



# C++



Algorithmic Trading & Quant Research Hub

 **YouTube**



**C++ Set-Up for Algo Quant Trading**  
By Nicholas Burgess

# C++ Set-Up for Algo Quant Trading



## ➤ Part 1 – Visual Studio for Windows

- Online C++ Emulators & Code Snippets
- Visual Studio Projects & Solutions
- C++ Building, Compilation & Linking

## ➤ Part 2 – CMake for Cross-platform Builds

- The CMake Build System
- How to use CMake
- Build Environments & Compilers



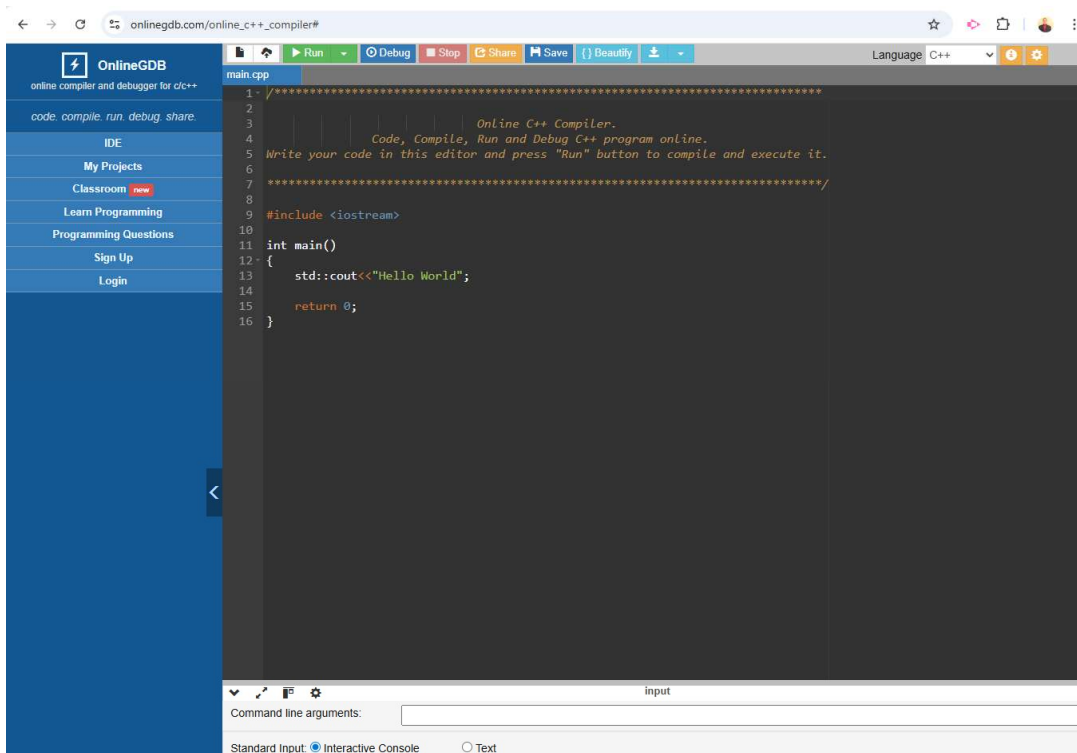
### Example: Visual Studio & CMake

<https://github.com/nburgessx/QuantResearch/tree/main/CMake%20Examples>

Algorithmic Trading & Quant Research Hub



# C++ Online Emulator



- Can Select Language (top-right)  
C++, Python, Java ...
- Can Select Version  
C++14, C++17, C++23 ...
- Great for Learning C++ Syntax & Testing Ideas
- Great for Code Snippets, Sharing & Debugging
- Links nicely with GitHub



Online C++ Compiler

[https://www.onlinegdb.com/online\\_c++\\_compiler#](https://www.onlinegdb.com/online_c++_compiler#)

# C++ Header and Source Files



## 1. Header Files (.h)

- Here we **declare** our functions, classes & interfaces

```
h
int add(int a, int b);
```

## 2. Source Files (.cpp)

- **Define** and implement the header file declarations
- Contain the actual code logic

```
cpp
int add(int a, int b) {
    return a + b;
}
```

## 3. Object Files (.obj)

- Source code from .cpp is compiled into an object file
- It is first translated into **assembly language** .asm
- The assembler then creates the object file .obj
- Object files contain **binary machine code** ready for linking

```
asm
add:
    mov eax, edi    ; move first argument a into register eax
    add eax, esi    ; add second argument b to eax
    ret            ; return value in eax
```

```
obj
B8 ?? ?? ?? ??    ; mov eax, ?
01 F0             ; add eax, esi
C3               ; ret
```

# C++ Build Process – Compile & Link



## 1. Compile (cl.exe)

- Expands #include directives and checks syntax and types
- Converts each translation unit (.cpp file) into an object file

## 2. Link (link.exe)

- Verifies all symbols (i.e. functions and global variables) are defined
- Combines object files into a single file (.exe | .lib | .dll)

## 3. Generated Output Files

- Object Files: These are compiled .cpp files (Windows .obj | Linux/macOS .o)
- Static Library: A library (.lib) or collection of object files merged together
- Dynamic Linked Library (DLL): Compiled code loaded at runtime (.dll)
- Executable: A fully linked program ready to run (.exe)

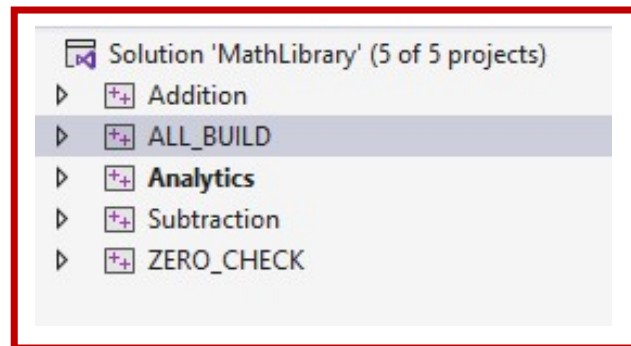
# Connecting Projects & Using Libraries



## Connecting Projects

- In C++ project folders are independent
- To share projects internally, we typically compile them as libraries (.lib)
- To use them we reference the **path to the include directory** (header files) and the **path to and name of the .lib file**
- To share projects externally we compile as them as a library, executable or DLL

## Example: Solution & Projects



## When using Libraries - Why are headers needed?

- **Headers** declare **what exists** (functions, classes, interfaces)
- **Libraries** contain **how it's implemented** (compiled machine code)

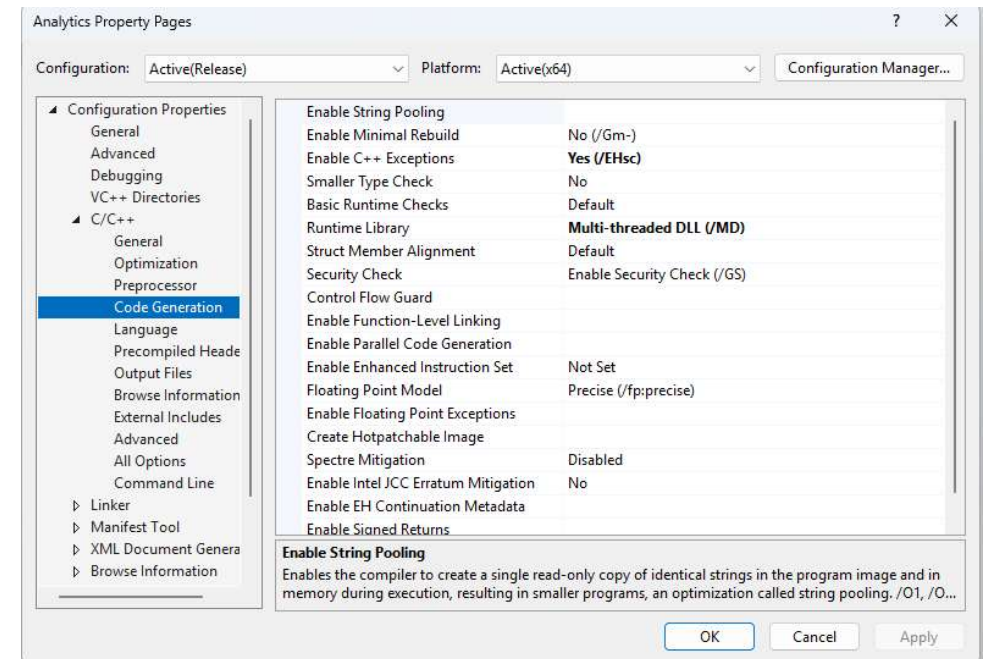
## Sharing Projects

- Header file path(s)
- Library file path



# Application Binary Interface (ABI)

- ABI defines how project binaries are linked and how they manage memory
- Projects sharing runtime resources e.g. `std::vector` or `FILE*` **must** use the same C++ Runtime library (CRT), which handles memory, I/O and startup support
- Dynamic Linkage (**/MD**) links against a shared C++ Runtime DLL (CRT)
- Static Linkage (**/MT**) embeds a private CRT into each binary
- Mixing /MD and /MT is unsafe – such code often builds successfully but fails and crashes at runtime



# C++



Algorithmic Trading & Quant Research Hub

 **YouTube**



Visual Studio







# Visual Studio

## ➤ Solution File

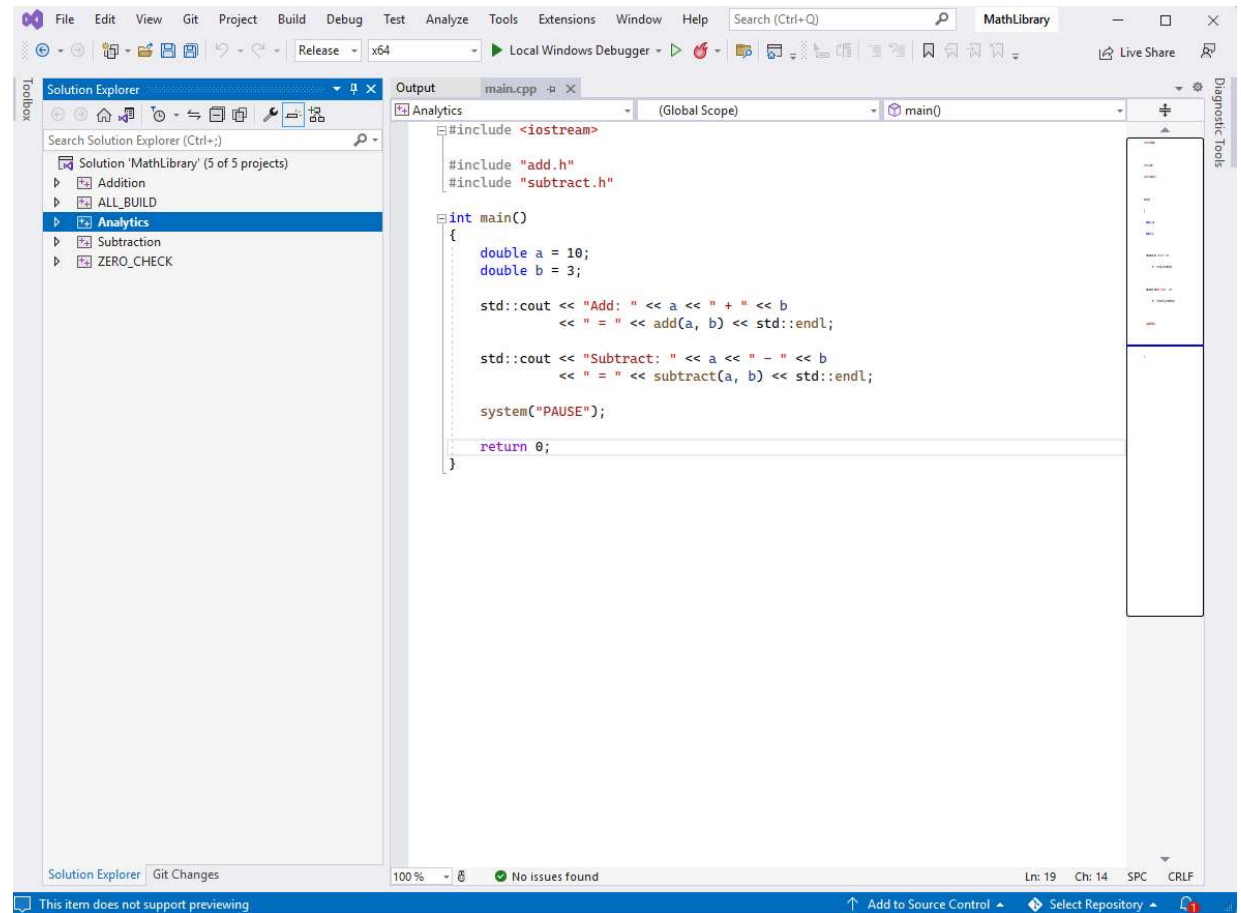
- Start-Up Project
- Project Dependencies (Build Order)
- Configuration
  - Debug, Release, Custom
  - Can Include/Exclude Projects

## ➤ Project Files

- Independent Code Project Groups

## ➤ Features

- Source Control – Git Integration
- Command Line – Dev Command Prompt
- External Tools – Custom Tools / Scripts
- Extensions – e.g. Incredibuild



# Visual Studio Solution & Projects Files

These are XML files in disguise – Try opening them in notepad!

The screenshot displays the Visual Studio IDE interface for a project named 'MathLibrary'. The Solution Explorer on the left shows a solution containing five projects: Addition, ALL\_BUILD, Analytics, Subtraction, and ZERO\_CHECK. The main code editor shows the 'main.cpp' file with the following C++ code:

```
#include <iostream>
#include "add.h"
#include "subtract.h"

int main()
{
    double a = 10;
    double b = 3;

    std::cout << "Add: " << a << " + " << b
    << " = " << add(a, b) << std::endl;

    std::cout << "Subtract: " << a << " - " << b
    << " = " << subtract(a, b) << std::endl;

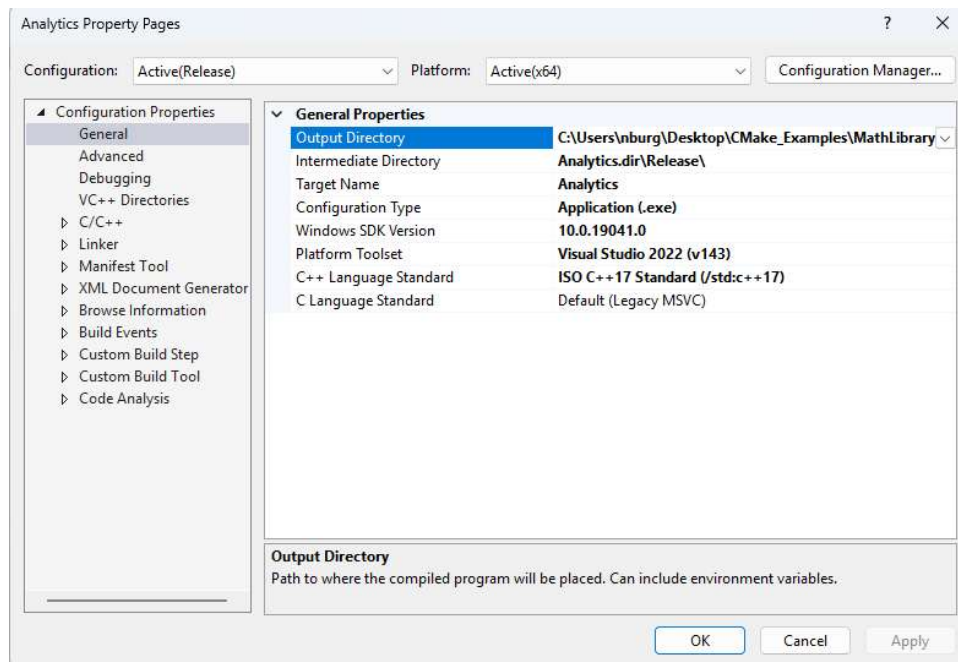
    system("PAUSE");

    return 0;
}
```

The Output window shows the execution results of the program. The 'Solution 'MathLibrary' Property Pages' dialog is open, showing the 'Configuration' tab. The configuration is set to 'Active(Relase)' and the platform is 'Active(x64)'. The 'Project contexts' table shows the build and deployment status for each project:

Project	Configuration	Platform	Build	Deploy
Addition	Release	x64	<input checked="" type="checkbox"/>	<input type="checkbox"/>
ALL_BUILD	Release	x64	<input type="checkbox"/>	<input type="checkbox"/>
Analytics	Release	x64	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Subtraction	Release	x64	<input checked="" type="checkbox"/>	<input type="checkbox"/>
ZERO_CHECK	Release	x64	<input checked="" type="checkbox"/>	<input type="checkbox"/>

# Visual Studio Project Properties



## Output type

- Configuration Type (.lib | .exe | .dll)

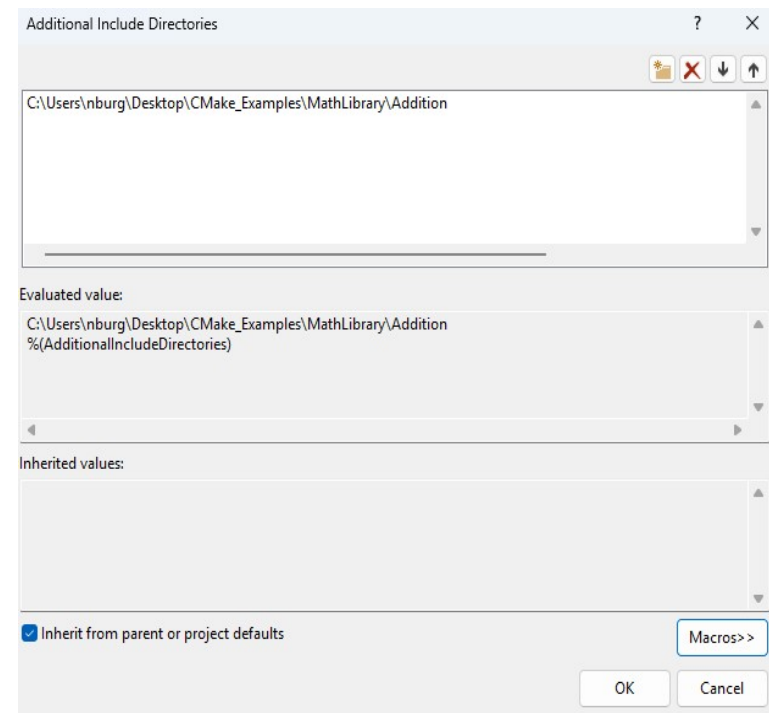
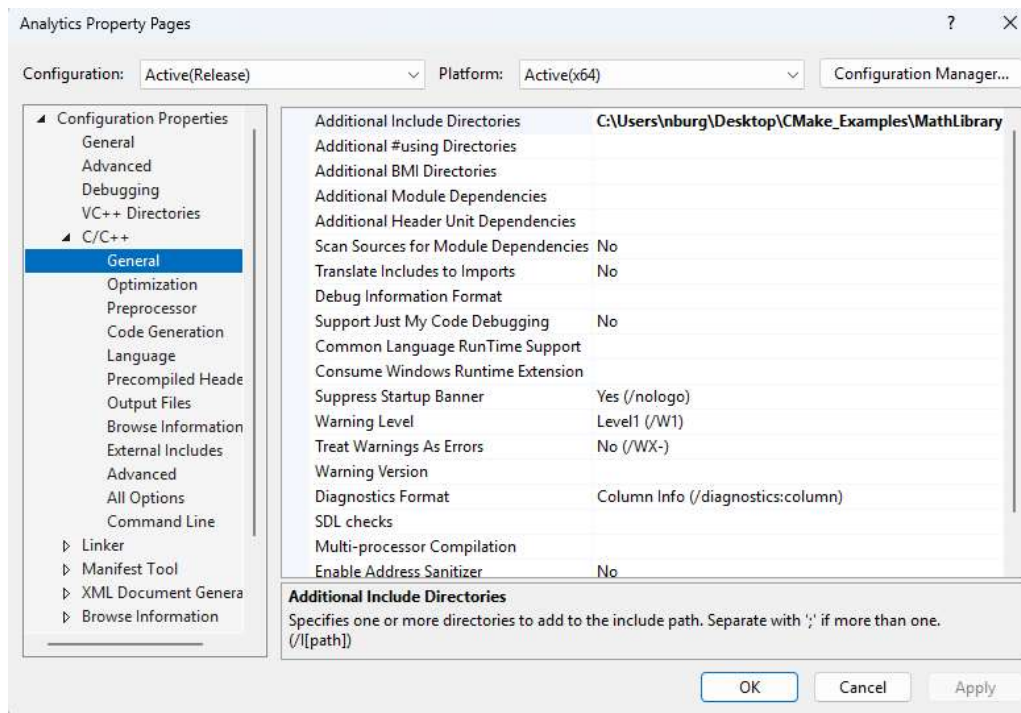
## Where outputs go

- Intermediate Directory (.obj)
- Output Directory (.lib | .exe | .dll)

## Solution and Project Files [TOP TIP]

- These are XML files that can be opened in Notepad
- XML supports extra features e.g. recursive file paths

# VS Project Properties – C/C++ Compiler

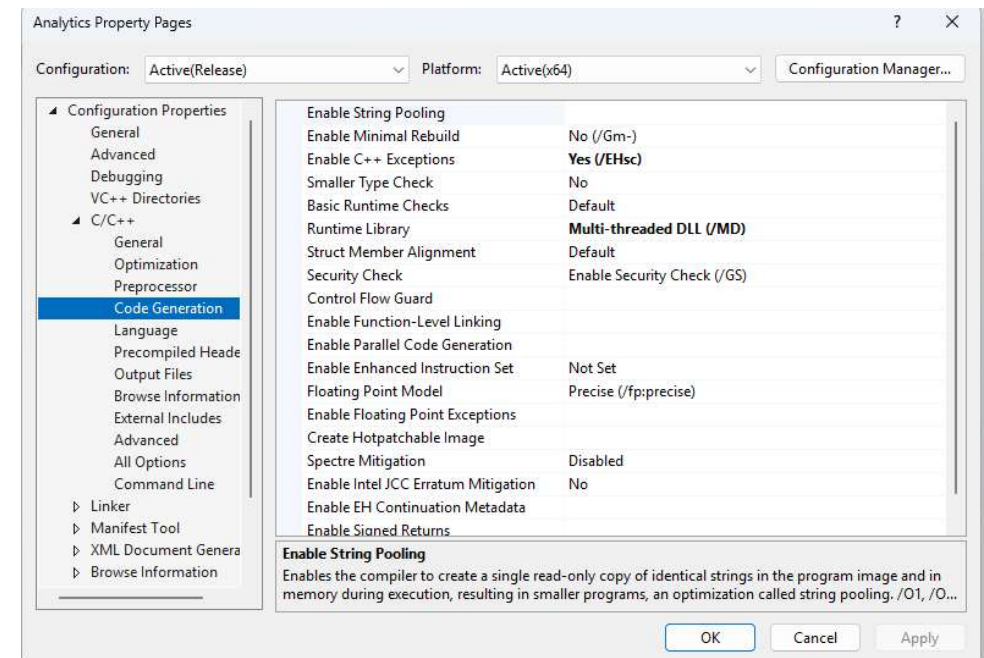
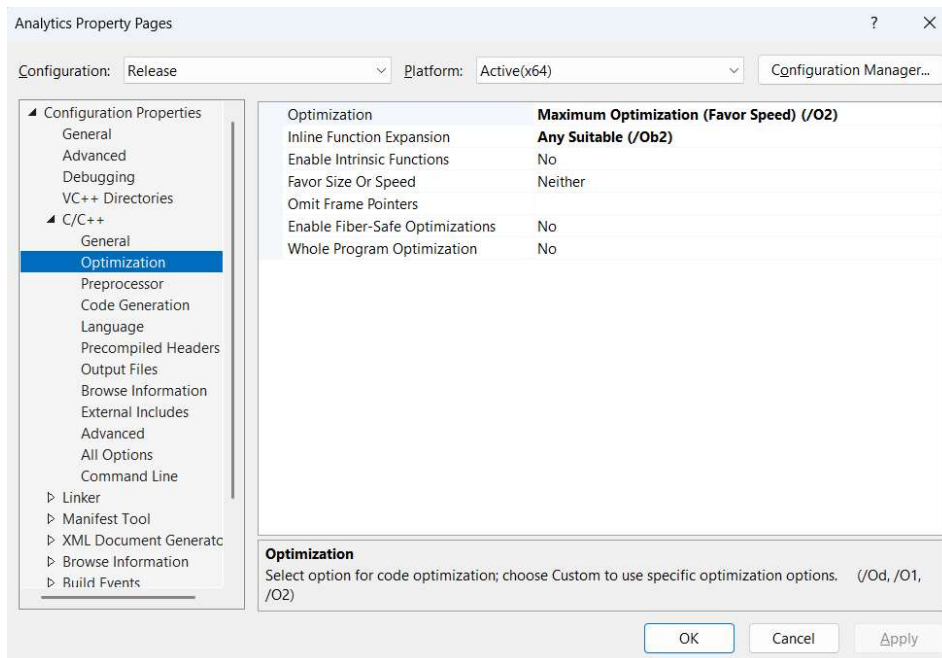


## File Path Macros [TOP TIP]

- Click the down arrow on any directory folder, then in the window pop-up press the **“Macros”** button
- View existing file path variables (macros) and/or add new ones e.g. \$(SolutionDir), \$(ProjectDir), ...

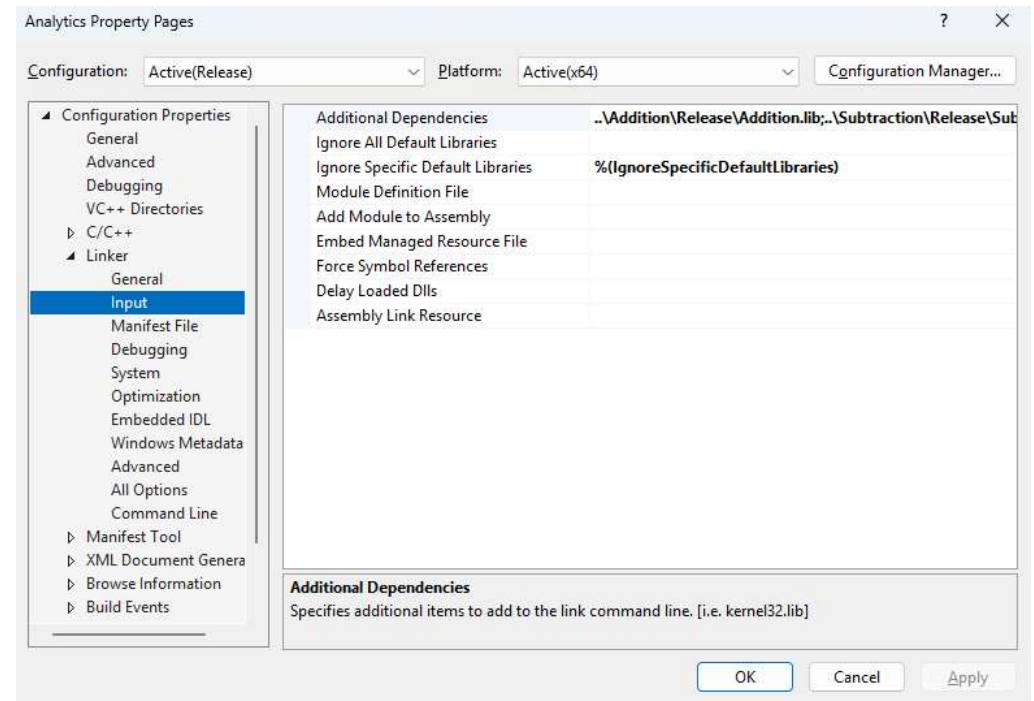
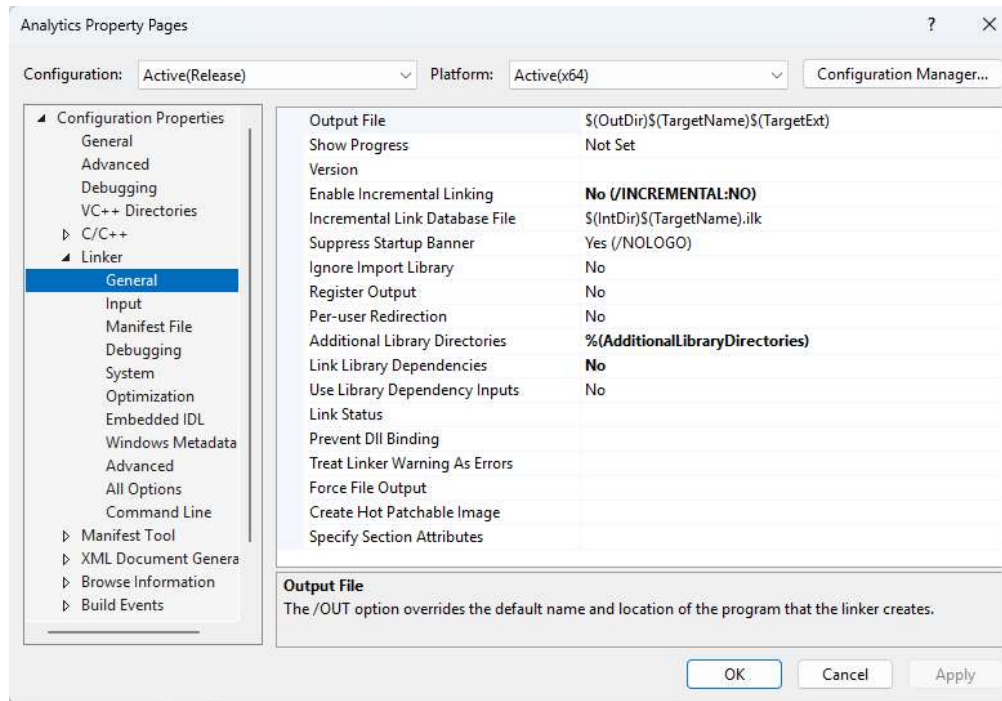


# VS Project Properties – C/C++ Compiler





# VS Project Properties – Linker/Librarian



# Summary – Key Project Properties



## ➤ General

- **Output Directory** – Specify output path
- **Configuration Type** – Specify the output file type .lib, .exe or .dll
- **C++ Language Standard** – C++14, C++17, C++20 ...

## ➤ C/C++ → General

- **Additional Include Directories** – To link projects, add include folder(s) here
- **Debug Information Format** – Edit and Continue (/ZI) this allows us to make minor modifications without rebuilding the project
- **Multi-processor Compilation (Yes /MP)** – allows parallel building of .cpp files

## ➤ C/C++ → Code Generation

- **Enable C++ Exceptions** – /EHsc allows structured exception handling and helps prevent crashes
- **Runtime Library** – Here we must specify dynamic or static linking of CRT (/MD or /MT), defaults to /MD

## ➤ Linker → General:

- **Additional Library Directories** - To link projects, add path to .lib files here

## ➤ Linker → Input:

- **Additional Dependencies** – To link projects, specify .lib path here

## ➤ Linker → Debugging

- **Generate Debug Info** – To test and debug a release project select /DEBUG

# C++



Algorithmic Trading & Quant Research Hub

 **YouTube**



[More Info ...](#)





# CMake Cross-Platform Build System

## CMake – What it is and what it does

- A **cross-platform** build system – not a compiler
- It uses platform-independent configuration files, **CMakeLists.txt**
- Generates native build files e.g. Visual Studio Solutions, Linux Make files, Ninja files, macOS Xcode projects
- Available as part of Visual Studio, see Tools -> Command line -> Developer Command Prompt

## How to generate a solution File for Visual Studio?

- Create the necessary CMakeLists.txt files
- Open Visual Studio command line and type:

```
cmake -G "Visual Studio 17 2022" <path-to-project-root>
```



# CMake Config Files – CMakeLists.txt

## CMake Commands for CMakeLists.txt

### ➤ Solution Config File

- Name the solution file ([project](#))
- Specify what projects to include ([add\\_subdirectory](#))

### ➤ Project Config Files

- Name the project ([project](#))
- Specify the project type and list the headers and source files to include ([add\\_library](#) | [add\\_executable](#))
- Provide the path to the include folder(s) and header files ([target\\_include\\_directories](#))
- List any dependency projects to include ([target\\_link\\_libraries](#))

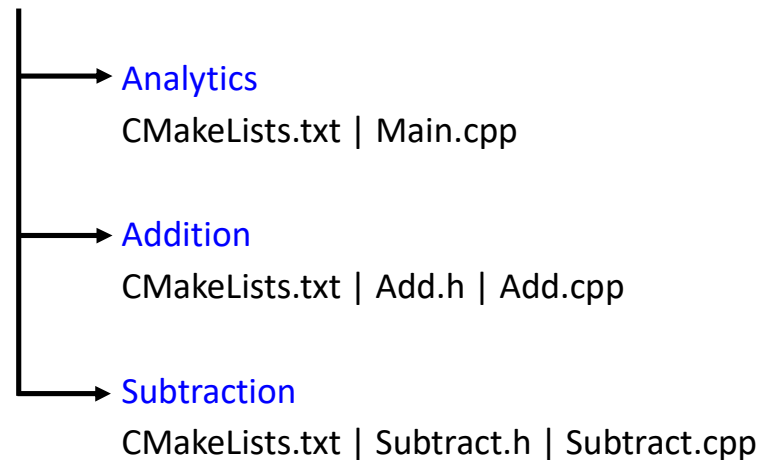


# Example: Create Solution File

- Consider a simple C++ maths library where the main project is called **Analytics** that depends on two projects named **Addition** and **Subtraction**. The folder structure looks as follows,

MathLibrary (Root Folder)

CMakeLists.txt



- The solution root folder and each project folder requires a **CMakeLists.txt** config file
- The config file defines the **project type** and specifies the **include paths** and **project dependencies**



# Solution Config File, CMakeLists.txt

```
1  cmake_minimum_required(VERSION 3.20)
2
3  project(MathLibrary LANGUAGES CXX)
4
5  # ---- Language standard ----
6  set(CMAKE_CXX_STANDARD 17)
7  set(CMAKE_CXX_STANDARD_REQUIRED ON)
8
9  # ---- Targets ----
10 add_subdirectory(Addition)
11 add_subdirectory(Subtraction)
12 add_subdirectory(Analytics)
```

- **project** – Name of the solution file
- **add\_subdirectory** – List project folders to include

C++

C++

C++

C++

# Main Project Config File, CMakeLists.txt

```
1  add_executable (Analytics
2      main.cpp
3  )
4
5  target_link_libraries (Analytics
6      PRIVATE
7          Addition
8          Subtraction
9  )
```

## ➤ add\_executable

- Creates project that outputs an executable called Analytics.
- List all the .h and .cpp files to include.

## ➤ target\_link\_libraries

- List the project name then the dependency projects to include
- Here we add the addition and subtraction projects to the analytics project

# Dependency Project Config File, CMakeLists.txt

```
1  add_library(Addition STATIC
2      add.h
3      add.cpp
4  )
5
6  target_include_directories(Addition
7      PUBLIC
8          $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}>
9  )
```

## ➤ add\_library

- Creates a project named Addition. Use **STATIC** to generate a .lib and **SHARED** to generate a .dll
- List all the .h and .cpp files to include.

## ➤ target\_include\_directories

- List the include directories for the Addition project
- **\$(CMAKE\_CURRENT\_SOURCE\_DIR)** means use the current folder



# Generating the Visual Studio Solution File

How to generate the Visual Studio solution and project files?

- After creating the necessary CMakeLists.txt configuration files
- Open Visual Studio command prompt and navigate to the solution root folder
- Type **mkdir build** to create a folder called 'build'
- Navigate to the build folder **cd build**

```
cmake -G "Visual Studio 17 2022" <path-to-project-root>
```

- To generate the solution file type: **cmake -G "Visual Studio 17 2022" ..**
- Note ".." means the root project is up one folder level

How to generate the native build projects for Linux, macOS and other platforms and compilers?

- Change the name of the compiler from "**Visual Studio 17 2022**" to the compiler of your choice
- Examples: For Linux "**Unix Makefiles**" or "**Ninja**" and for macOS use "**Xcode**"



# C++



Algorithmic Trading & Quant Research Hub

 **YouTube**



CMake Resources



# Getting Started with CMake

## Professional C++ with CMake

- Outlines how professional Quants use CMake
- Includes canonical stylized working examples
- Intentionally simple and easy to follow

AlgoQuantHub Weekly Deep Dive

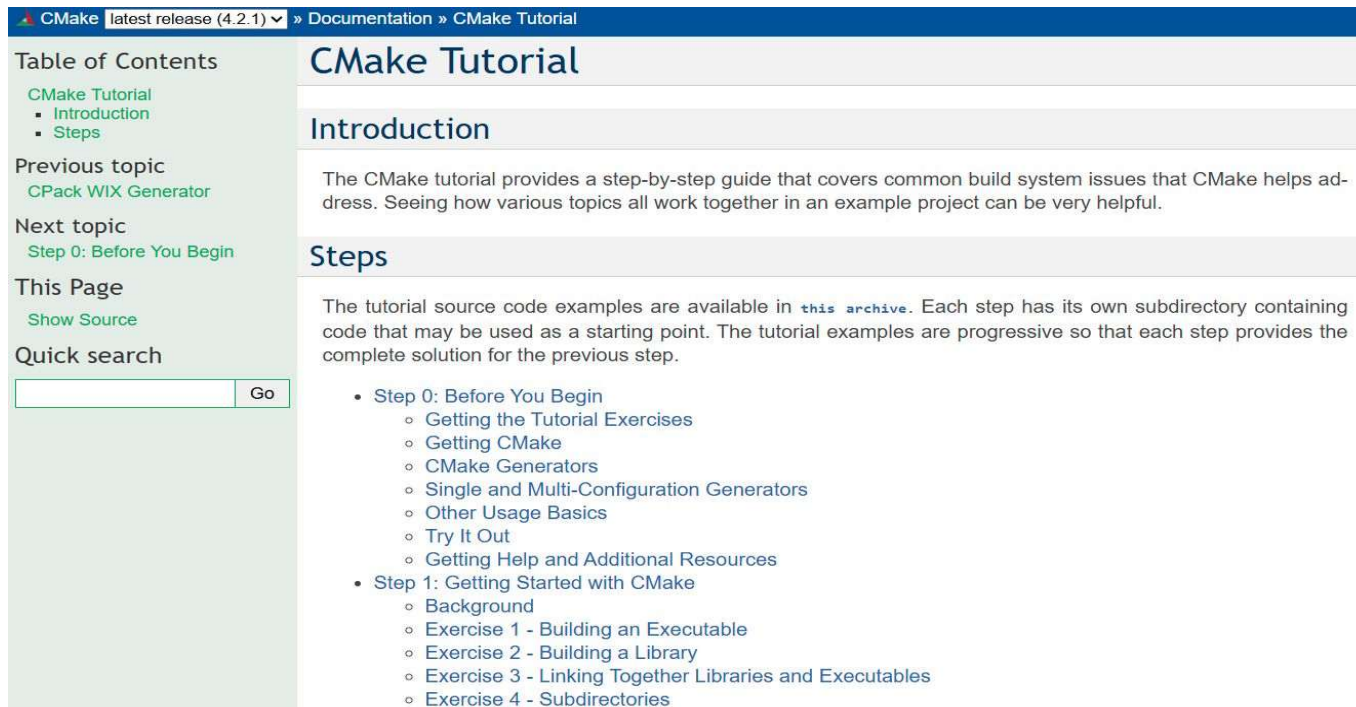


Professional C++ with CMake for  
Quants & Algo Trading

Link: <https://algoquanthub.beehiiv.com/p/professional-c-with-cmake-for-quants-algo-trading>

Examples: <https://github.com/nburgessx/QuantResearch/tree/main/CMake%20Examples>

# CMake Tutorial – cmake.org



The screenshot shows the CMake Tutorial page on the cmake.org website. The page has a blue header with the CMake logo and the text "latest release (4.2.1) » Documentation » CMake Tutorial". The main content area is titled "CMake Tutorial" and "Introduction". The introduction text states: "The CMake tutorial provides a step-by-step guide that covers common build system issues that CMake helps address. Seeing how various topics all work together in an example project can be very helpful." Below the introduction is a section titled "Steps" which contains a list of links to the tutorial steps. The left sidebar contains a "Table of Contents" with links to "CMake Tutorial", "Introduction", and "Steps". It also has sections for "Previous topic" (CPack WIX Generator), "Next topic" (Step 0: Before You Begin), "This Page" (Show Source), and a "Quick search" box with a "Go" button.

CMake latest release (4.2.1) » Documentation » CMake Tutorial

## CMake Tutorial

### Introduction

The CMake tutorial provides a step-by-step guide that covers common build system issues that CMake helps address. Seeing how various topics all work together in an example project can be very helpful.

### Steps

The tutorial source code examples are available in [this archive](#). Each step has its own subdirectory containing code that may be used as a starting point. The tutorial examples are progressive so that each step provides the complete solution for the previous step.

- Step 0: Before You Begin
  - [Getting the Tutorial Exercises](#)
  - [Getting CMake](#)
  - [CMake Generators](#)
  - [Single and Multi-Configuration Generators](#)
  - [Other Usage Basics](#)
  - [Try It Out](#)
  - [Getting Help and Additional Resources](#)
- Step 1: Getting Started with CMake
  - [Background](#)
  - [Exercise 1 - Building an Executable](#)
  - [Exercise 2 - Building a Library](#)
  - [Exercise 3 - Linking Together Libraries and Executables](#)
  - [Exercise 4 - Subdirectories](#)

**Table of Contents**

- CMake Tutorial
  - [Introduction](#)
  - [Steps](#)

**Previous topic**  
[CPack WIX Generator](#)

**Next topic**  
[Step 0: Before You Begin](#)

**This Page**  
[Show Source](#)

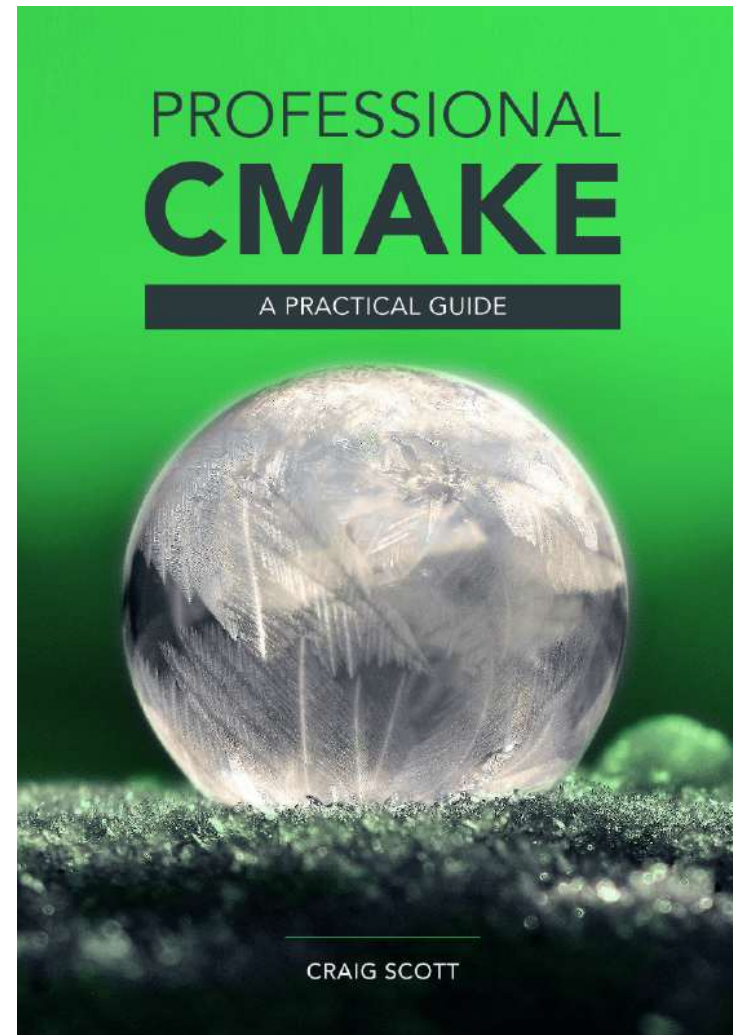
**Quick search**

## ➤ CMake Tutorial – cmake.org

- Provides a step-by-step guides and tutorials on how to use CMake
- Link: <https://cmake.org/cmake/help/book/mastering-cmake/cmake/Help/guide/tutorial/index.html>

# Professional CMake

- Professional CMake – A Practical Guide
  - Free Book
  - By Craig Scott
  - <https://crascit.com/professional-cmake/>





# C++



Algorithmic Trading & Quant Research Hub

 **YouTube**



[More Info ...](#)







Subscribe to my Quant Newsletter

<https://algoquanthub.beehiiv.com/subscribe>

Subscribe to my Quant YouTube Channel

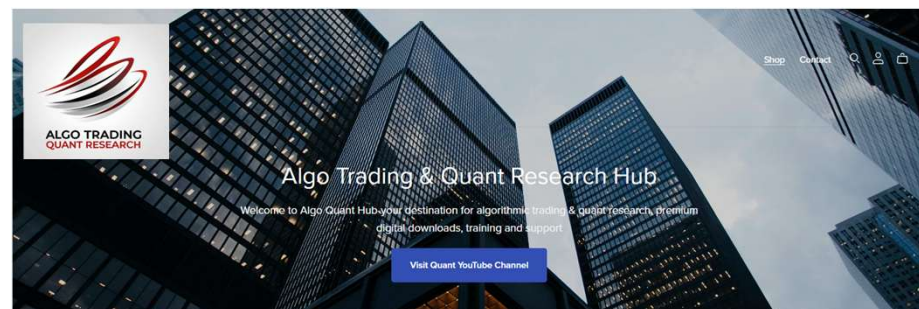
<https://www.youtube.com/@algoquanthub>

Algo Trading & Quant Store

<https://payhip.com/AlgoQuantHub>

Follow me on LinkedIn

<https://www.linkedin.com/in/nburgessx>



Have questions or want further info?

## Contact

LinkedIn: [www.linkedin.com/in/nburgessx](https://www.linkedin.com/in/nburgessx)