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

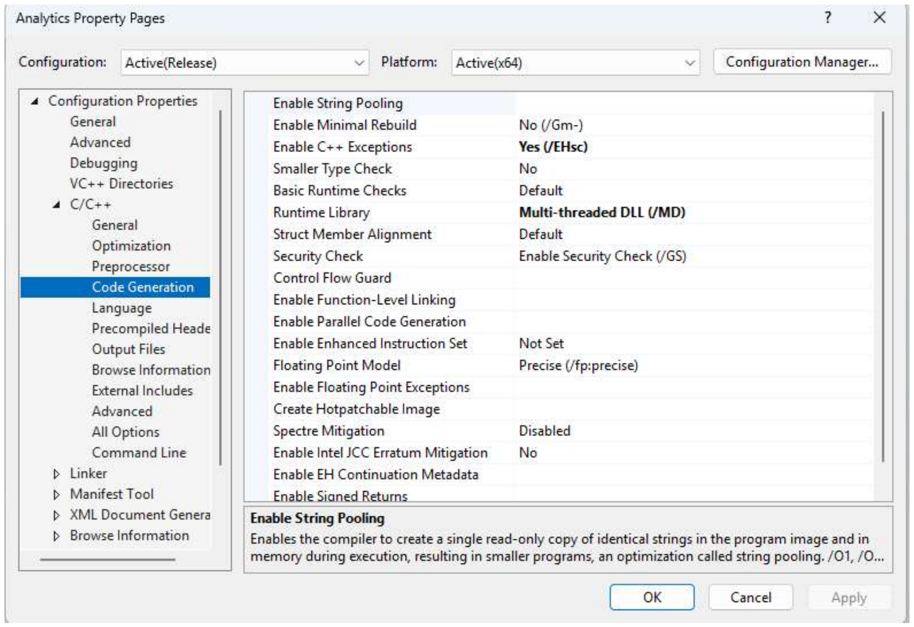
ALGO TRADING
QUANT RESEARCH

Example: Visual Studio & CMake
https://github.com/nburgessx/QuantResearch/tree/main/CMake%20Examples

Algorithmic Trading & Quant Research Hub
YouTube

Application Binary Interface (ABI)

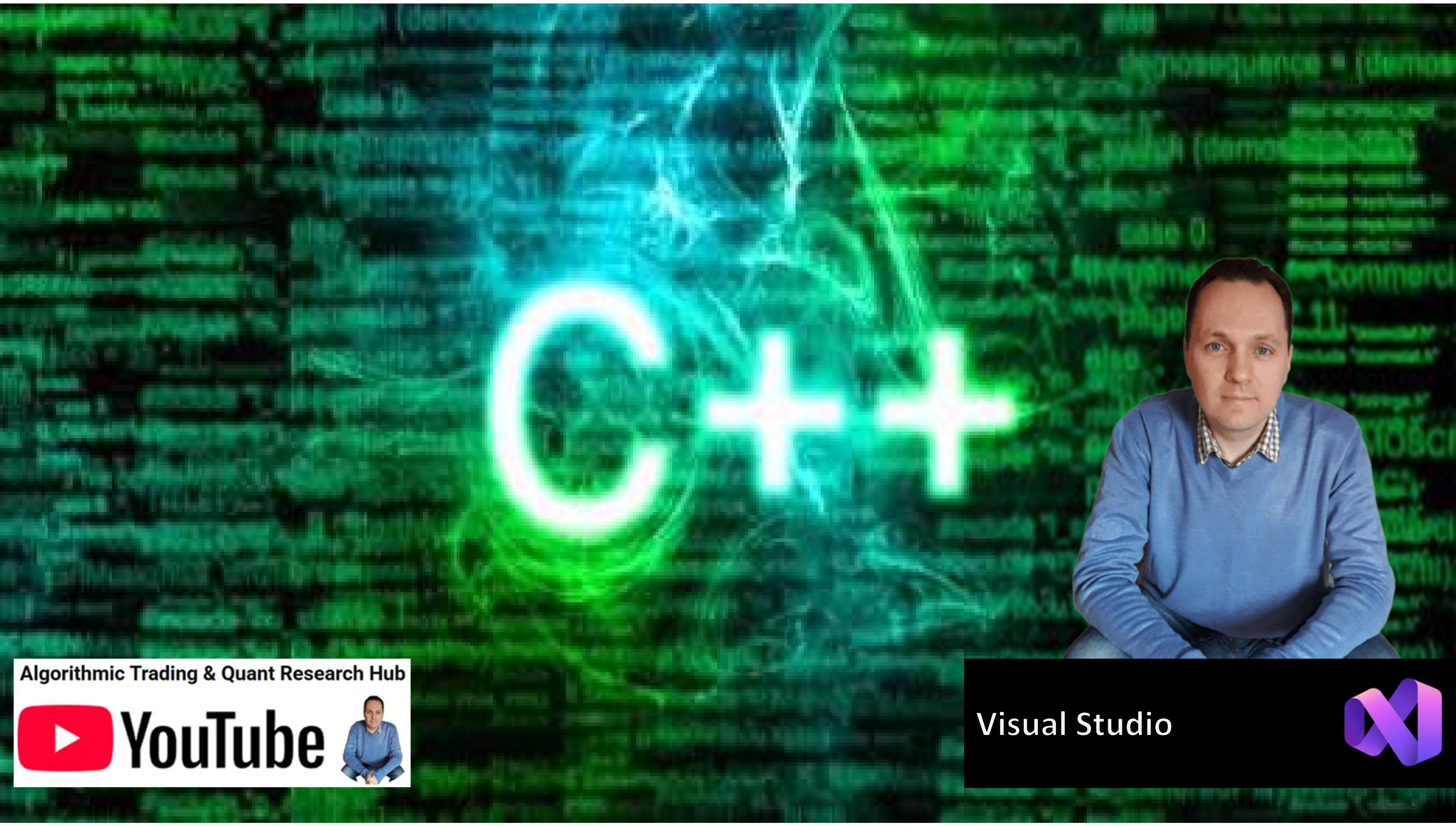
- ABI defines how project binaries are linked and how they manage memory
- Projects sharing runtime resources e.g. std::vector or FILE* **must** use the same C++ Runtime library (CRT), which handles memory, I/O and startup support
- Dynamic Linkage **(/MD)** links against a shared C++ Runtime DLL (CRT)
- 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



Enable String Pooling	
Enable Minimal Rebuild	No (/Gm-)
Enable C++ Exceptions	**Yes (/EHsc)**
Smaller Type Check	No
Basic Runtime Checks	Default
Runtime Library	**Multi-threaded DLL (/MD)**
Struct Member Alignment	Default
Security Check	Enable Security Check (/GS)
Control Flow Guard	
Enable Function-Level Linking	
Enable Parallel Code Generation	
Enable Enhanced Instruction Set	Not Set
Floating Point Model	Precise (/fp:precise)
Enable Floating Point Exceptions	
Create Hotpatchable Image	
Spectre Mitigation	Disabled
Enable Intel JCC Erratum Mitigation	No
Enable EH Continuation Metadata	
Enable Signed Returns	

Enable String Pooling
Enables the compiler to create a single read-only copy of identical strings in the program image and in memory during execution, resulting in smaller programs, an optimization called string pooling. /O1, /O...

Rule of thumb:
/MD → DLLs & large apps (shared runtime, one-heap)
/MT → Fully self-contained tools (no shared ownership)

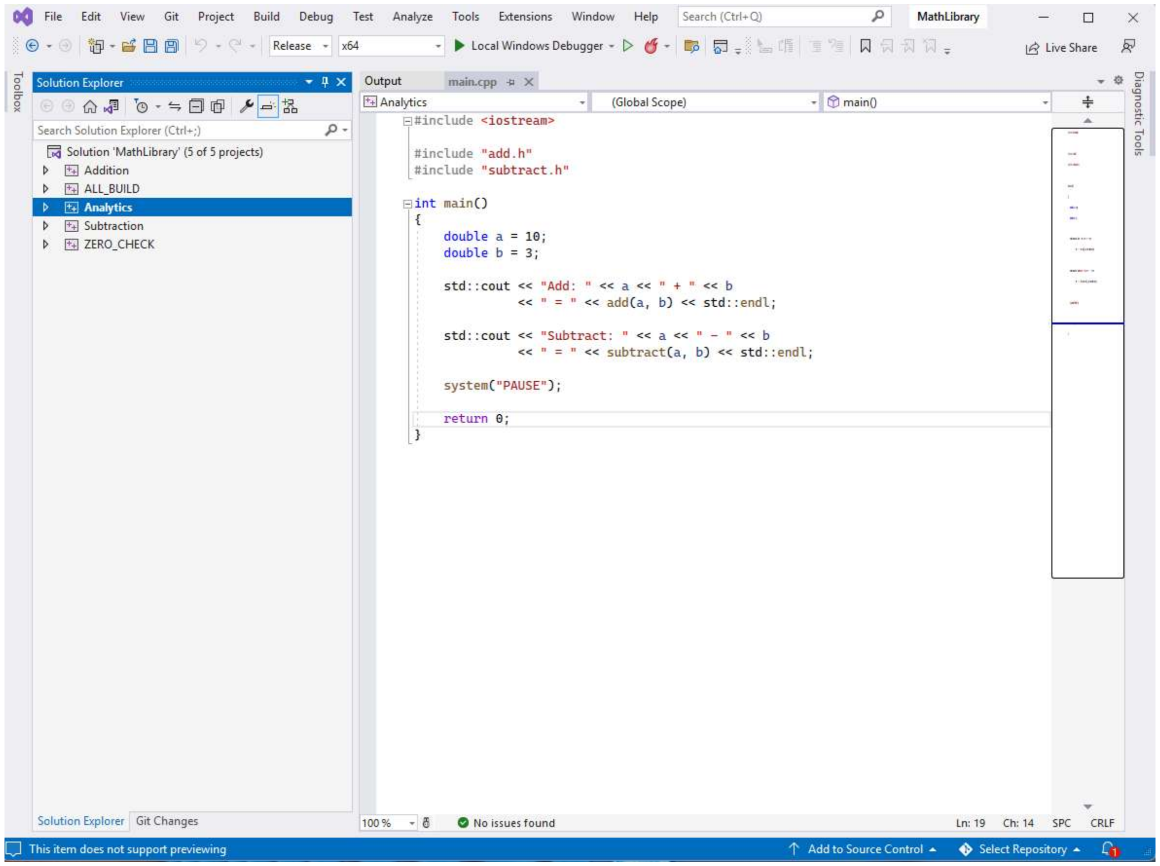

C++
Algorithmic Trading & Quant Research Hub
YouTube
Visual Studio

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



Release | x64

Solution 'MathLibrary' (5 of 5 projects)
- Addition
- ALL_BUILD
- Analytics
- Subtraction
- ZERO_CHECK

Analytics

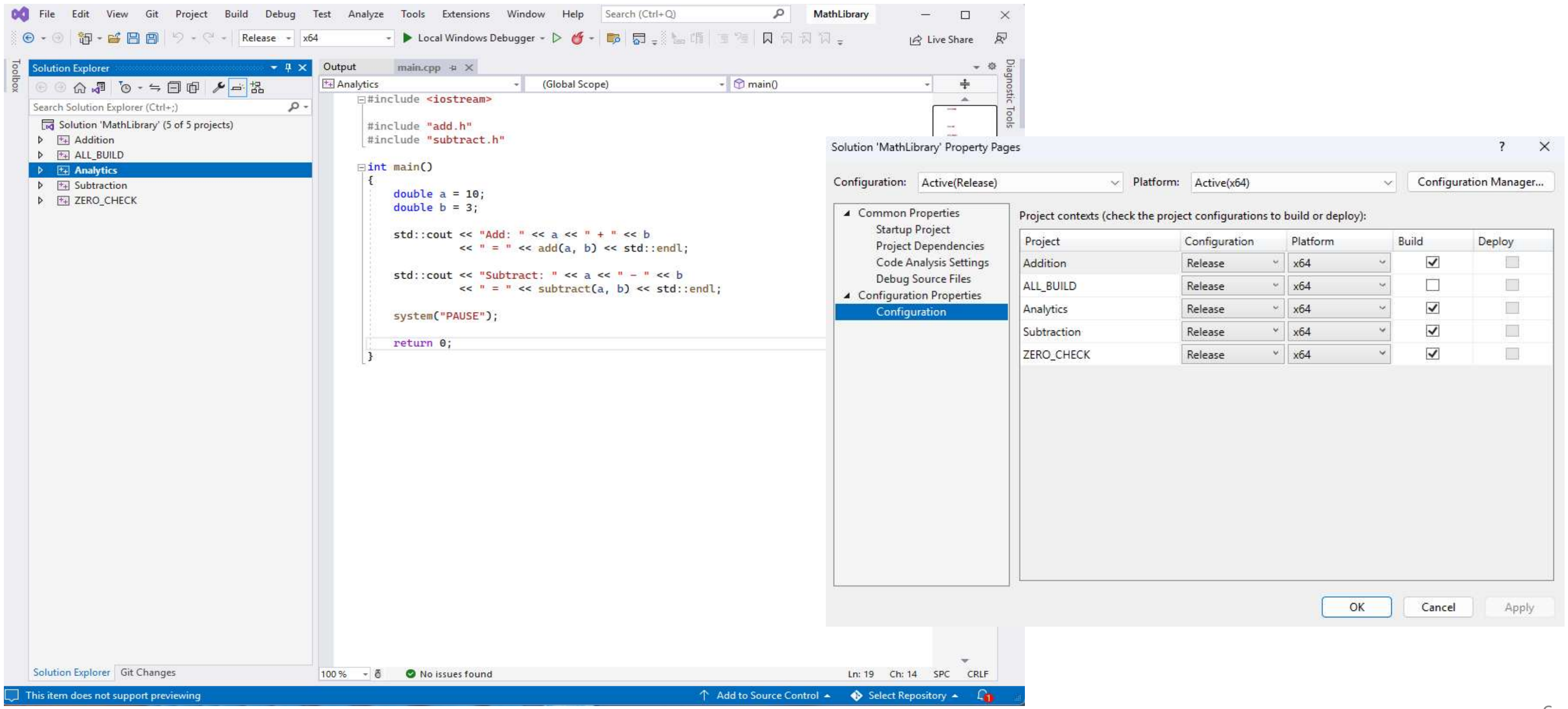
```cpp
#include <iostream>

#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;
}
```

No issues found

This item does not support previewing

Visual Studio Solution & Projects Files

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



Solution 'MathLibrary' (5 of 5 projects)
- Addition
- ALL_BUILD
- **Analytics**
- Subtraction
- ZERO_CHECK

```
#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;
}
```

Solution 'MathLibrary' Property Pages

Configuration: Active(Release)   Platform: Active(x64)   Configuration Manager...

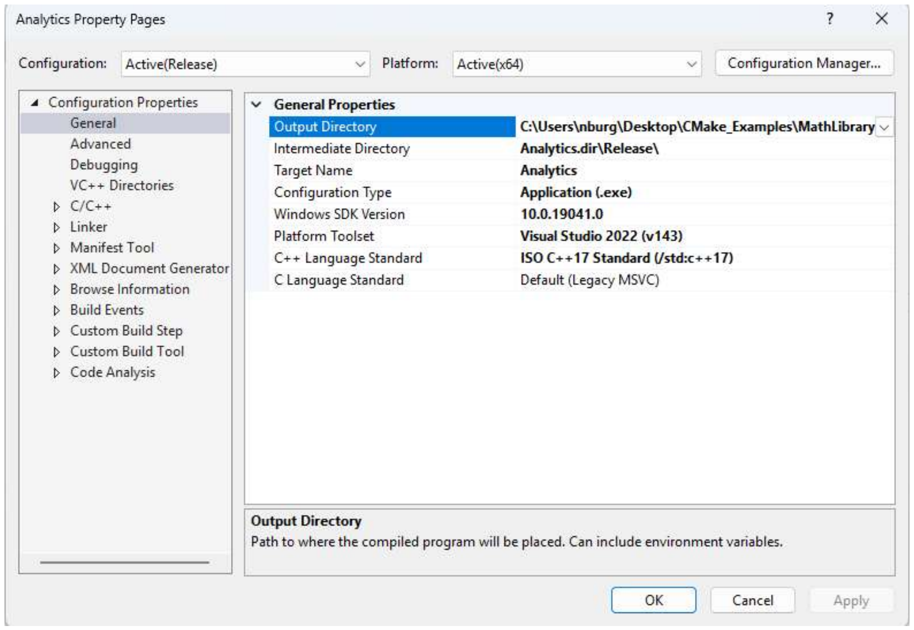
- Common Properties
  - Startup Project
  - Project Dependencies
  - Code Analysis Settings
  - Debug Source Files
- Configuration Properties
  - Configuration

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build	Deploy
Addition	Release	x64	✓	
ALL_BUILD	Release	x64	☐	
Analytics	Release	x64	✓	
Subtraction	Release	x64	✓	
ZERO_CHECK	Release	x64	✓	

OK   Cancel   Apply

Visual Studio Project Properties



General Properties

Output Directory	C:\Users\nburg\Desktop\CMake_Examples\MathLibrary
Intermediate Directory	Analytics.dir\Release\
Target Name	Analytics
Configuration Type	Application (.exe)
Windows SDK Version	10.0.19041.0
Platform Toolset	Visual Studio 2022 (v143)
C++ Language Standard	ISO C++17 Standard (/std:c++17)
C Language Standard	Default (Legacy MSVC)

Output Directory
Path to where the compiled program will be placed. Can include environment variables.

Output type

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

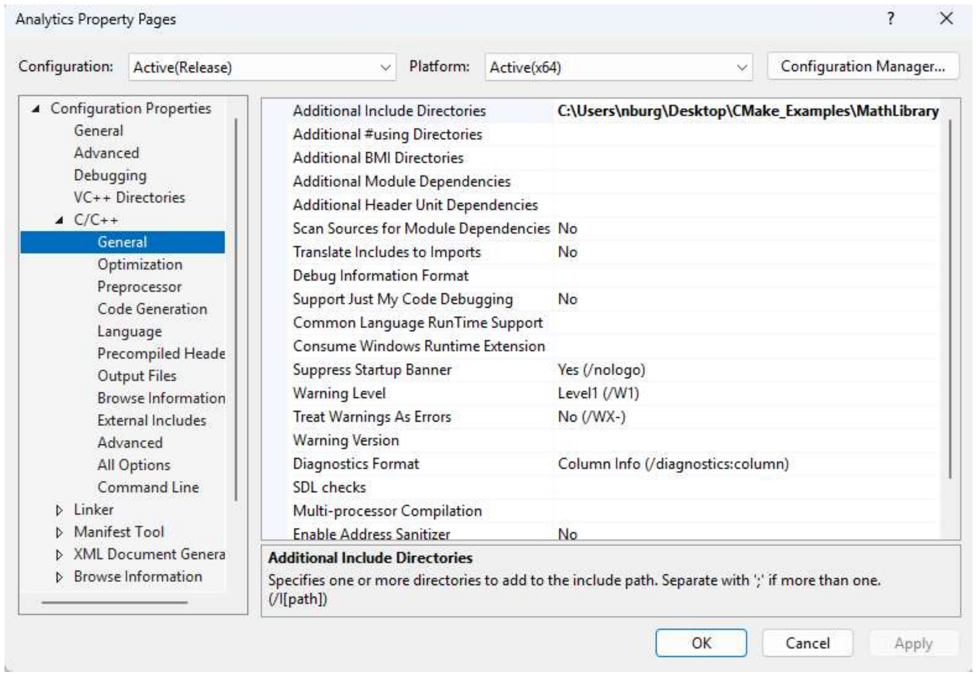
Where outputs go

- 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



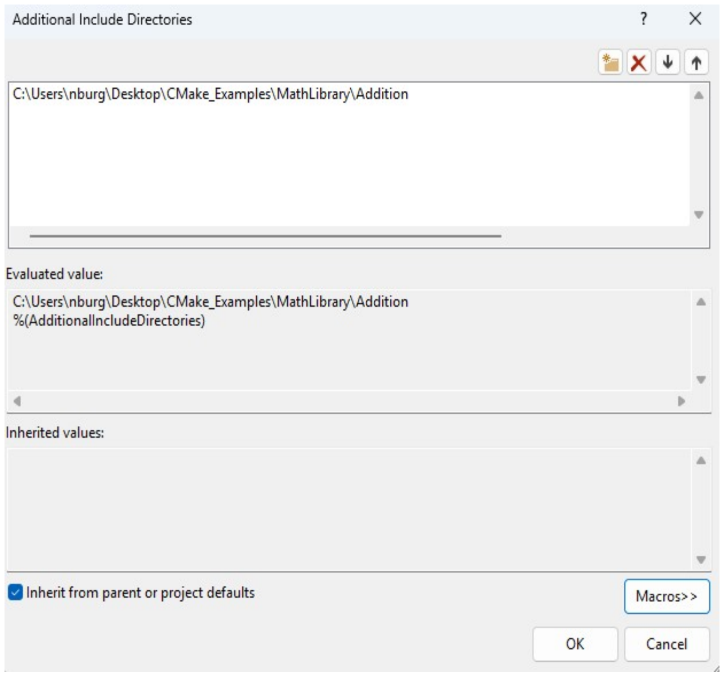
Analytics Property Pages

Configuration: Active(Release) Platform: Active(x64) Configuration Manager...

Additional Include Directories	**C:\Users\nburg\Desktop\CMake_Examples\MathLibrary**
Additional #using Directories	
Additional BMI Directories	
Additional Module Dependencies	
Additional Header Unit Dependencies	
Scan Sources for Module Dependencies	No
Translate Includes to Imports	No
Debug Information Format	
Support Just My Code Debugging	No
Common Language RunTime Support	
Consume Windows Runtime Extension	
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level1 (/W1)
Treat Warnings As Errors	No (/WX-)
Warning Version	
Diagnostics Format	Column Info (/diagnostics:column)
SDL checks	
Multi-processor Compilation	
Enable Address Sanitizer	No

Additional Include Directories

Specifies one or more directories to add to the include path. Separate with ';' if more than one. (/I[path])



Additional Include Directories

C:\Users\nburg\Desktop\CMake_Examples\MathLibrary\Addition

Evaluated value:

C:\Users\nburg\Desktop\CMake_Examples\MathLibrary\Addition
%(AdditionalIncludeDirectories)

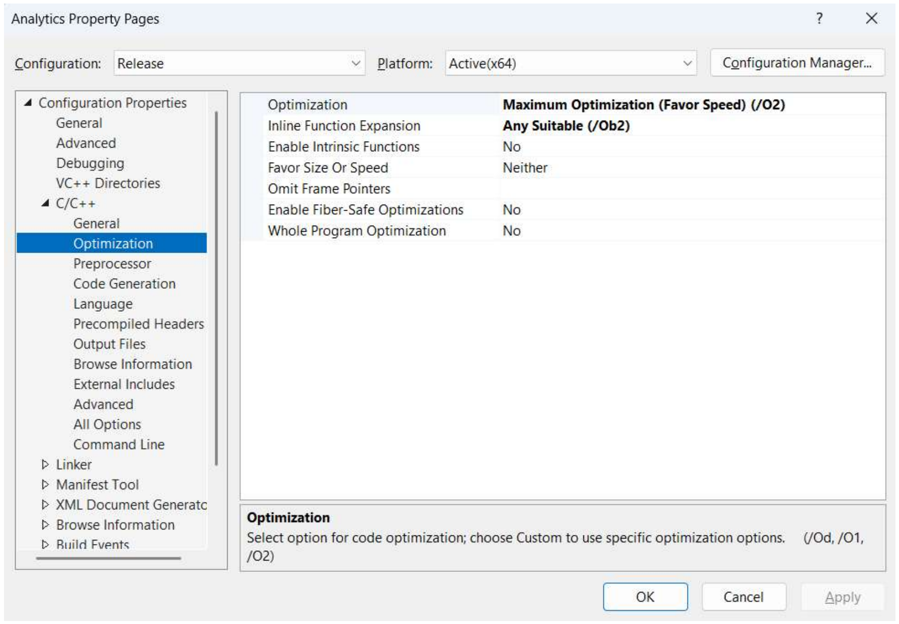
Inherited values:

☑ Inherit from parent or project defaults

File Path Macros [TOP TIP]

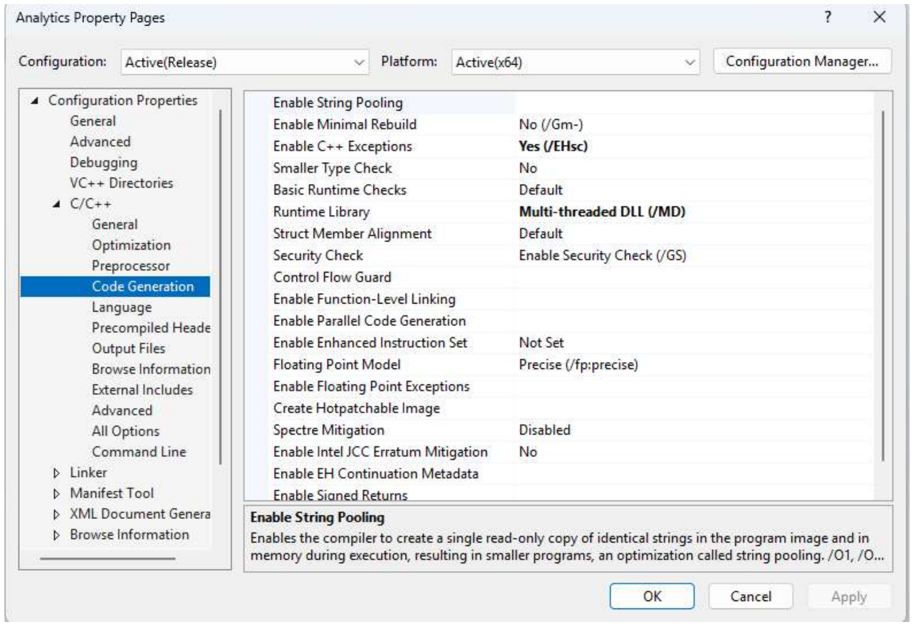
- Click the down arrow on any directory folder, then in the window pop-up press the "**Macros**" button
- View existing file path variables (macros) and/or add new ones e.g. $(SolutionDir), $(ProjectDir), …

VS Project Properties – C/C++ Compiler



Optimization	**Maximum Optimization (Favor Speed) (/O2)**
Inline Function Expansion	**Any Suitable (/Ob2)**
Enable Intrinsic Functions	No
Favor Size Or Speed	Neither
Omit Frame Pointers	
Enable Fiber-Safe Optimizations	No
Whole Program Optimization	No

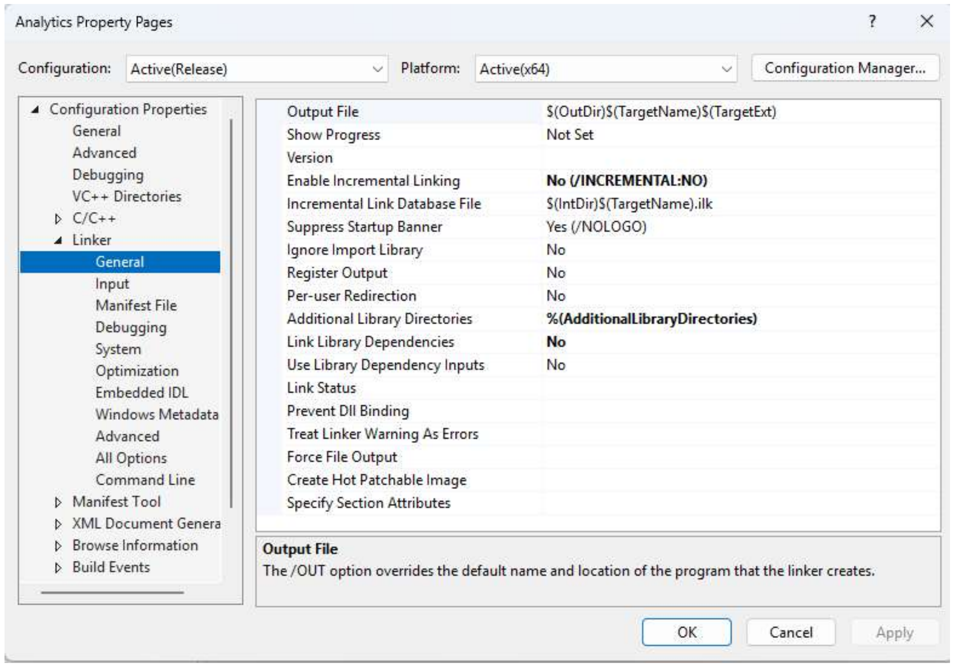
Optimization
Select option for code optimization; choose Custom to use specific optimization options. (/Od, /O1, /O2)



Enable String Pooling	
Enable Minimal Rebuild	No (/Gm-)
Enable C++ Exceptions	**Yes (/EHsc)**
Smaller Type Check	No
Basic Runtime Checks	Default
Runtime Library	**Multi-threaded DLL (/MD)**
Struct Member Alignment	Default
Security Check	Enable Security Check (/GS)
Control Flow Guard	
Enable Function-Level Linking	
Enable Parallel Code Generation	
Enable Enhanced Instruction Set	Not Set
Floating Point Model	Precise (/fp:precise)
Enable Floating Point Exceptions	
Create Hotpatchable Image	
Spectre Mitigation	Disabled
Enable Intel JCC Erratum Mitigation	No
Enable EH Continuation Metadata	
Enable Signed Returns	

Enable String Pooling
Enables the compiler to create a single read-only copy of identical strings in the program image and in memory during execution, resulting in smaller programs, an optimization called string pooling. /O1, /O...

VS Project Properties – Linker/Librarian



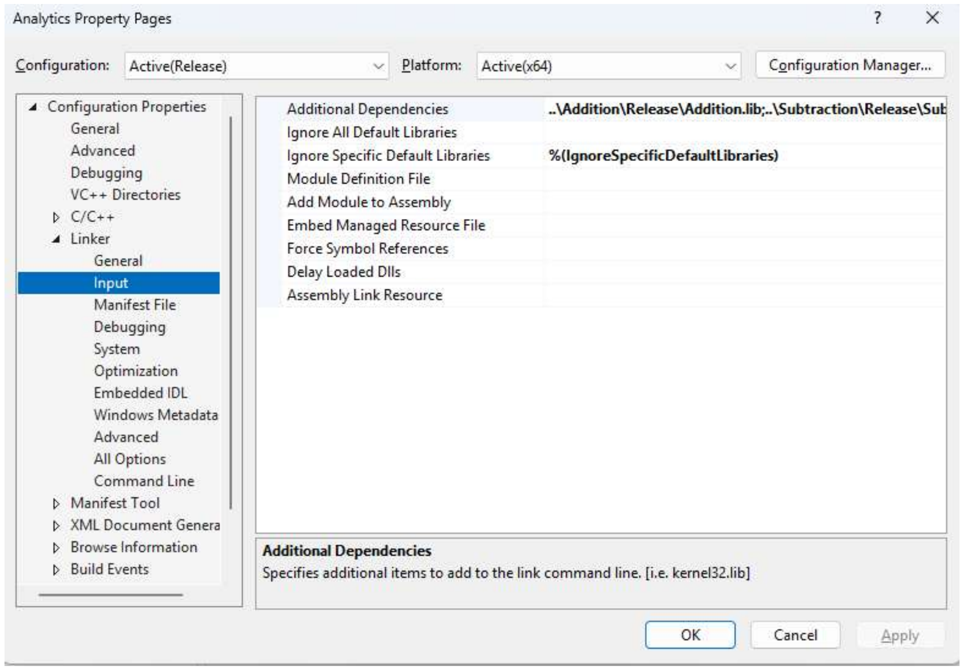
Analytics Property Pages

Configuration: Active(Release) Platform: Active(x64) Configuration Manager...

Configuration Properties > Linker > General

Output File	$(OutDir)$(TargetName)$(TargetExt)
Show Progress	Not Set
Version	
Enable Incremental Linking	**No (/INCREMENTAL:NO)**
Incremental Link Database File	$(IntDir)$(TargetName).ilk
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	**%(AdditionalLibraryDirectories)**
Link Library Dependencies	**No**
Use Library Dependency Inputs	No
Link Status	
Prevent Dll Binding	
Treat Linker Warning As Errors	
Force File Output	
Create Hot Patchable Image	
Specify Section Attributes	

Output File
The /OUT option overrides the default name and location of the program that the linker creates.



Analytics Property Pages

Configuration: Active(Release) Platform: Active(x64) Configuration Manager...

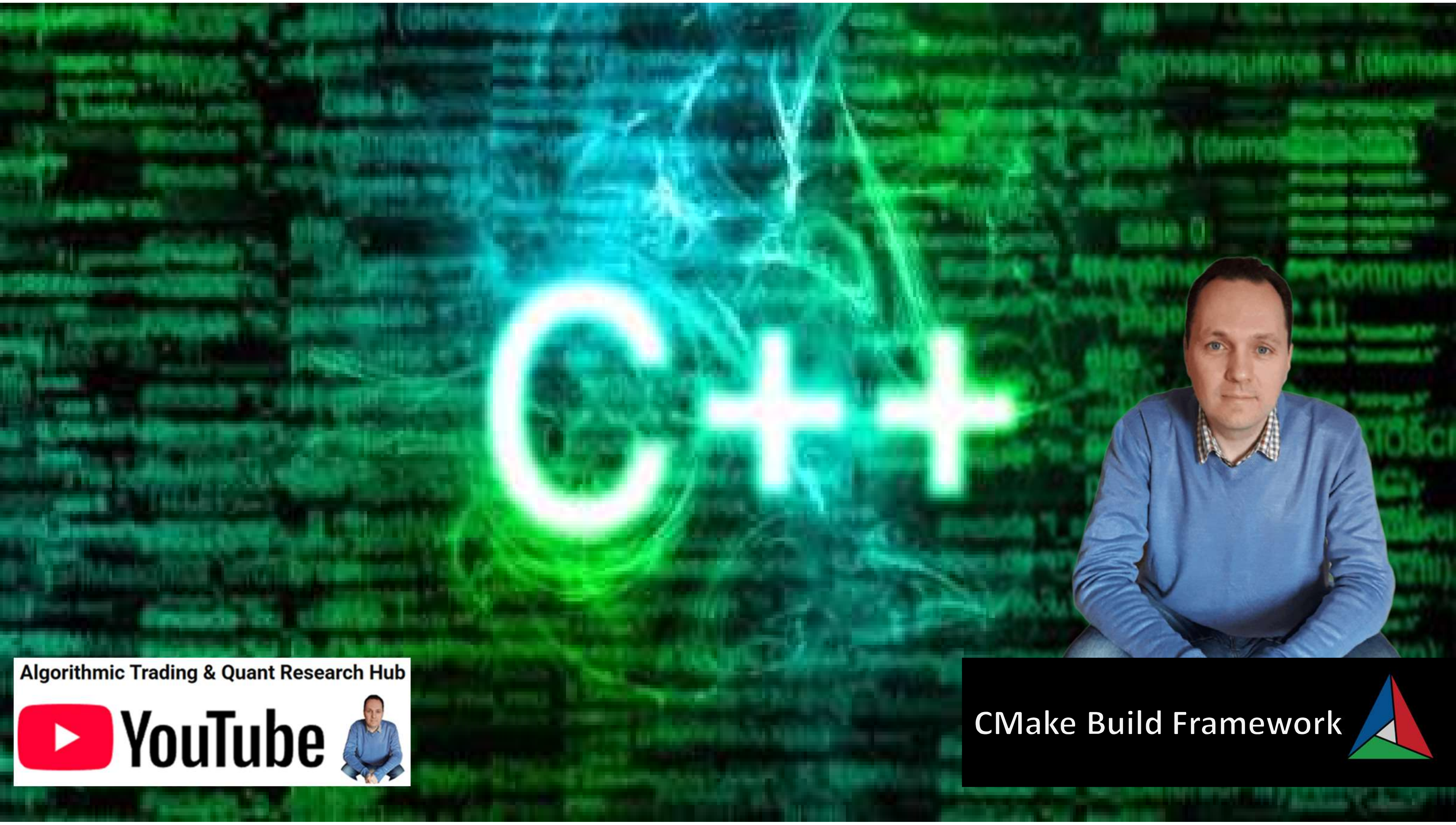
Configuration Properties > Linker > Input

Additional Dependencies	**..\Addition\Release\Addition.lib;..\Subtraction\Release\Sub**
Ignore All Default Libraries	
Ignore Specific Default Libraries	**%(IgnoreSpecificDefaultLibraries)**
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]

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 with out 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
CMake Build Framework

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 the solution File using 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

Creating CMakeLists.txt Files – A summary of main CMake commands

- **Solution Config File**
  - Name the solution file (project) and specify what projects to include (add_subdirectory)

- **Project Config Files**
  - Name the project and list the .h and .cpp files to include (add_library | add_executable)
  - We provide the path to the include folder(s) with our header files (target_include_directories)
  - List any dependency projects to include (target_link_libraries)

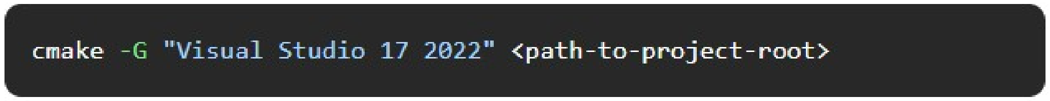
Example Cake Solution G

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 the solution File using 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>
```

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

Analytics

CMakeLists.txt | Main.cpp

Addition

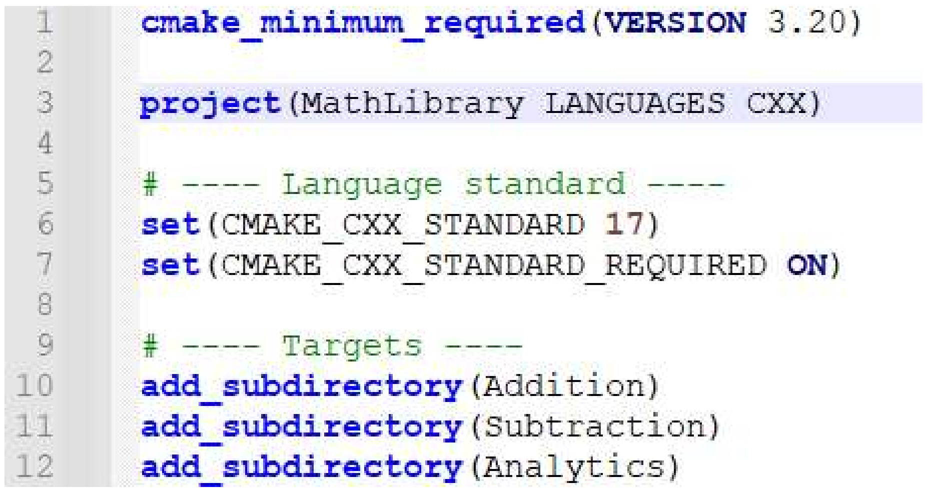
CMakeLists.txt | Add.h | Add.cpp

Subtraction

CMakeLists.txt | Subtract.h | Subtract.cpp

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

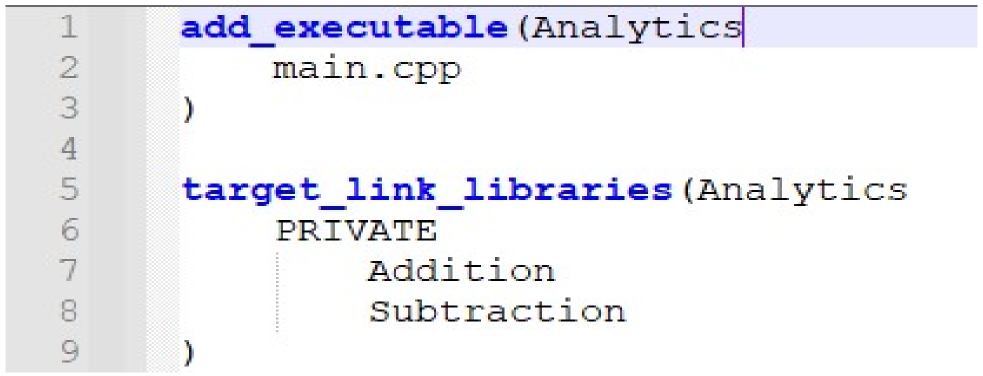
Solution Config File, CMakeLists.txt



```cmake
cmake_minimum_required(VERSION 3.20)

project(MathLibrary LANGUAGES CXX)

# ---- Language standard ----
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

# ---- Targets ----
add_subdirectory(Addition)
add_subdirectory(Subtraction)
add_subdirectory(Analytics)
```

- project – Name of the solution file
- add_subdirectory – List project folders to include

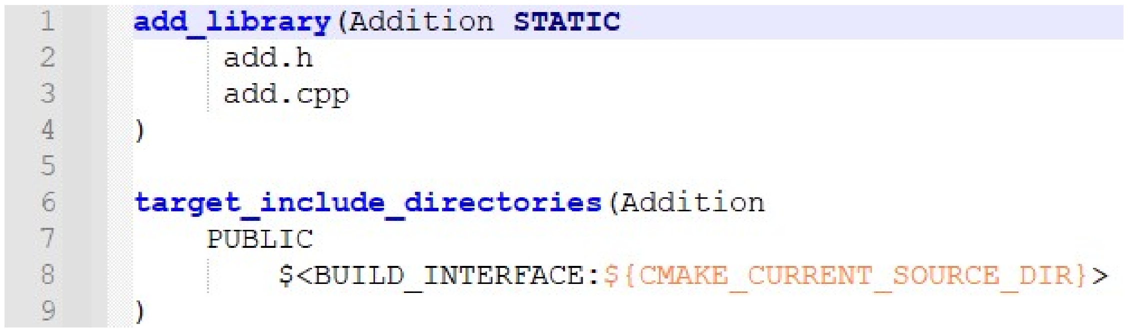
Main Project Config File, CMakeLists.txt



```
add_executable(Analytics
    main.cpp
)

target_link_libraries(Analytics
    PRIVATE
        Addition
        Subtraction
)
```

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



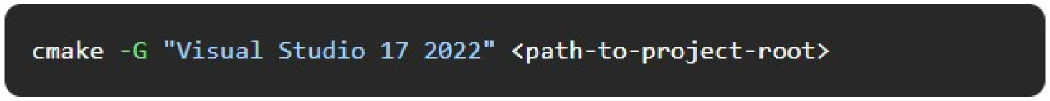
```
add_library(Addition STATIC
        add.h
        add.cpp
)

target_include_directories(Addition
        PUBLIC
            $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}>
)
```

- 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 solution File using Visual Studio?

- Create the necessary CMakeLists.txt 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 on non-windows 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**”

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

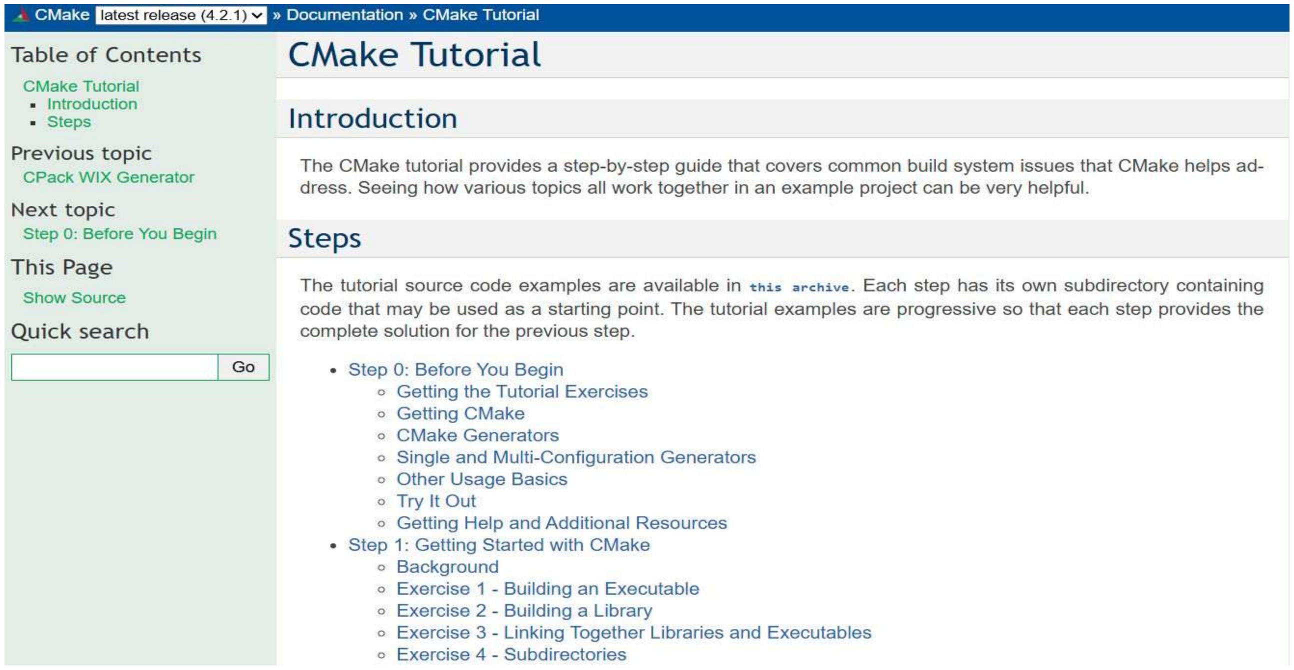
AlgoQuantHub Weekly Deep Dive

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



CMake latest release (4.2.1) » Documentation » CMake Tutorial

Table of Contents

Previous topic
CPack WIX Generator

Next topic
Step 0: Before You Begin

This Page
Show Source

Quick search

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.

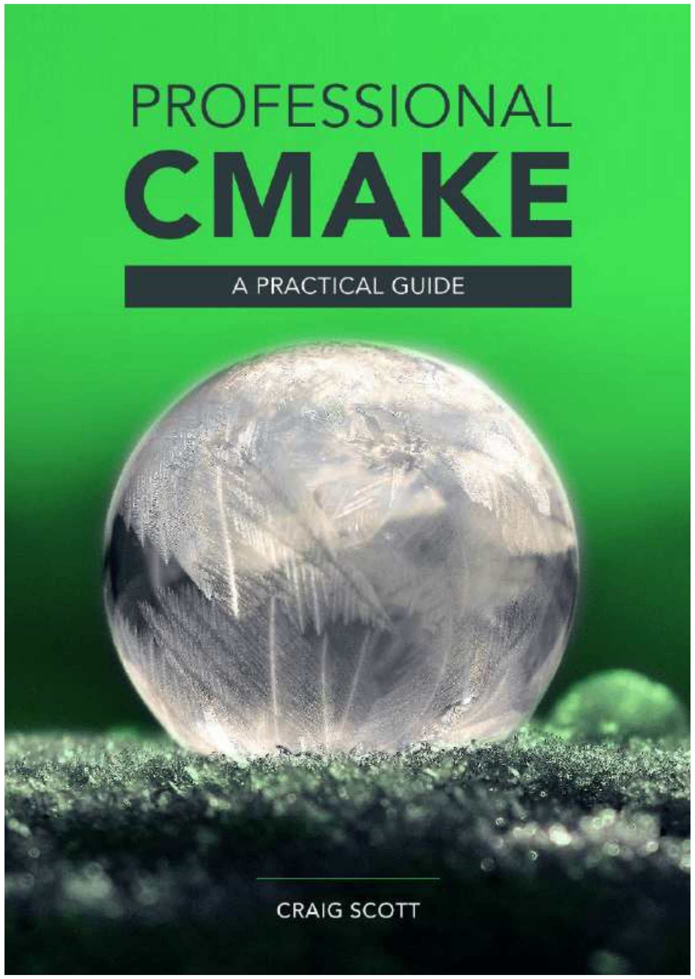
- Step 0: Before You Begin
  - Getting the Tutorial Exercises
  - Getting CMake
  - CMake Generators
  - Single and Multi-Configuration Generators
  - Other Usage Basics
  - Try It Out
  - Getting Help and Additional Resources
- Step 1: Getting Started with CMake
  - Background
  - Exercise 1 - Building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

➢ CMake Tutorial – cmake.org

- Provides a step-by-step guides and tutorials on how to use CMake

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 ...

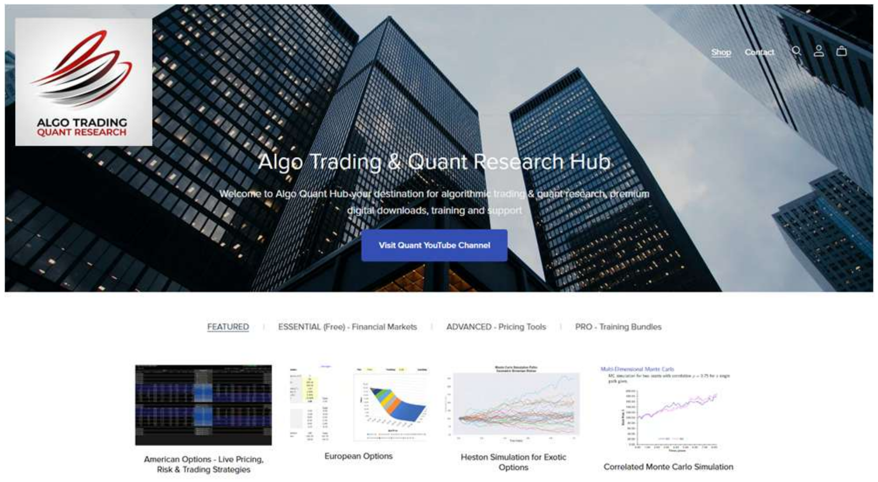
Algorithmic Trading & Quant Research Hub
@AlgoQuantHub ·
Algorithmic Trading & Quant Research .

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


ALGO TRADING
QUANT RESEARCH
Shop
Contact
Algo Trading & Quant Research Hub
Welcome to Algo Quant Hub-your destination for algorithmic trading & quant research, premium digital downloads, training and support
Visit Quant YouTube Channel
FEATURED
ESSENTIAL (Free) - Financial Markets
ADVANCED - Pricing Tools
PRO - Training Bundles
American Options - Live Pricing, Risk & Trading Strategies
European Options
Heston Simulation for Exotic Options
Correlated Monte Carlo Simulation

Have questions or want further info?

Contact

LinkedIn: www.linkedin.com/in/nburgessx