Courses

Four courses and one capstone project comprise the C++ Developer Nanodegree Program.

- Foundations
- Object-Oriented Programming
- Memory Management
- Concurrency
- Capstone Project

Foundations

Learn basic C++ syntax, functions, containers, and compiling and linking with multiple files. Use OpenStreetMap and the 2D visualization library IO2D to build a route planner that displays a path between two points on a map.

Object-Oriented Programming

Explore Object-Oriented Programming (OOP) in C++ with examples and exercises covering the essentials of OOP like abstraction and inheritance all the way through to advanced topics like polymorphism and templates. In the end, you'll build a Linux system monitor application to demonstrate C++ OOP in action!

Memory Management

Discover the complexity of memory management in C++ by diving deep into stack vs. heap, pointers, references, new, delete and much more. By the end, you'll write your very own smart pointer!

Concurrency

Concurrent programming runs multiple threads of execution in parallel. Concurrency is an advanced programming technique that, when properly implemented, can dramatically accelerate your C++ programs.

Capstone Project

Put your C++ skills to use on a project of your own! You'll utilize the core concepts from this Nanodegree program - object-oriented programming, memory management, and concurrency - to build your own application using C++.

Bjarne Stroustrup appears throughout the C++ Nanodegree Program, offering his perspective and insight on aspects of the programming language.

Bjarne Stroustrup is the designer and original implementer of C++ as well as the author of The C++ Programming Language (Fourth Edition), A Tour of C++ (Second edition), Programming: Principles and Practice using C++ (Second Edition), and many popular and academic publications. Dr. Stroustrup is a Technical Fellow and a Managing Director in the technology division of Morgan Stanley in New York City as well as a visiting professor at Columbia University. He is a member of the US National Academy of Engineering, and an IEEE, ACM, and CHM fellow. He received the 2018 Charles Stark Draper Prize, the IEEE Computer Society's 2018 Computer Pioneer Award, and the 2017 IET Faraday Medal. His research interests include distributed systems, design, programming techniques, software development tools, and programming languages. He is actively involved in the ISO standardization of C++. He holds a masters in Mathematics from Aarhus University, where he is an honorary professor, and a PhD in Computer Science from Cambridge University, where he is an honorary fellow of Churchill College.


In 2011, the C++ Standards Committee
1:03 / 1:44
YouTube


released C++11 which was a major update to the programming language standard.
1:08 / 1:44
YouTube


Move semantics, variadic templates,
0:13 / 1:05
YouTube


initializer lists, the auto keyword,
0:14 / 1:05
YouTube


lambda expressions, null pointer,
0:17 / 1:05
YouTube


constant expression, range-based four loops, and smart pointers,
0:20 / 1:05
YouTube


all came into the C++ programming language as part
0:24 / 1:05
YouTube


of C++ 11 and changed the language dramatically.
0:26 / 1:05
YouTube


C++ 17 includes standard file system within the standard library,
0:43 / 1:05
YouTube


standard string view, parallel implementations of
0:46 / 1:05
YouTube


many of the standard library algorithms, and inline variables.
0:49 / 1:05
CC
HD
YouTube

The C++ Core Guidelines are a set of best practices, recommendations, and rules for coding in C++ which have been developed by Bjarne Stroustrup and hundreds of other experts in the field. These guidelines are an important part of the language, as they help users to write the best, most up-to-date C++ possible. In the next couple of videos, Bjarne will talk about the origin of the guidelines and give some advice on how to use them.

Standard Library

> *"The C++ Standard Library is a collection of classes and functions, which are written in the core language and part of the C++ ISO Standard itself." Wikipedia*

Learning how to use the Standard Library is an important part of becoming a proficient C++ software engineer. In almost all cases, it is preferable to utilize functionality that already exists in the Standard Library, instead of implementing functionality from scratch. This is both because using the Standard Library is faster (it is well-documented) and because many expert software engineers have worked on the Standard Library. The performance of Standard Library facilities is optimized, robust, and almost always as fast or faster than an initial re-implementation of the same functionality.

In fact, guideline SL.1 of the C++ Core Guidelines is:

> ***Use libraries wherever possible***
>
> *Reason Save time. Don't re-invent the wheel. Don't replicate the work of others. Benefit from other people's work when they make improvements. Help other people when you make improvements.*

And guideline SL.2 is:

> ***Prefer the standard library to other libraries***
>
> *Reason More people know the standard library. It is more likely to be stable, well-maintained, and widely available than your own code or most other libraries.*

We use Standard Library features throughout the program, since proficiency with the Standard Library is a critical for C++ developers.

Namespace

Standard Library functions and classes exist in the `std::` namespace. `std::vector`, for example, refers to the `vector` class within the Standard Library. Typically, in order to use a Standard Library feature we must both include the necessary header file (e.g. `#include <vector>`) and also namespace the class with `std::` (e.g. `std::vector`).

Compilers

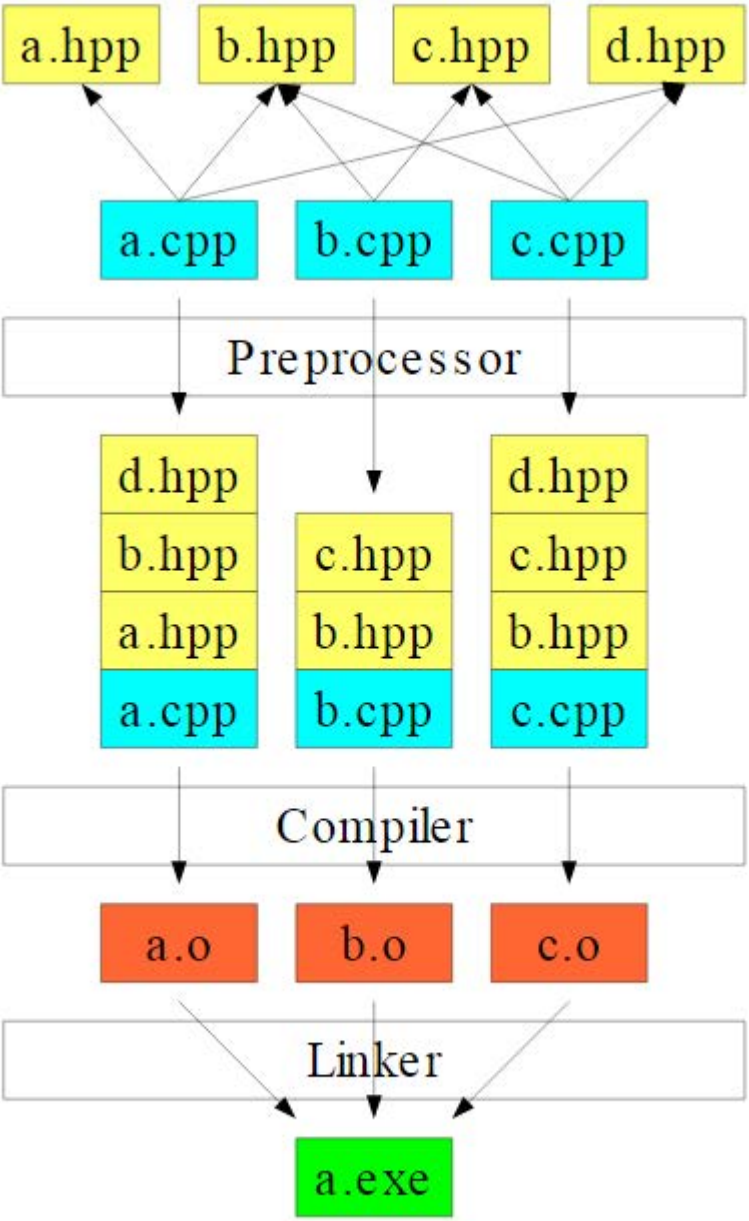
C++ is a compiled programming language, which means that programmers use a program to compile their human-readable source code into machine-readable object and executable files. The program that performs this task is called a compiler.

C++ does not have an "official" compiler. Instead, there are many different compilers that a programmer can use.

GNU Compiler Collection (GCC)

In this program we primarily use the GNU Compiler Collection, which is a popular, open-source, cross-platform compiler from the larger GNU Project. In particular, we use the `g++` program, which is a command line executable that compiles C++ source code and automatically links the C++ Standard Library.

Linking



C++ Compilation Process (Wikimedia)

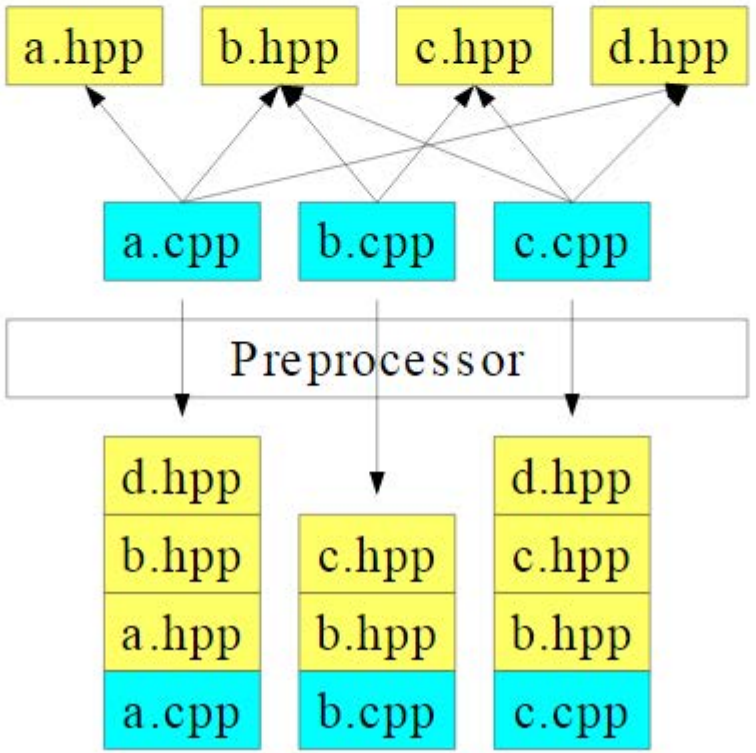
In order to use classes and functions from the C++ Standard Library, the compiler must have access to a compiled version of the standard library, stored in object files. Most compiler implementations, including GCC, include those object files as part of the installation process. In order to use the Standard Library facilities, the compiler must "link" the standard library object files to the object files created from the programmer's source code.

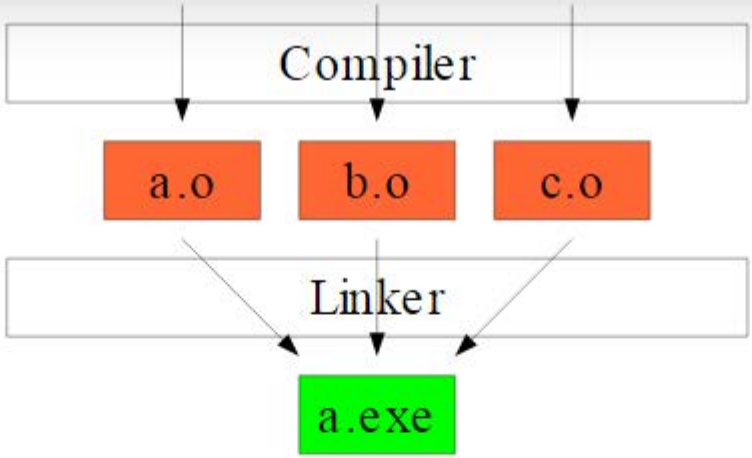
Once the compiler links together the necessary object files, it is able to generate a standalone executable file that can run on the operating system.

Build Tools

Make and CMake are two separate and similar build tools that both serve to help simplify the process of building software.

In particular, build tools automate the process of compiling multiple source code files into object files, linking those object files together, and generating an executable. Build tools also often automate the process of determining which files have changed since the last build and thus need to be recompiled.





C++ Build Process

Make

GNU Make is a widely-used build tool that relies on `Makefile`s to automate the process of building a project.

A `Makefile` typically includes one or more "targets". Each target performs a different action.

`build` is a common target name that is configured in the `Makefile` to compile all of the project's source code into an executable file. `clean`, on the other hand, is a common target to delete all object files and other artifacts of the build process, resulting in a clean, unbuilt project state.

Running either `make build` or `make clean` (or any other target) on the command line would cause Make to search for a local Makefile, search for a matching target within that Makefile, and then execute the target.

GNU provides

CMake

CMake is a built tool that facilitates cross-platform builds, so that it is straightforward to build the same source code on Linux, macOS, Windows, or any other operating system. CMake relies on a CMakeLists.txt file, which configures appropriate cross-platform targets.

Building a `CMakeLists.txt` file can be a bit daunting, but CMake provides a helpful tutorial.

In this Nanodegree program, you will not need to build your own `Makefile`s or `CMakeLists.txt` files. We provide the appropriate configuration files for each project and instruct you as to their usage.

Installation

You are welcome to write all of your code in Udacity's web-based Workspaces. If, however, you prefer to work locally on your machine, you will need to install certain software.

g++, gdb, make

macOS

macOS includes g++ as part of Command Line Tools.

- Launch **Terminal**, which can be found in the **Utilities** folder in **Applications**.
- Type `xcode-select --install` into the Terminal window and press "Enter"
- If you don't already have Xcode or Command Line Tools installed, a window will pop up. Press the **Install** button.
- **Verify:** Type `g++` into Terminal and press enter. If the output is `clang: error: no input files`, then the installation was successful.

Linux

These programs are typically available through the default package manager for each Linux distribution. For example, we can use APT on Ubuntu systems.

- `sudo apt update`
- `sudo apt install build-essential`
- `sudo apt install gdb`

Windows

MinGW provides the necessary software.

- Proceed from Section 3.2 of these linked instructions.

Microsoft Visual Studio (VSCode):

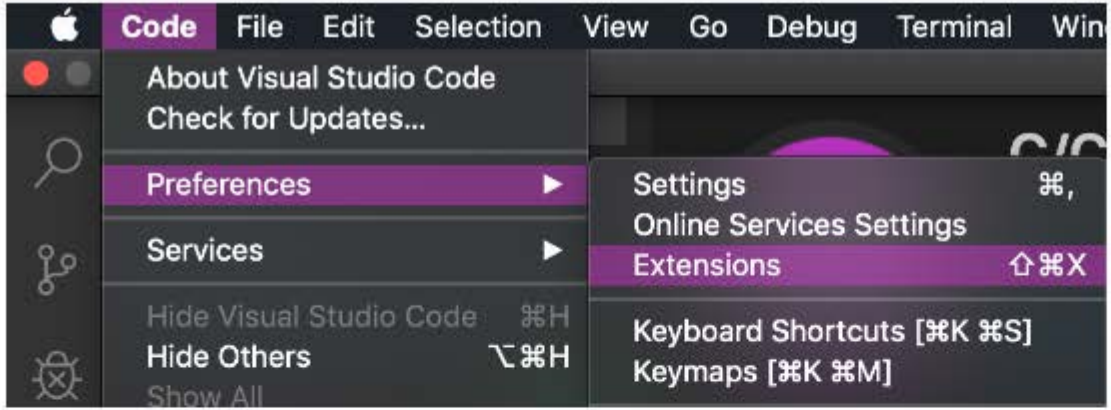
The instructions for this are the same for all machines:

- Go to the Visual Studio Code download webpage
- Select your operating system.

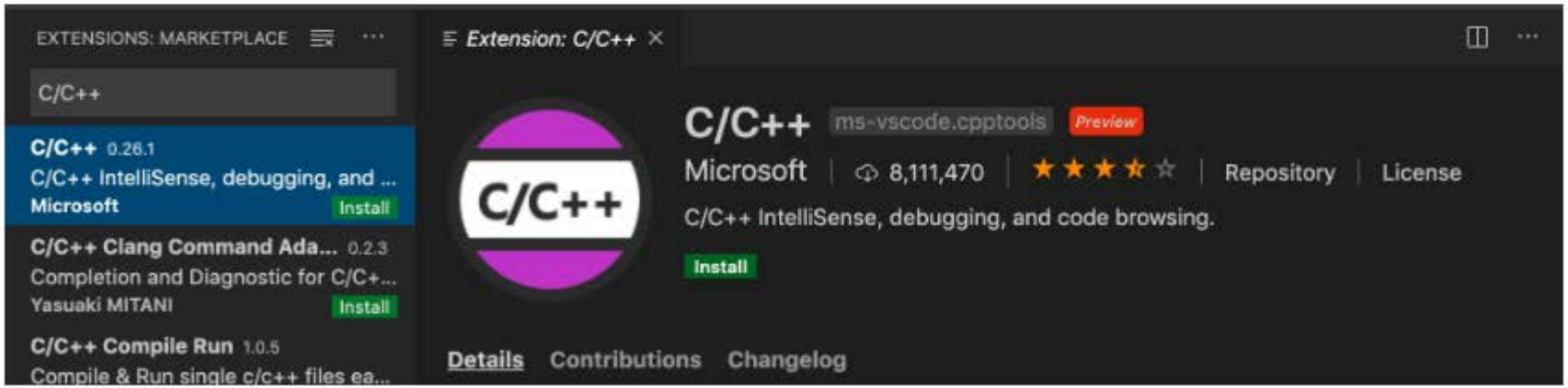
- Click on the downloaded file
- Complete the installation instructions.

VSCode C/C++ Extension

- Open VSCode
- Navigate to VSCode extensions by clicking into the following menus:
    - Code > Preferences > Extensions



- In the search bar that says "*Search Extensions in Marketplace*" type "**C/C++**"

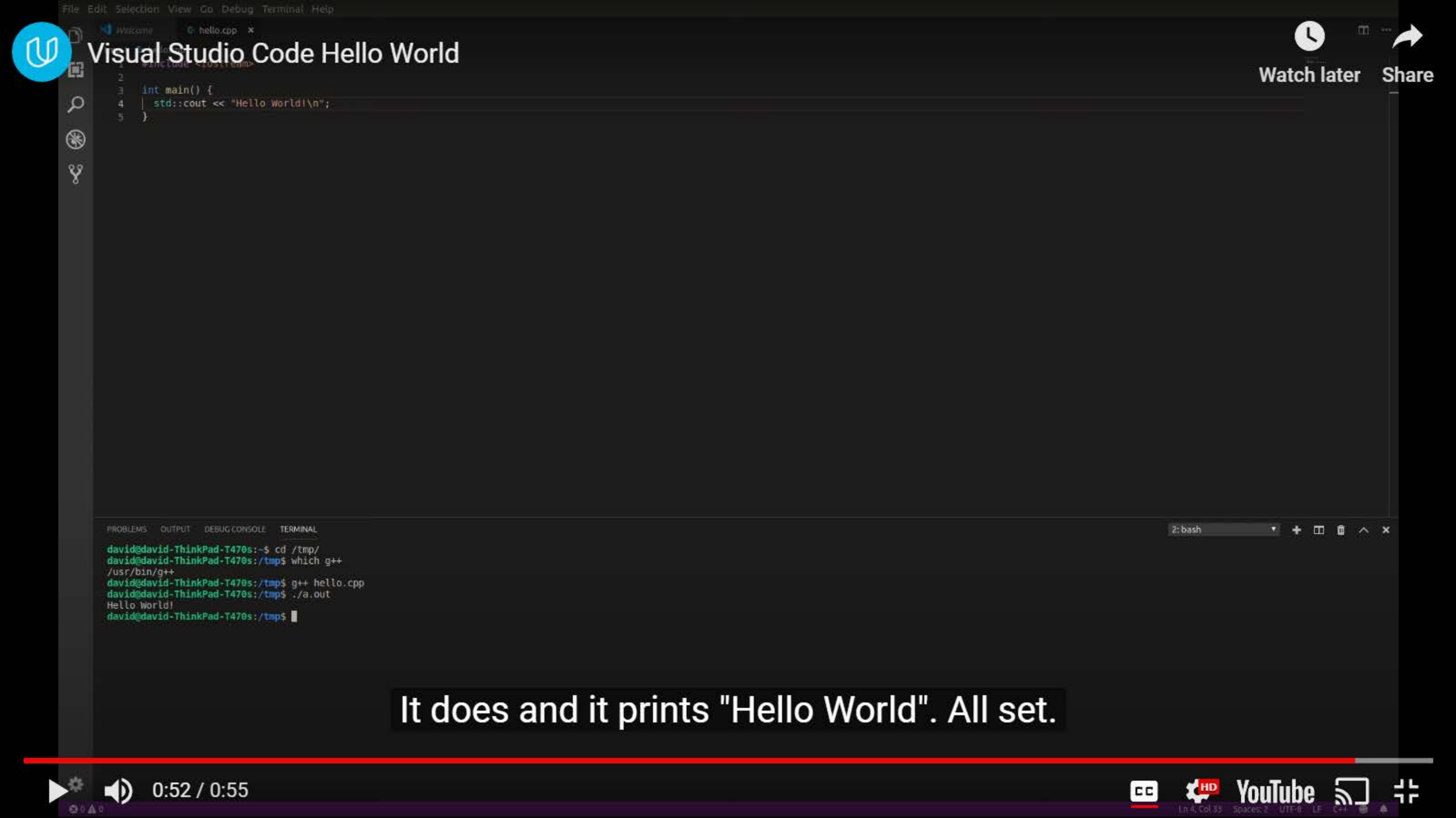


- Select the C/C++ extension and press the **Install** button to install this extension.

clang-format

- This is also a VSCode extension, so navigate to **Extensions** using the same process as the previous section.
- In the search bar that says "Search Extensions in Marketplace", type: "**clang-format**"

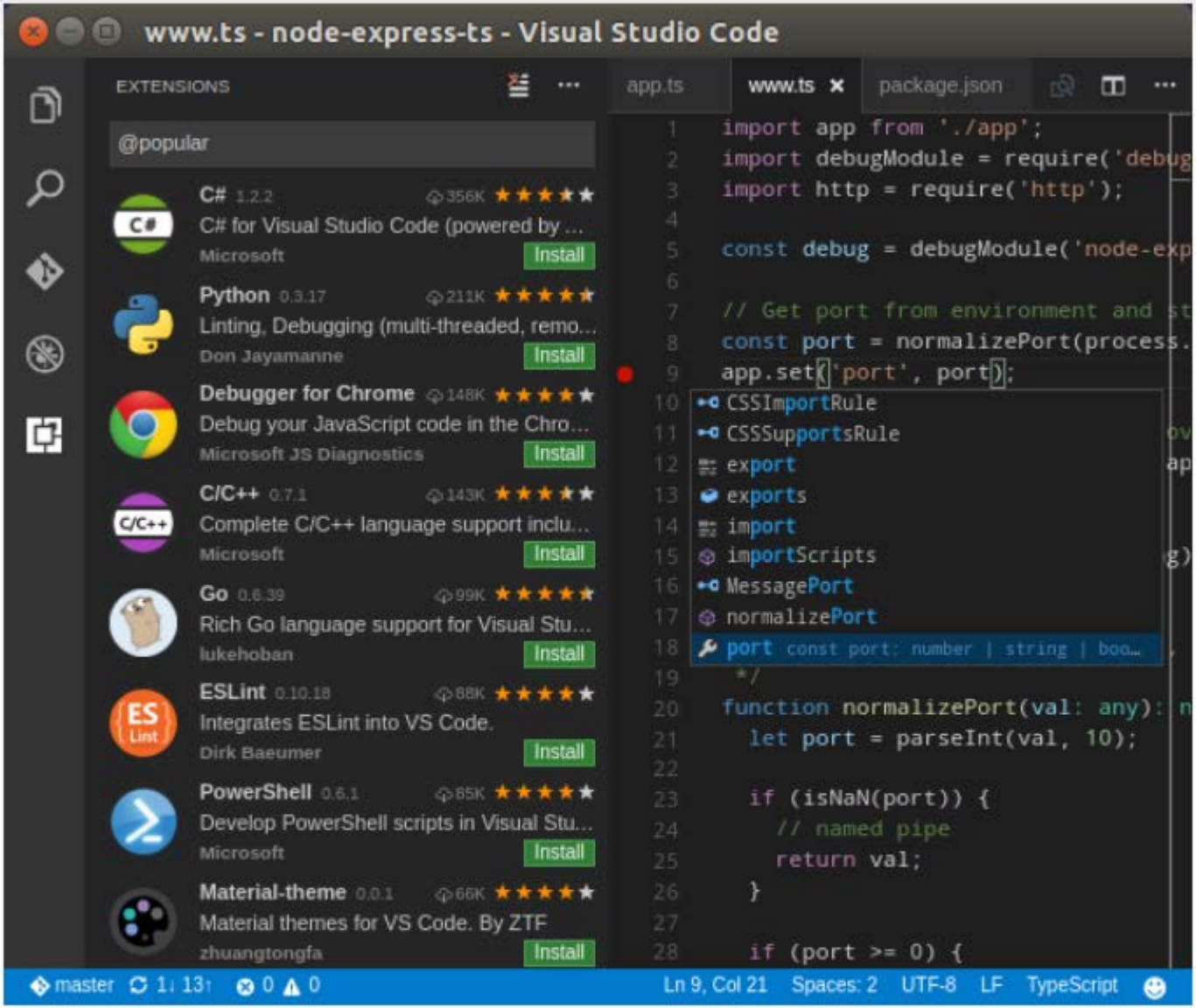
- Select the Clang-Format extension and press the **Install** button to install this extension.


Visual Studio Code Hello World
Watch later
Share
File Edit Selection View Go Debug Terminal Help
hello.cpp
#include <iostream>
int main() {
std::cout << "Hello World!\n";
}
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: bash
david@david-ThinkPad-T470s:~$ cd /tmp/
david@david-ThinkPad-T470s:/tmp$ which g++
/usr/bin/g++
david@david-ThinkPad-T470s:/tmp$ g++ hello.cpp
david@david-ThinkPad-T470s:/tmp$ ./a.out
Hello World!
david@david-ThinkPad-T470s:/tmp$
It does and it prints "Hello World". All set.
0:52 / 0:55
HD
YouTube

Editors

There are many different C++ code editors, and in fact it is at least theoretically possible to develop C++ source code in any text editor.

Microsoft Visual Studio Code

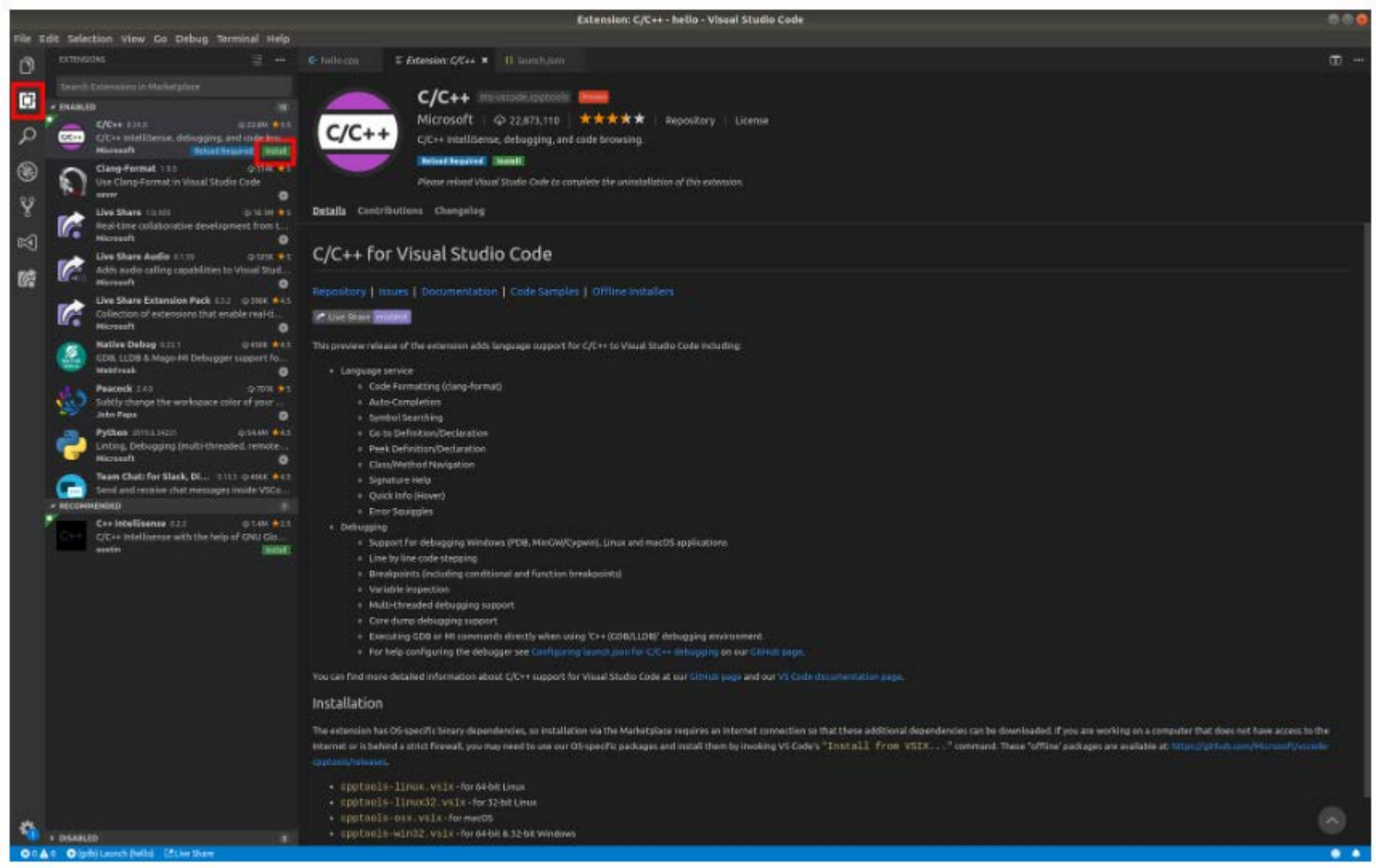


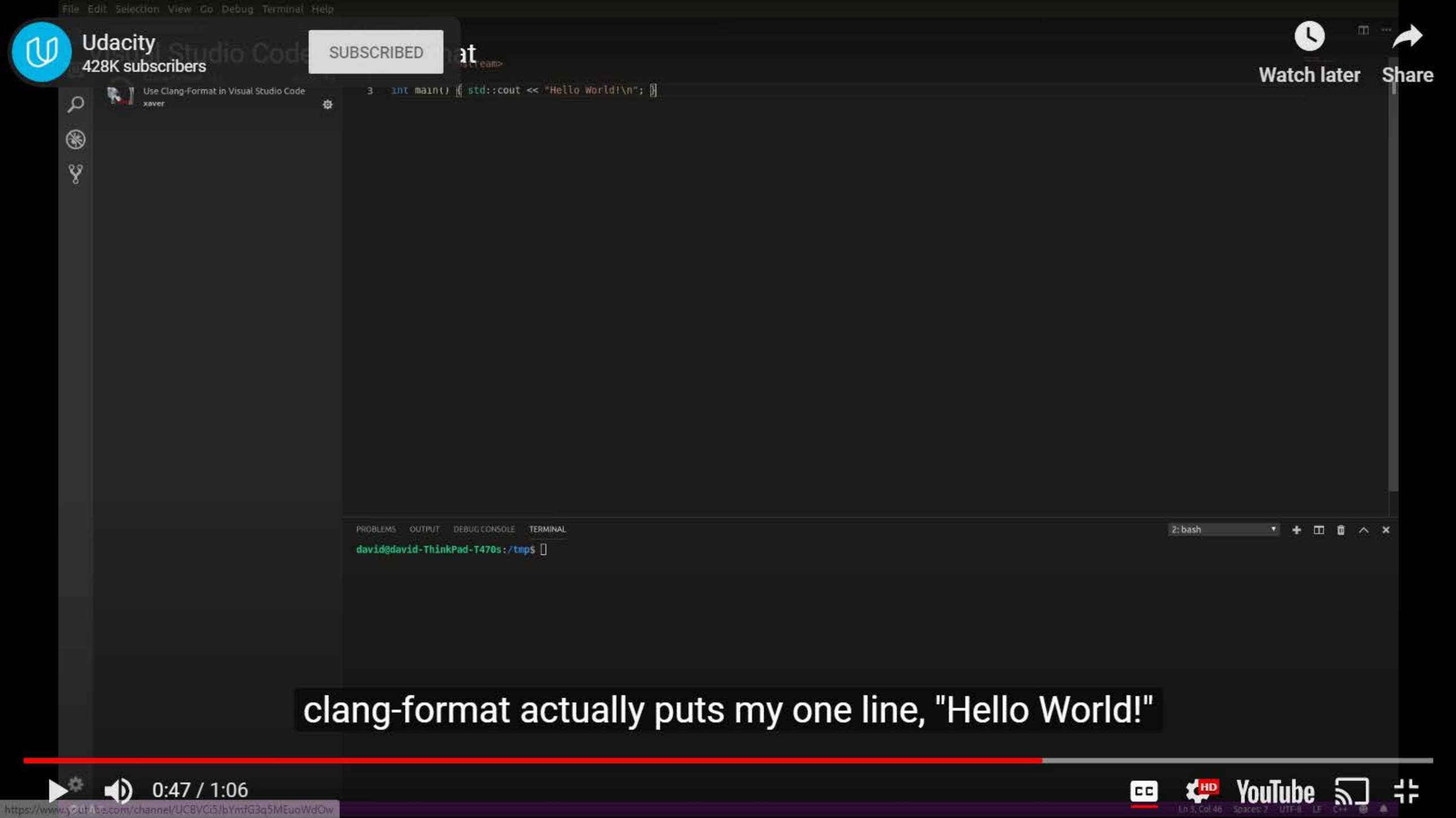
In this Nanodegree Program, we will demonstrate many of the programming examples using Microsoft Visual Studio Code. You are welcome, but not required, to use this editor.

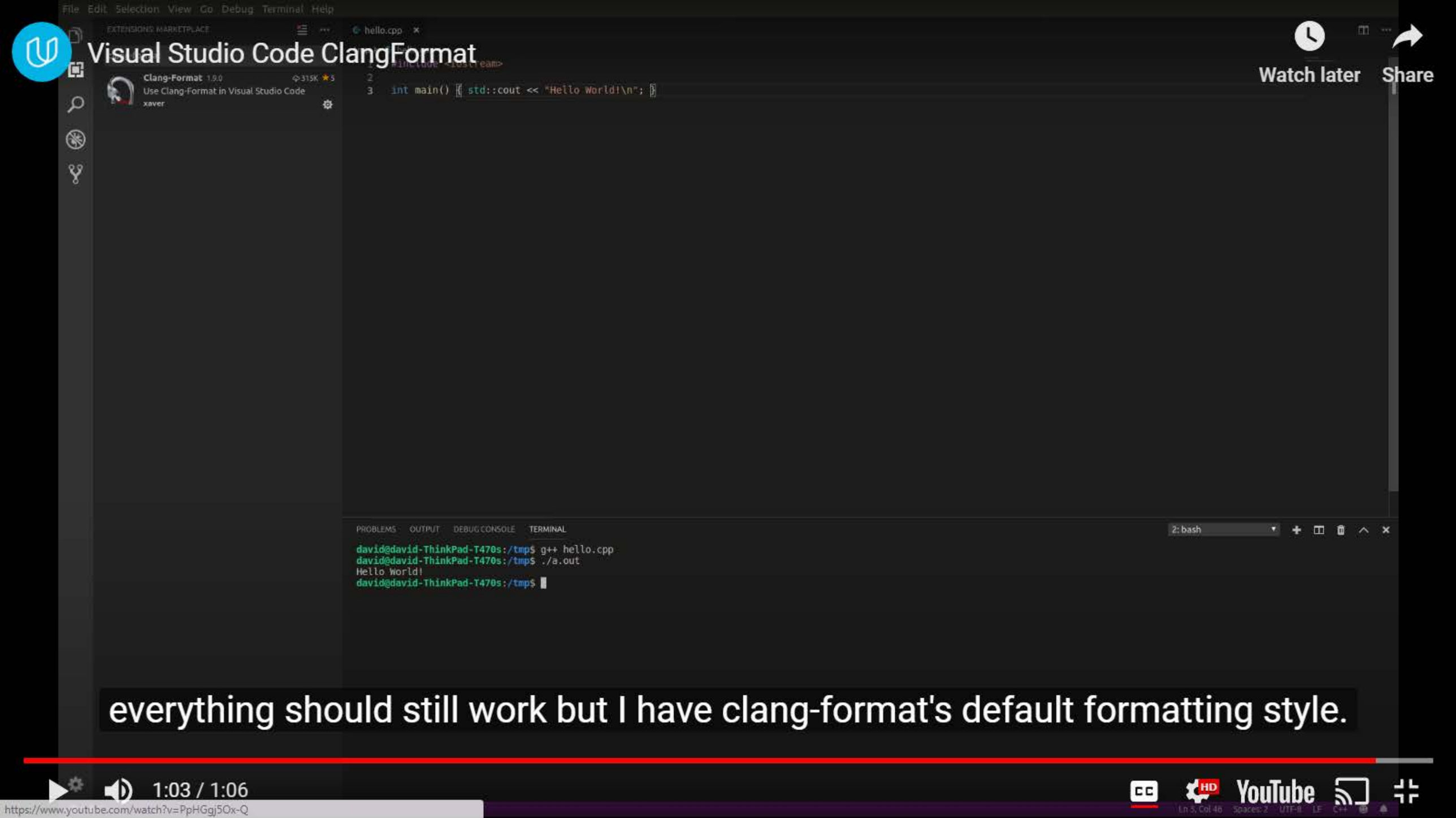
Visual Studio Code is a powerful, free, and open-source editor that runs on Linux, macOS, and Windows.

> *"Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity). Begin your journey with VS Code with these introductory videos."*
>
> *Microsoft*

If you use Visual Studio Code to develop C++ applications, Microsoft suggests installing the free C/C++ extension.


Extension: C/C++ - hello - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
EXTENSIONS
Search Extensions in Marketplace
ENABLED
C/C++
C/C++ IntelliSense, debugging, and code bro...
Microsoft
Reload Required
Install
Clang-Format
Use Clang-Format in Visual Studio Code
Live Share
Real-time collaborative development from t...
Microsoft
Live Share Audio
Adds audio calling capabilities to Visual Stud...
Microsoft
Live Share Extension Pack
Collection of extensions that enable real-ti...
Microsoft
Native Debug
GDB, LLDB & Mago-MI Debugger support fo...
WebFreak
Peacock
Subtly change the workspace color of your ...
John Papa
Python
Linting, Debugging (multi-threaded, remote ...
Microsoft
Team Chat: for Slack, Di...
Send and receive chat messages inside VSCo...
RECOMMENDED
C++ Intellisense
C/C++ Intellisense with the help of GNU Glo...
austin
Install
DISABLED
hello.cpp
Extension: C/C++
launch.json
C/C++
ms-vscode.cpptools
Preview
Microsoft
22,873,110
Repository
License
C/C++ IntelliSense, debugging, and code browsing.
Reload Required
Install
Please reload Visual Studio Code to complete the uninstallation of this extension.
Details
Contributions
Changelog
C/C++ for Visual Studio Code
Repository | Issues | Documentation | Code Samples | Offline Installers
Live Share Installed
This preview release of the extension adds language support for C/C++ to Visual Studio Code including:
Language service
Code Formatting (clang-format)
Auto-Completion
Symbol Searching
Go to Definition/Declaration
Peek Definition/Declaration
Class/Method Navigation
Signature Help
Quick Info (Hover)
Error Squiggles
Debugging
Support for debugging Windows (PDB, MinGW/Cygwin), Linux and macOS applications
Line by line code stepping
Breakpoints (including conditional and function breakpoints)
Variable inspection
Multi-threaded debugging support
Core dump debugging support
Executing GDB or MI commands directly when using 'C++ (GDB/LLDB)' debugging environment
For help configuring the debugger see Configuring launch.json for C/C++ debugging on our GitHub page.
You can find more detailed information about C/C++ support for Visual Studio Code at our GitHub page and our VS Code documentation page.
Installation
The extension has OS-specific binary dependencies, so installation via the Marketplace requires an internet connection so that these additional dependencies can be downloaded. If you are working on a computer that does not have access to the Internet or is behind a strict firewall, you may need to use our OS-specific packages and install them by invoking VS Code's "Install from VSIX..." command. These "offline" packages are available at: https://github.com/Microsoft/vscode-cpptools/releases.
cpptools-linux.vsix - for 64-bit Linux
cpptools-linux32.vsix - for 32-bit Linux
cpptools-osx.vsix - for macOS
cpptools-win32.vsix - for 64-bit & 32-bit Windows
(gdb) Launch (hello)
Live Share


Udacity
428K subscribers
SUBSCRIBED
Watch later
Share
Use Clang-Format in Visual Studio Code
xaver
int main() { std::cout << "Hello World!\n"; }
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
2: bash
david@david-ThinkPad-T470s:/tmp$
clang-format actually puts my one line, "Hello World!"
0:47 / 1:06


Visual Studio Code ClangFormat
Watch later
Share
File Edit Selection View Go Debug Terminal Help
EXTENSIONS MARKETPLACE
Clang-Format 1.9.0
Use Clang-Format in Visual Studio Code
xaver
hello.cpp
#include <iostream>
int main() { std::cout << "Hello World!\n"; }
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: bash
david@david-ThinkPad-T470s:/tmp$ g++ hello.cpp
david@david-ThinkPad-T470s:/tmp$ ./a.out
Hello World!
david@david-ThinkPad-T470s:/tmp$
everything should still work but I have clang-format's default formatting style.
1:03 / 1:06
YouTube
https://www.youtube.com/watch?v=PpHGgj5Ox-Q

Style

A consistent style (hopefully) helps improve and make your code more readable.

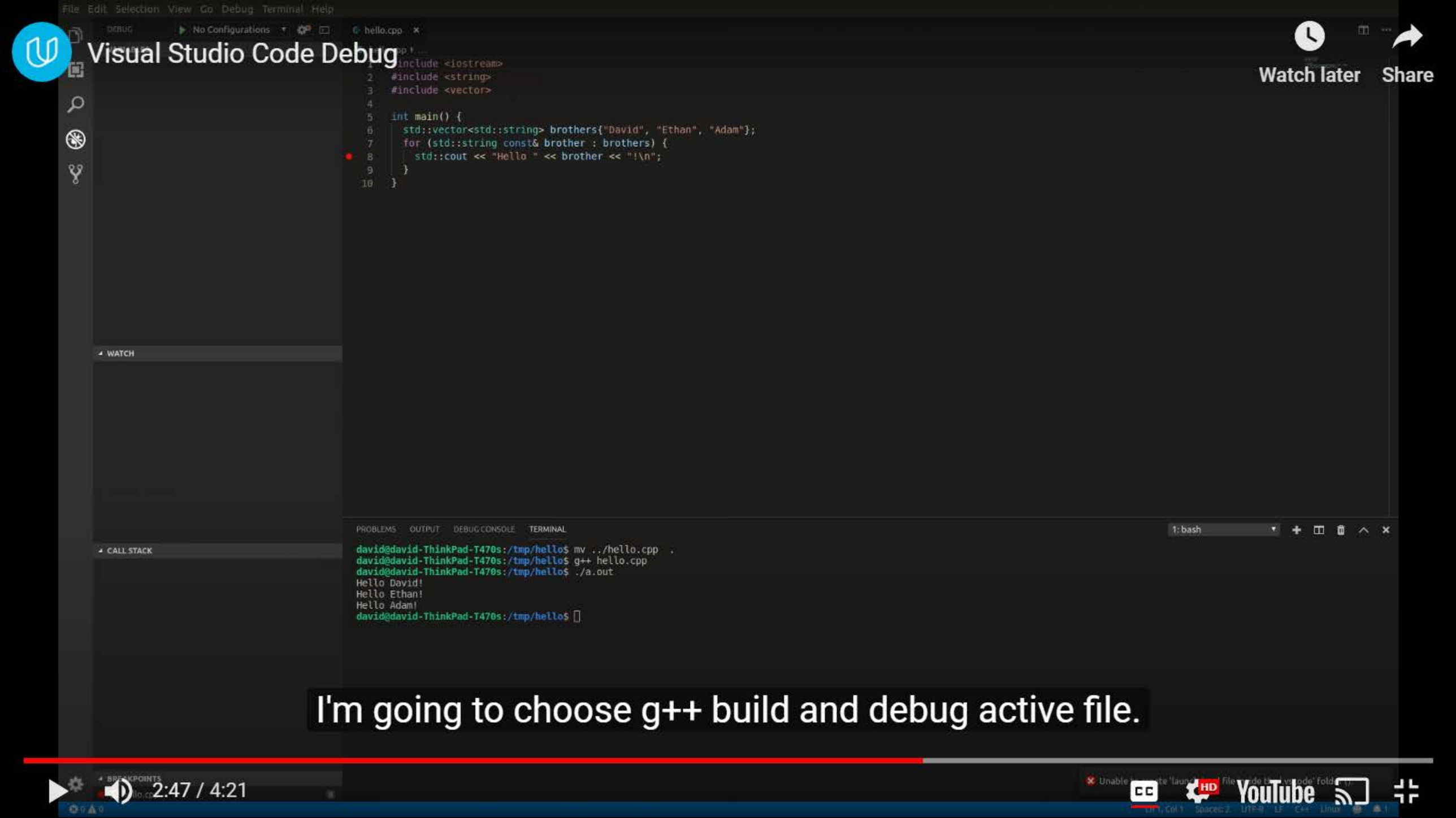
There are many different C++ styles, none of which is authoritative.

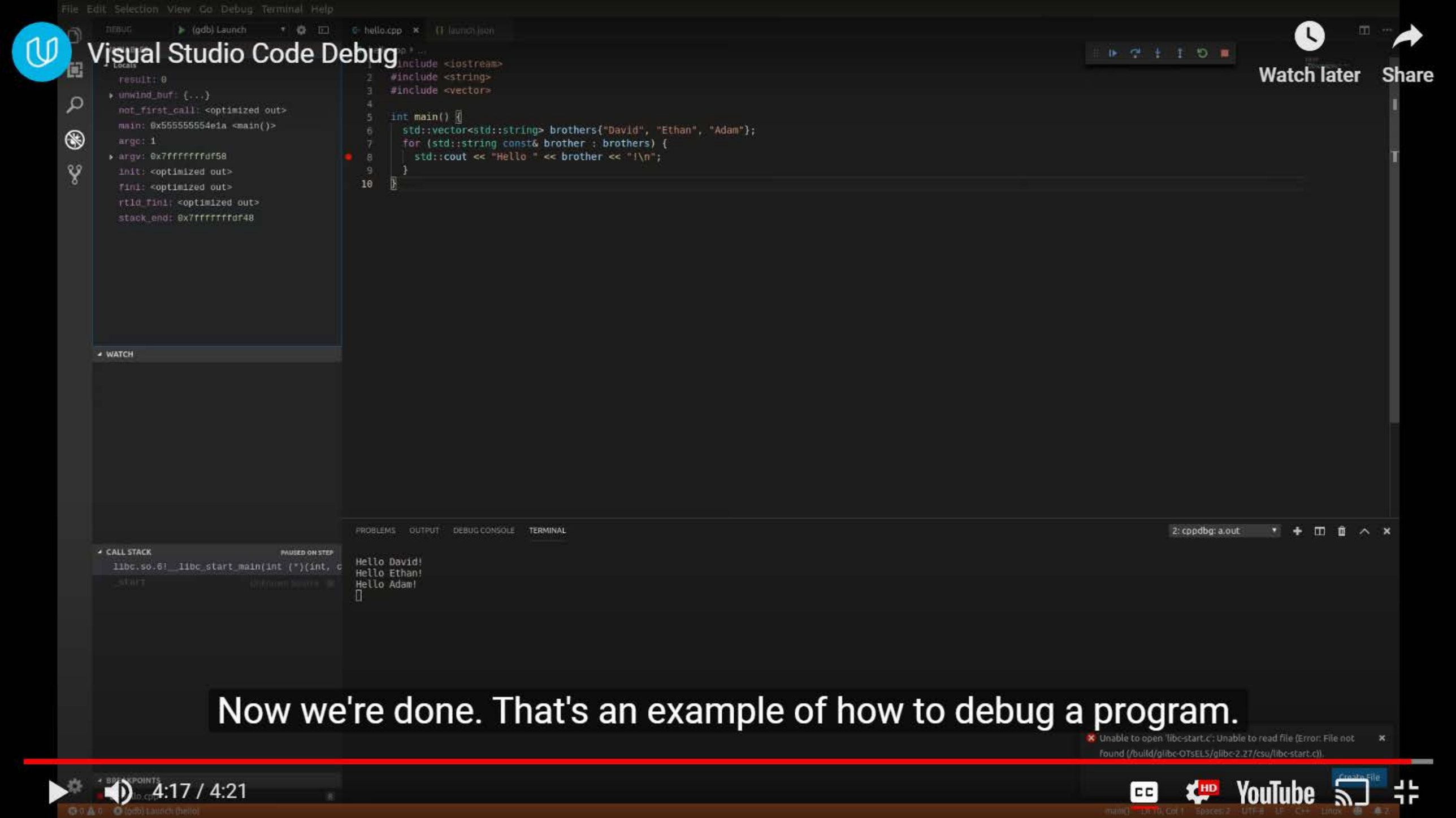
- C++ Core Guidlines: Naming and layout rules
- Google C++ Style Guide
- Mozilla Coding Style: C/C++ practices

ClangFormat

`clang-format` is a command line text formatter that automatically reformats source code according to configurable set of policies. The tool includes several pre-configured styles, or you can create your own.

`clang-format` is an open-source application that you can install on your system, or it is straightforward to install as a Visual Studio Code extension.


Visual Studio Code Debug
Watch later
Share
File Edit Selection View Go Debug Terminal Help
DEBUG
No Configurations
hello.cpp
#include <iostream>
#include <string>
#include <vector>
int main() {
std::vector<std::string> brothers{"David", "Ethan", "Adam"};
for (std::string const& brother : brothers) {
std::cout << "Hello " << brother << "!\n";
}
}
WATCH
CALL STACK
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
david@david-ThinkPad-T470s:/tmp/hello$ mv ../hello.cpp .
david@david-ThinkPad-T470s:/tmp/hello$ g++ hello.cpp
david@david-ThinkPad-T470s:/tmp/hello$ ./a.out
Hello David!
Hello Ethan!
Hello Adam!
david@david-ThinkPad-T470s:/tmp/hello$
I'm going to choose g++ build and debug active file.
2:47 / 4:21
HD
YouTube



Now we're done. That's an example of how to debug a program.

Debugging

Debugging is an important part of software development! Therefore, learning how to use a debugger is an important part of becoming a software developer

Debuggers

Debuggers are tools that allow you to pause the execution of your code in various locations, inspect the state of the program, and step through your code line-by-line.

GDB and LLDB are two popular, open-source debuggers for C++. Integrating them into a code editor often makes debugging easier.

Visual Studio Code Debugging

In order to use Visual Studio Code's debugger with C++ files, you must install the free C/C++ extension.