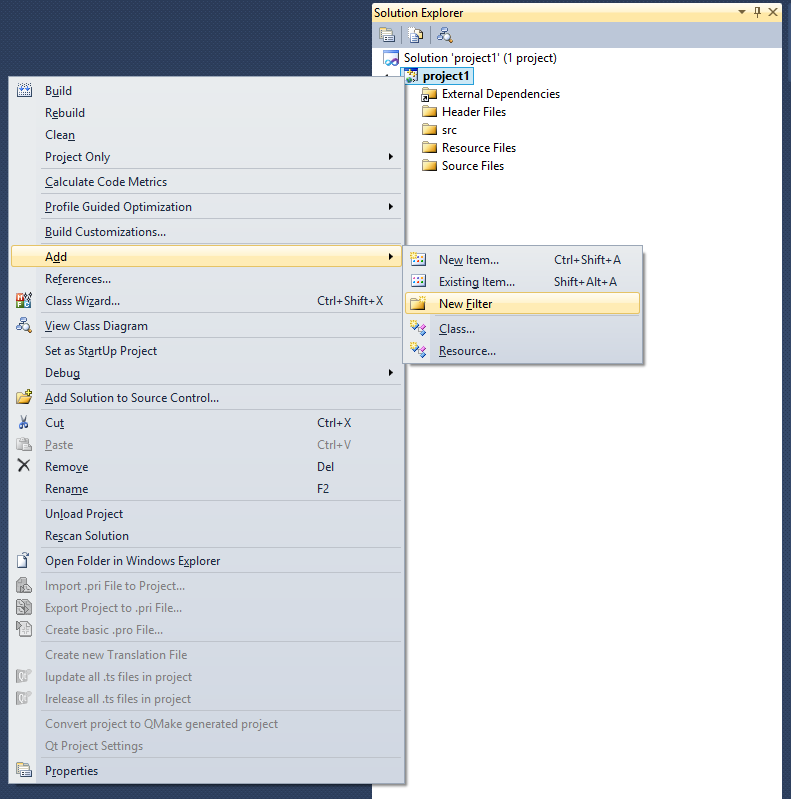




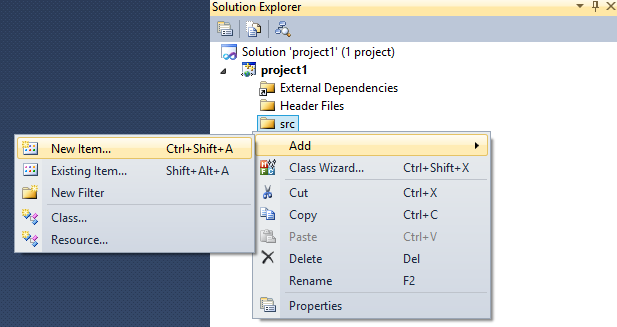
2) Создать папку src в корне проекта:

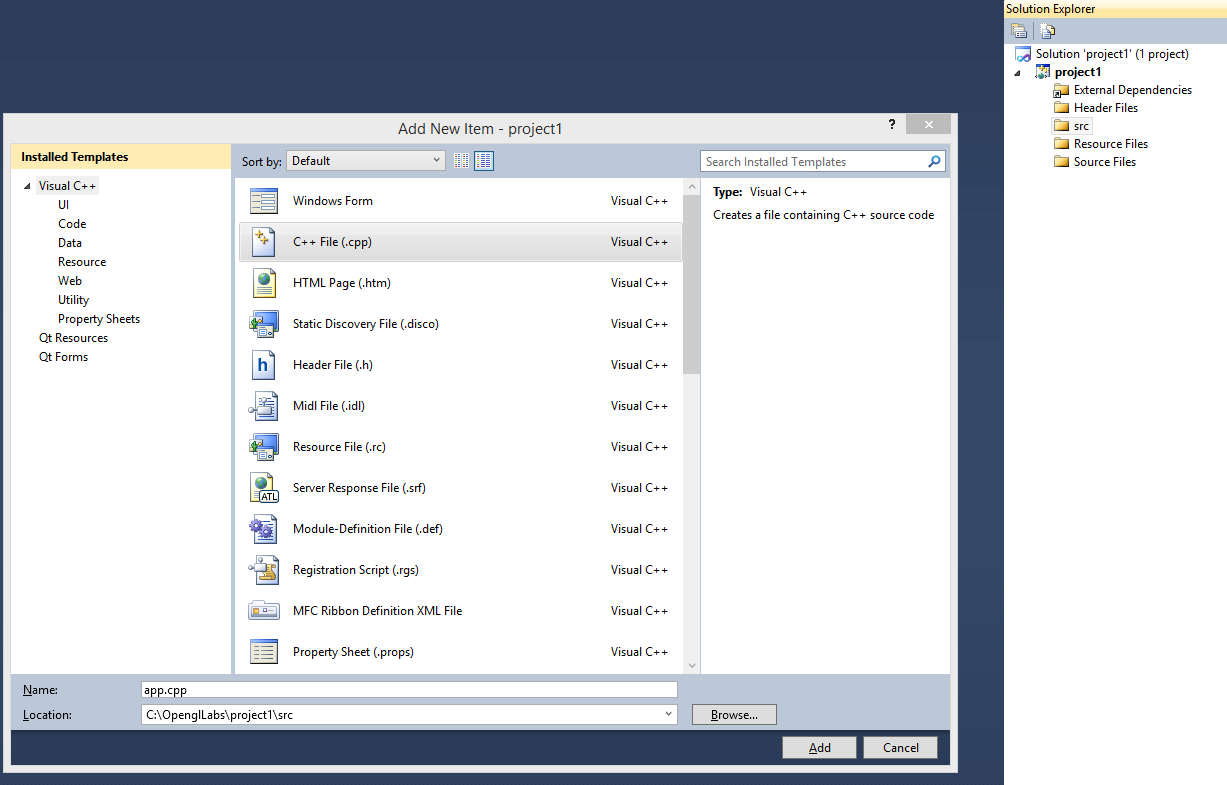


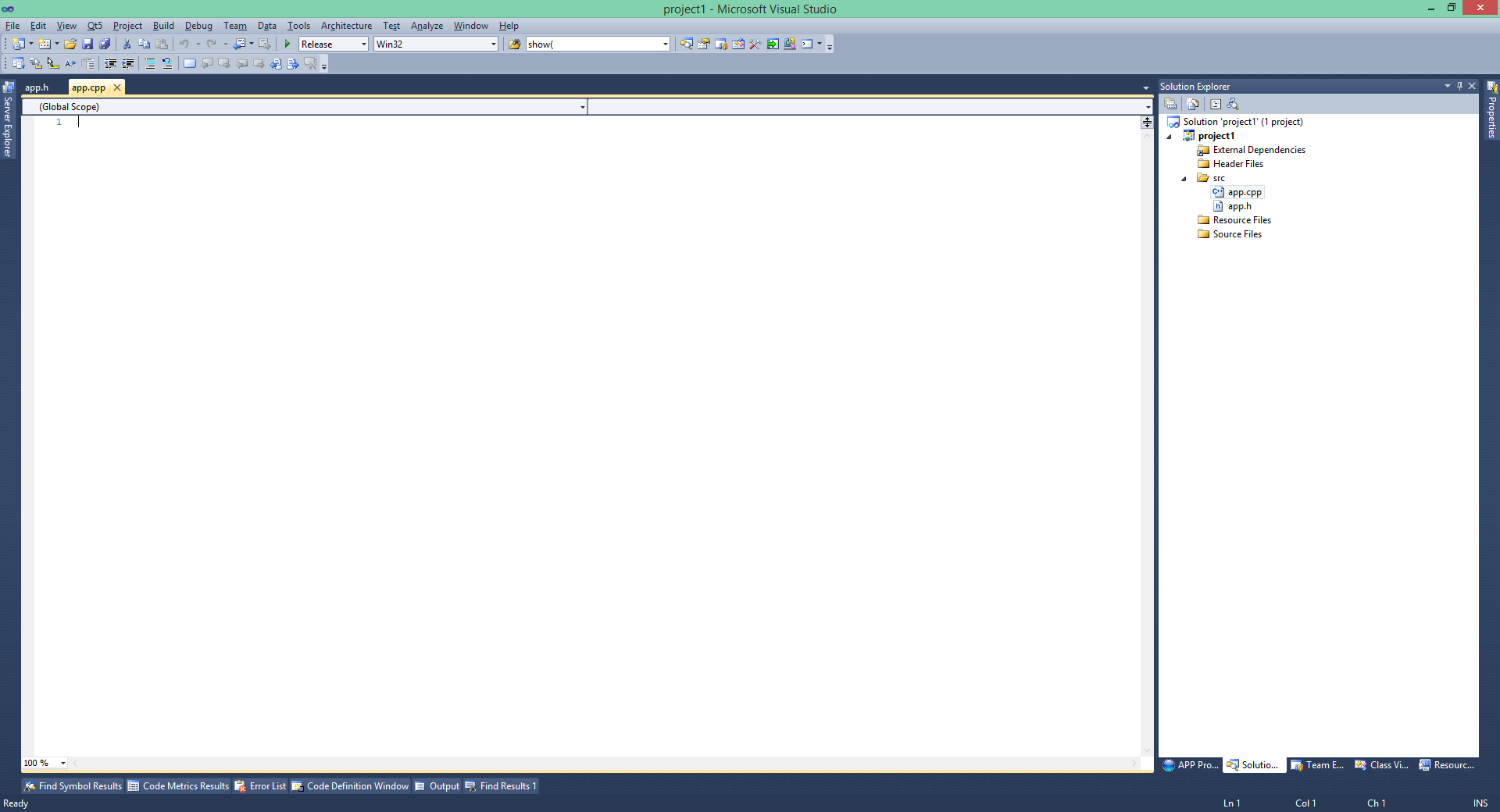
3) Добавить в проект новый фильтр с названием src в котором визуально будут лежать наши файлы:



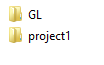
4) В проекте для фильтра "src" добавить два новых файла app.cpp и app.h. В этих файлах и будет находится код простейшего приложения opengl. В .h файле будут находится #include файлы, описание классов, константы, #define и прочее. В .cpp файле будет исходный код.

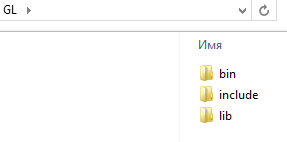




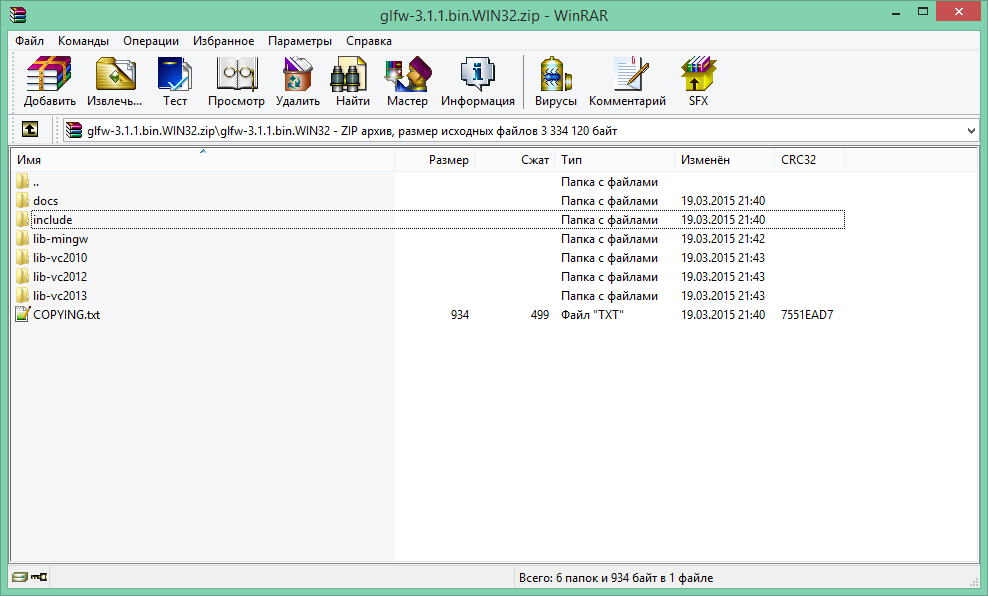


5) На уровень выше каталога вашего проекта "project1" создайте папку GL. В этой папке создайте папку include, папку lib и папку bin. В этих папках будет содержатся необходимый исходные код, статические и динамические библиотеки позволяющие получить доступ к OpenGL, и использовать его в вашем приложении:

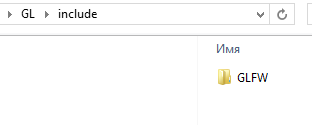




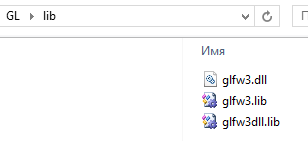
6) Распакуйте прекомпилированную библиотеку GLFW 3 в архиве glfw-3.1.1.bin.WIN32.zip:



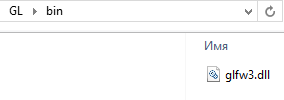
7) Содержимое папки include(папка GLFW) архива glfw-3.1.1.bin.WIN32.zip поместите в папку GL\include :



8) В зависимости от версии студии которую вы используете скопируйте содержимое одной из папок lib-vc201\* в папку GL\lib :



9) Файл glfw3.dll перенесите из папки GL\lib в папку GL\bin



10) В студии откройте файл app.h и скопируйте туда следующий исходный код:

#define GLFW\_INCLUDE\_GLU

#include <GL/glfw3.h>

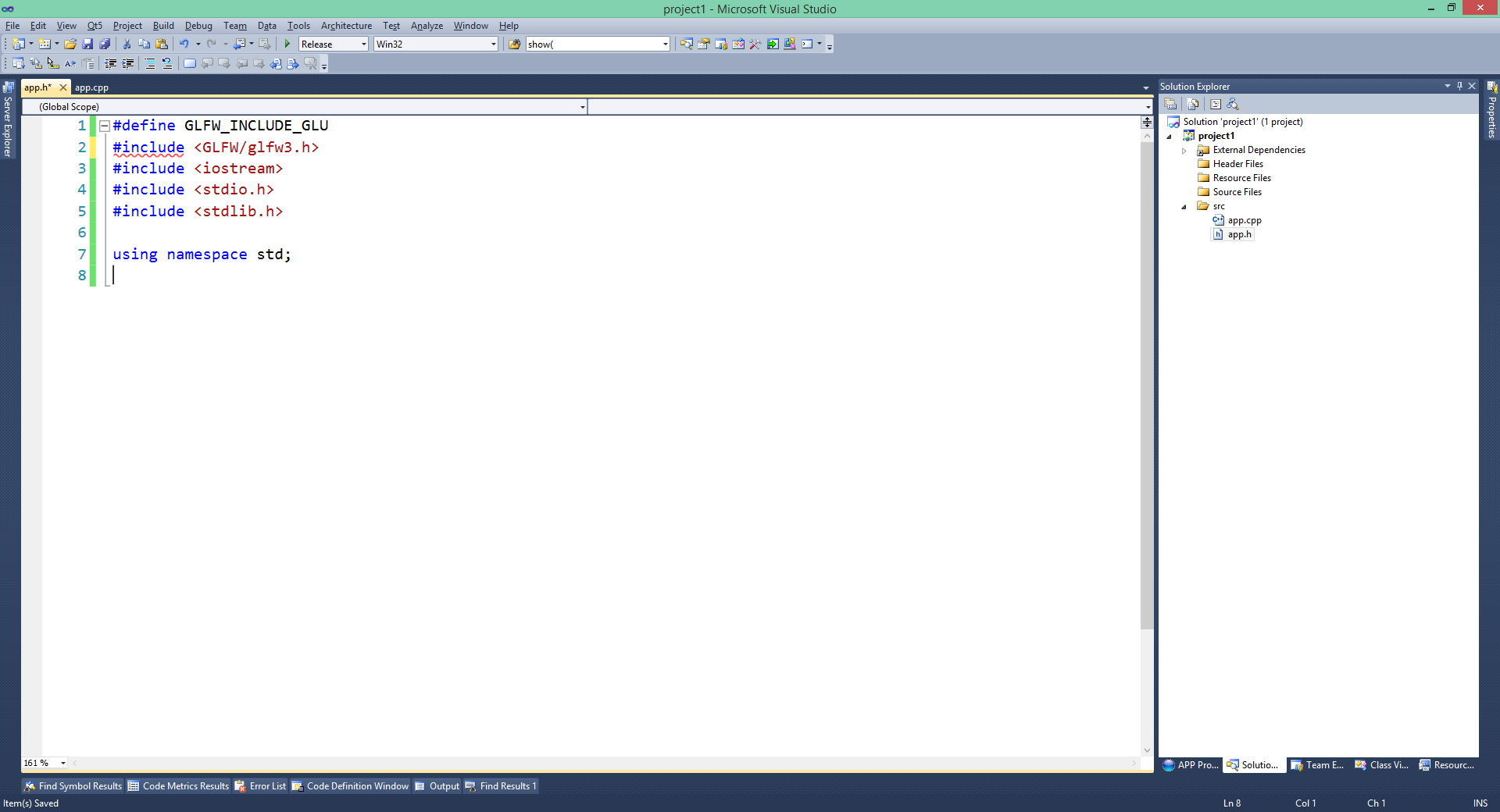
#include <iostream>

#include <stdio.h>

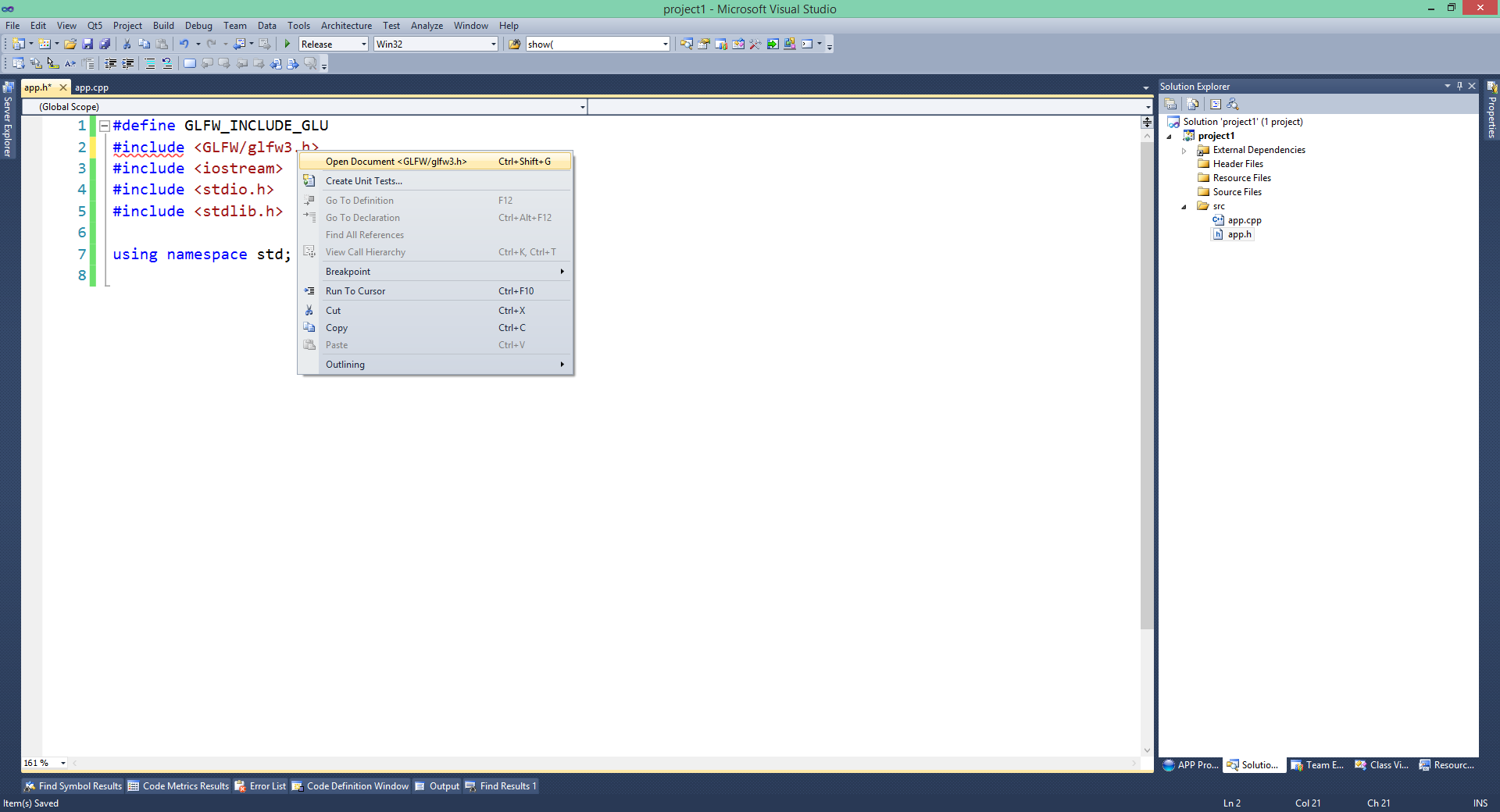
#include <stdlib.h>

using namespace std;

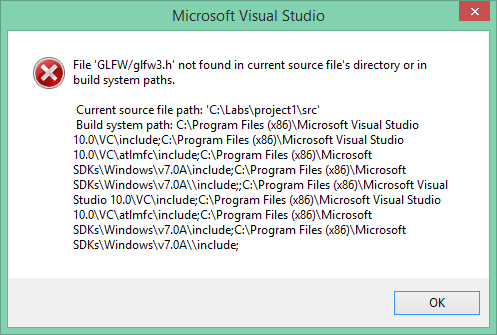
Этот код подключает необходимые файлы к вашему проекту чтобы можно было использовать OpenGL , и вспомогательную подсистему GLU и также стандартные вспомогательные библиотеки C++ :



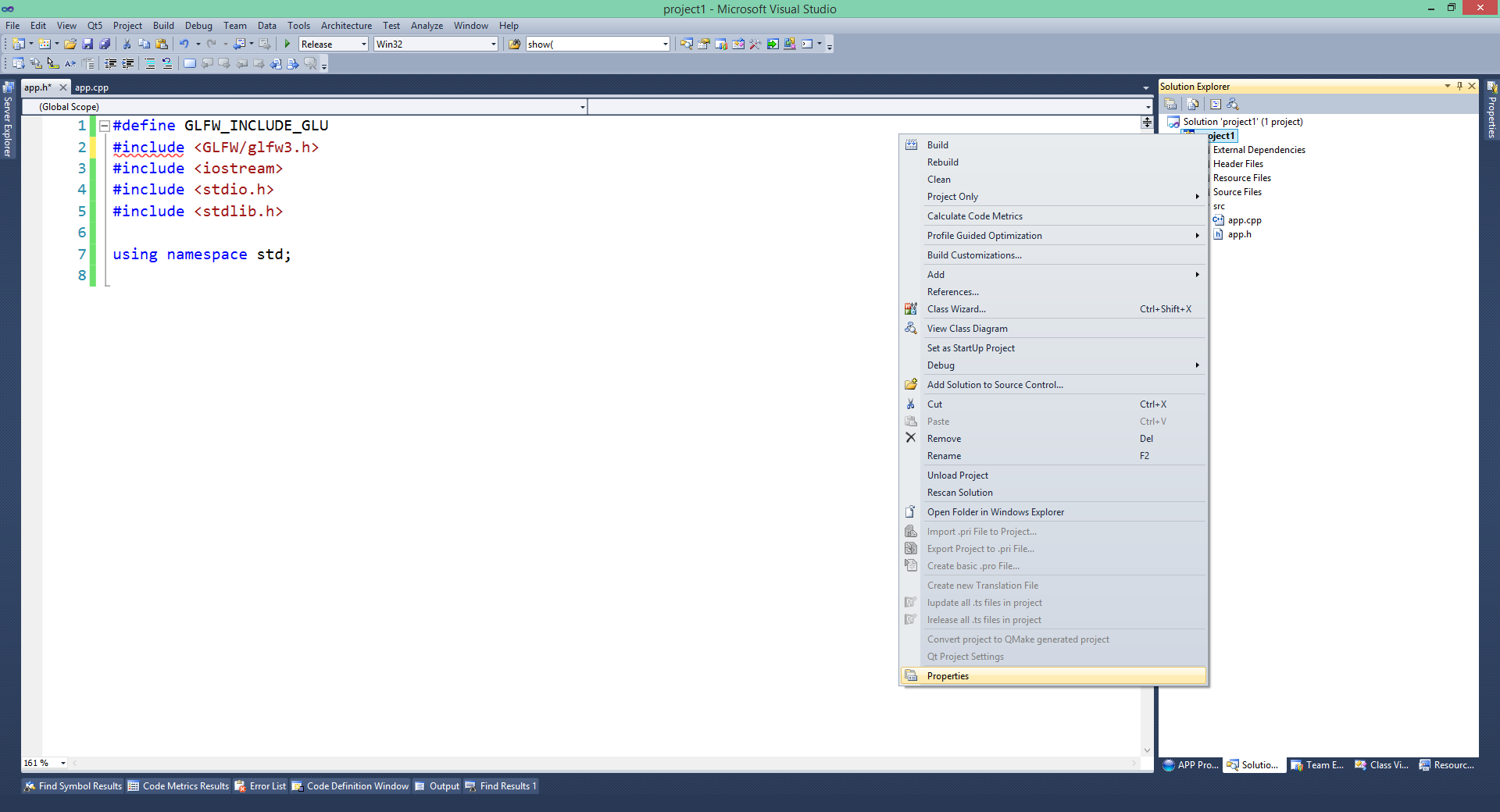
обратите внимание на красное подчеркивание файла glfw3.h - это означает что студия не видит данный файл и требуется прописать дополнительный каталог файлов включения(include). Это также легко проверить кликнув правой кнопкой мыши по данному файла и нажать открыть:



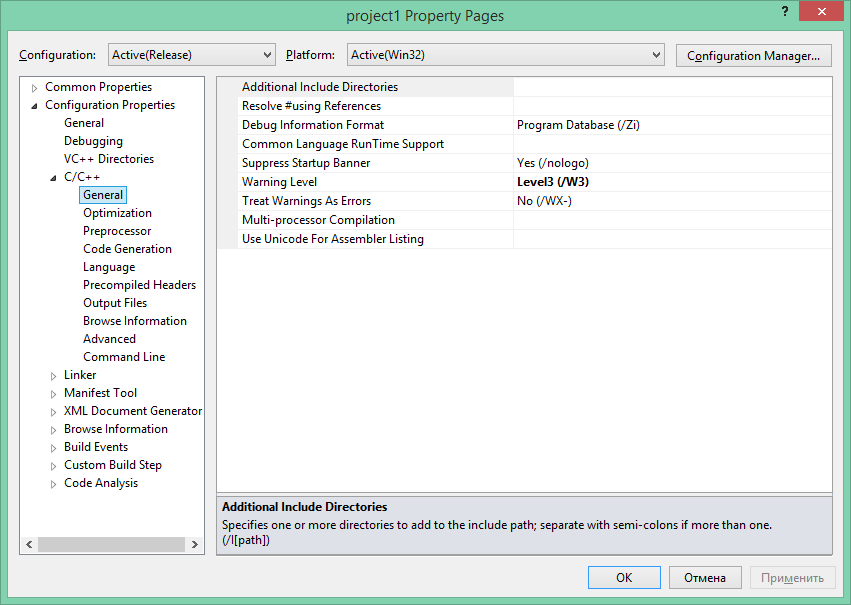
Результатом этого будет ошибка, поскольку студия не может найти этот файл:



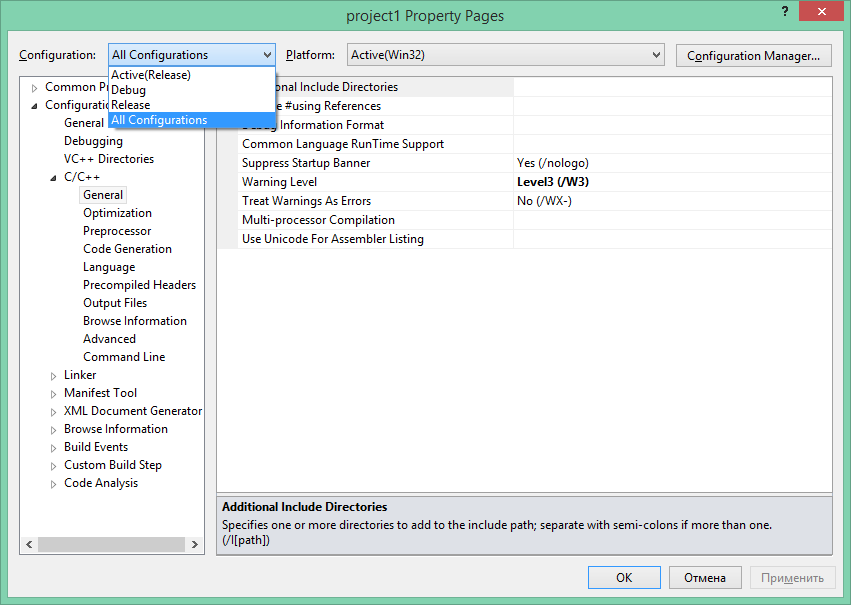
11) Для того чтобы устранить эту ошибку и дать студии возможность увидеть этот файл нужно прописать дополнительный каталог включения. Правой кнопкой кликните на названии проекта и кликните по "свойствам" :



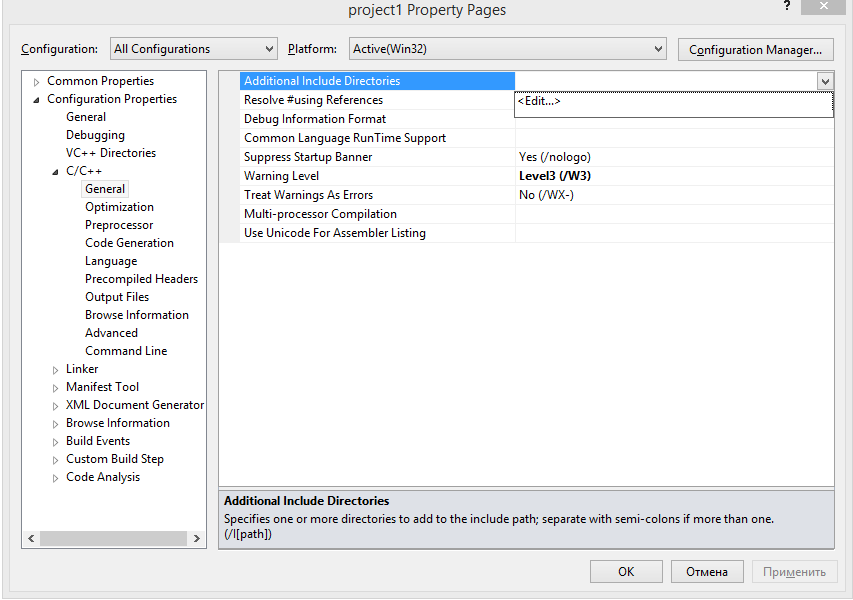
Далее нужно перейти в раздел C++ \ General:



после чего в configuration нужно выбрать все конфигурации:

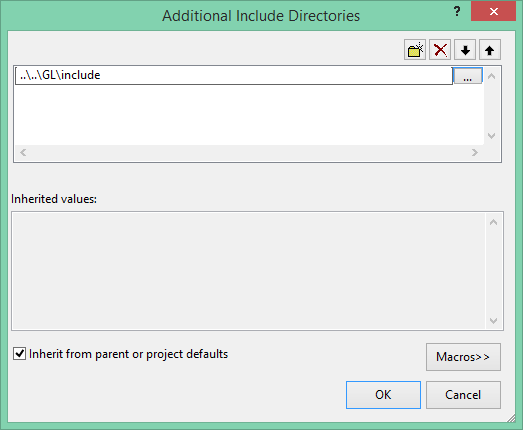
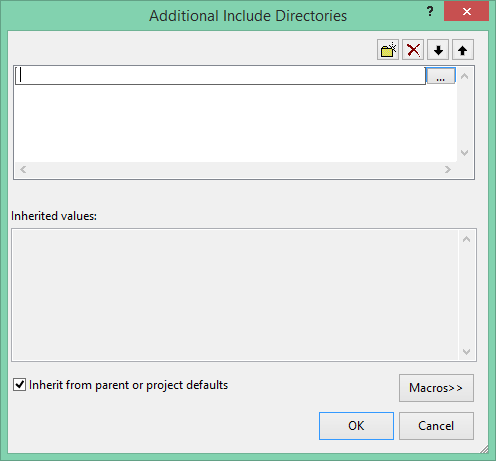


Это необходимо чтобы настройки которые вы будете сейчас устанавливать применилсь для двух режимов сборки: режима release и debug. После чего заходите в раздел "дополнительные каталоги включаемых файлов" :

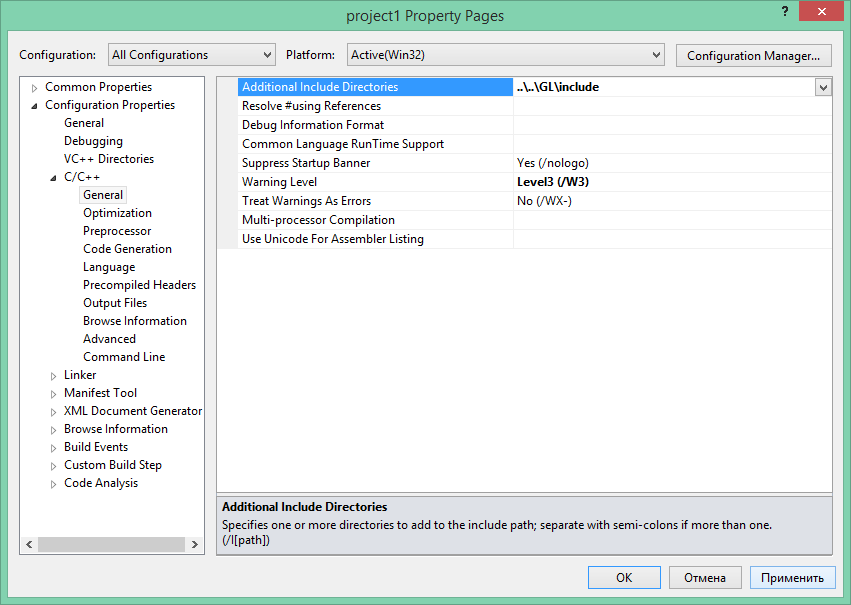


Кликайте по кнопке <Edit..>

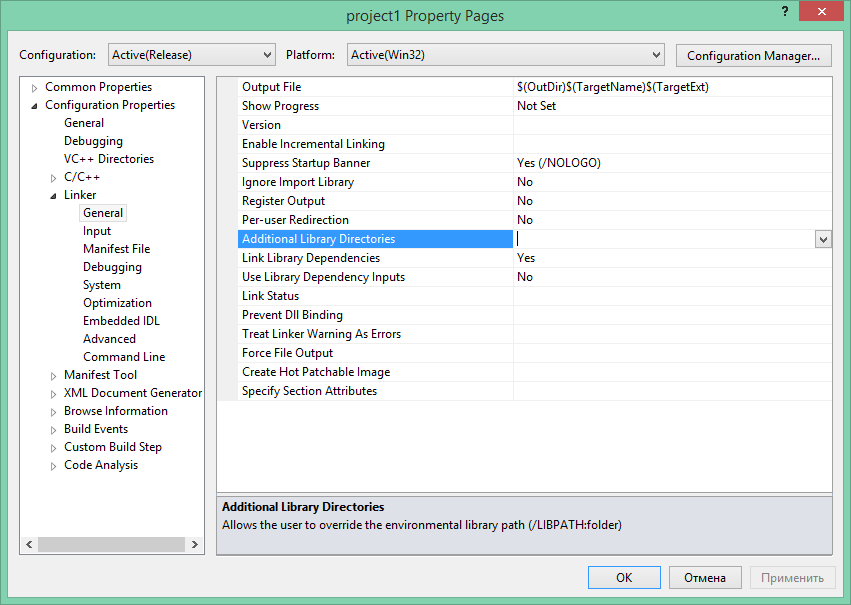
После чего в окне прописывайте путь к директории GL\include – ..\..\GL\include



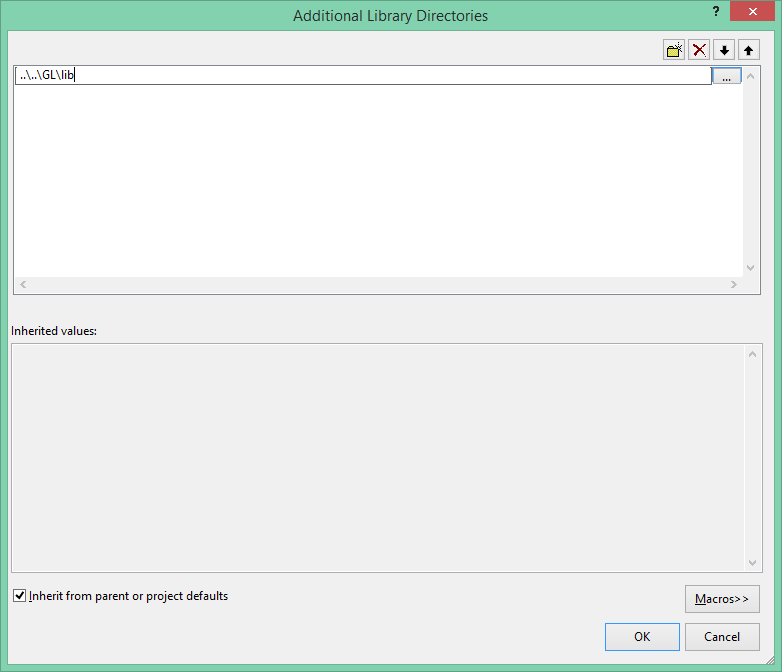
После чего нажимайте на ok, затем нажмите применить:

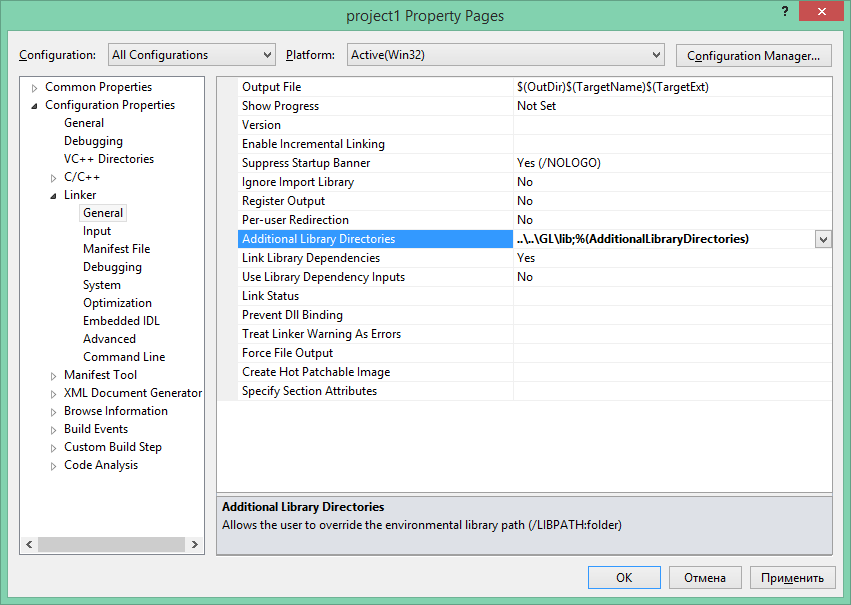


12) Следующий шаг: необходимо прописать каталог статических библиотек GL\lib и саму статическую библиотеку GLFW. Переходите в раздел linker(компоновщик) в раздел general(общее) и там, по аналогии в раздел "дополнительные каталоги библиотек":



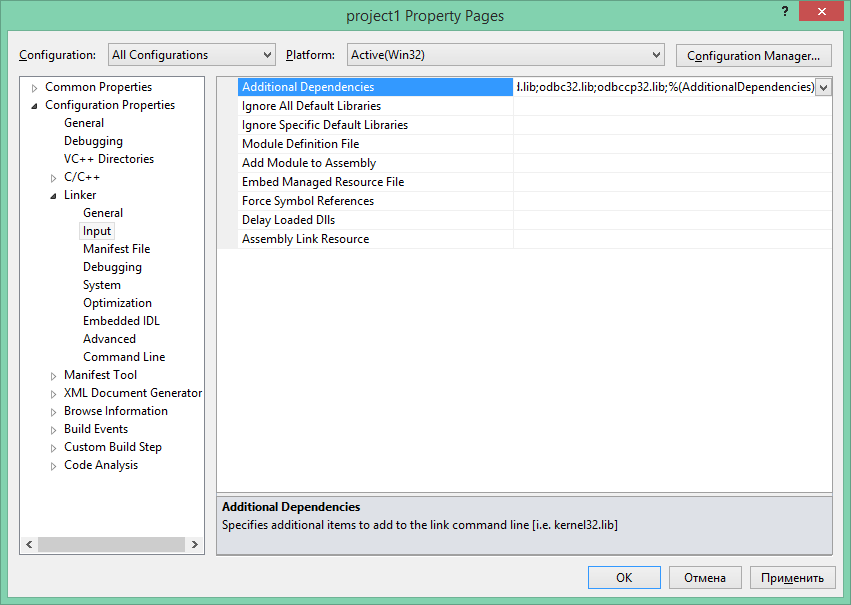
Далее пропишите относительный путь до папки GL\lib:



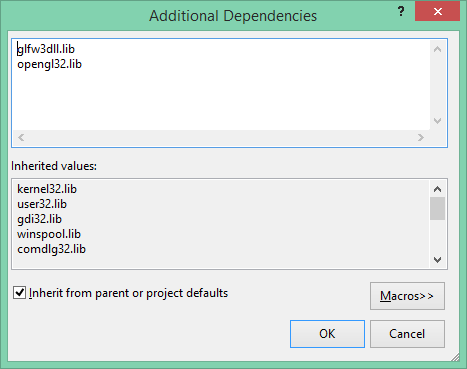


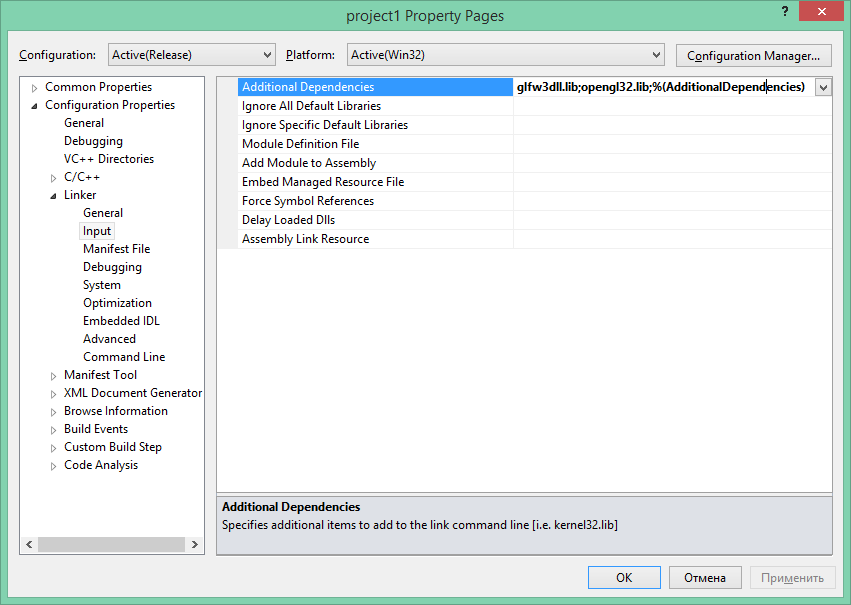
По аналогии нажмите кнопку применить.

13) Теперь необходимо указать конкретную статическую библиотеку из каталога GL\lib которую будет использовать наше приложение. Переходите во вкладку ввод(input) в разделе компоновщика(linker):



14) В текстовом поле дополнительные зависимости пропишите имя файла статической библиотеки glfw3dll.lib которая лежит в папке GL\lib, также нужно прописать саму библиотеку opengl - opengl32.lib:





Исходя из названия данная статическая библиотека предполагает работу с динамической библиотекой (DLL) которая подгружается при запуске приложения.

15) Теперь почти все готово для набора исходного кода приложения OpenGL которое будет рисовать треугольник. Откройте файл app.cpp и введите туда следующий код:

/// Подключаем файл app.h чтобы app.cpp видел небходимые внешние файлы библиотеки GLFW:

#include "app.h"

/// Данные функции вызываются когда происходят события нажатия на клавиатуру, события с мышью и возмыжные неполадки:

void error\_callback(int error, const char\* description)

{

fputs(description, stderr);

}

void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mods)

{

if (key == GLFW\_KEY\_ESCAPE && action == GLFW\_PRESS)

glfwSetWindowShouldClose(window, GL\_TRUE);

}

void mouse\_callback(GLFWwindow\* window, int a, int b, int c){

cout << "button a = " << a << endl;

cout << "button b = " << b << endl;

cout << "button c = " << c << endl;

}

/// Это основная функция main() которая является точкой входа в наше приложения, откуда мы уже запускаем все другие подсистемы нашего

/// приложения:

int main(){

/// Указатель на используемое окно:

GLFWwindow\* window;

/// Подключаем функцию error\_callback для отслеживания возможных ошибок связанных с GL или

/// конкретно библиотекой GLFW

glfwSetErrorCallback(error\_callback);

/// Выполняем инициализацию библиотеки GLFW:

if (!glfwInit())

exit(EXIT\_FAILURE);

/// Включаем 4 кратное сглаживания для устранения эффекта зазубренности изображения:

glfwWindowHint(GLFW\_SAMPLES,4);

/// Включаем продвинутую свежею версию OpenGL 3.3:

//glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR,3);

//glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR,3);

//glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);

/// Создаем GLFW окно с нужным размером и заголовком, записываем указатель на область памяти где

/// где находится данное окно:

window = glfwCreateWindow(1024, 768, "Simple example", NULL, NULL);

/// Если окно не создано, то есть указатель содержит NULL - аварийно завершаем наше приложение:

if (!window){

glfwTerminate();

exit(EXIT\_FAILURE);

}

/// Подключаем выполняет контекст OpenGL для окна window:

glfwMakeContextCurrent(window);

/// Задаем количество экранных обновлений прежде:

glfwSwapInterval(1);

/// Подключаем отслеживания событий мыши и клавиатуры:

glfwSetKeyCallback(window, key\_callback);

glfwSetMouseButtonCallback(window, mouse\_callback);

/// Запускаем цикл обновления картинки в окне:

/// Выполняем цикл до тех пор пока окно не закрывается:

while (!glfwWindowShouldClose(window))

{

/// Коэфицент соотношения сторон окна:

float ratio;

/// Ширина и высота окна:

int width, height;

/// В реальном времени получаем размер окна:

glfwGetFramebufferSize(window, &width, &height);

/// Получаем позицию курсора:

double xpos,ypos;

glfwGetCursorPos(window,&xpos,&ypos);

/// Вычисляем коэфицент соотношения сторон:

ratio = width / (float) height;

/// Далее идут базовые GL функции:

/// Задаем двумерную область вывода изображения на экране:

/// То есть та область где будет визуализация изображения OpenGL:

glViewport(0, 0, width, height);

/// Выполняем очистки буфера цвета:

glClear(GL\_COLOR\_BUFFER\_BIT);

/// Выбираем для работы проэкционную матрицу:

glMatrixMode(GL\_PROJECTION);

/// Устанавливем единичную(стандартную) проекционную матрицу:

glLoadIdentity();

/// Установим ортогональную проекцию:

/// функцией void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);

/// То есть все трехмерное пространство будет ортогонально проецироваться

/// на плоскость экрана в диапазоне координат (0,0,width,height);

/// Для этого данная функция задает параллельный объем отсечения.

/// Если рассматривать объем отсечения в декартовых координатах,

/// left и right будут минимальным и максимальным значениями х;

/// top и bottom — минимальным и максимальным значениями у,

/// a near и far — минимальным и максимальным значениями г.

/// То есть такая проекция означает, что объекты удаленные от пользователя, не кажутся меньше

/// (в перспективной проекции объекты уменьшаются)

glOrtho(-ratio, ratio, -1.f, 1.f, 1.f, -1.f);

/// Выбираем для работы модельно видовую матрицу:

glMatrixMode(GL\_MODELVIEW);

/// Устанавливем единичную(стандартную) модельно видовую матрицу:

/// По сути выполняем сброс всех ранее выполненных преобразования матрицы

glLoadIdentity();

/// Выполняем поворот пространства на угол glfwGetTime() \* 50.f

/// Где glfwGetTime() возвращает время в секундах с момента запуска приложения:

glRotatef((float) glfwGetTime() \* 50.f, 0.f, 0.f, 1.f);

/// Рисуем треугольник по вершинам, для каждой вершины задаем произвольный цвет:

glBegin(GL\_TRIANGLES);

glColor3f(1.f, 0.f, 0.f);

glVertex3f(-0.6f, -0.4f, 0.f);

glColor3f(0.f, 1.f, 0.f);

glVertex3f(0.6f, -0.4f, 0.f);

glColor3f(0.f, 0.f, 1.f);

glVertex3f(0.f, 0.6f, 0.f);

glEnd();

/// Выполняем свап буфера:

glfwSwapBuffers(window);

/// И обновления событий окна( в том числе клавиатуры и мыши):

glfwPollEvents();

}

/// Если цикл не выполняется, значит окно закрывается и нужно его удалить из памяти:

glfwDestroyWindow(window);

/// Также нужно завершить работу с glfw, высвободить все ресурсы связанные с OpenGL, окном и прочее:

glfwTerminate();

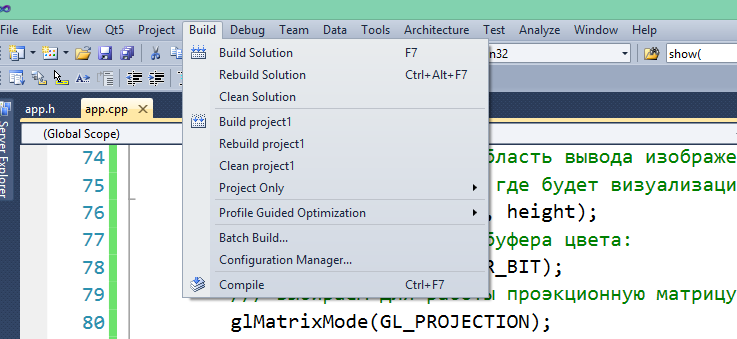
/// Выйти из приложении сообщив о нормальном(штатном) завершении работы программы:

exit(EXIT\_SUCCESS);

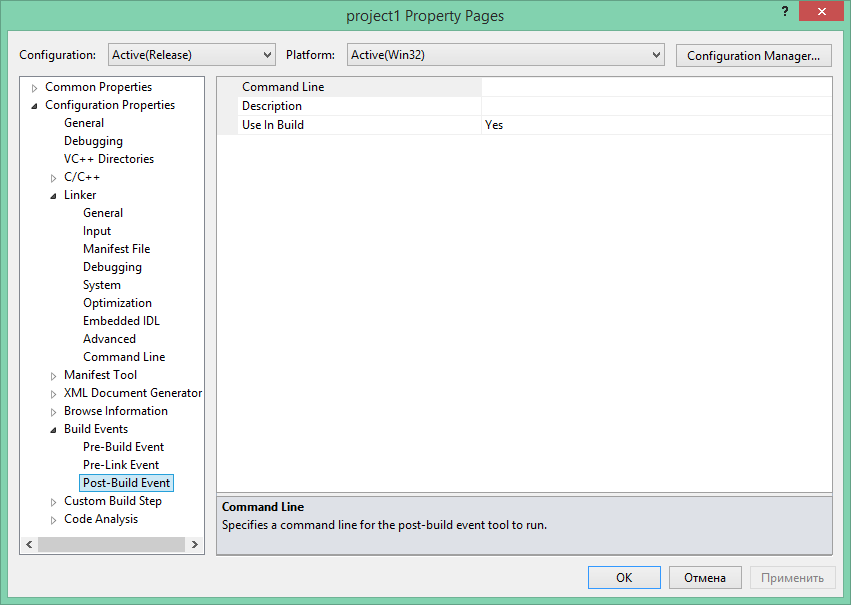
}

16) Далее необходимо выполнить release сборку решения(solution):

при компиляции установите режим release сборки



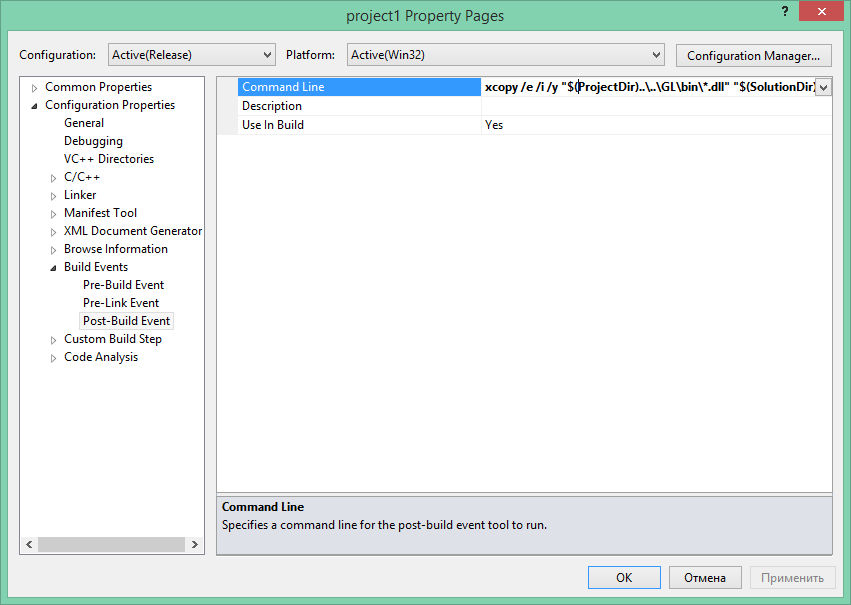
17) Последний шаг - необходимо настроить авто копирование нужного DLL файла из папки GL\bin в папку с исполняемым файлом приложения. Для этого заходите в свойства проекта, события построения(Build Events), События после постройки:



18) В этом разделе нужно прописать командную строку(Command Line) :

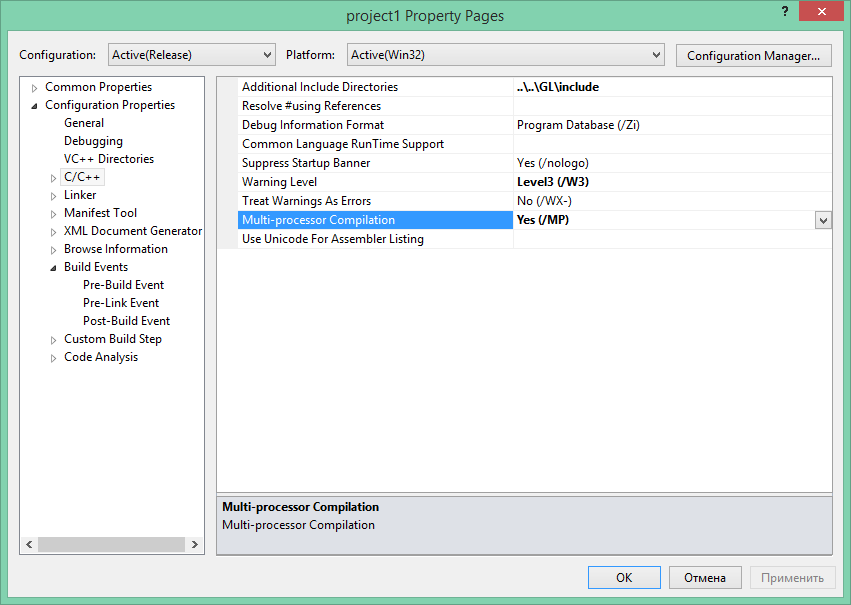
а именно команду копирования \*.dll файлов:

xcopy /e /i /y "$(ProjectDir)..\..\GL\bin\\*.dll" "$(SolutionDir)$(Configuration)"



19) Сейчас время сборки небольшое, но с ростом размера проекта, увеличении количества исходного кода, используемых внешних библиотек время компиляции существенно возрастет.

Хороший способ уменьшить время компиляции - использовать многопроцессорную компиляцию:



20) Теперь можно запустить приложение, собранный исполняемый файл exe приложения должен лежать в папке: project1\Release

**Окно приложения:**

