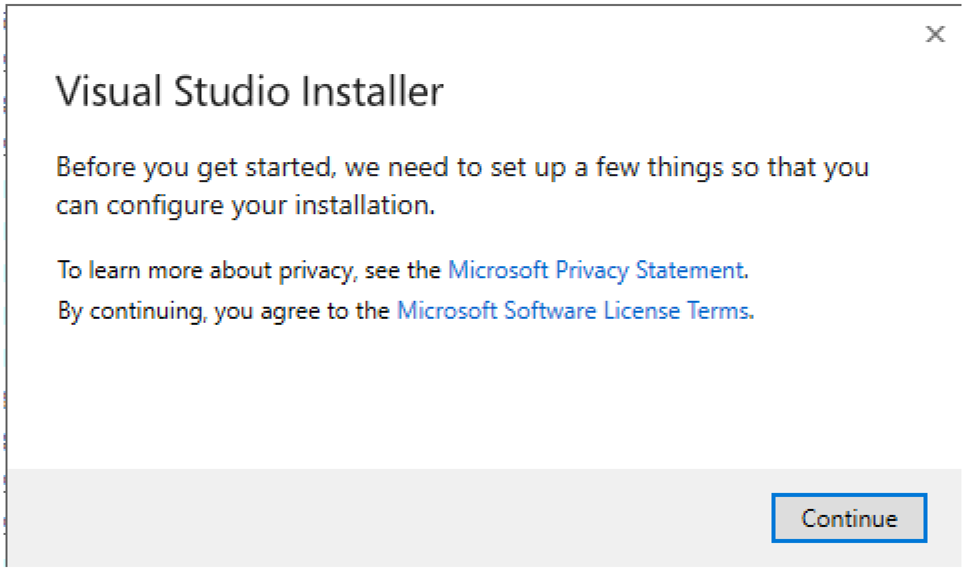
Environment setup, IDE, introduction to OpenGL libraries.

OpenGL Environment Setup

1. Install Visual Studio Code

Step -1 : Double click on the VisualStudioSetup.exe file to install visual studio community 2022.

Step -2:



Visual Studio Installer

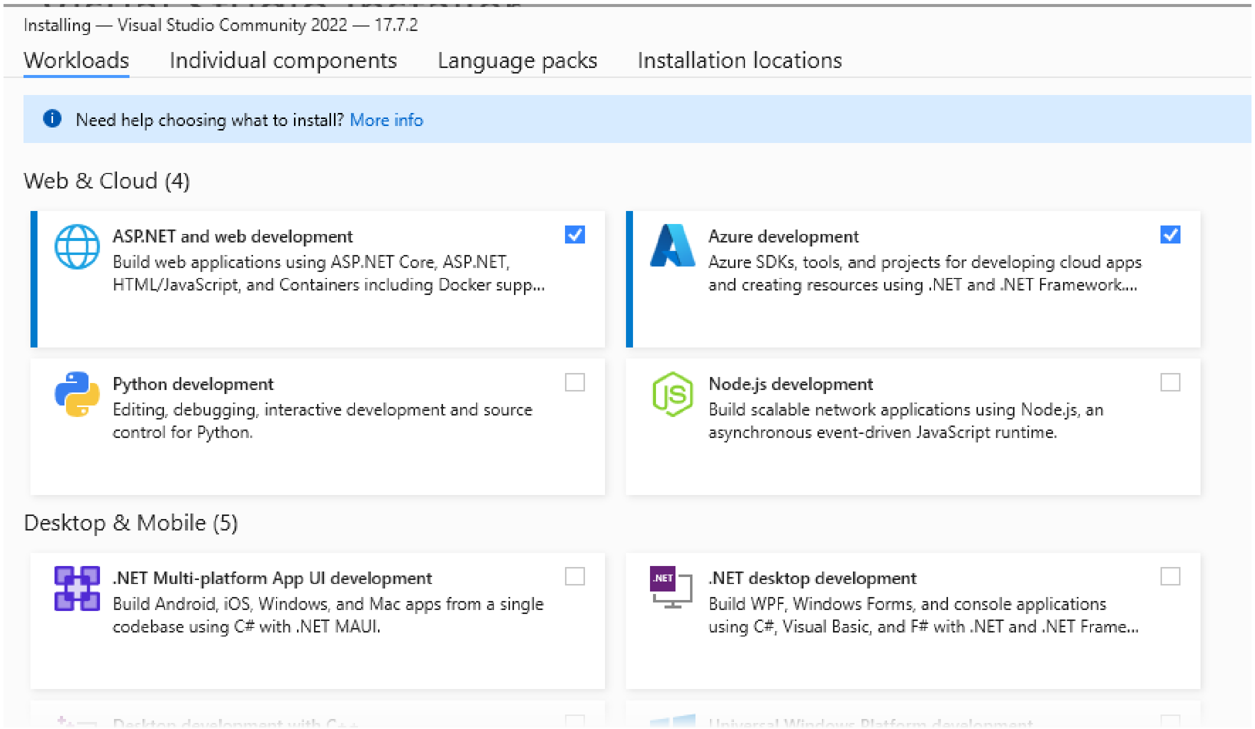
Before you get started, we need to set up a few things so that you can configure your installation.

To learn more about privacy, see the Microsoft Privacy Statement.

By continuing, you agree to the Microsoft Software License Terms.

Click “Continue”.

Step – 3:



Installing — Visual Studio Community 2022 — 17.7.2

Need help choosing what to install? More info

Web & Cloud (4)

☑ ASP.NET and web development
Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker supp...

☑ Azure development
Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET and .NET Framework....

□ Python development
Editing, debugging, interactive development and source control for Python.

□ Node.js development
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Desktop & Mobile (5)

□ .NET Multi-platform App UI development
Build Android, iOS, Windows, and Mac apps from a single codebase using C# with .NET MAUI.

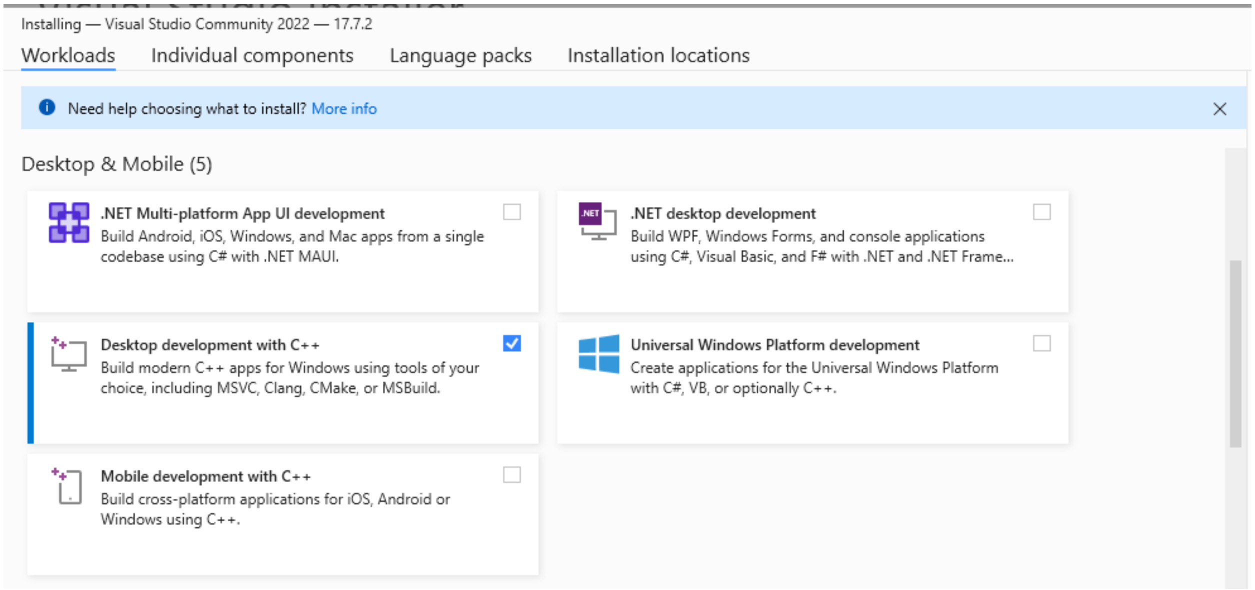
□ .NET desktop development
Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...

□ Desktop development with C++

□ Universal Windows Platform development

Environment setup, IDE, introduction to OpenGL libraries.

Uncheck ASP.NET and web development, Azure development.



Installing — Visual Studio Community 2022 — 17.7.2

Workloads | Individual components | Language packs | Installation locations

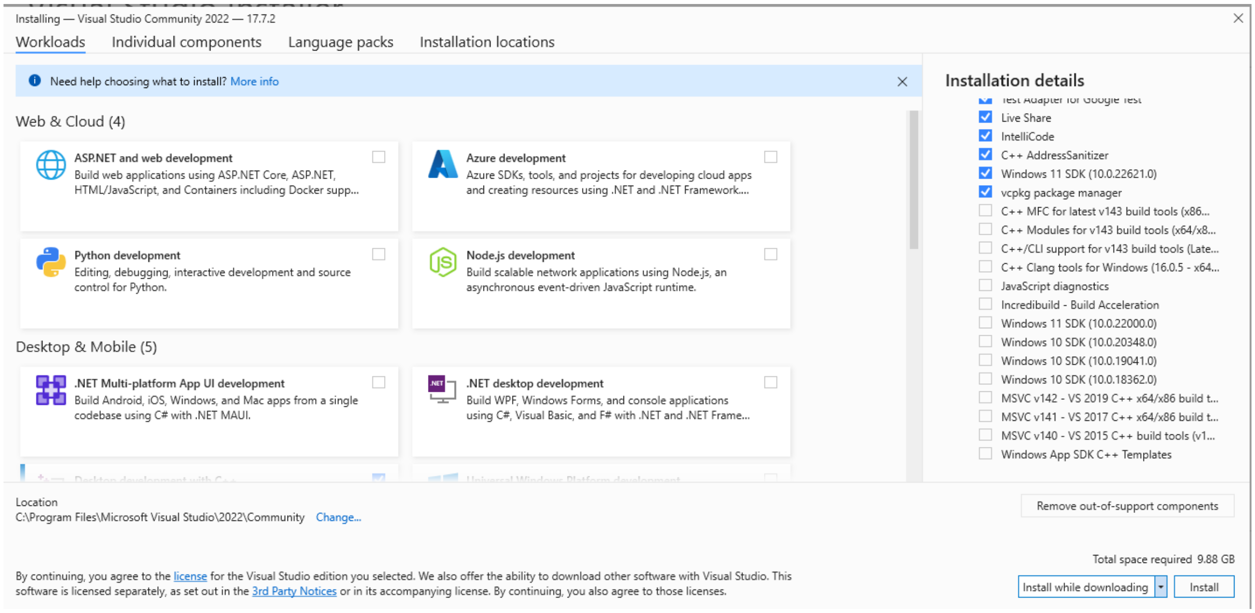
Need help choosing what to install? More info

Desktop & Mobile (5)

- .NET Multi-platform App UI development □ — Build Android, iOS, Windows, and Mac apps from a single codebase using C# with .NET MAUI.
- .NET desktop development □ — Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...
- Desktop development with C++ ☑ — Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.
- Universal Windows Platform development □ — Create applications for the Universal Windows Platform with C#, VB, or optionally C++.
- Mobile development with C++ □ — Build cross-platform applications for iOS, Android or Windows using C++.

Check Desktop development with C++.

Step -4:



Installing — Visual Studio Community 2022 — 17.7.2

Workloads | Individual components | Language packs | Installation locations

Need help choosing what to install? More info

Web & Cloud (4)

- ASP.NET and web development □ — Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker supp...
- Azure development □ — Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET and .NET Framework....
- Python development □ — Editing, debugging, interactive development and source control for Python.
- Node.js development □ — Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Desktop & Mobile (5)

- .NET Multi-platform App UI development □ — Build Android, iOS, Windows, and Mac apps from a single codebase using C# with .NET MAUI.
- .NET desktop development □ — Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...
- Desktop development with C++ ☑
- Universal Windows Platform development □

Installation details

- ☑ Test Adapter for Google Test
- ☑ Live Share
- ☑ IntelliCode
- ☑ C++ AddressSanitizer
- ☑ Windows 11 SDK (10.0.22621.0)
- ☑ vcpkg package manager
- □ C++ MFC for latest v143 build tools (x86...
- □ C++ Modules for v143 build tools (x64/x8...
- □ C++/CLI support for v143 build tools (Late...
- □ C++ Clang tools for Windows (16.0.5 - x64...
- □ JavaScript diagnostics
- □ Incredibuild - Build Acceleration
- □ Windows 11 SDK (10.0.22000.0)
- □ Windows 10 SDK (10.0.20348.0)
- □ Windows 10 SDK (10.0.19041.0)
- □ Windows 10 SDK (10.0.18362.0)
- □ MSVC v142 - VS 2019 C++ x64/x86 build t...
- □ MSVC v141 - VS 2017 C++ x64/x86 build t...
- □ MSVC v140 - VS 2015 C++ build tools (v1...
- □ Windows App SDK C++ Templates

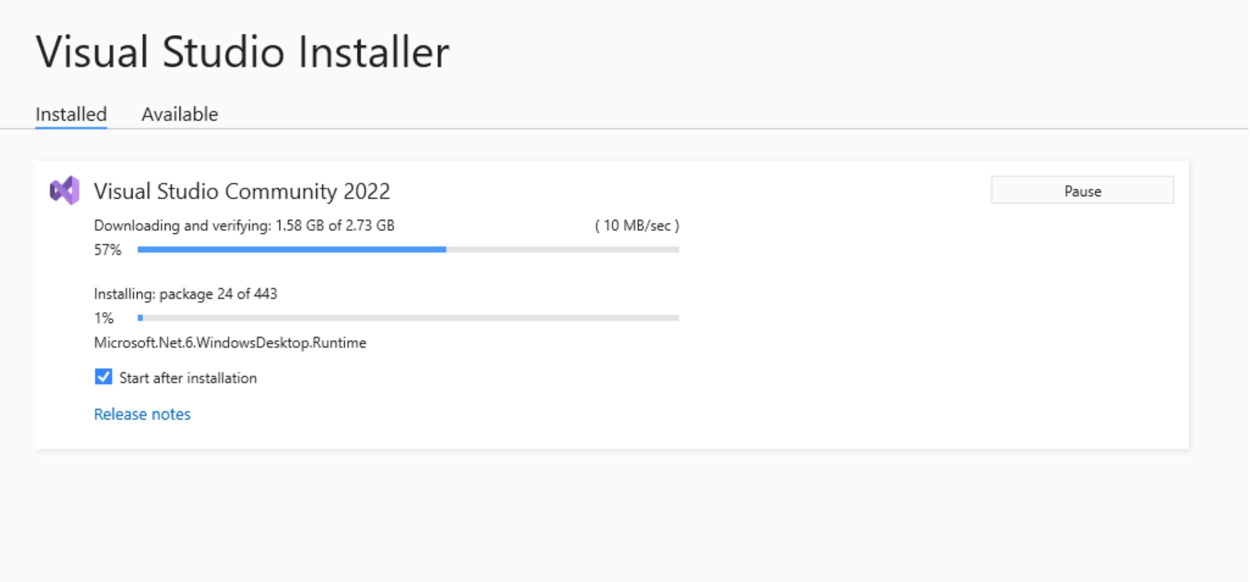
Location
C:\Program Files\Microsoft Visual Studio\2022\Community Change...

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Remove out-of-support components
Total space required 9.88 GB
Install while downloading
Install

Click on the “Install” button.

Environment setup, IDE, introduction to OpenGL libraries.



Visual Studio Installer

Installed Available

Visual Studio Community 2022

Downloading and verifying: 1.58 GB of 2.73 GB (10 MB/sec)

57%

Installing: package 24 of 443

1%

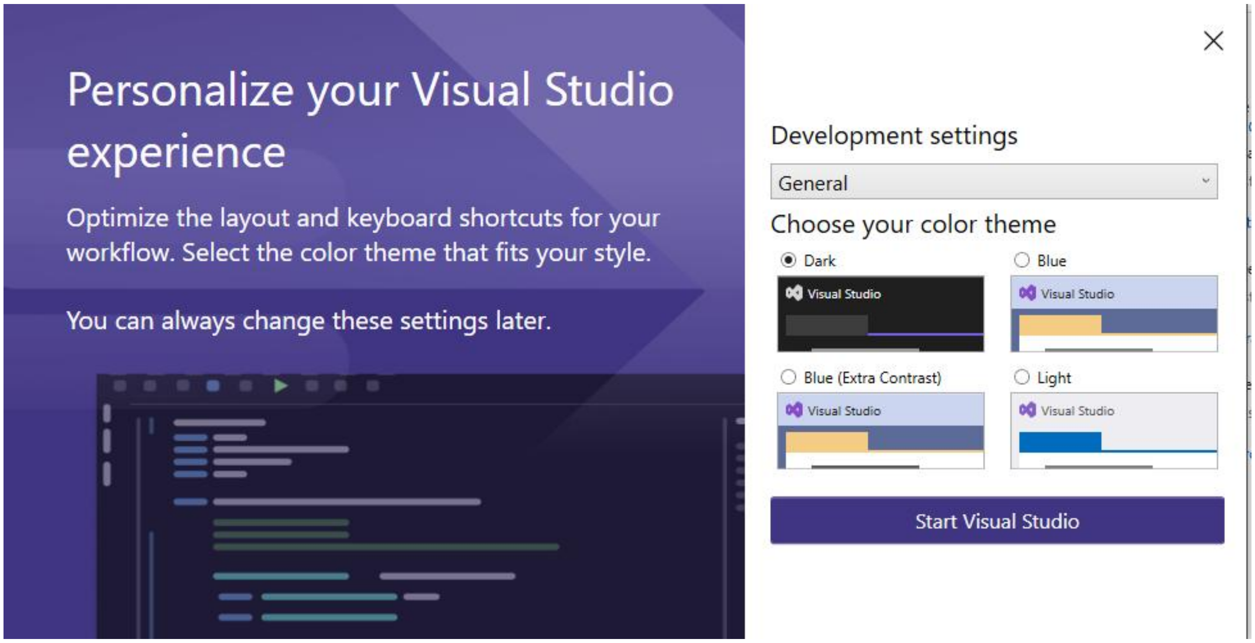
Microsoft.Net.6.WindowsDesktop.Runtime

☑ Start after installation

Release notes

Installation is going on………

Step -5:



Personalize your Visual Studio experience

Optimize the layout and keyboard shortcuts for your workflow. Select the color theme that fits your style.

You can always change these settings later.

Development settings

General

Choose your color theme

● Dark ○ Blue

○ Blue (Extra Contrast) ○ Light

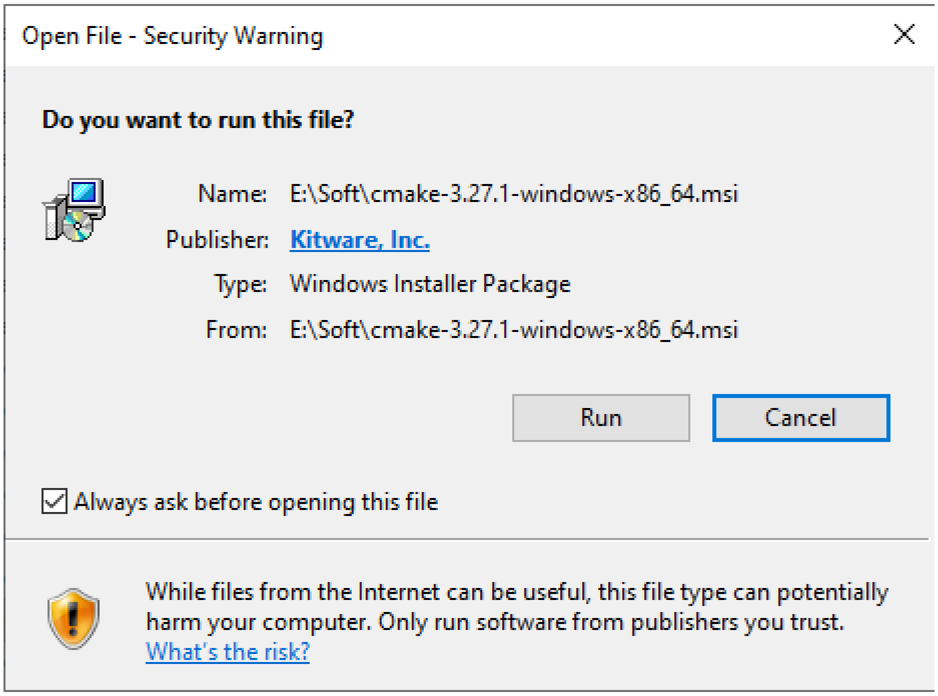
Choose your favorite color theme and click on “Start Visual Studio”.

2. Install CMake

Step – 1: Double click on cmake-3.27.1-windows-x86_64.msi.

Step -2:

Environment setup, IDE, introduction to OpenGL libraries.



Do you want to run this file?

Name: E:\Soft\cmake-3.27.1-windows-x86_64.msi

Publisher: Kitware, Inc.

Type: Windows Installer Package

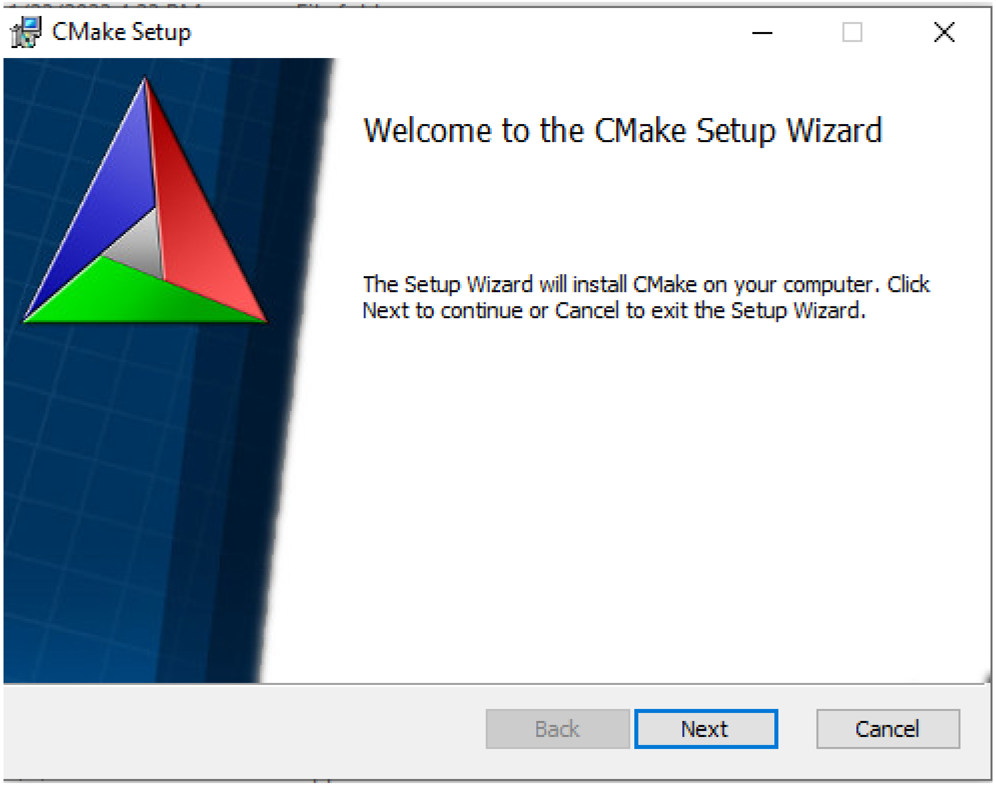
From: E:\Soft\cmake-3.27.1-windows-x86_64.msi

☑ Always ask before opening this file

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

Click on “Run”.

Step -3:

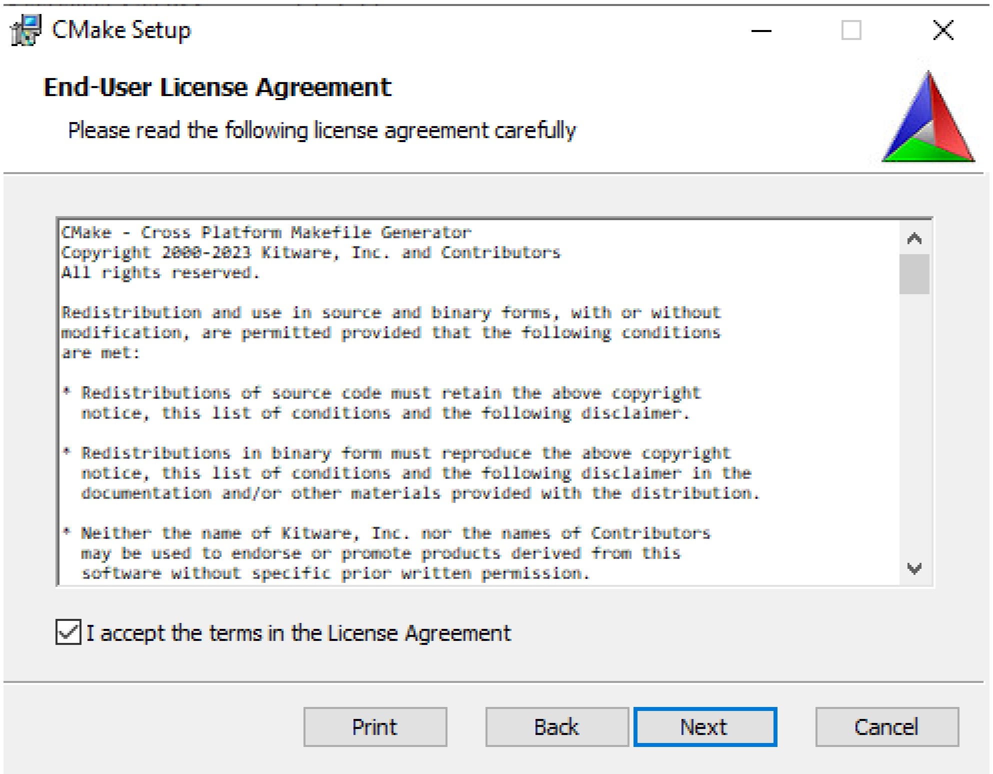


Welcome to the CMake Setup Wizard

The Setup Wizard will install CMake on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Click on “Next”.

Step – 4:



End-User License Agreement

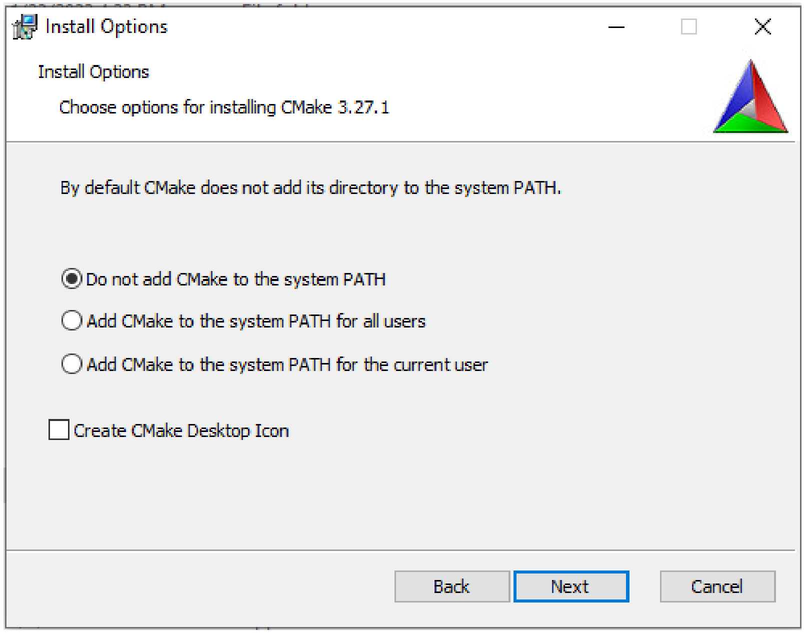
Please read the following license agreement carefully

```
CMake - Cross Platform Makefile Generator
Copyright 2000-2023 Kitware, Inc. and Contributors
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

* Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions and the following disclaimer in the
  documentation and/or other materials provided with the distribution.

* Neither the name of Kitware, Inc. nor the names of Contributors
  may be used to endorse or promote products derived from this
  software without specific prior written permission.
```

☑ I accept the terms in the License Agreement

Accept the terms and license Agreement and click on “Next”.

Step – 5:



Install Options

Choose options for installing CMake 3.27.1

By default CMake does not add its directory to the system PATH.

● Do not add CMake to the system PATH

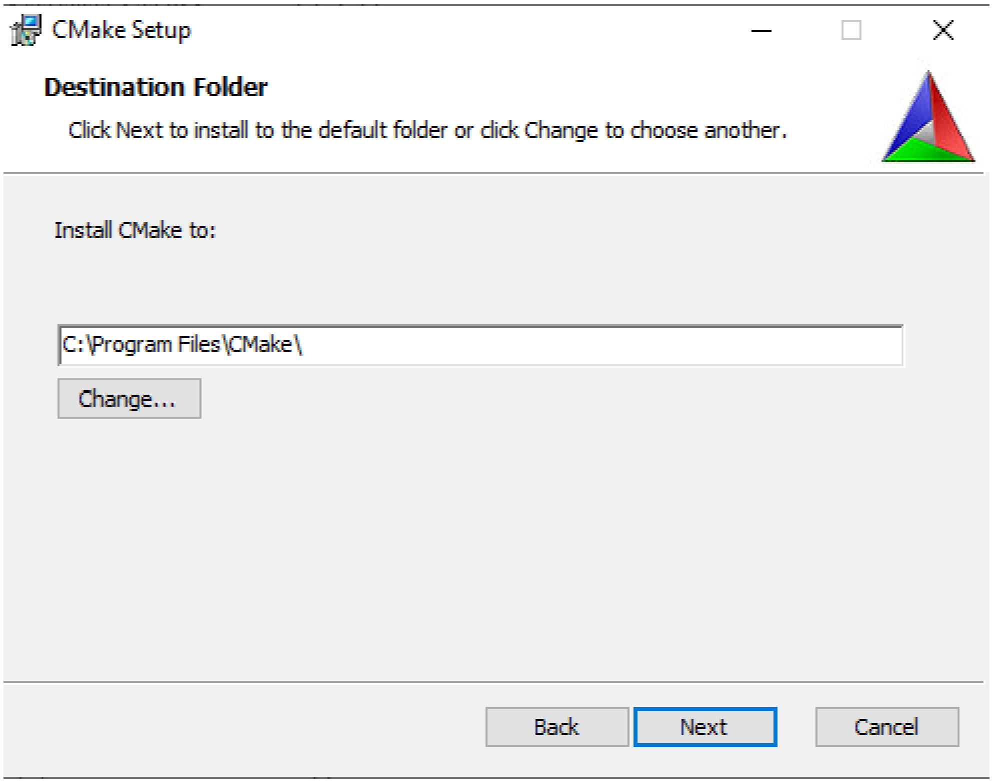
○ Add CMake to the system PATH for all users

○ Add CMake to the system PATH for the current user

□ Create CMake Desktop Icon

Leave everything as default and click on “Next”.

Step – 6:



CMake Setup

Destination Folder

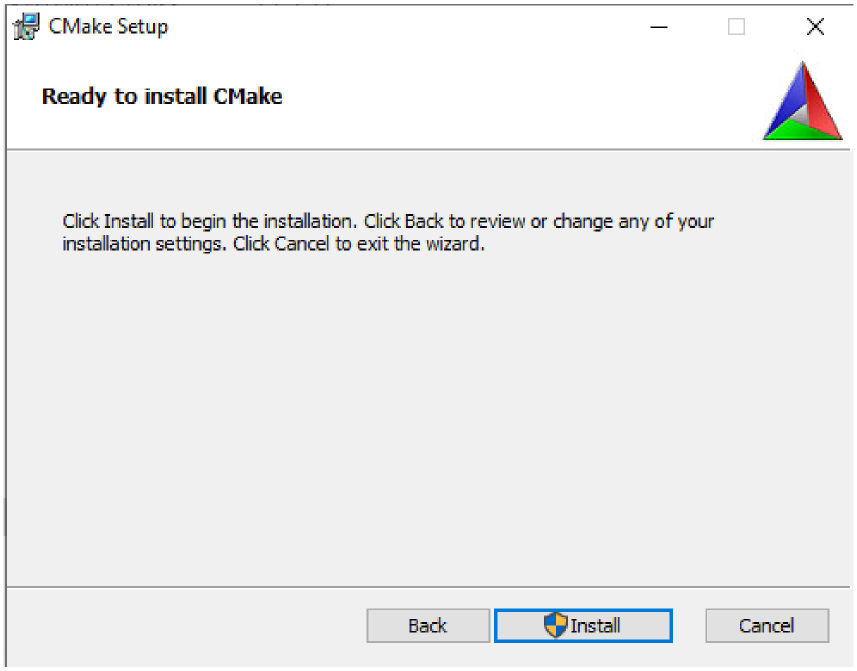
Click Next to install to the default folder or click Change to choose another.

Install CMake to:

C:\Program Files\CMake\

Click on "Next".

Step – 7:



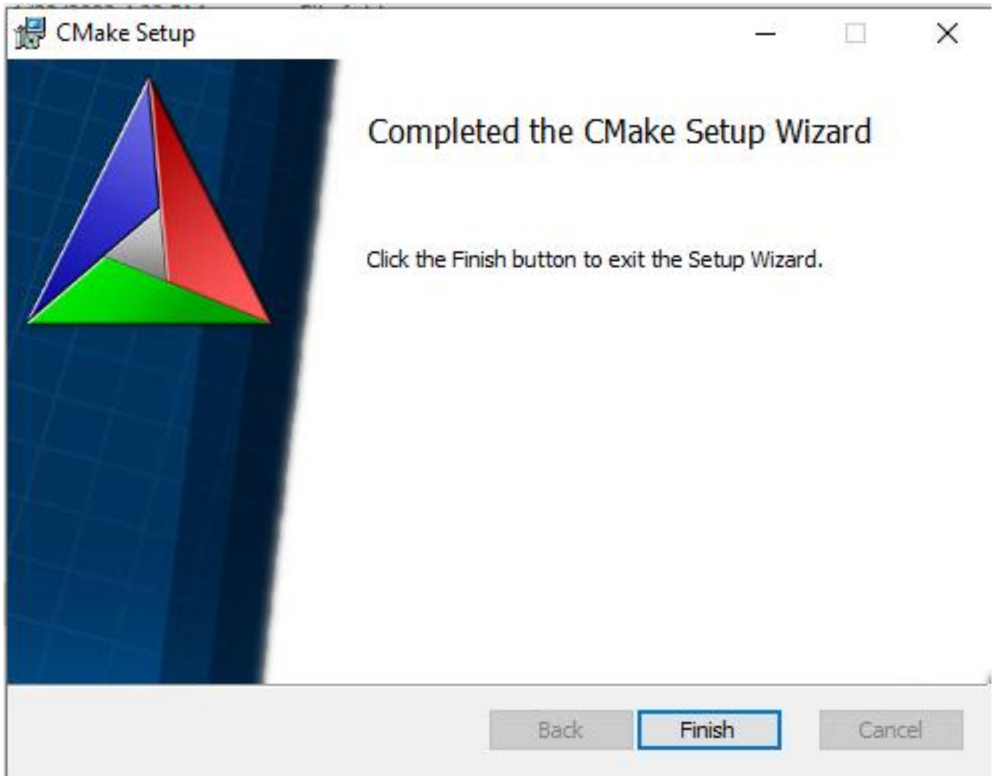
CMake Setup

Ready to install CMake

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Click on "Install".

Step -8:



Completed the CMake Setup Wizard

Click the Finish button to exit the Setup Wizard.

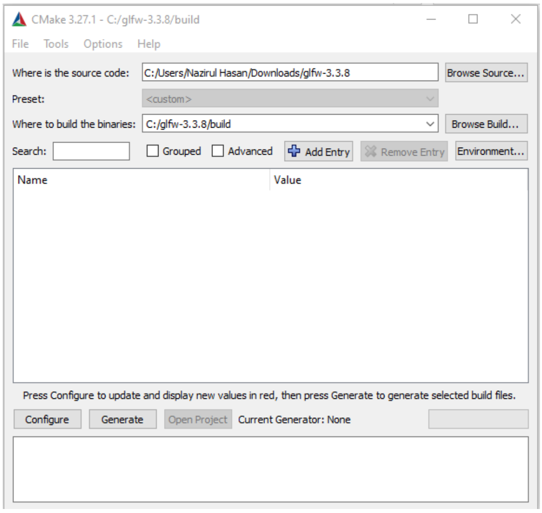
After the installation is over click on the “Finish” button.

3. Install GLFW

Step – 1: Go to https://www.glfw.org/download.html and download the “source package”.

Step – 2: Unzip the downloaded file.

Step – 3:



Where is the source code: C:/Users/Nazirul Hasan/Downloads/glfw-3.3.8

Preset: <custom>

Where to build the binaries: C:/glfw-3.3.8/build

Search:

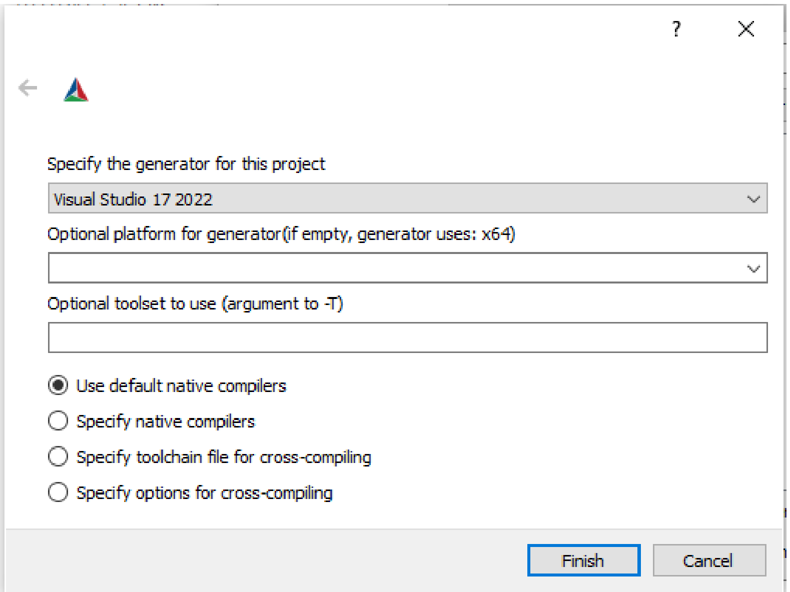
Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: None

Environment setup, IDE, introduction to OpenGL libraries.

Open CMake. For the source code folder we’re going to choose the root folder of the downloaded GLFW source package and for the build folder we’re creating a new directory build and then select that directory. Once the source and destination folders have been set, click the Configure button.

Step – 4:



Specify the generator for this project

Visual Studio 17 2022

Optional platform for generator(if empty, generator uses: x64)

Optional toolset to use (argument to -T)

● Use default native compilers

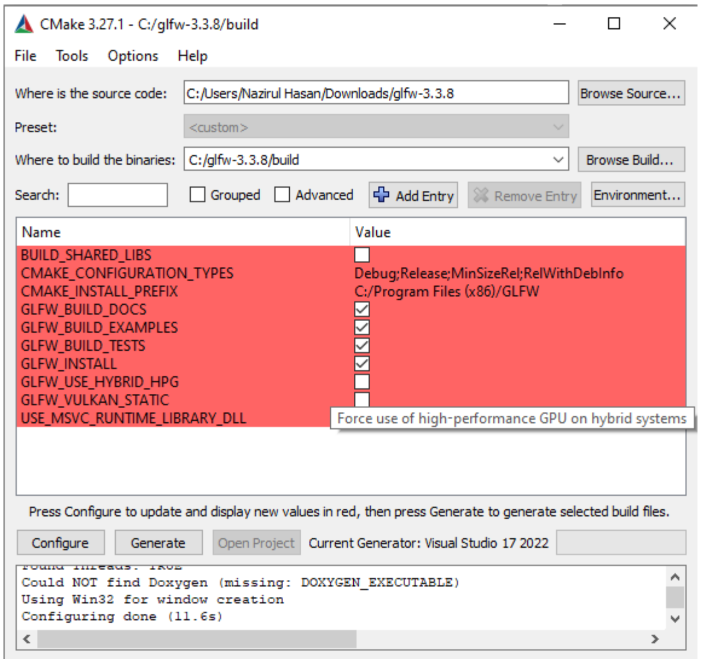
○ Specify native compilers

○ Specify toolchain file for cross-compiling

○ Specify options for cross-compiling

Choose visual Studio 17 2022. Leave everything as default and click “Finish”.

Step – 5:



CMake 3.27.1 - C:/glfw-3.3.8/build

Where is the source code: C:/Users/Nazirul Hasan/Downloads/glfw-3.3.8

Preset: <custom>

Where to build the binaries: C:/glfw-3.3.8/build

Search: □ Grouped □ Advanced

Name	Value
BUILD_SHARED_LIBS	□
CMAKE_CONFIGURATION_TYPES	Debug;Release;MinSizeRel;RelWithDebInfo
CMAKE_INSTALL_PREFIX	C:/Program Files (x86)/GLFW
GLFW_BUILD_DOCS	☑
GLFW_BUILD_EXAMPLES	☑
GLFW_BUILD_TESTS	☑
GLFW_INSTALL	☑
GLFW_USE_HYBRID_HPG	□
GLFW_VULKAN_STATIC	□
USE_MSVC_RUNTIME_LIBRARY_DLL	

Force use of high-performance GPU on hybrid systems

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 17 2022

```
Found Threads: TRUE
Could NOT find Doxygen (missing: DOXYGEN_EXECUTABLE)
Using Win32 for window creation
Configuring done (11.6s)
```

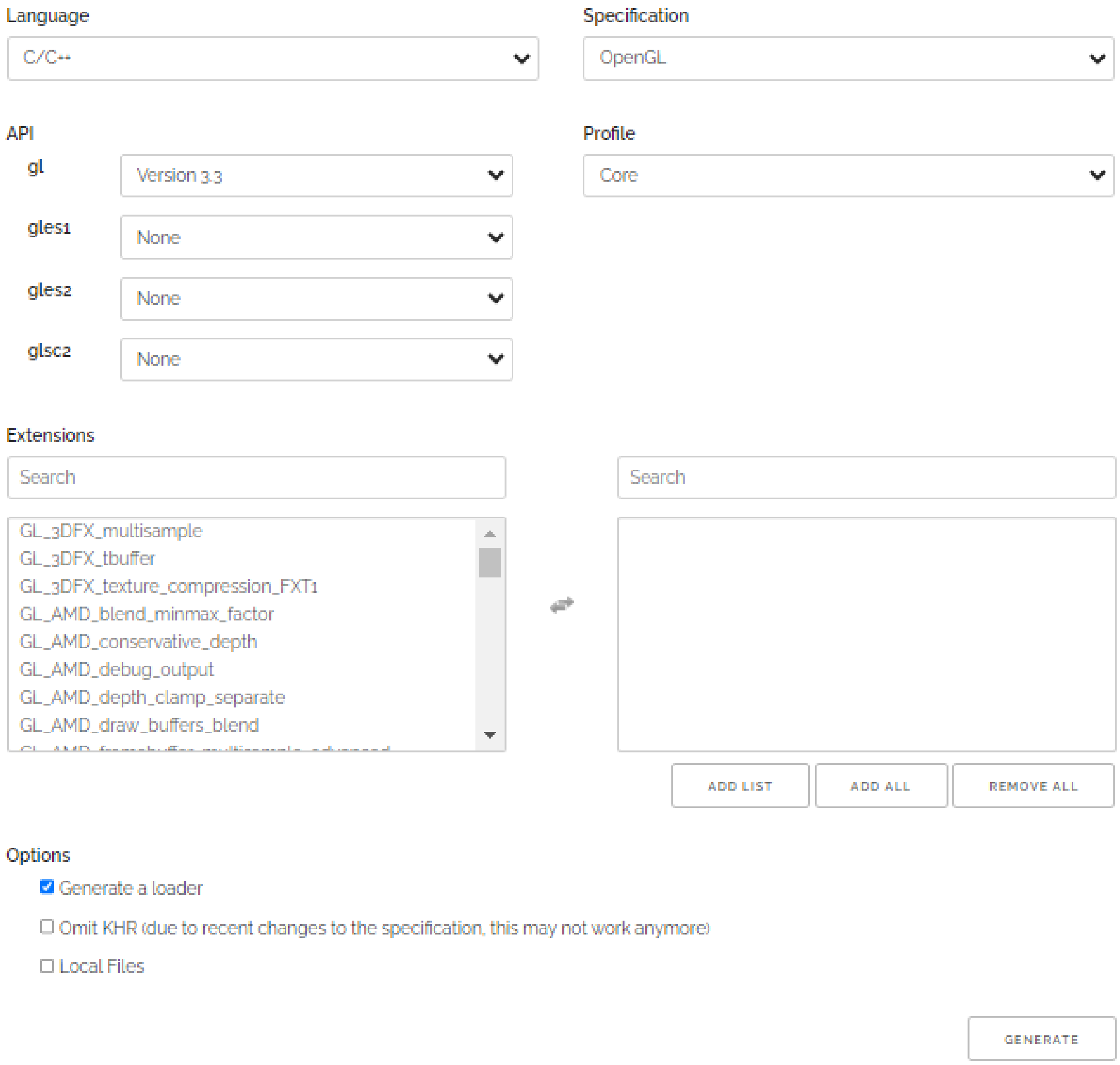
CMake will then display the possible build options to configure the resulting library. We can leave them to their default values and click Configure again to store the settings. Once the settings have been set, we click Generate and the resulting project files will be generated in your build folder.

Step -6: In the build folder a file named GLFW.sln can now be found and we open it with Visual Studio 2022. Now hit build solution. This will give us a compiled library file that can be found in build/src/Debug named glfw3.lib.

Step -7: Create a directory to store the header and library files. I named it “opengl”. Inside opengl, I created two more directories named “Include” and “Lib”. Copy the glfw3.lib file in this Lib folder. After that go to your downloaded glfw directory. Then go to the “include” folder and copy the “GLFW” folder. Now, go to “opengl/Include” directory and paste “GLFW” there.

4. Install GLAD

Step – 1: Go to https://glad.dav1d.de/ .



Language: C/C++

Specification: OpenGL

API

gl: Version 3.3

gles1: None

gles2: None

glsc2: None

Profile: Core

Extensions

GL_3DFX_multisample
GL_3DFX_tbuffer
GL_3DFX_texture_compression_FXT1
GL_AMD_blend_minmax_factor
GL_AMD_conservative_depth
GL_AMD_debug_output
GL_AMD_depth_clamp_separate
GL_AMD_draw_buffers_blend

Options

☑ Generate a loader

☐ Omit KHR (due to recent changes to the specification, this may not work anymore)

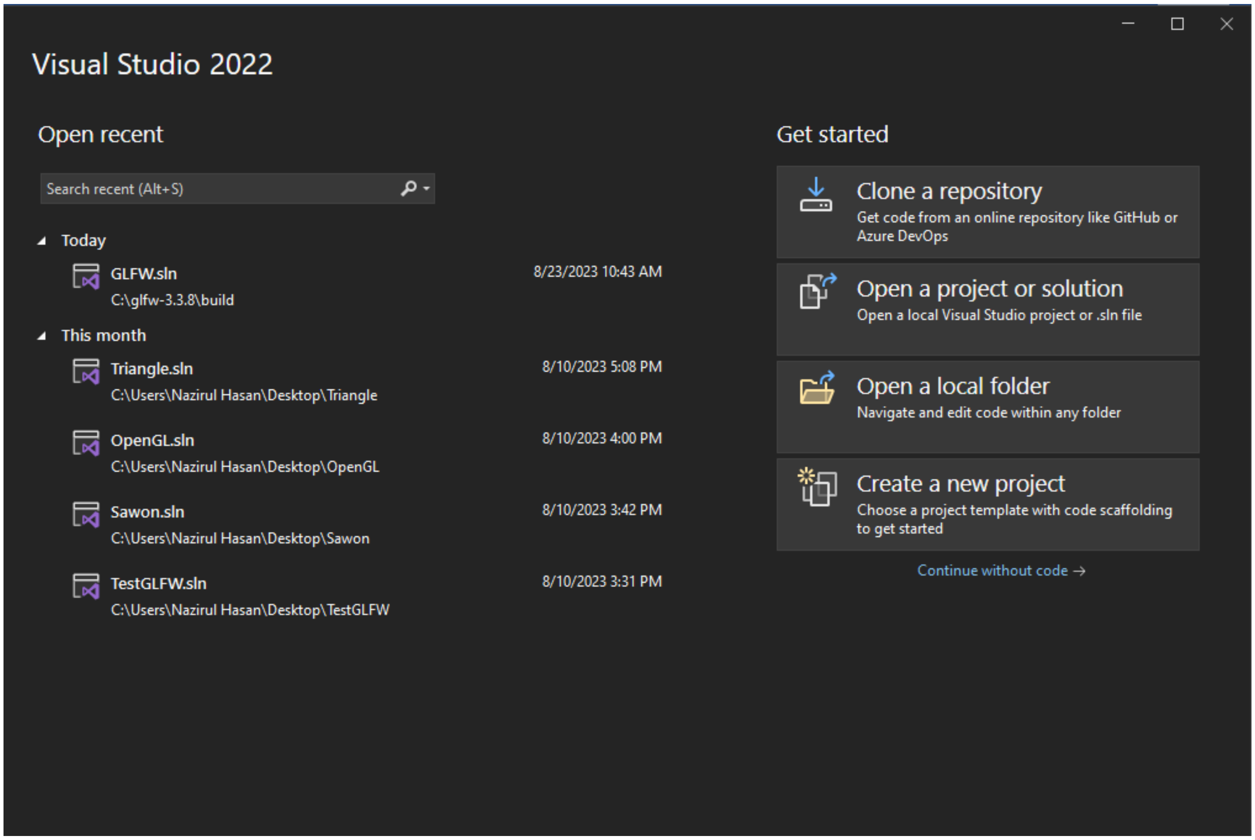
☐ Local Files

Select language as “c++”, specification “opengl”, opengl version “3.3” and profile “core”. Make sure that generate loader is checked. Now Click “GENERATE” button and download the glad.zip file.

Step – 2: Unzip the glad.zip file. Open it and copy the folders inside “include” file. Now, paste those folders inside “opengl/Include” directory. Again, open the “glad” folder. Inside “src” folder you will find glad.c file. Copy it and paste inside opengl folder.

5. Linking

Step – 1:



Visual Studio 2022

Open recent

Search recent (Alt+S)

Today

GLFW.sln
C:\glfw-3.3.8\build
8/23/2023 10:43 AM

This month

Triangle.sln
C:\Users\Nazirul Hasan\Desktop\Triangle
8/10/2023 5:08 PM

OpenGL.sln
C:\Users\Nazirul Hasan\Desktop\OpenGL
8/10/2023 4:00 PM

Sawon.sln
C:\Users\Nazirul Hasan\Desktop\Sawon
8/10/2023 3:42 PM

TestGLFW.sln
C:\Users\Nazirul Hasan\Desktop\TestGLFW
8/10/2023 3:31 PM

Get started

Clone a repository
Get code from an online repository like GitHub or Azure DevOps

Open a project or solution
Open a local Visual Studio project or .sln file

Open a local folder
Navigate and edit code within any folder

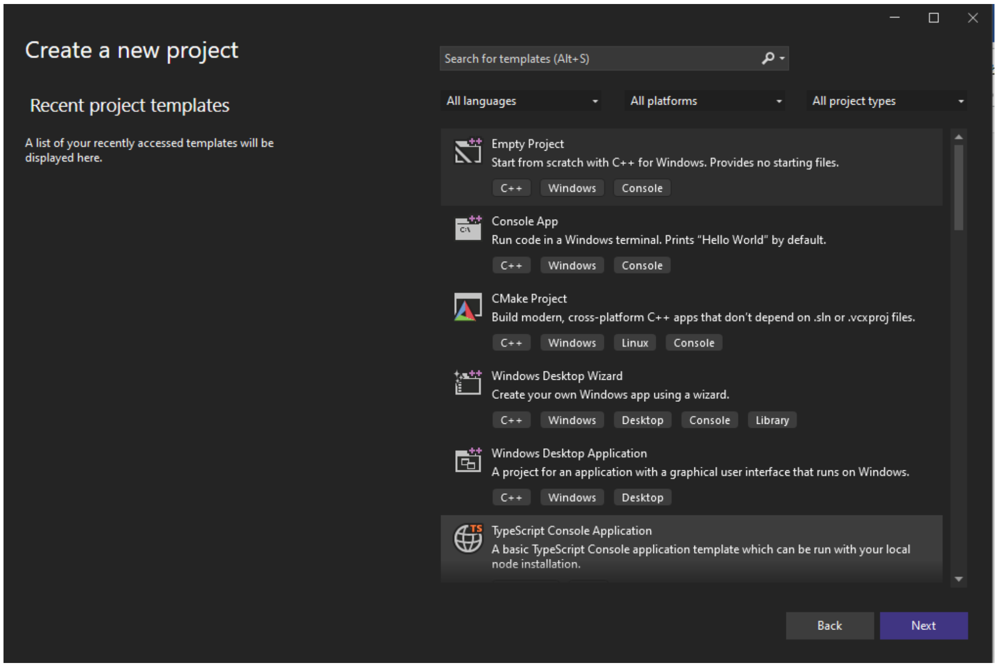
Create a new project
Choose a project template with code scaffolding to get started

Continue without code →

Open visual studio 2022 and click “Create a new project”.

Environment setup, IDE, introduction to OpenGL libraries.

Step -2:



Create a new project

Recent project templates

A list of your recently accessed templates will be displayed here.

Search for templates (Alt+S)

All languages | All platforms | All project types

Empty Project
Start from scratch with C++ for Windows. Provides no starting files.
C++ | Windows | Console

Console App
Run code in a Windows terminal. Prints "Hello World" by default.
C++ | Windows | Console

CMake Project
Build modern, cross-platform C++ apps that don't depend on .sln or .vcxproj files.
C++ | Windows | Linux | Console

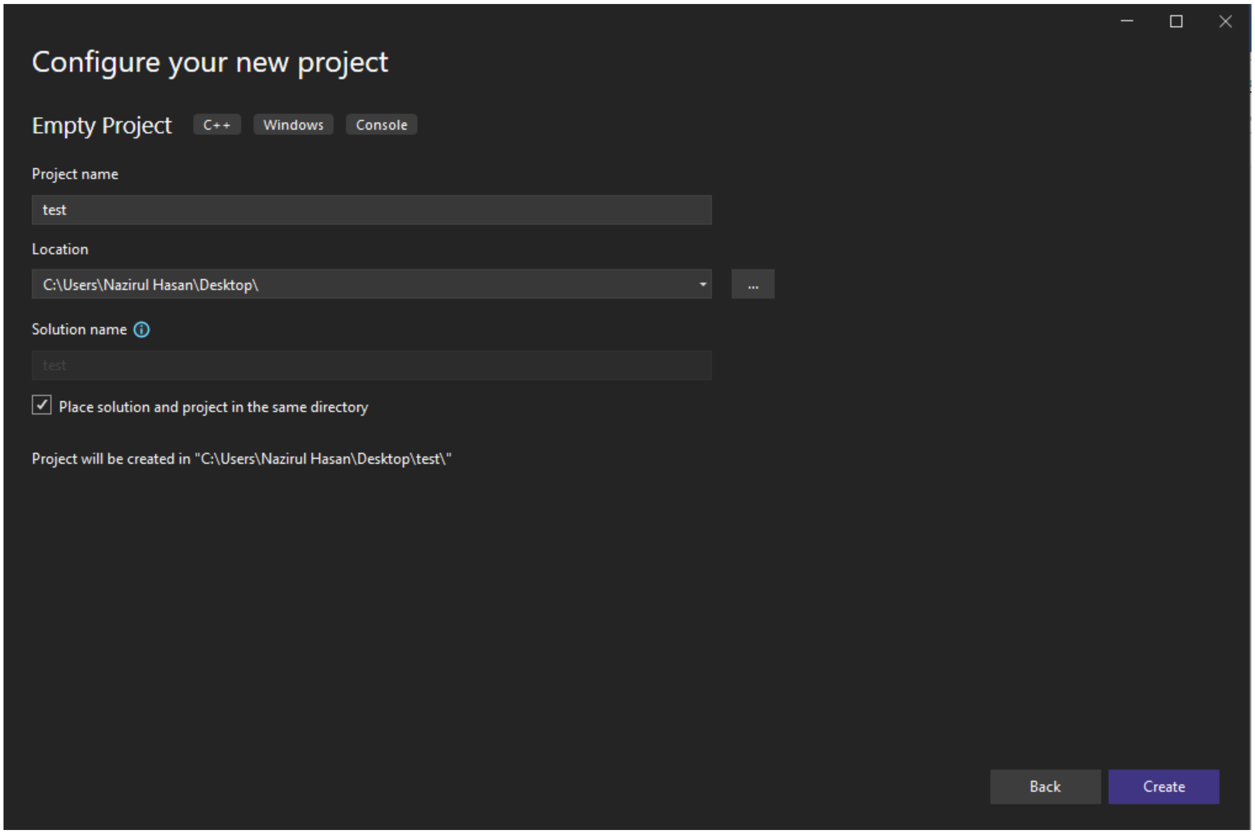
Windows Desktop Wizard
Create your own Windows app using a wizard.
C++ | Windows | Desktop | Console | Library

Windows Desktop Application
A project for an application with a graphical user interface that runs on Windows.
C++ | Windows | Desktop

TypeScript Console Application
A basic TypeScript Console application template which can be run with your local node installation.

Select “Empty project” and then click “Next”.

Step – 3:



Configure your new project

Empty Project | C++ | Windows | Console

Project name: test

Location: C:\Users\Nazirul Hasan\Desktop\

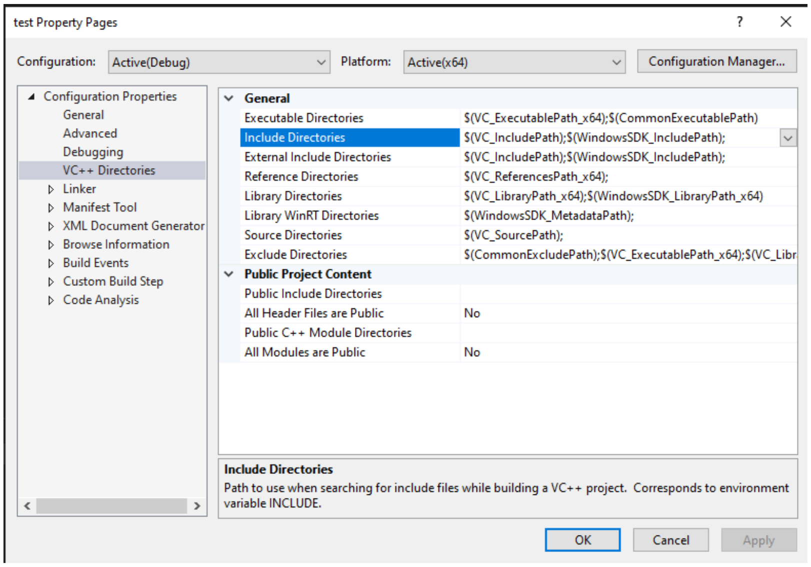
Solution name: test

☑ Place solution and project in the same directory

Project will be created in "C:\Users\Nazirul Hasan\Desktop\test\"

Give a name of your project. Select the location where you want to save it and then click “Create”.

Step – 4: Right-click the project name in the solution explorer and then go to properties. Select VC++ Directories as seen in the image below:

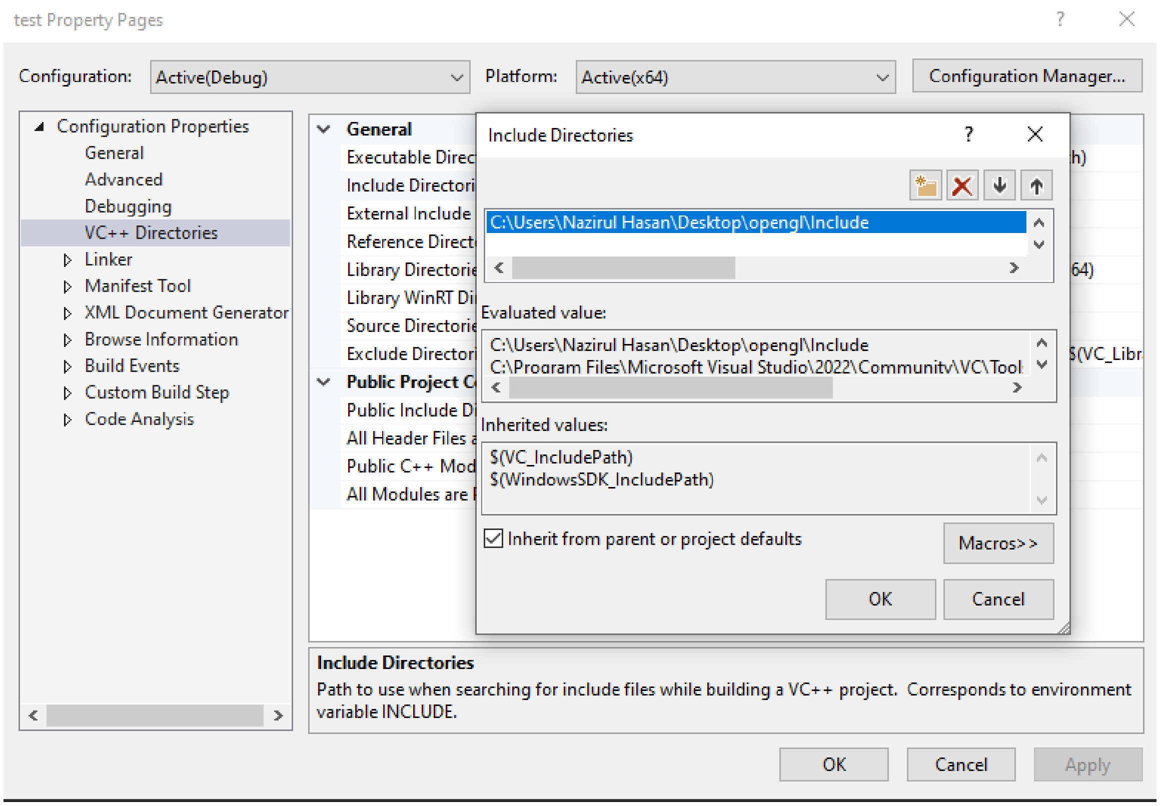


Configuration: Active(Debug)   Platform: Active(x64)

General	
Executable Directories	$(VC_ExecutablePath_x64);$(CommonExecutablePath)
Include Directories	$(VC_IncludePath);$(WindowsSDK_IncludePath);
External Include Directories	$(VC_IncludePath);$(WindowsSDK_IncludePath);
Reference Directories	$(VC_ReferencesPath_x64);
Library Directories	$(VC_LibraryPath_x64);$(WindowsSDK_LibraryPath_x64)
Library WinRT Directories	$(WindowsSDK_MetadataPath);
Source Directories	$(VC_SourcePath);
Exclude Directories	$(CommonExcludePath);$(VC_ExecutablePath_x64);$(VC_Libr
Public Project Content	
Public Include Directories	
All Header Files are Public	No
Public C++ Module Directories	
All Modules are Public	No

Include Directories
Path to use when searching for include files while building a VC++ project. Corresponds to environment variable INCLUDE.

From there on out you can add your own directories to let the project know where to search. This can be done by manually inserting it into the text or clicking the appropriate location string and selecting the <Edit..> option. Do this for both the Library Directories and Include Directories:



Configuration: Active(Debug)   Platform: Active(x64)

Include Directories

C:\Users\Nazirul Hasan\Desktop\opengl\Include

Evaluated value:

C:\Users\Nazirul Hasan\Desktop\opengl\Include
C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tool:

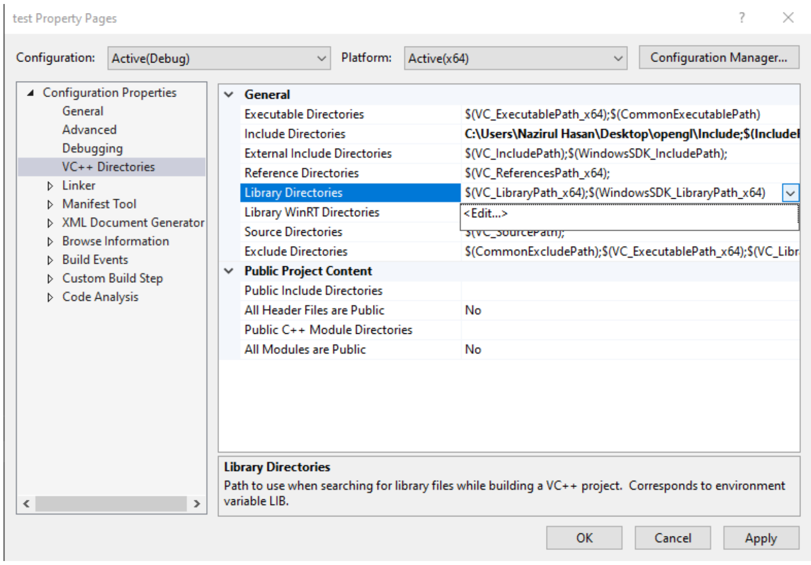
Inherited values:

$(VC_IncludePath)
$(WindowsSDK_IncludePath)

☑ Inherit from parent or project defaults

Include Directories
Path to use when searching for include files while building a VC++ project. Corresponds to environment variable INCLUDE.

For Include Directories, paste the path of the “Include” folder which is inside your “opengl” folder. For Library Directories, paste the path of the “Lib” folder which is inside your “opengl” folder.



test Property Pages

Configuration: Active(Debug) Platform: Active(x64)

Configuration Properties
- General
- Advanced
- Debugging
- VC++ Directories
- Linker
- Manifest Tool
- XML Document Generator
- Browse Information
- Build Events
- Custom Build Step
- Code Analysis

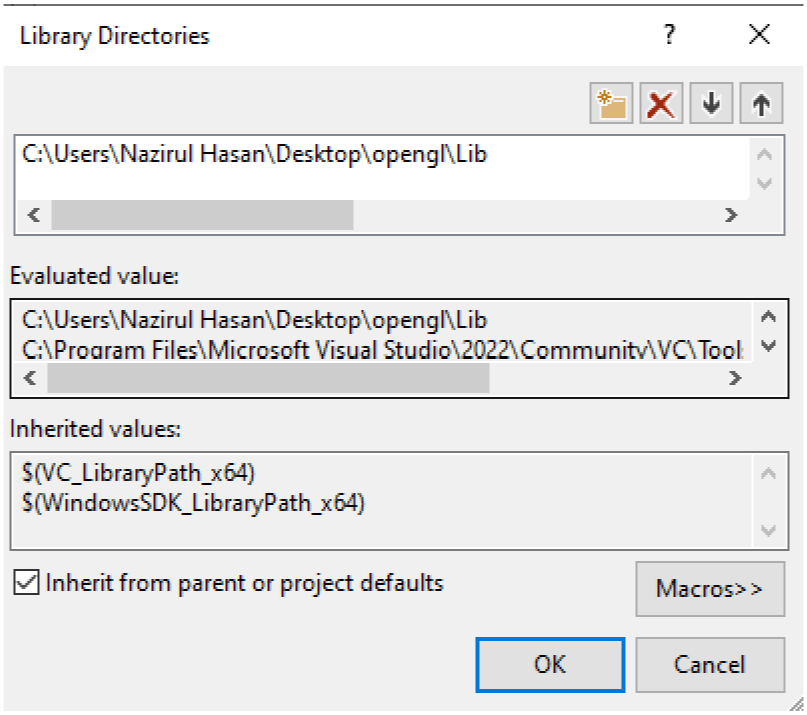
General

Executable Directories	$(VC_ExecutablePath_x64);$(CommonExecutablePath)
Include Directories	**C:\Users\Nazirul Hasan\Desktop\opengl\Include;$(IncludeI**
External Include Directories	$(VC_IncludePath);$(WindowsSDK_IncludePath);
Reference Directories	$(VC_ReferencesPath_x64);
Library Directories	$(VC_LibraryPath_x64);$(WindowsSDK_LibraryPath_x64) <Edit...>
Library WinRT Directories	
Source Directories	$(VC_SourcePath);
Exclude Directories	$(CommonExcludePath);$(VC_ExecutablePath_x64);$(VC_Libr

Public Project Content

Public Include Directories	
All Header Files are Public	No
Public C++ Module Directories	
All Modules are Public	No

Library Directories
Path to use when searching for library files while building a VC++ project. Corresponds to environment variable LIB.



Library Directories

C:\Users\Nazirul Hasan\Desktop\opengl\Lib

Evaluated value:

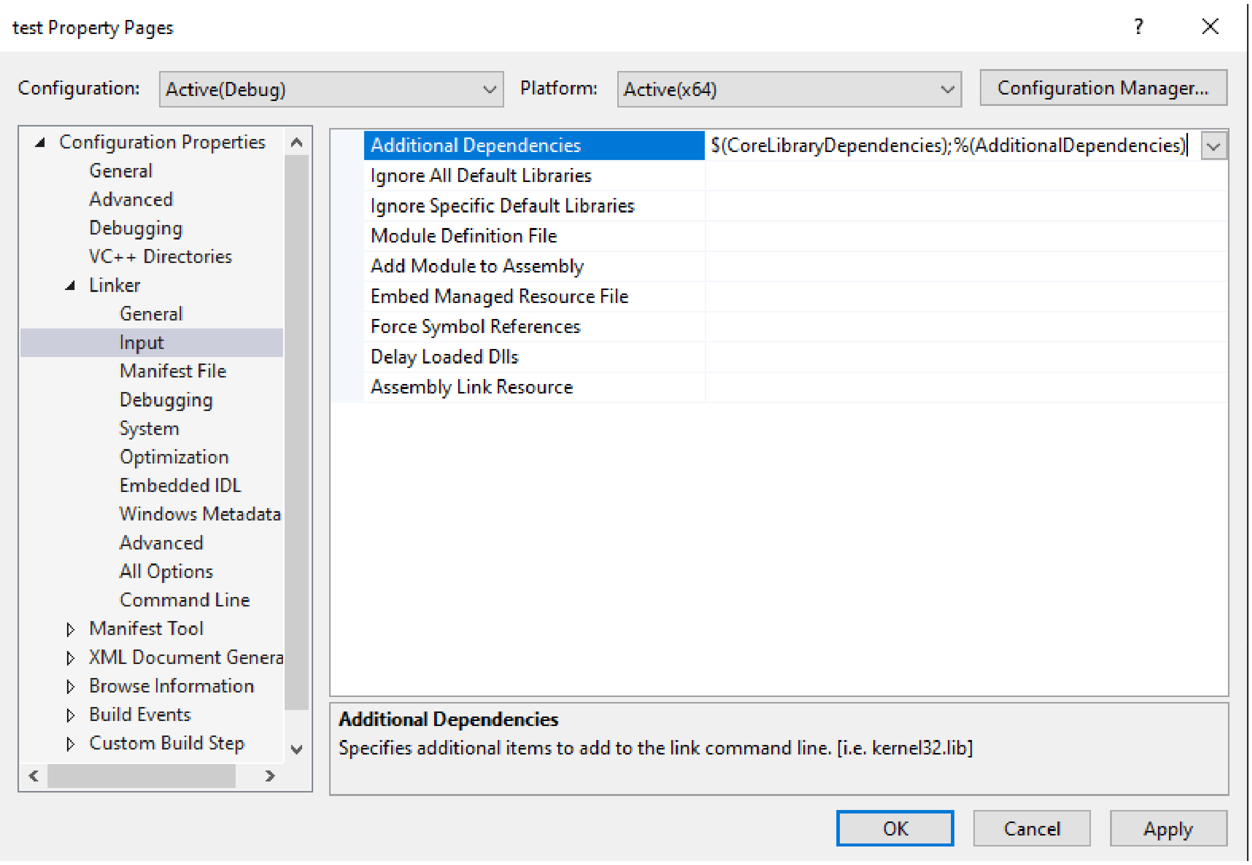
C:\Users\Nazirul Hasan\Desktop\opengl\Lib
C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tool

Inherited values:

$(VC_LibraryPath_x64)
$(WindowsSDK_LibraryPath_x64)

☑ Inherit from parent or project defaults

Since VS can now find all the required files we can finally link GLFW to the project by going to the Linker tab and Input:



test Property Pages

Configuration: Active(Debug)

Platform: Active(x64)

Additional Dependencies: $(CoreLibraryDependencies);%(AdditionalDependencies)

Ignore All Default Libraries

Ignore Specific Default Libraries

Module Definition File

Add Module to Assembly

Embed Managed Resource File

Force Symbol References

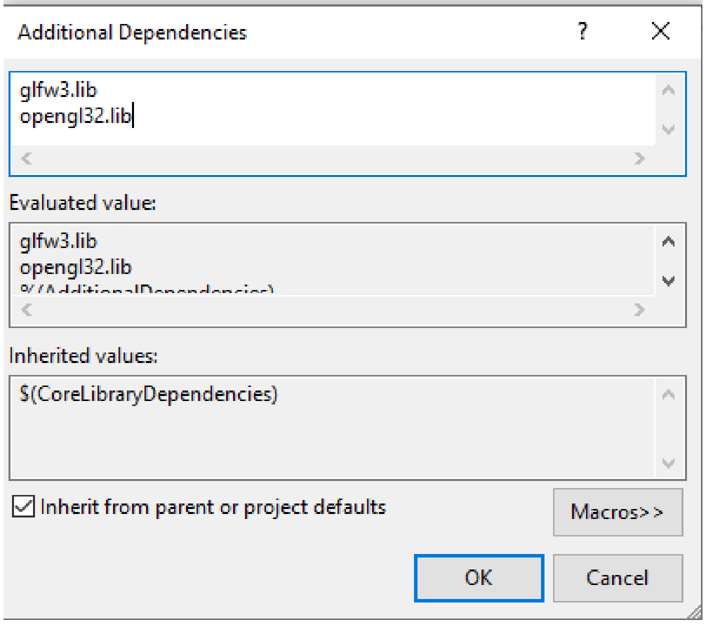
Delay Loaded Dlls

Assembly Link Resource

Additional Dependencies

Specifies additional items to add to the link command line. [i.e. kernel32.lib]

To then link to a library you'd have to specify the name of the library to the linker. Since the library name is glfw3.lib, we add that to the Additional Dependencies field (either manually or using the <Edit..> option) and from that point on GLFW will be linked when we compile. In addition to GLFW we should also add a link entry to the OpenGL library, but this may differ per operating system.



Additional Dependencies

```
glfw3.lib
opengl32.lib
```

Evaluated value:

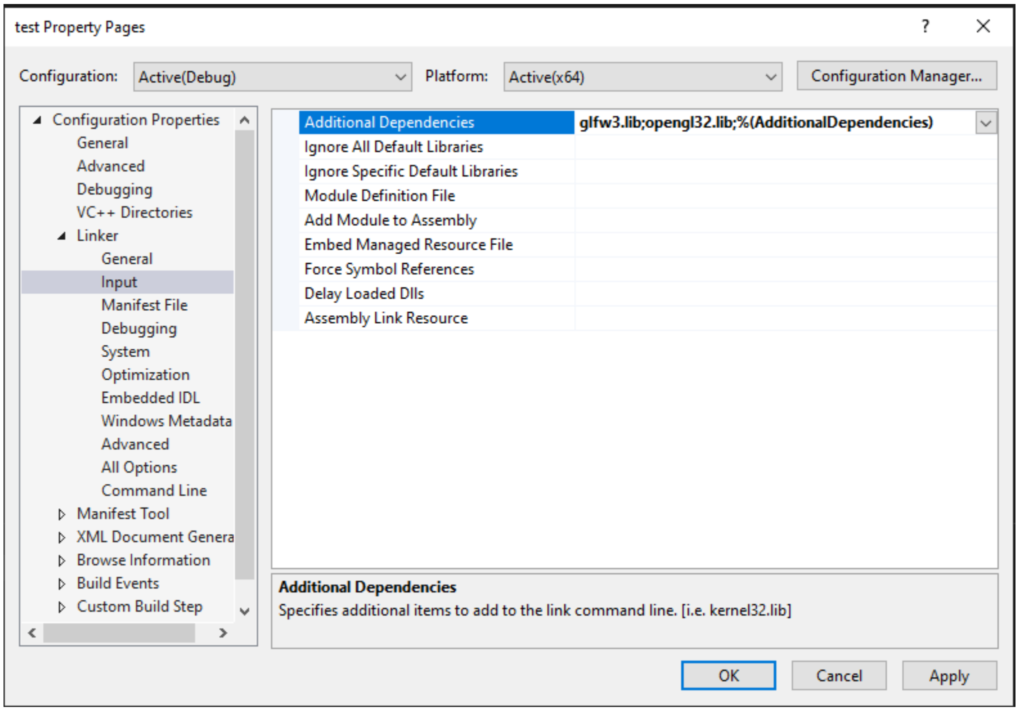
```
glfw3.lib
opengl32.lib
%(AdditionalDependencies)
```

Inherited values:

```
$(CoreLibraryDependencies)
```

☑ Inherit from parent or project defaults

Click “OK”.



test Property Pages

Configuration: Active(Debug)

Platform: Active(x64)

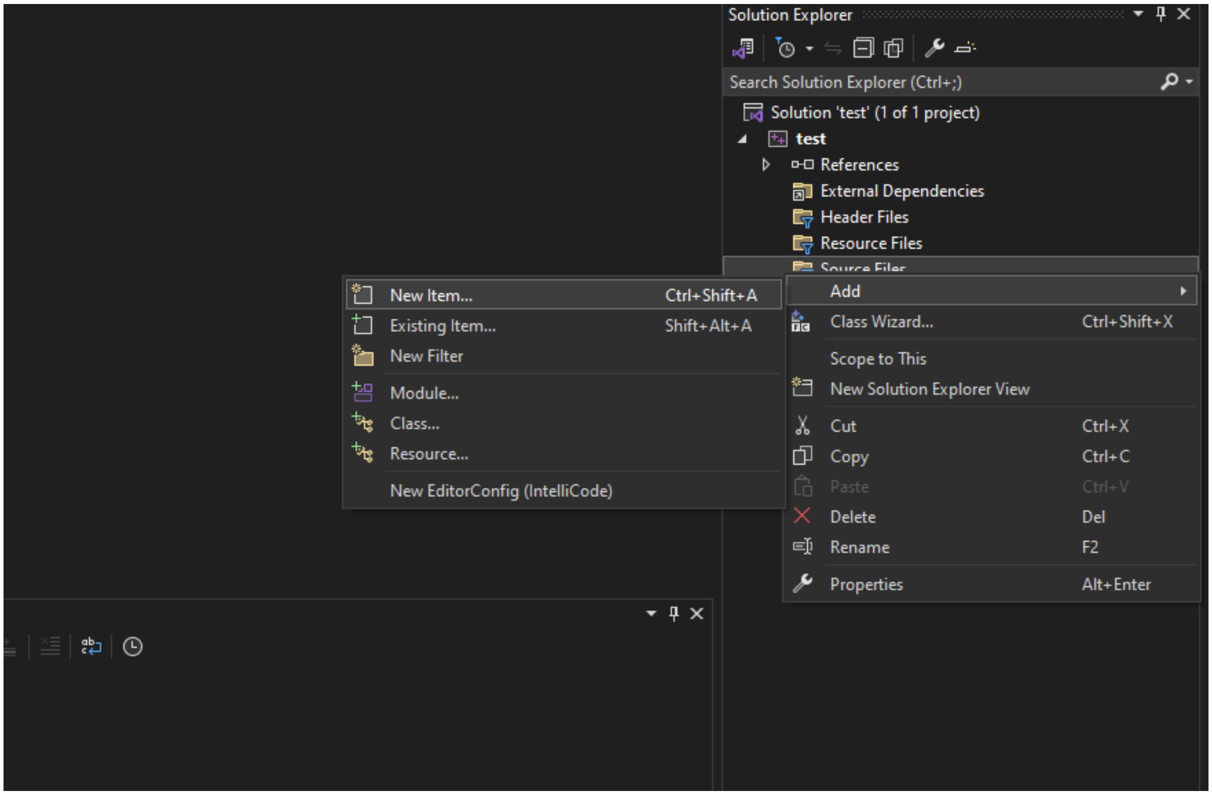
Additional Dependencies: glfw3.lib;opengl32.lib;%(AdditionalDependencies)

Additional Dependencies

Specifies additional items to add to the link command line. [i.e. kernel32.lib]

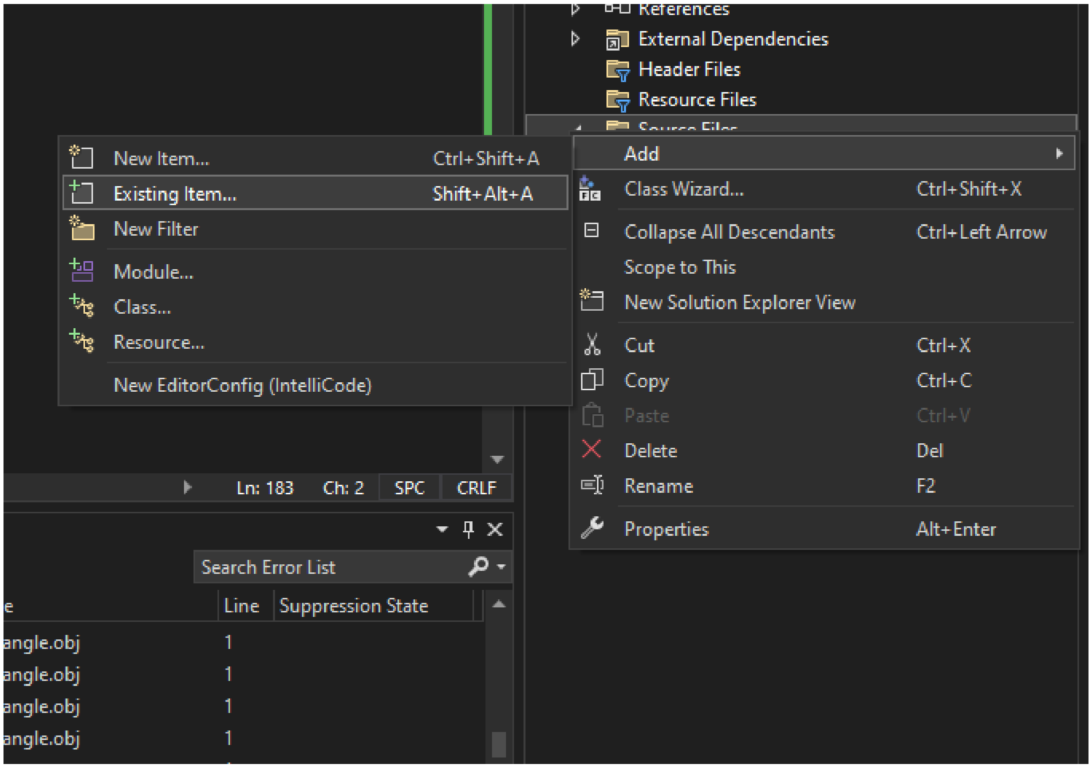
Now, Click “Apply”.

Step – 5 (Lab Task):

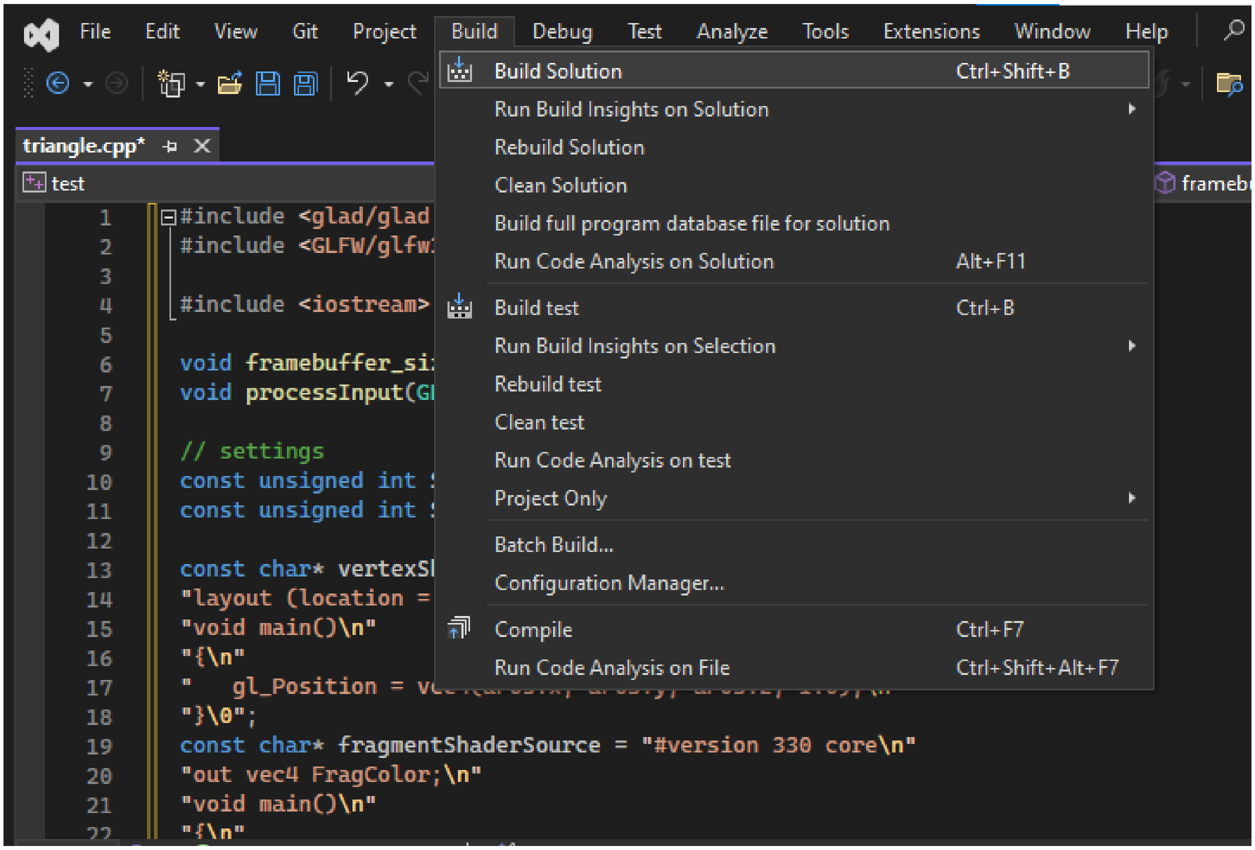


Now, In your project click on “Source Files”, the click on “Add” and click on “New item” to create a new cpp file. Paste the following code inside the file.

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>

#include <iostream>

void framebuffer_size_callback(GLFWwindow* window, int width, int height);
void processInput(GLFWwindow* window);

// settings
const unsigned int SCR_WIDTH = 800;
const unsigned int SCR_HEIGHT = 600;

const char* vertexShaderSource = "#version 330 core\n"
"layout (location = 0) in vec3 aPos;\n"
"void main()\n"
"{\n"
"   gl_Position = vec4(aPos.x, aPos.y, aPos.z, 1.0);\n"
"}\0";
const char* fragmentShaderSource = "#version 330 core\n"
"out vec4 FragColor;\n"
"void main()\n"
"{\n"
"   FragColor = vec4(1.0f, 0.5f, 0.2f, 1.0f);\n"
"}\n\0";

int main()
{
    // glfw: initialize and configure
    // ------------------------------
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

#ifdef __APPLE__
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif

    // glfw window creation
    // --------------------
    GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT, "LearnOpenGL",
NULL, NULL);
    if (window == NULL)
    {
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);

    // glad: load all OpenGL function pointers
    // ---------------------------------------
    if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
    {
        std::cout << "Failed to initialize GLAD" << std::endl;
        return -1;
    }


    // build and compile our shader program
    // ------------------------------------
    // vertex shader
    unsigned int vertexShader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
    glCompileShader(vertexShader);
    // check for shader compile errors
    int success;
    char infoLog[512];
    glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &success);
    if (!success)
    {
        glGetShaderInfoLog(vertexShader, 512, NULL, infoLog);
        std::cout << "ERROR::SHADER::VERTEX::COMPILATION_FAILED\n" << infoLog <<
std::endl;
    }
    // fragment shader
    unsigned int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
    glCompileShader(fragmentShader);
    // check for shader compile errors
    glGetShaderiv(fragmentShader, GL_COMPILE_STATUS, &success);
    if (!success)
    {
        glGetShaderInfoLog(fragmentShader, 512, NULL, infoLog);
        std::cout << "ERROR::SHADER::FRAGMENT::COMPILATION_FAILED\n" << infoLog <<
std::endl;
    }
    // link shaders
    unsigned int shaderProgram = glCreateProgram();
    glAttachShader(shaderProgram, vertexShader);
    glAttachShader(shaderProgram, fragmentShader);
    glLinkProgram(shaderProgram);
    // check for linking errors
    glGetProgramiv(shaderProgram, GL_LINK_STATUS, &success);
    if (!success) {
        glGetProgramInfoLog(shaderProgram, 512, NULL, infoLog);
        std::cout << "ERROR::SHADER::PROGRAM::LINKING_FAILED\n" << infoLog <<
std::endl;
    }
    glDeleteShader(vertexShader);
    glDeleteShader(fragmentShader);

    // set up vertex data (and buffer(s)) and configure vertex attributes
    // ------------------------------------------------------------------
    float vertices[] = {
        -0.5f, -0.5f, 0.0f, // left
         0.5f, -0.5f, 0.0f, // right
         0.0f,  0.5f, 0.0f  // top
    };

    unsigned int VBO, VAO;
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    // bind the Vertex Array Object first, then bind and set vertex buffer(s), and
then configure vertex attributes(s).
    glBindVertexArray(VAO);

    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);

    // note that this is allowed, the call to glVertexAttribPointer registered VBO
as the vertex attribute's bound vertex buffer object so afterwards we can safely
unbind
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    // You can unbind the VAO afterwards so other VAO calls won't accidentally
modify this VAO, but this rarely happens. Modifying other
    // VAOs requires a call to glBindVertexArray anyways so we generally don't
unbind VAOs (nor VBOs) when it's not directly necessary.
    glBindVertexArray(0);


    // uncomment this call to draw in wireframe polygons.
    //glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

    // render loop
    // -----------
    while (!glfwWindowShouldClose(window))
    {
        // input
        // -----
        processInput(window);

        // render
        // ------
        glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        // draw our first triangle
        glUseProgram(shaderProgram);
        glBindVertexArray(VAO); // seeing as we only have a single VAO there's no
need to bind it every time, but we'll do so to keep things a bit more organized
        glDrawArrays(GL_TRIANGLES, 0, 3);
        // glBindVertexArray(0); // no need to unbind it every time

        // glfw: swap buffers and poll IO events (keys pressed/released, mouse moved
etc.)
        // -------------------------------------------------------------------------
------
        glfwSwapBuffers(window);
        glfwPollEvents();
    }

    // optional: de-allocate all resources once they've outlived their purpose:
    // ------------------------------------------------------------------------
    glDeleteVertexArrays(1, &VAO);
    glDeleteBuffers(1, &VBO);
    glDeleteProgram(shaderProgram);

    // glfw: terminate, clearing all previously allocated GLFW resources.
    // ------------------------------------------------------------------
    glfwTerminate();
    return 0;
}

// process all input: query GLFW whether relevant keys are pressed/released this
frame and react accordingly
// ---------------------------------------------------------------------------------
------------------------
void processInput(GLFWwindow* window)
{
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
}

// glfw: whenever the window size changed (by OS or user resize) this callback
function executes
// ---------------------------------------------------------------------------------
------------
void framebuffer_size_callback(GLFWwindow* window, int width, int height)
{
    // make sure the viewport matches the new window dimensions; note that width and
    // height will be significantly larger than specified on retina displays.
    glViewport(0, 0, width, height);
      }
```

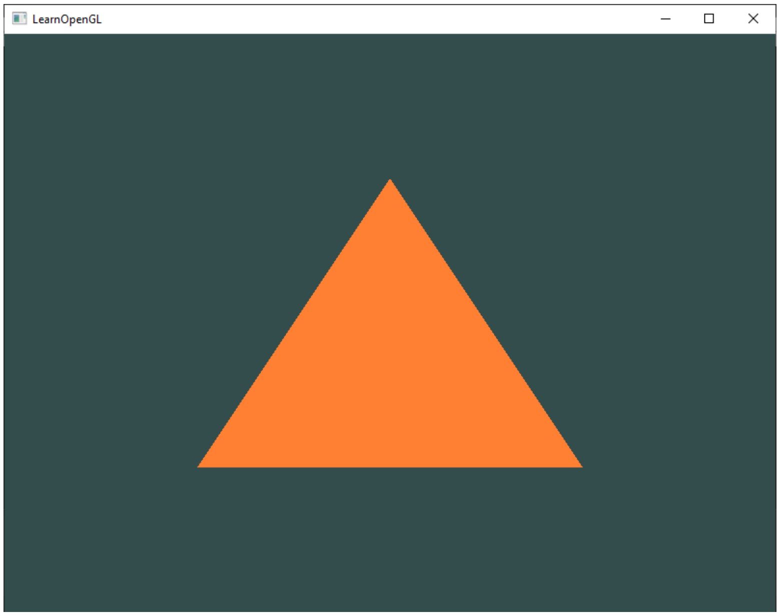


Again, In your project click on “Source Files”, the click on “Add” and click on “Existing item” to add the glad.c file.


File
Edit
View
Git
Project
Build
Debug
Test
Analyze
Tools
Extensions
Window
Help
triangle.cpp*
test
framebı
Build Solution Ctrl+Shift+B
Run Build Insights on Solution
Rebuild Solution
Clean Solution
Build full program database file for solution
Run Code Analysis on Solution Alt+F11
Build test Ctrl+B
Run Build Insights on Selection
Rebuild test
Clean test
Run Code Analysis on test
Project Only
Batch Build...
Configuration Manager...
Compile Ctrl+F7
Run Code Analysis on File Ctrl+Shift+Alt+F7

```
#include <glad/glad
#include <GLFW/glfw

#include <iostream>

void framebuffer_si
void processInput(Gl

// settings
const unsigned int S
const unsigned int S

const char* vertexSh
"layout (location =
"void main()\n"
"{\n"
"   gl_Position = ve
"}\0";
const char* fragmentShaderSource = "#version 330 core\n"
"out vec4 FragColor;\n"
"void main()\n"
"{\n"
```

Now, Build the solution and run it. If you see the below image as output you are good to go.


LearnOpenGL