
C++
Algorithmic Trading & Quant Research Hub
YouTube
C++ Set-Up for Algo Quant Trading
By Nicholas Burgess

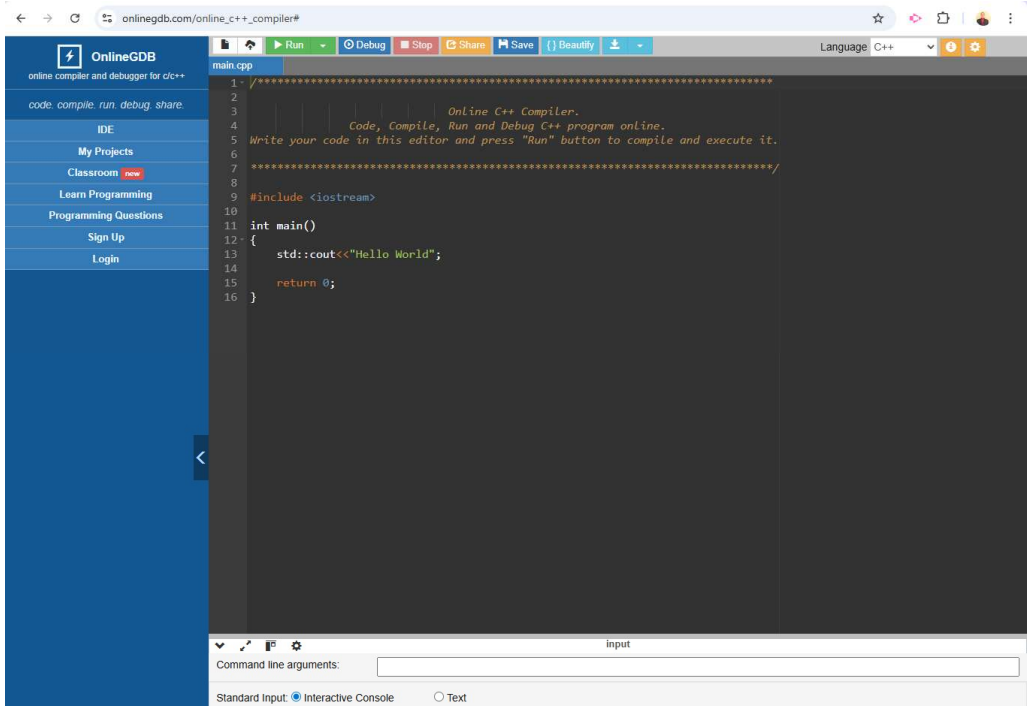
C++ Set-Up for Algo Quant Trading

- **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



Online C++ Compiler
https://www.onlinegdb.com/online_c++_compiler#

- 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

C++ Header and Source Files

1. Header Files (.h)

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

2. Source Files (.cpp)

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

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

h

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

cpp

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

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

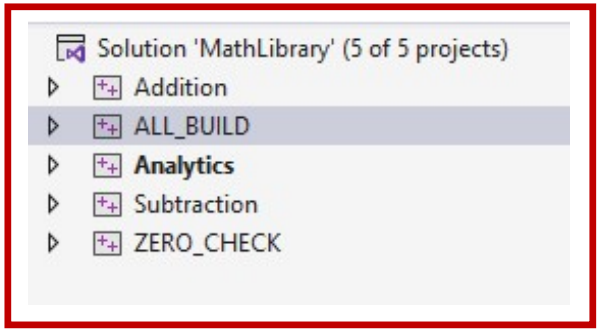
Connecting Projects

- 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?

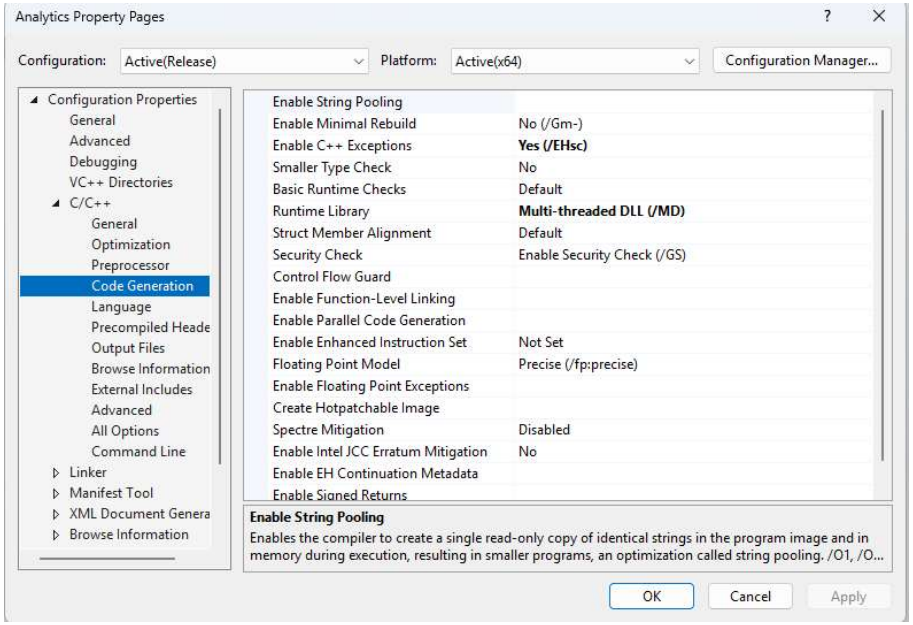
- **Headers** declare what exists (functions, classes, interfaces)
- **Libraries** contain how it's implemented (compiled machine code)

Sharing Projects

- Header file path(s)
- Library file path

Application Binary Interface (ABI)

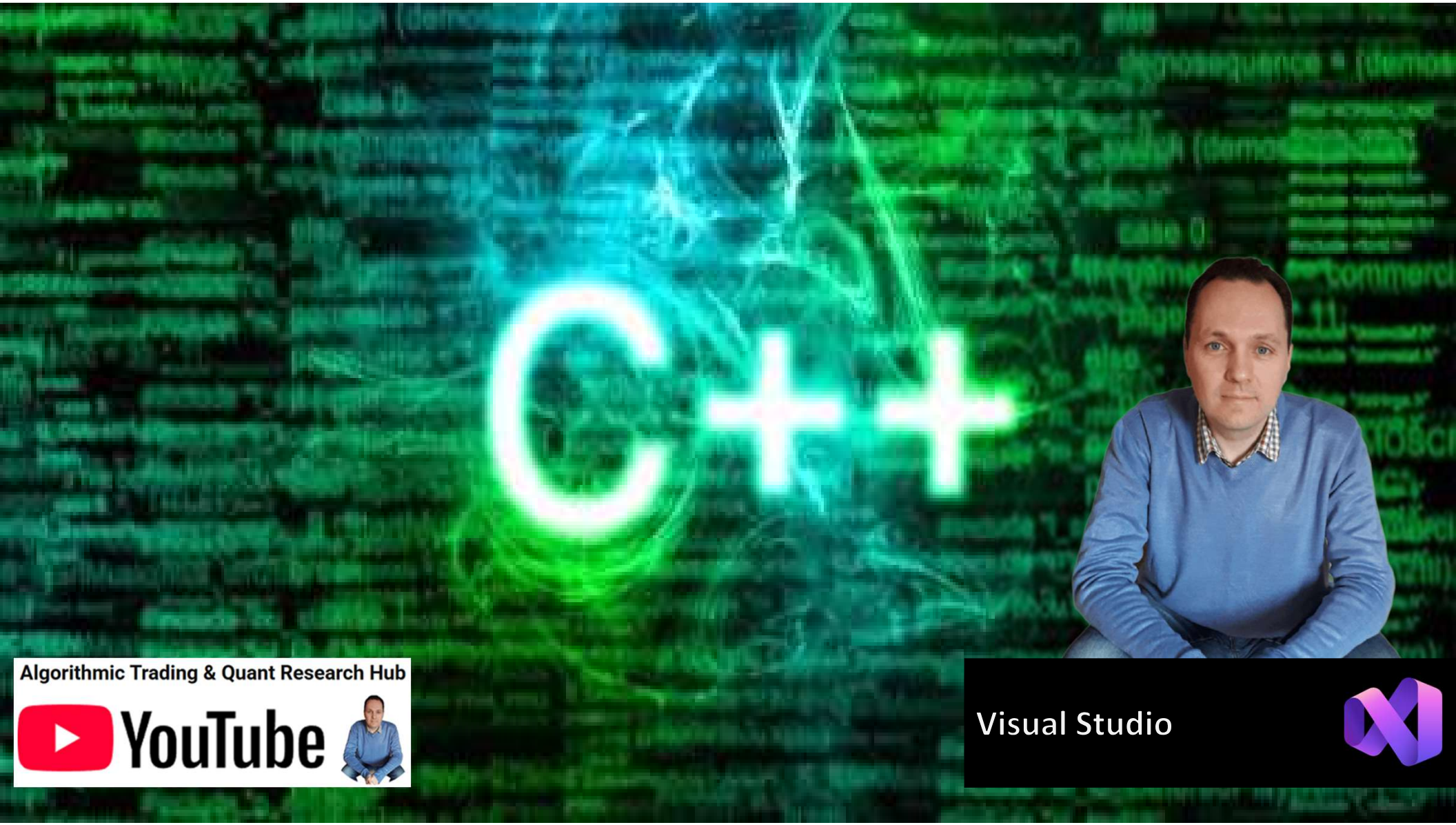
- ABI defines how project binaries are linked and how they manage memory
- Projects sharing runtime resources e.g. std::vector or FILE* **must** use the same C++ Runtime library (CRT), which handles memory, I/O and startup support
- Dynamic Linkage **(/MD)** links against a shared C++ Runtime DLL (CRT)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



Rule of thumb:

/MD → DLLs & large apps (shared runtime, one-heap)

/MT → Fully self-contained tools (no shared ownership)

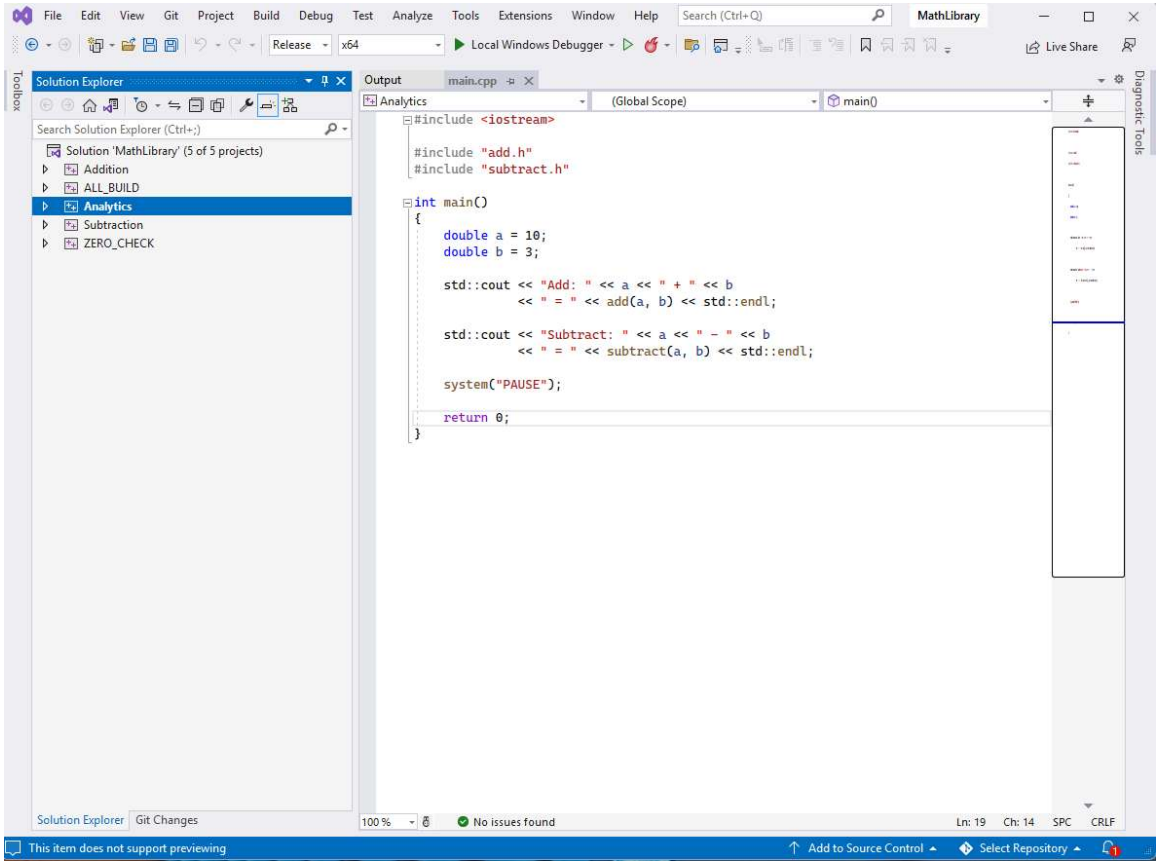

C++
Algorithmic Trading & Quant Research Hub
YouTube
Visual Studio

Visual Studio

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

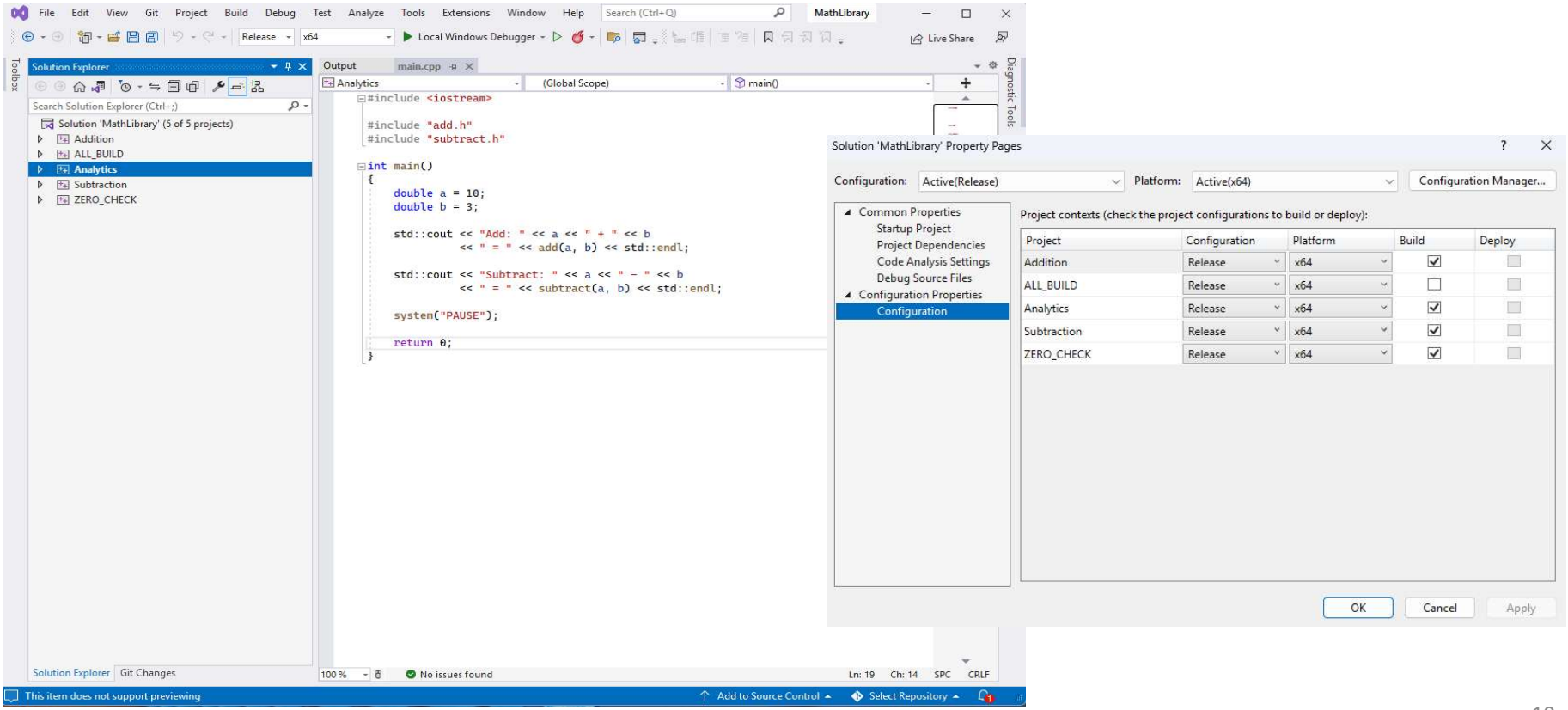
- **Project Files**
  - Independent Code Project Groups

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

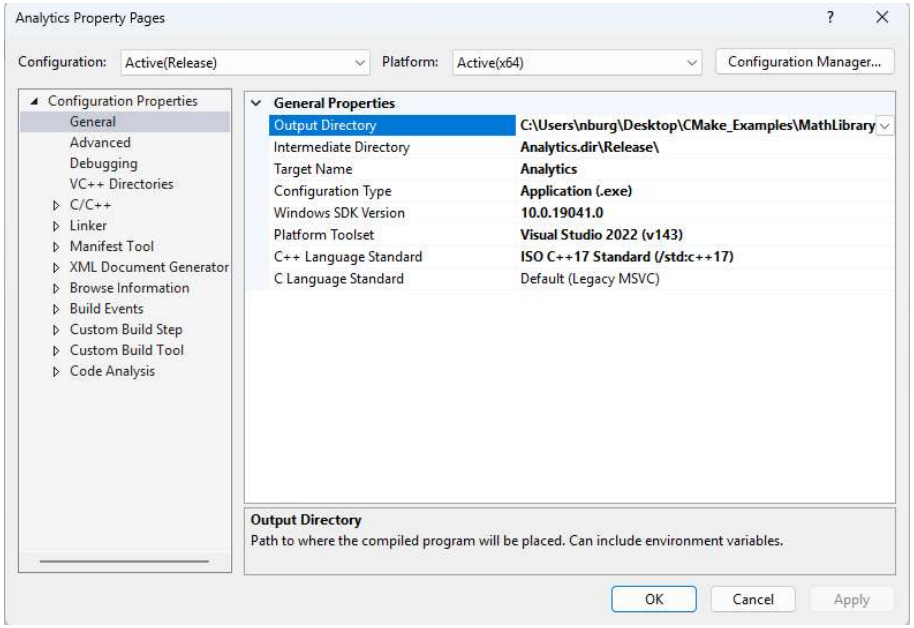


Visual Studio Solution & Projects Files

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



Visual Studio Project Properties



Output type

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

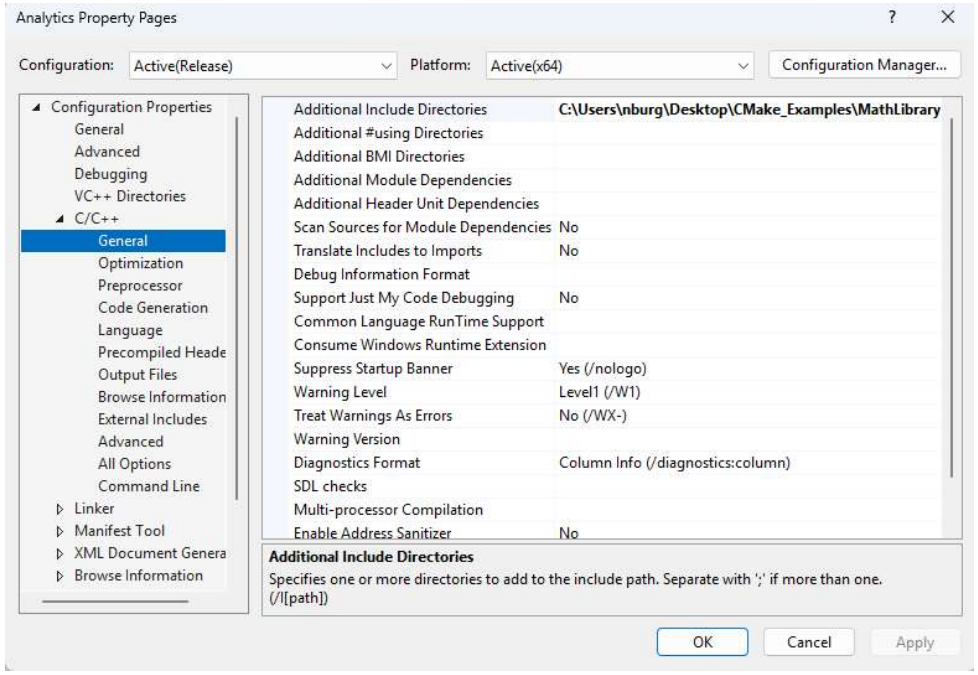
Where outputs go

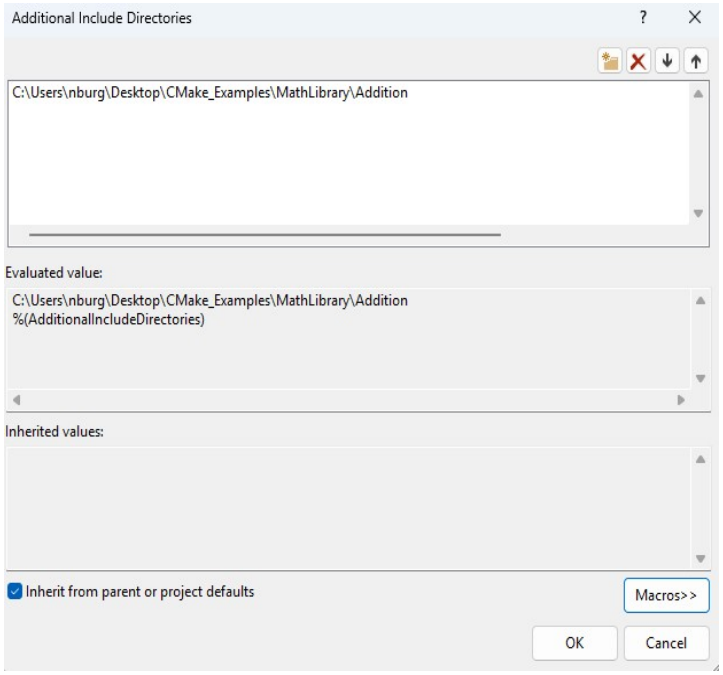
- Intermediate Directory (.obj)
- Output Directory (.lib | .exe | .dll)

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

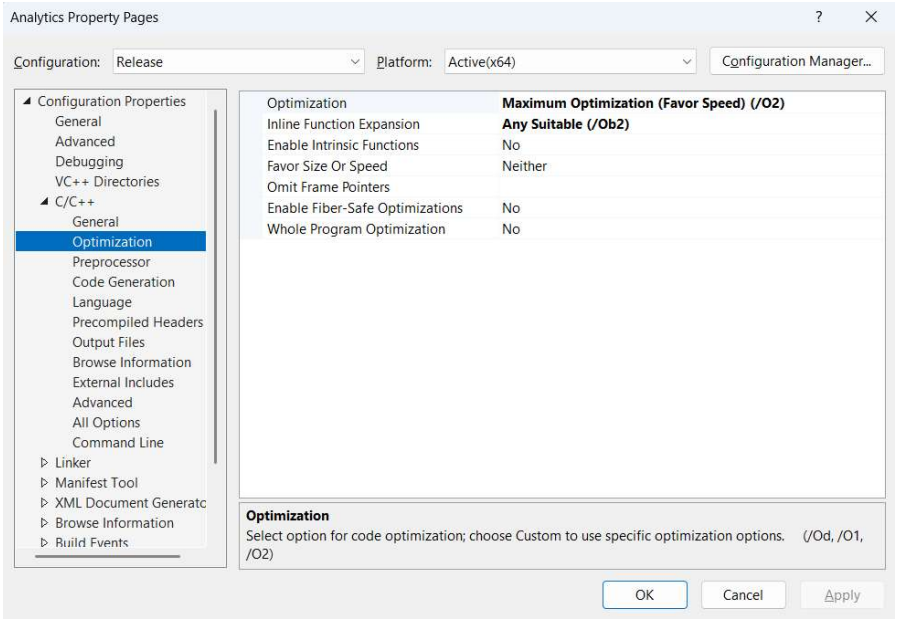


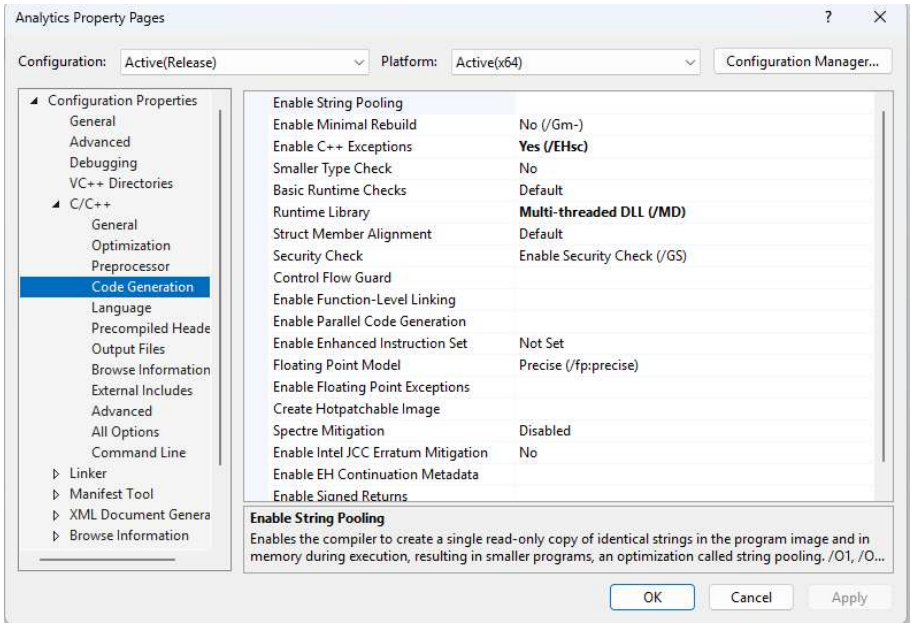


File Path Macros [TOP TIP]

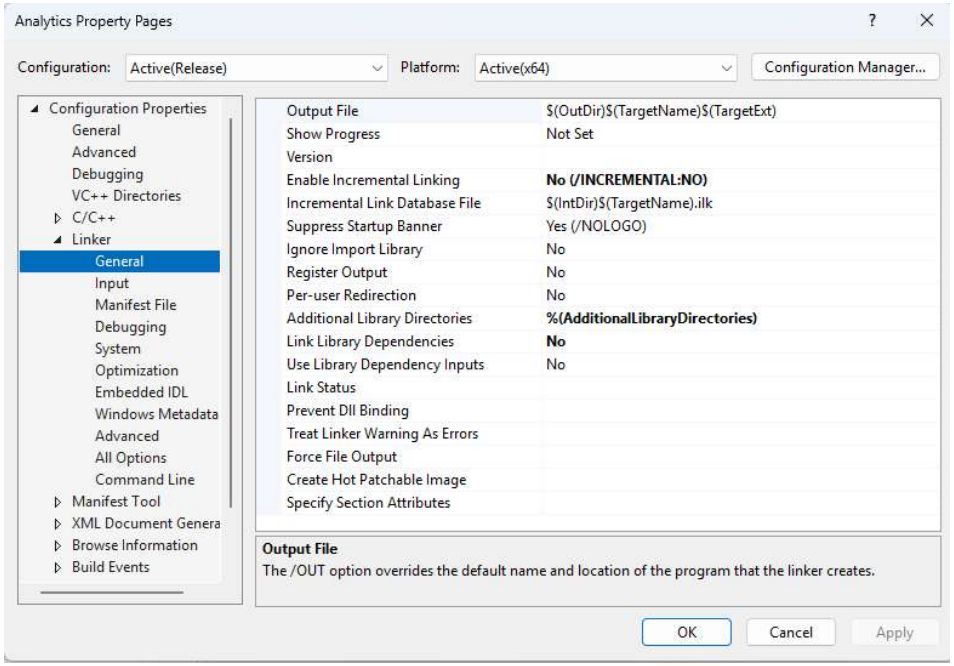
- Click the down arrow on any directory folder, then in the window pop-up press the "**Macros"** button
- View existing file path variables (macros) and/or add new ones e.g. $(SolutionDir), $(ProjectDir), …

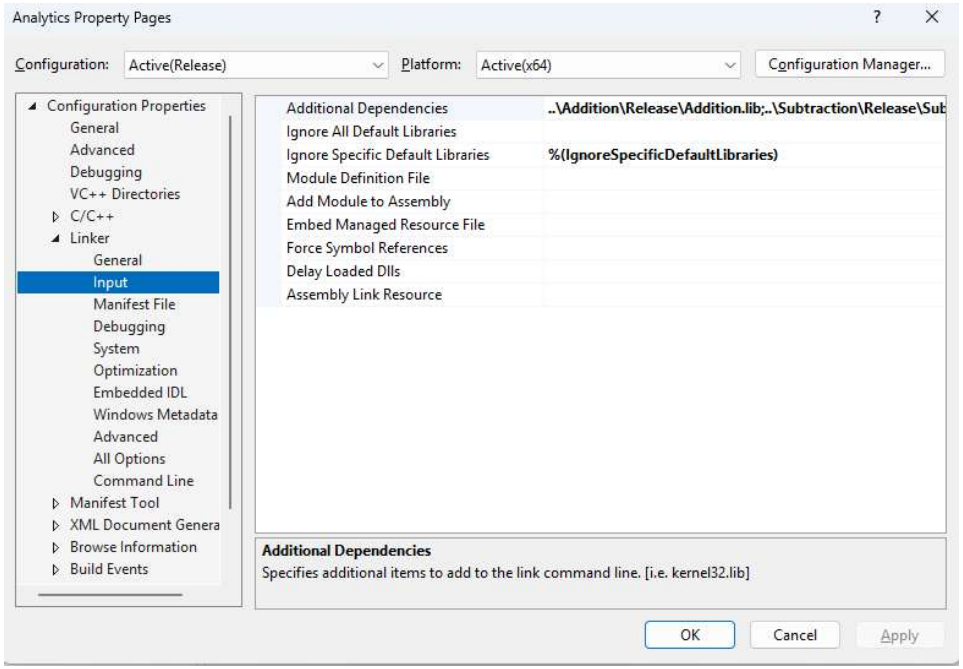
VS Project Properties – C/C++ Compiler





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 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
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

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** 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

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

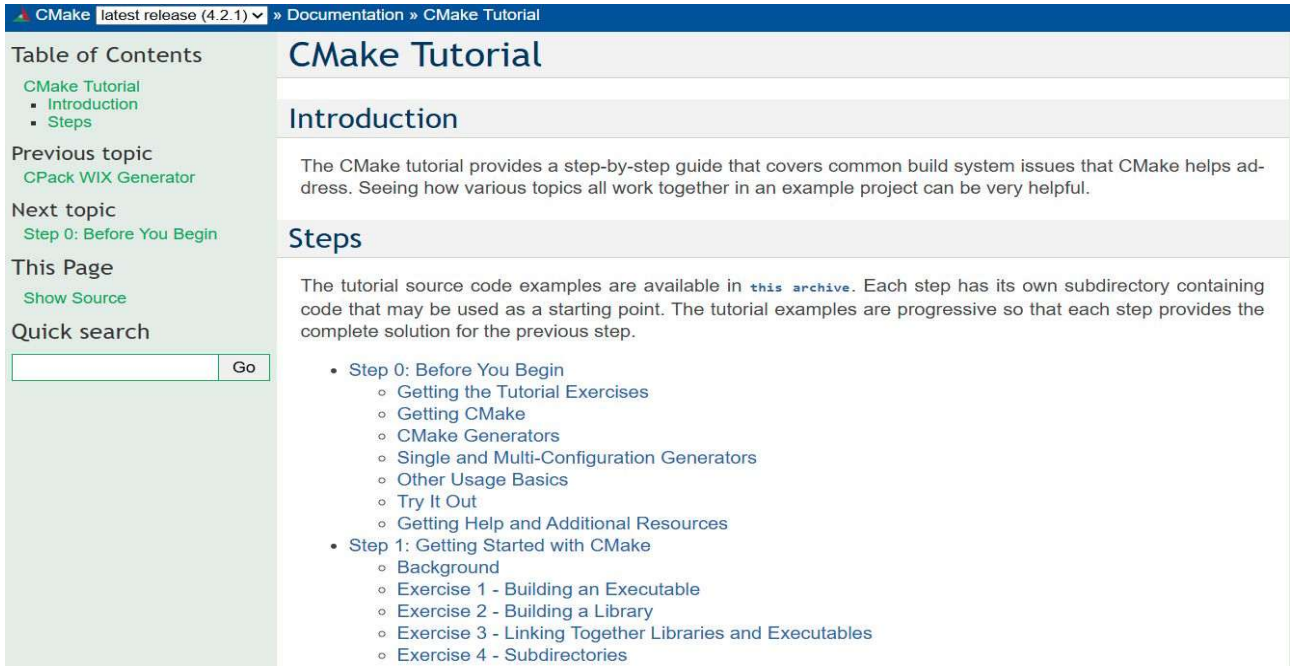
AlgoQuantHub Weekly Deep Dive

Professional C++ with CMake for Quants & Algo Trading

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

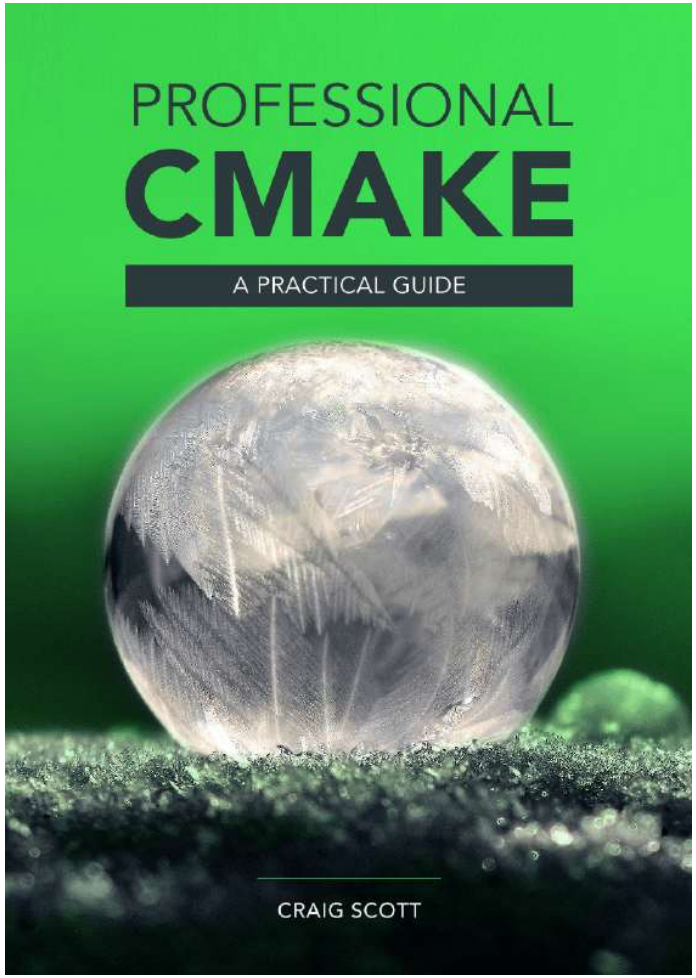
Examples: *https://github.com/nburgessx/QuantResearch/tree/main/CMake%20Examples*

CMake Tutorial – cmake.org



- 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/*

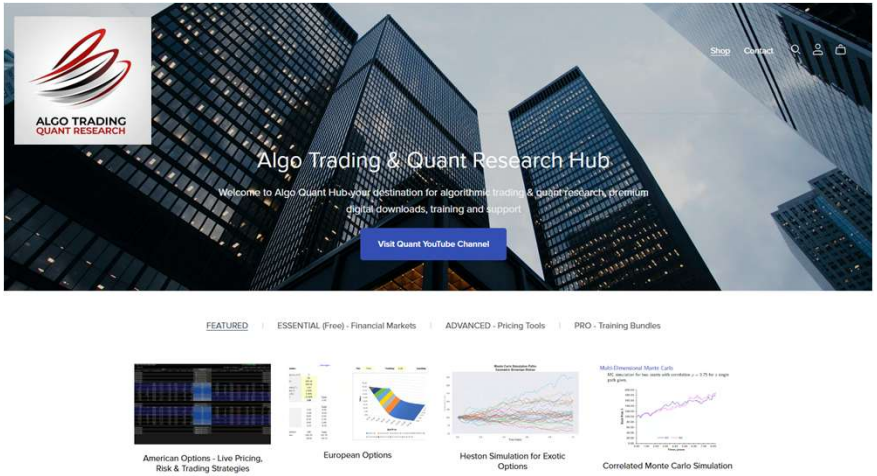

C++
Algorithmic Trading & Quant Research Hub
YouTube
More Info ...

Subscribe to my Quant Newsletter
https://algoquantthub.beehiiv.com/subscribe

Subscribe to my Quant YouTube Channel
https://www.youtube.com/@algoquantthub

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