UNLV Howard R. Hughes College of Engineering

DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

Week 4-2

1. Headers
2. Library
3. Make and CMake

Instructor: Truc Tran

02-18-2023

UNLV Howard R. Hughes College of Engineering

DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

1. Headers

- In C++, a header is a file that contains function and class declarations, constants, and other definitions that can be used by other parts of a program. It helps making programming easier and more efficient.
- Header files typically have the ".h" and they are included in a C++ source file using the #include directive.
- Header files allow the separation of interface and implementation in a C++ program. Therefore, it is easier to change the implementation without affecting the interface.
- There are two types of headers:
  - Pre-existing header files: Already exists in C/C++ compiler, we just need to import them. For example, #include <iostream> or #include <string>.
  - User-defined header files: Defined by the user and can be imported using “include”.

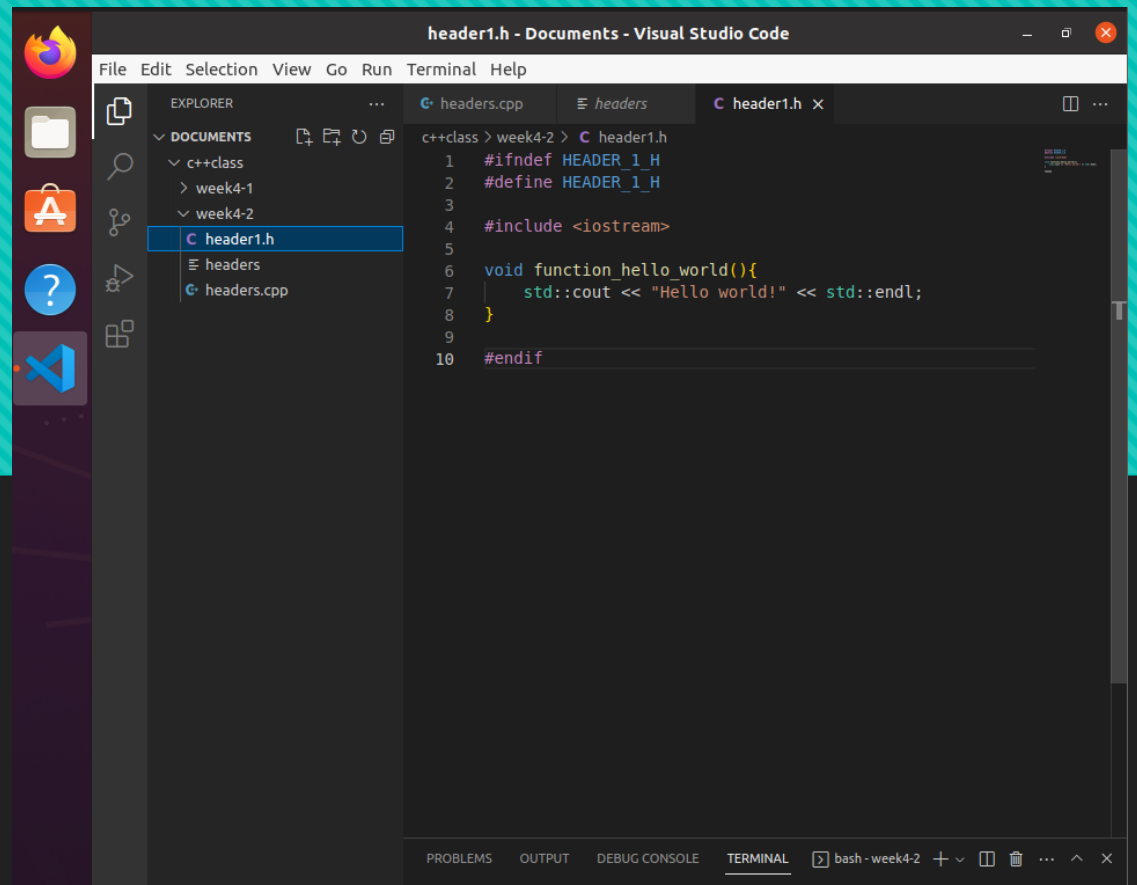
UNLV Howard R. Hughes College of Engineering

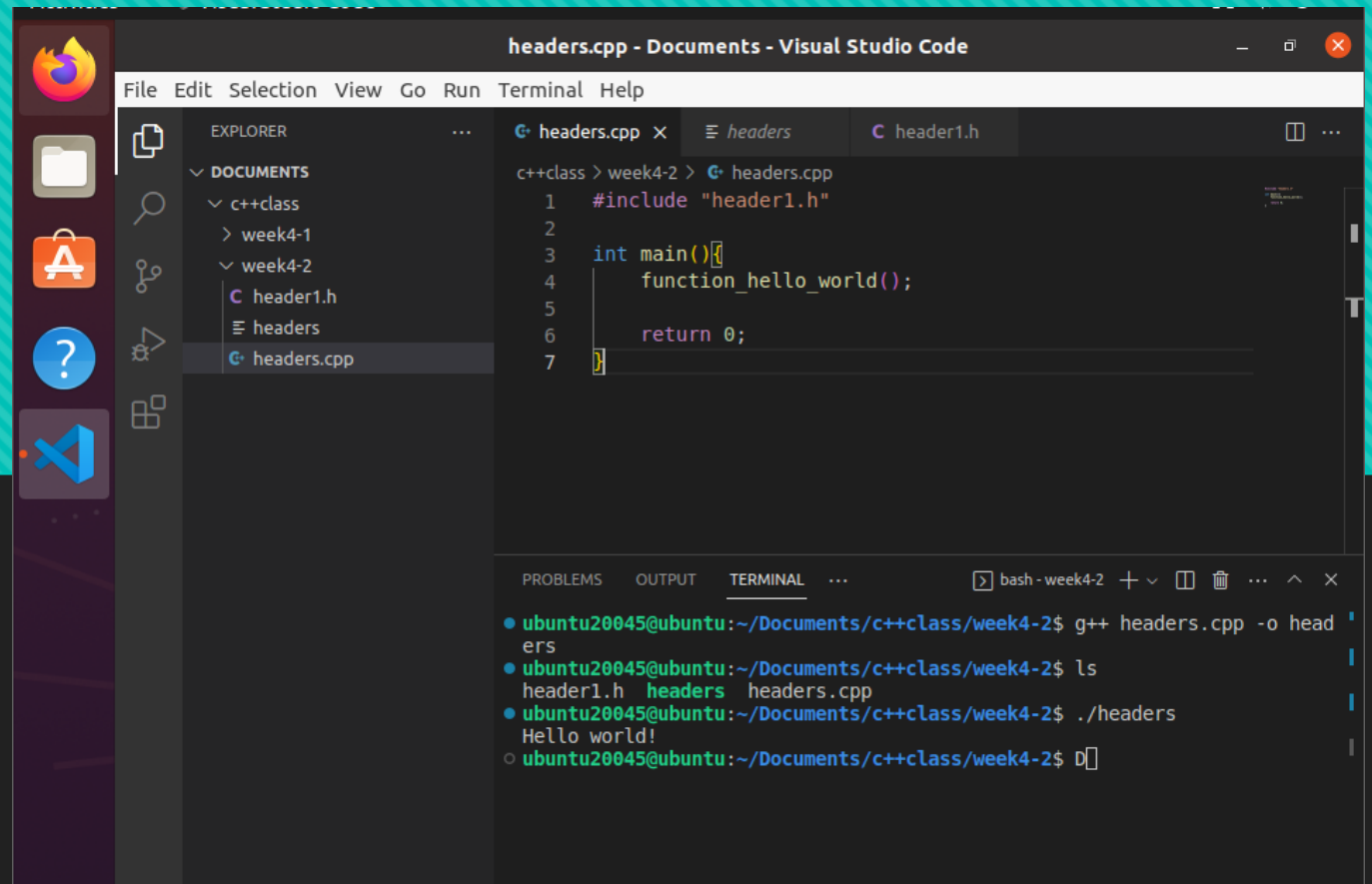
DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

1. Headers

- We will be exploring the user-defined header files.
- Example 1, a single header file (.h) without additional source file (.cpp).





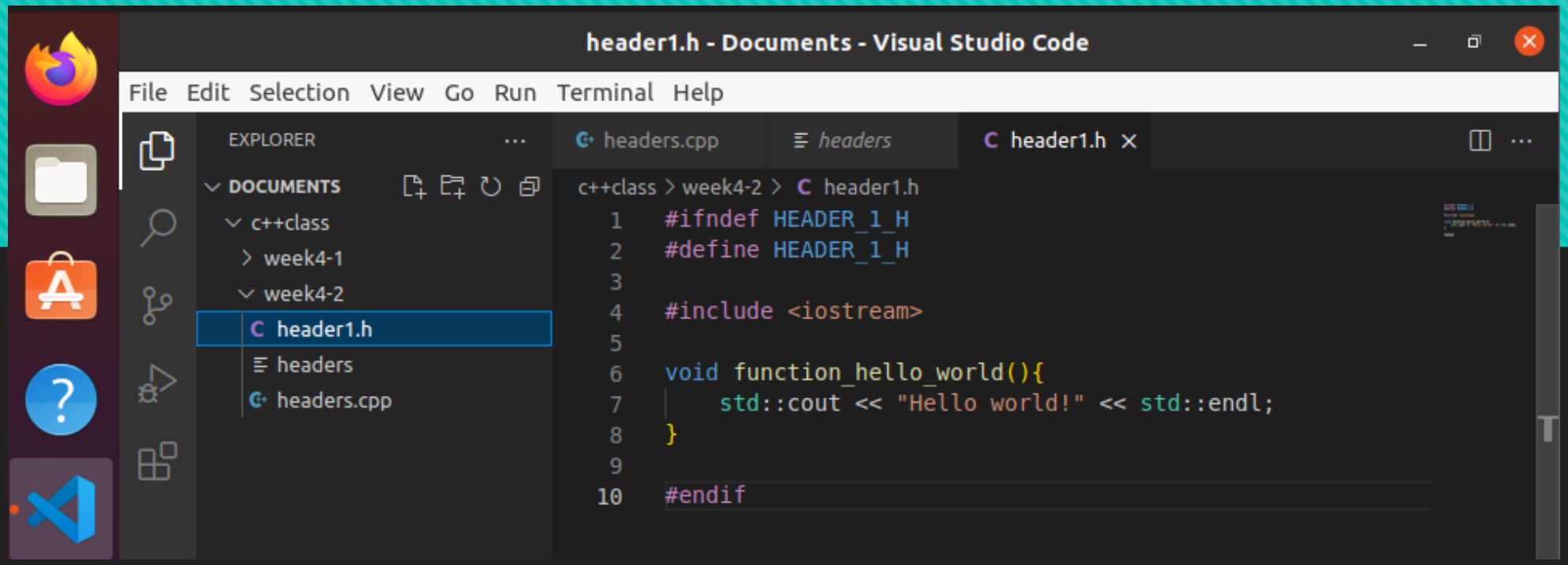
UNLV Howard R. Hughes College of ENGINEERING

DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

1. Headers

- In the header file header1.h, #ifndef and #define are preprocessor directives in C++ that are commonly used together in header files to prevent multiple inclusions of the same header file.
- The #ifndef directive stands for "if not defined".
- The #define directive is used to define a macro or identifier.



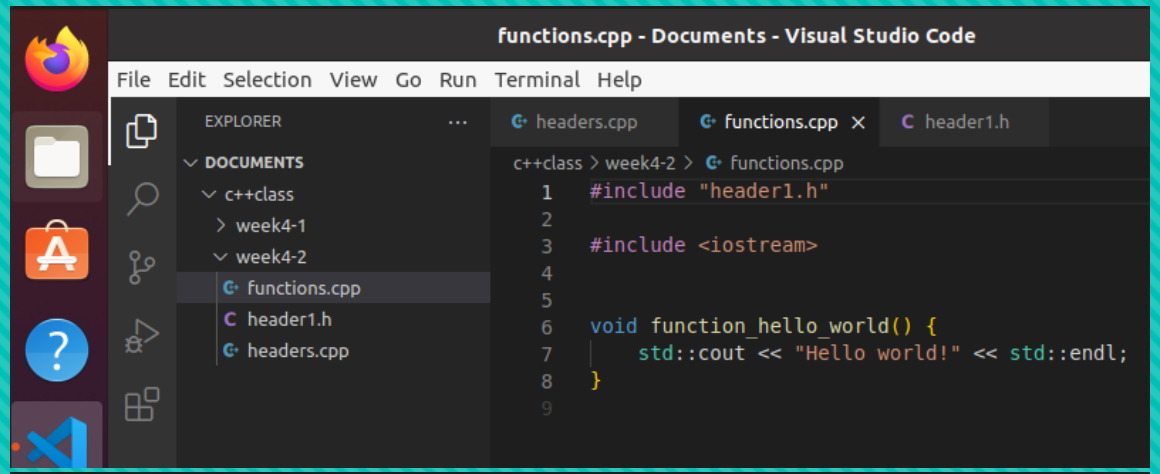
UNLV Howard R. Hughes College of Engineering

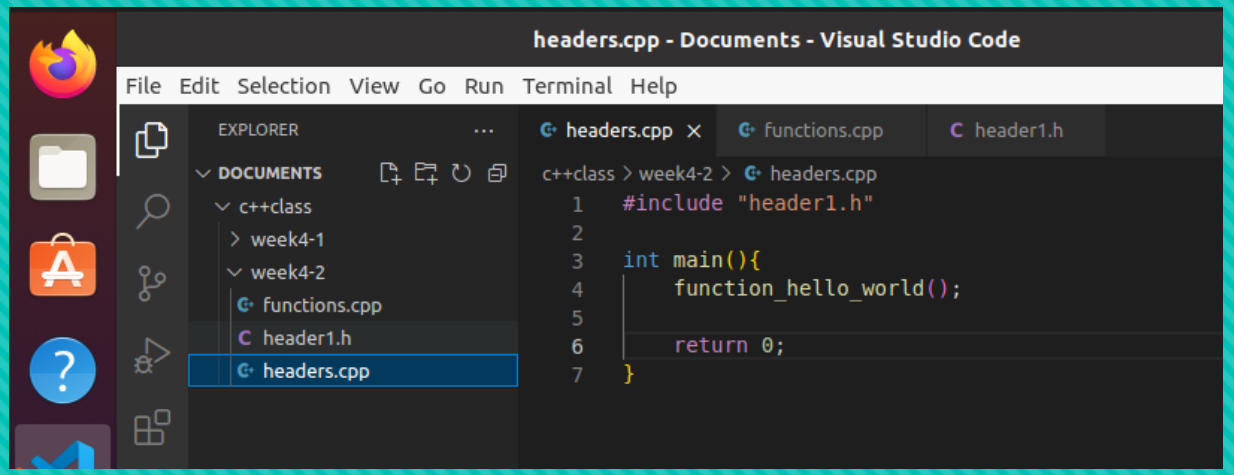
DASL-100.2 C++ Programming and Linux

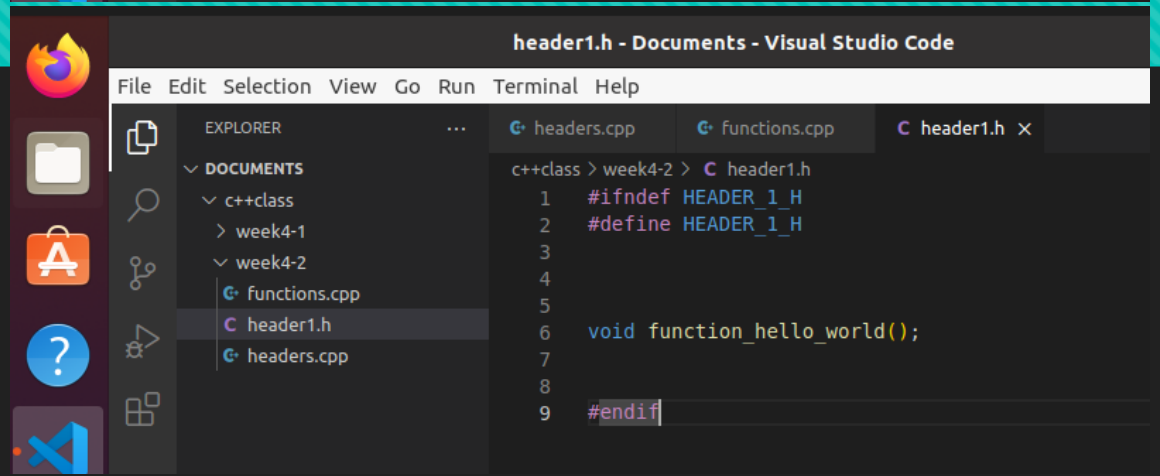
DASL Drones and Autonomous Systems Lab @ UNLV

1. Headers

- Example 2, a header file (.h) with additional source file (.cpp).







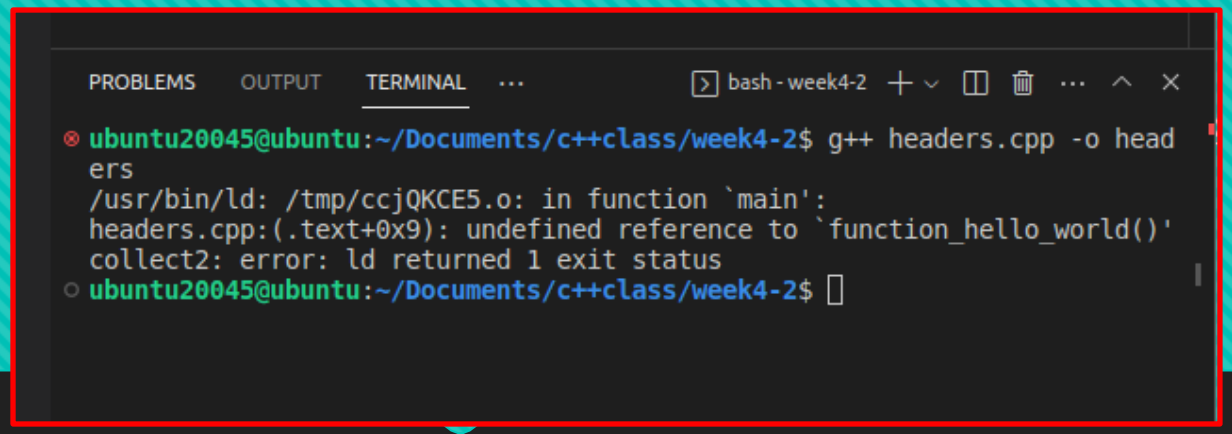
UNLV Howard R. Hughes College of ENGINEERING

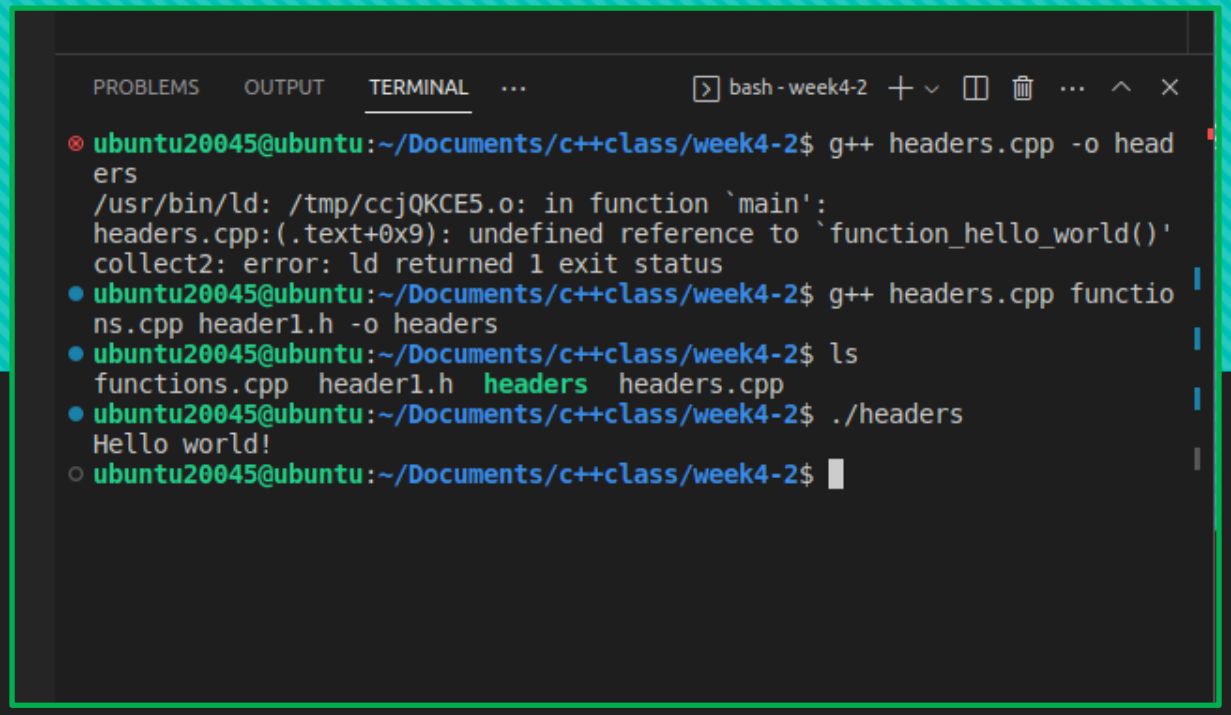
DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

1. Headers

- Note that if we use the traditional way "g++ headers.cpp –o headers" to compile this, we will get an error.
- Because the additional source file (.cpp) is not able to be linked to our executable file.
- Use "g++ headers.cpp functions.cpp header1. h –o headers" instead.





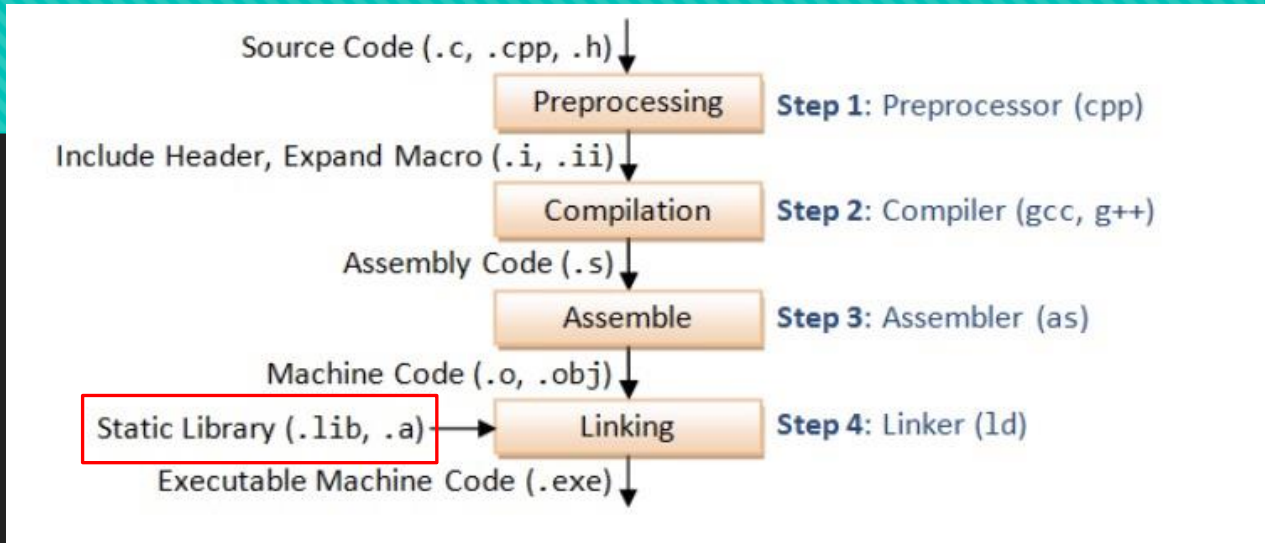
UNLV Howard R. Hughes College of Engineering

DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

1. Libraries

- C++ class libraries are collections of pre-written C++ classes that provide developers with pre-built solutions for common programming tasks. These libraries can help to speed up development time and reduce the amount of code needed to build an application.
- The object library provides compiled functions and data that are linked with your program to produce an executable program. Types of Libraries include:
  - Standard Libraries: provides several generic, function objects, generic strings and streams (including interactive and file I/O), etc.
  - Static Libraries.
  - Dynamic (Shared) Libraries.



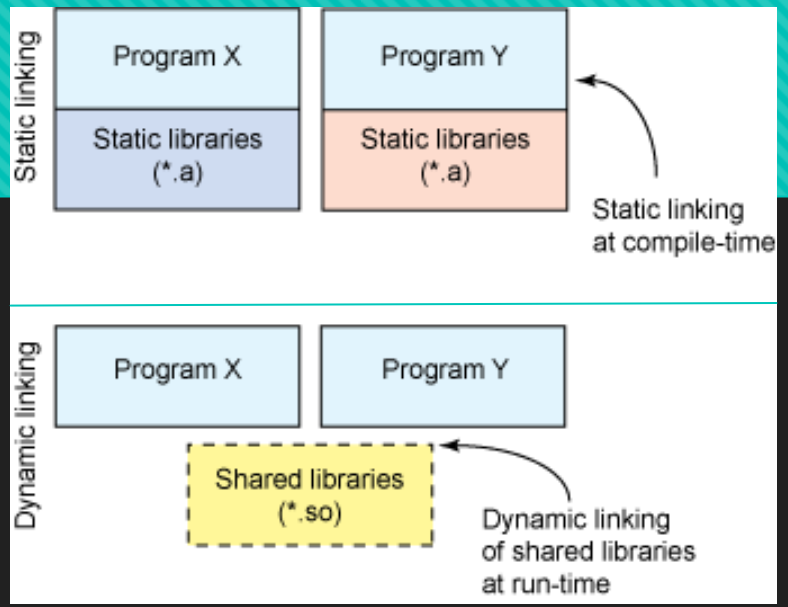
UNLV Howard R. Hughes College of ENGINEERING

DASL-100.2 C++ Programming and Linux

DASL
Drones and Autonomous Systems Lab @ UNLV

1. Libraries

- A **static** library is linked directly into an executable during the build process, resulting in a larger executable file that includes all the library code. The library code is loaded directly into memory at runtime, making it more efficient.
- A **dynamic** library is loaded at runtime by an executable or another shared library, resulting in a smaller executable file that only contains a reference to the library code. The library code is loaded into memory at runtime by the operating system, making it more flexible.
- Overall, **static** libraries are good for small projects where performance is critical and library size is not a concern. **Dynamic** libraries are good for larger projects where library size and flexibility are important.
- **Static** libraries are .a files in Linux and .lib files in Windows.
- **Dynamic** libraries are .so in Linux and .dll in Windows.



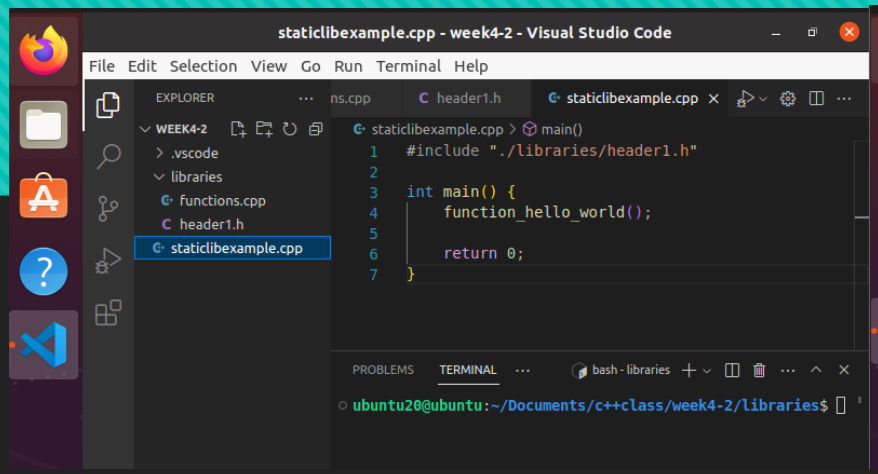
UNLV Howard R. Hughes College of Engineering

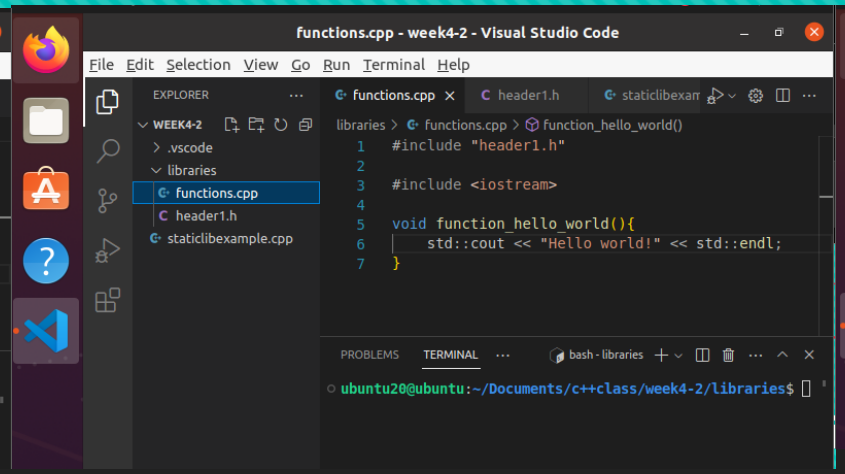
DASL-100.2 C++ Programming and Linux

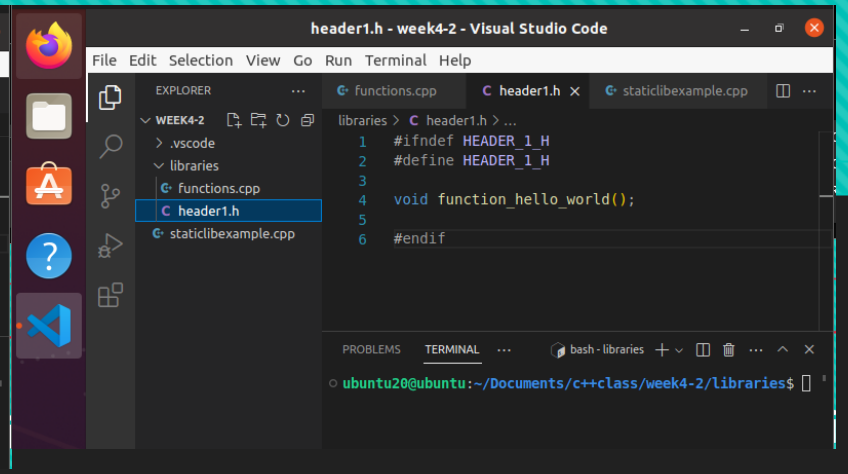
DASL Drones and Autonomous Systems Lab @ UNLV

2. Libraries - Static

- Example 1, static libraries:
  - Step 1, create a main source code “staticlibexample.cpp”.
  - Create a folder name “libraries”.
  - Create functions.cpp file and header1.h file inside libraries folder.







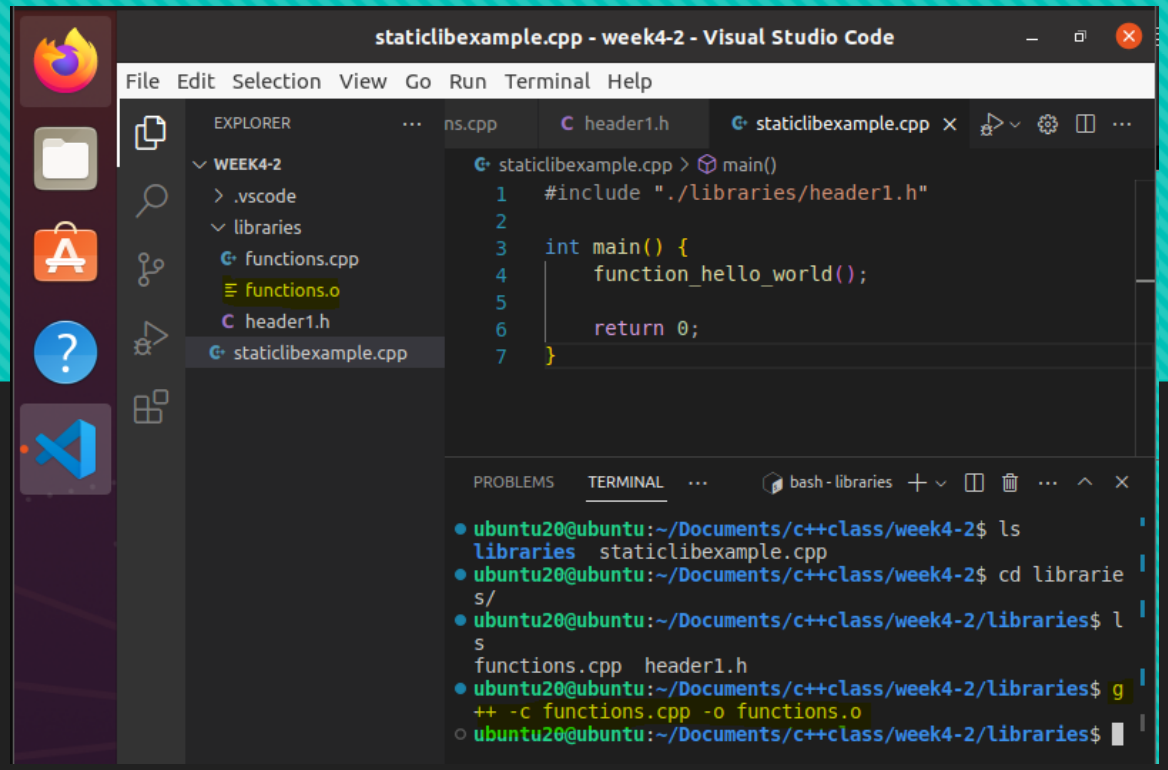
UNLV Howard R. Hughes College of ENGINEERING

DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

2. Libraries - Static

- Example 1, static libraries:
  - Step 2, navigate to libraries folder and generate an object file (.o) from the function.cpp file by using the command: "g++ -c functions.cpp -o functions.o".



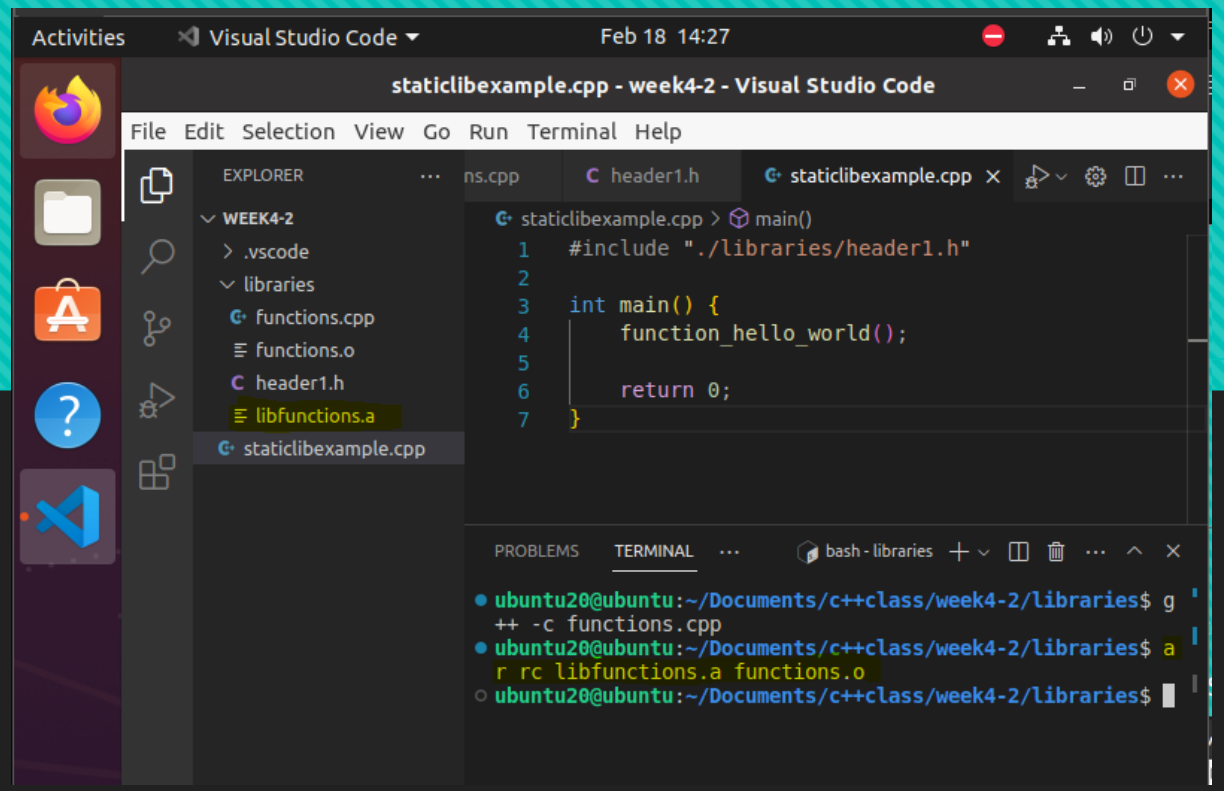
UNLV Howard R. Hughes College of ENGINEERING

DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

2. Libraries - Static

- Example 1, static libraries:
  - Step 3, generate a static library file (.a) to contain all the object files (.o) by using the command: "ar rc libfunctions.a functions.o".



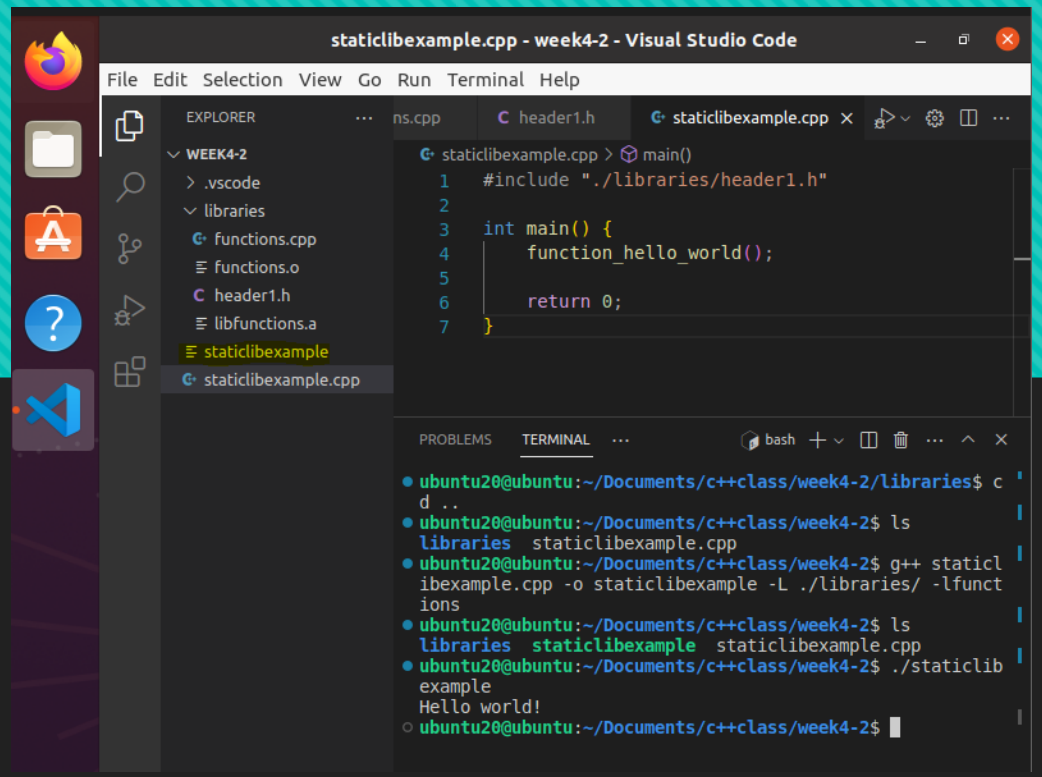
UNLV Howard R. Hughes College of Engineering

DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

2. Libraries - Static

- Example 1, static libraries:
  - Step 4, navigate back to the main source code folder and generate an executable file with the static library file by using the command :"g++ staticlibexample.cpp -o staticlibexample -L ./libraries/ -lfunctions"



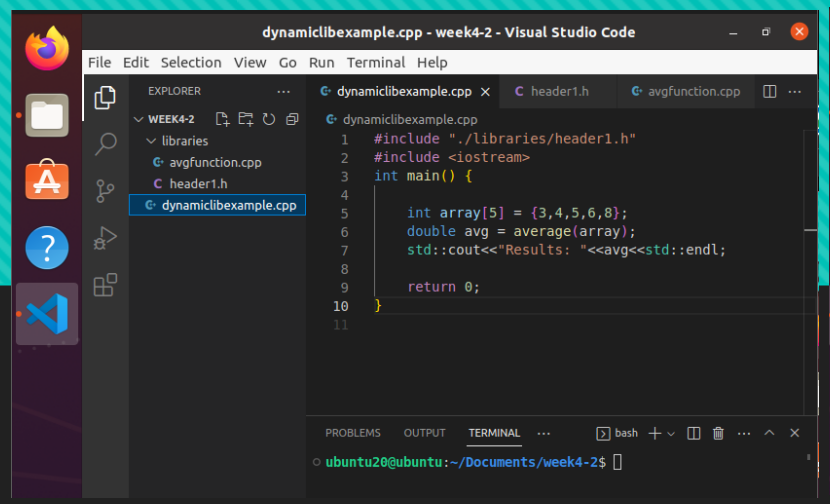
UNLV Howard R. Hughes College of Engineering

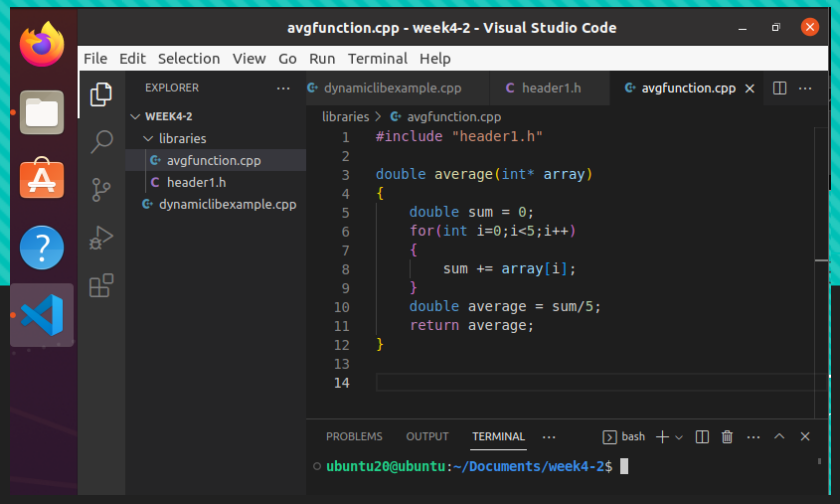
DASL-100.2 C++ Programming and Linux

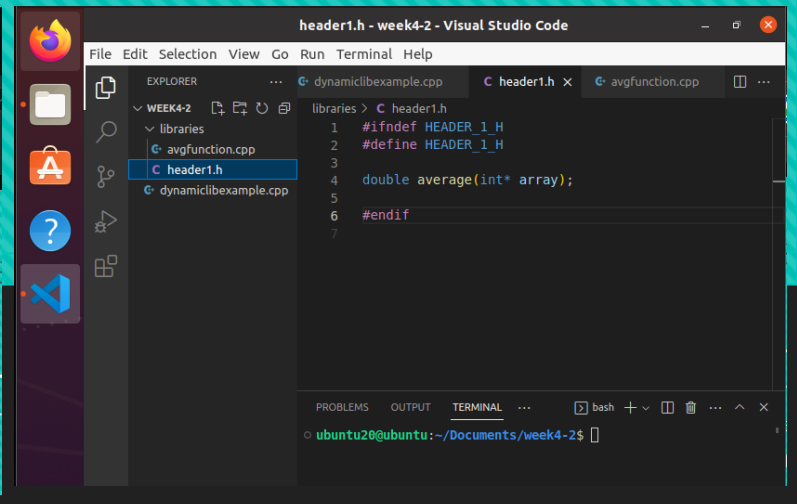
DASL Drones and Autonomous Systems Lab @ UNLV

2. Libraries - Dynamic

- Example 2, dynamic libraries:
  - Step 1, create a main source code “dynamiclibexample.cpp”.
  - Create a folder name “libraries”.
  - Create avgfunction.cpp file and header1.h file inside libraries folder.







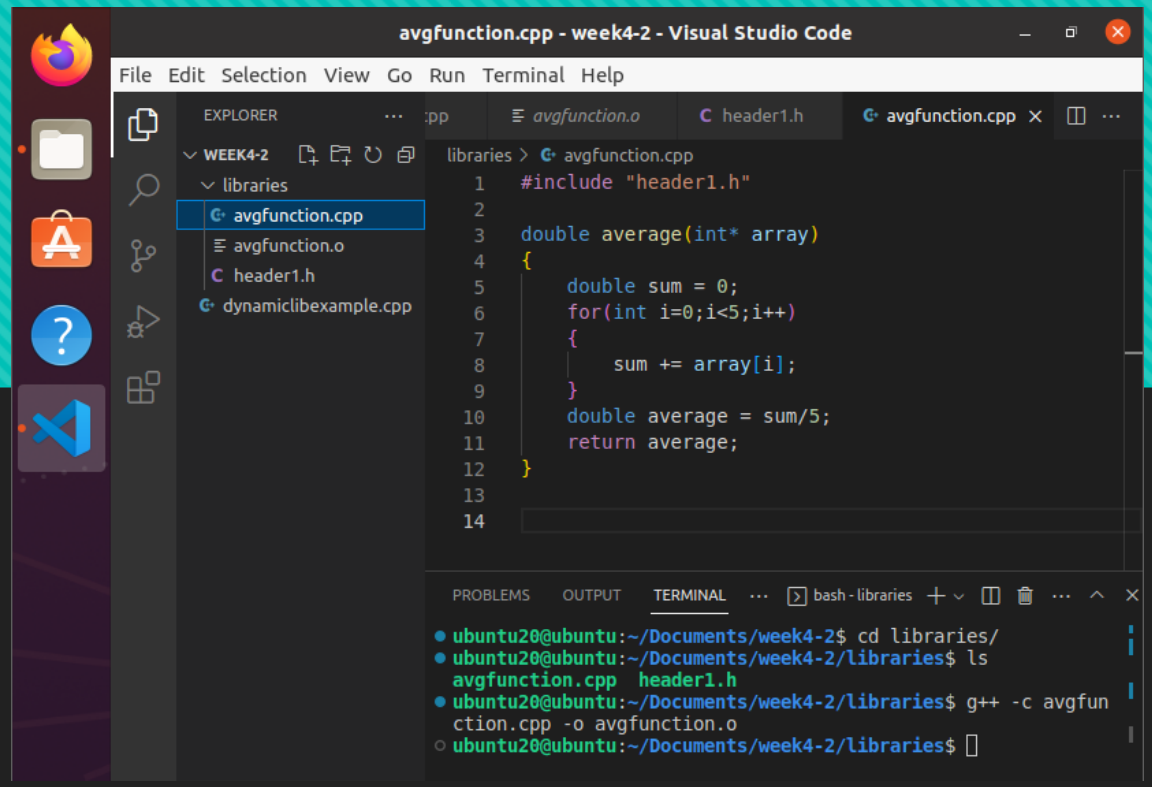
UNLV Howard R. Hughes College of Engineering

DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

2. Libraries - Dynamic

- Example 2, dynamic libraries:
  - Step 2, navigate to libraries folder and generate an object file (.o) from the sum.cpp file by using the command: "g++ -c avgfunction.cpp -o avgfunction.o".



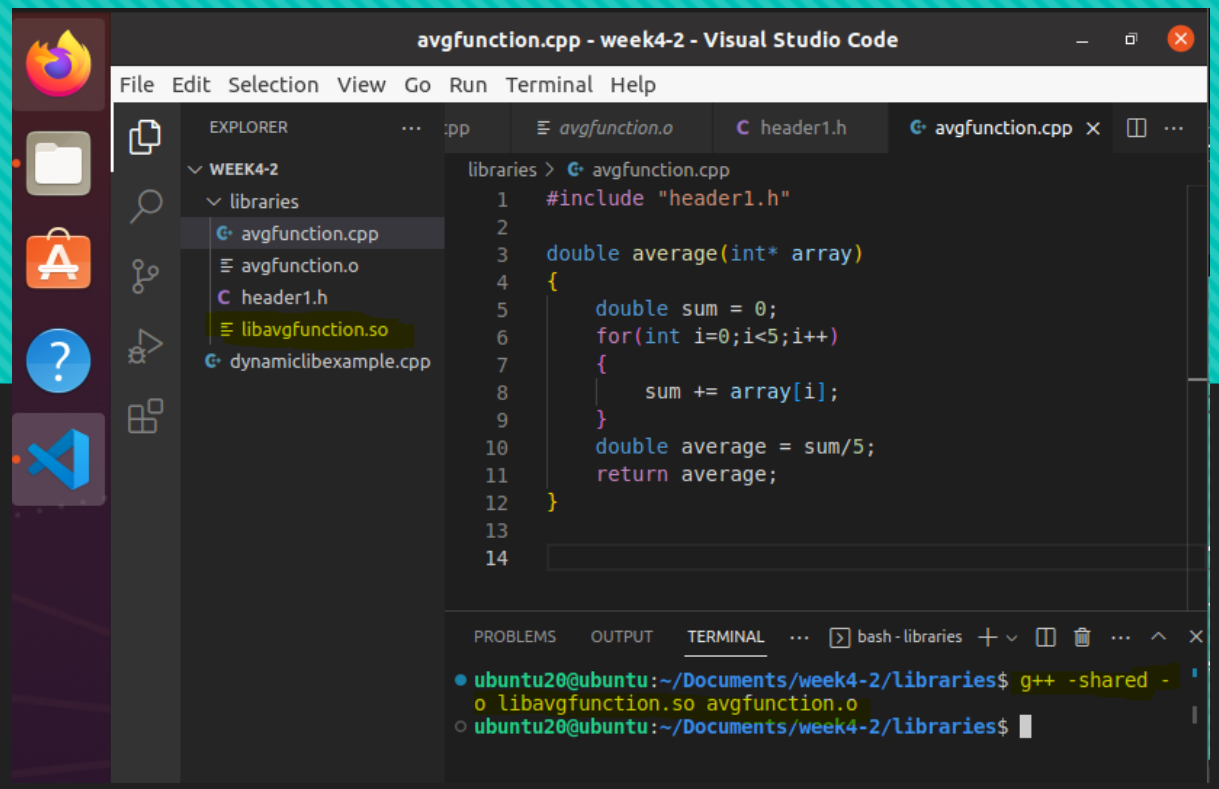
UNLV Howard R. Hughes College of Engineering

DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

2. Libraries - Dynamic

- Example 2, dynamic libraries:
  - Step 3, generate a dynamic library file (.so) to contain all the object files (.o) by using the command: "g++ -shared -o **lib**avgfunction.so avgfunction.o"



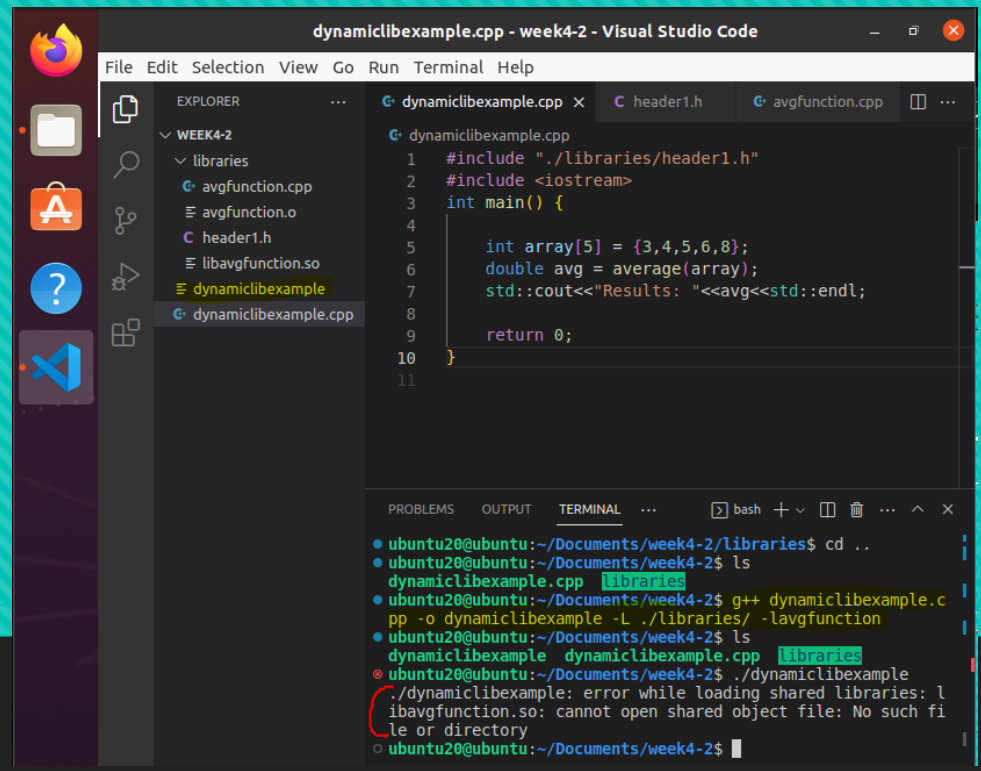
UNLV Howard R. Hughes College of Engineering

DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

2. Libraries - Dynamic

- Example 2, dynamic libraries:
  - Step 4, navigate back to the main source code folder and generate an executable file with the dynamic library file by using the command :"g++ dynamiclibexample.cpp -o dynamiclibexample -L ./libraries/ -lavgfunction"
  - Note that when we try to run the executable file it gives us an error.



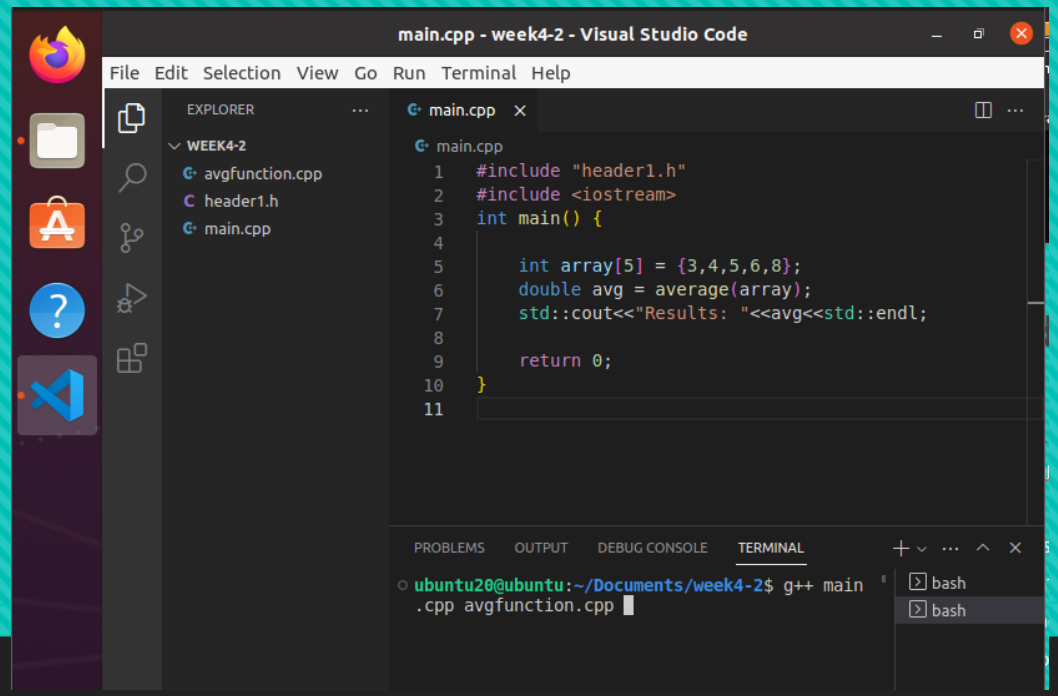
UNLV Howard R. Hughes College of Engineering

DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

3. Make

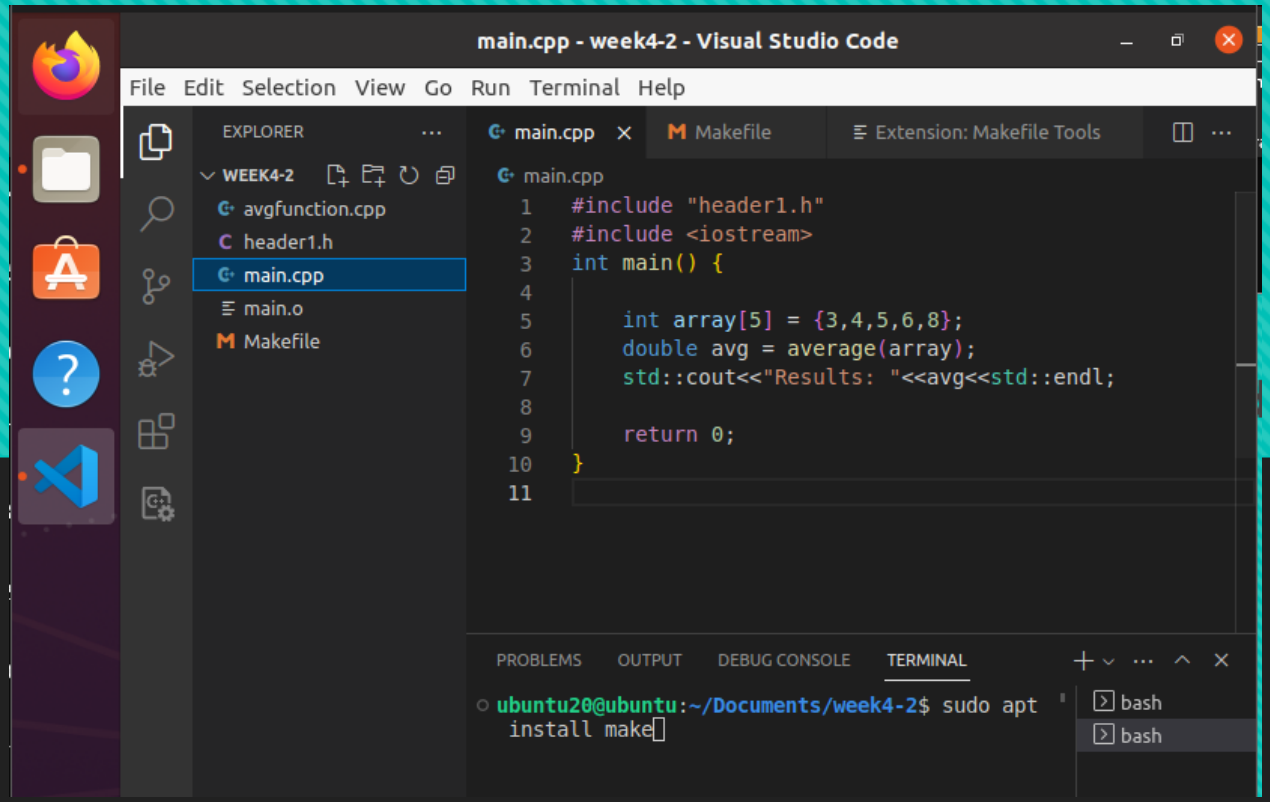
- Often, a program is considered of several files. It is painful to link and complie as we demonstrated in the libraries.
- Make is a build automation tool that is used to build software projects by specifying a set of rules and dependencies between files. Make reads a file called "Makefile" that contains the rules and dependencies for the project. The Makefile specifies the targets, dependencies, and commands needed to build the software.
- We can simply do “g++ main.cpp avgfunction.cpp” but if there are a lot of files, we would have to incorporate them all in the command line.

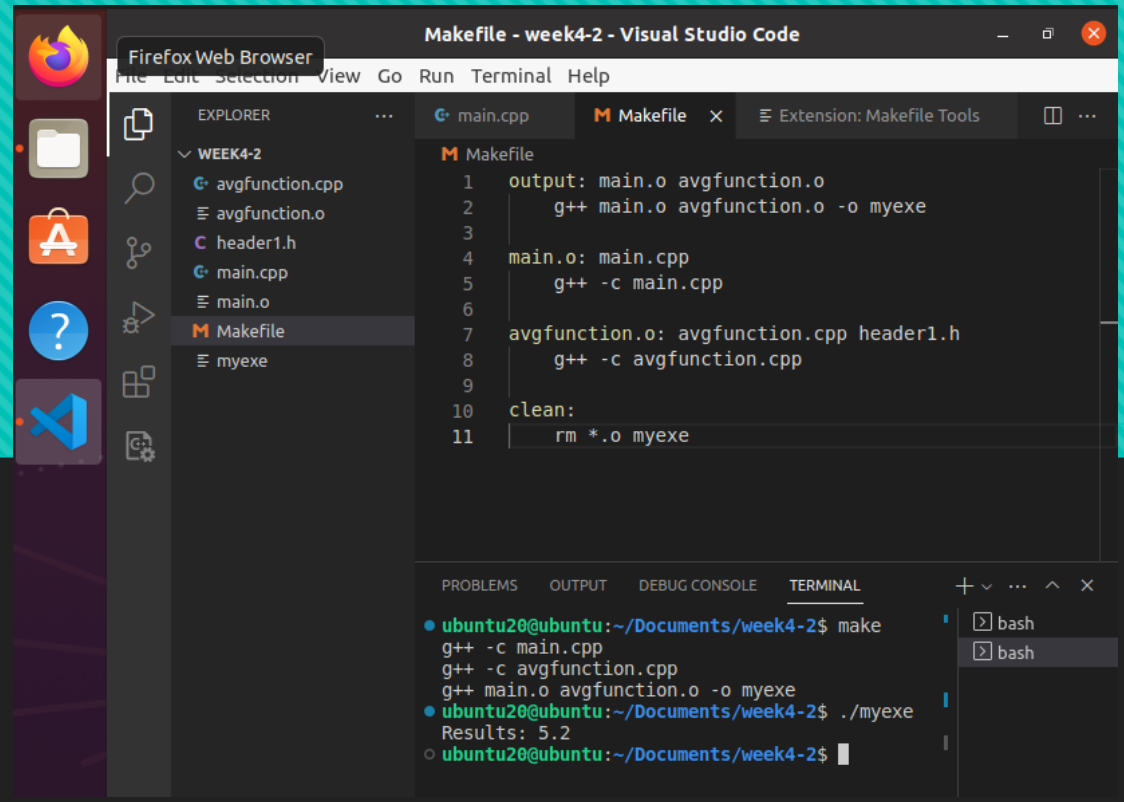


DASL-100.2 C++ Programming and Linux

3. Make

- Install make, type in command : "sudo apt install make"
- Create a Makefile by click new file and name it as "Makefile"





UNLV Howard R. Hughes College of ENGINEERING

DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

3. CMake

- Similar to Make, CMake is a cross-platform build system that is used to manage the build process for C++ projects. CMake generates native build files for various platforms such as Unix, Windows, and macOS. The build process for C++ projects typically involves compiling the source code, linking object files, and generating executables or libraries.
- CMake uses a file called "CMakeLists.txt" to define the build process for a project. The CMakeLists.txt file specifies the source files, libraries, and dependencies for the project, and defines how the project should be built. CMake can also be used to generate project files for various integrated development environments (IDEs), such as Visual Studio, Eclipse, and Xcode.

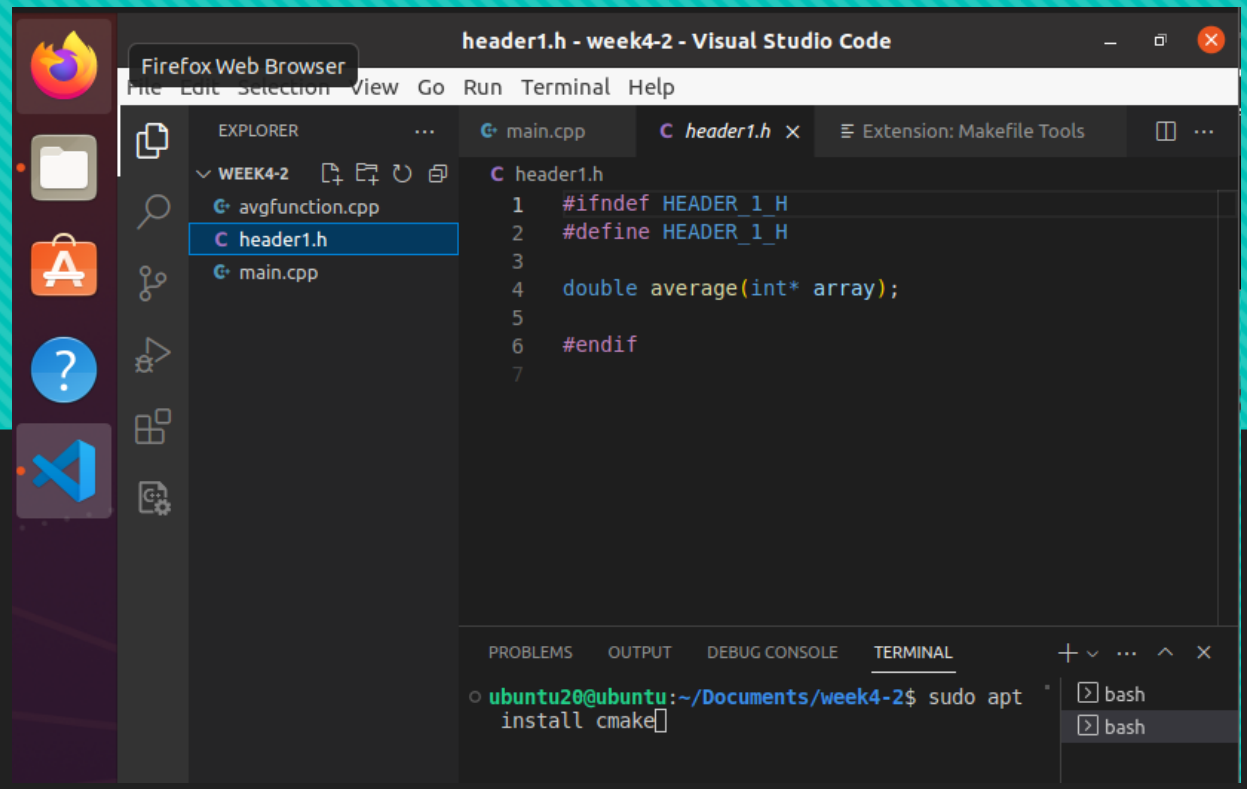
UNLV Howard R. Hughes College of Engineering

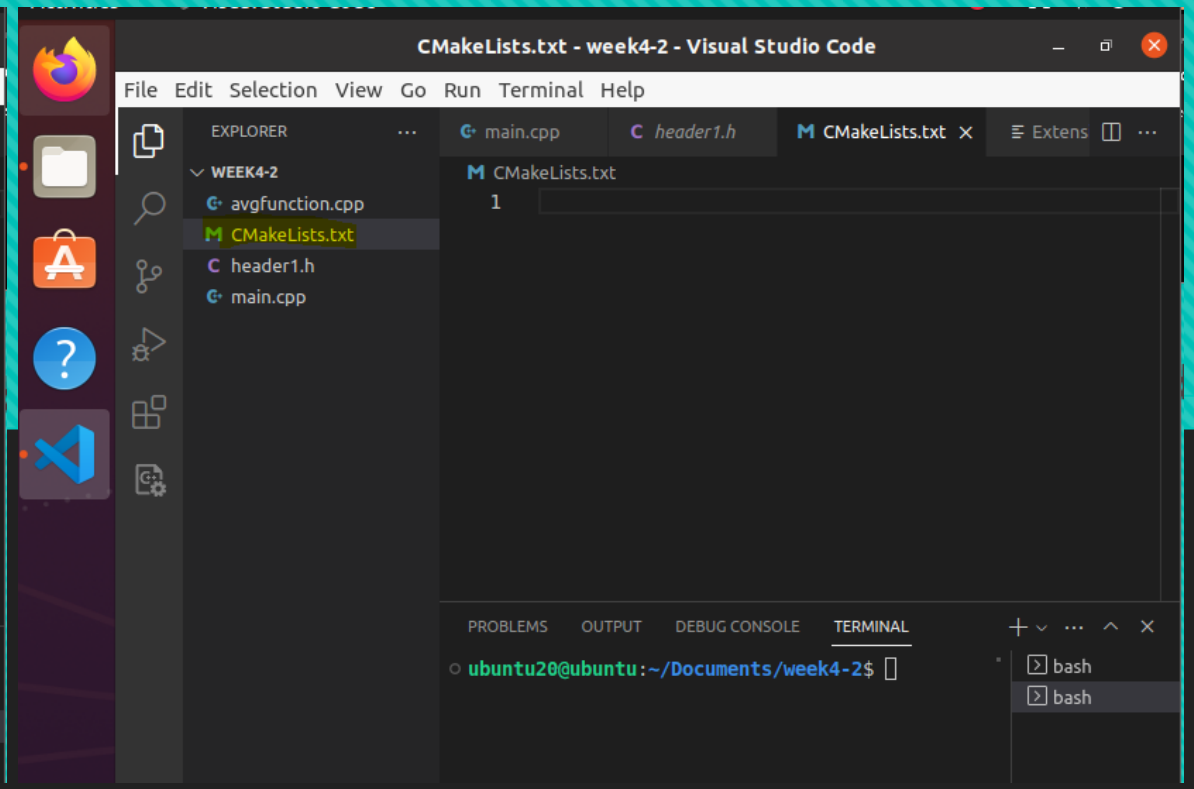
DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

3. CMake

- Install cmake, type in the command "sudo apt install cmake".
- Create a new file "CMakeLists.txt".





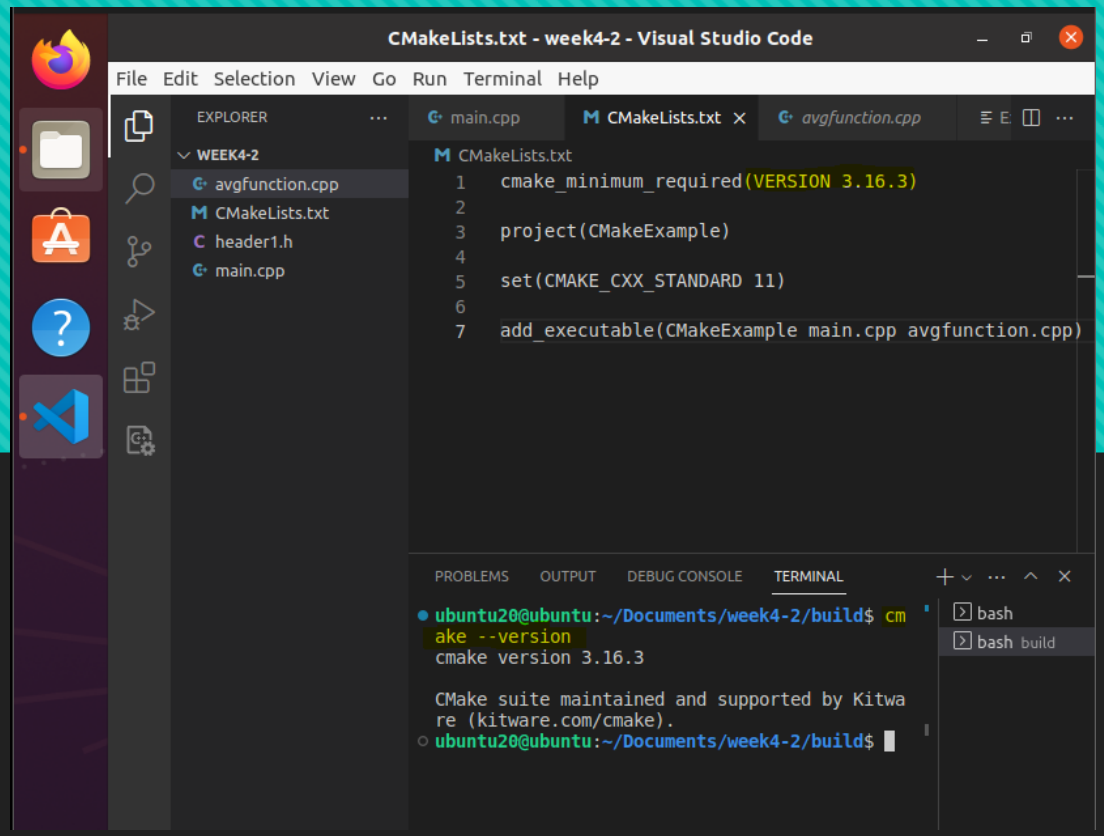
UNLV Howard R. Hughes College of Engineering

DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

3. CMake

- Check CMake version by type in command : "cmake --version"
- Type in the CMake file as follow:



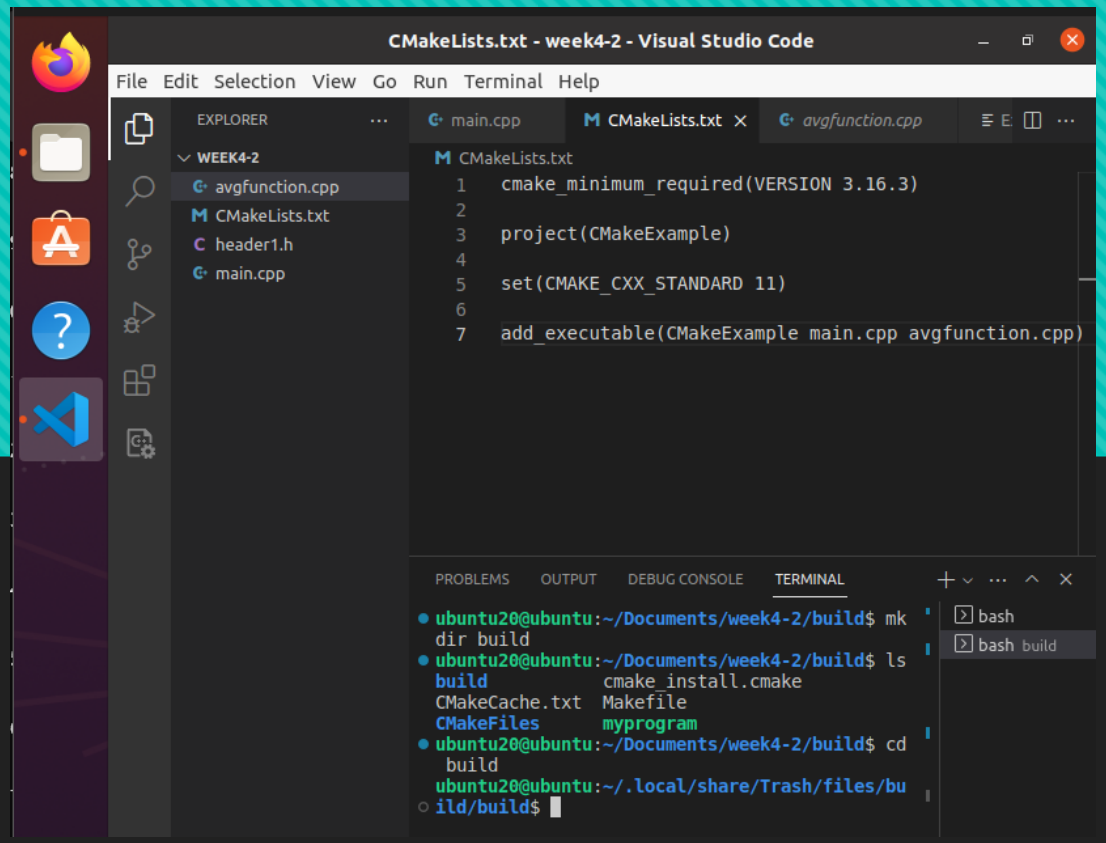
UNLV Howard R. Hughes College of ENGINEERING

DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

3. CMake

- Create a build folder by type in command : "mkdir build"
- Navigate to build folder.



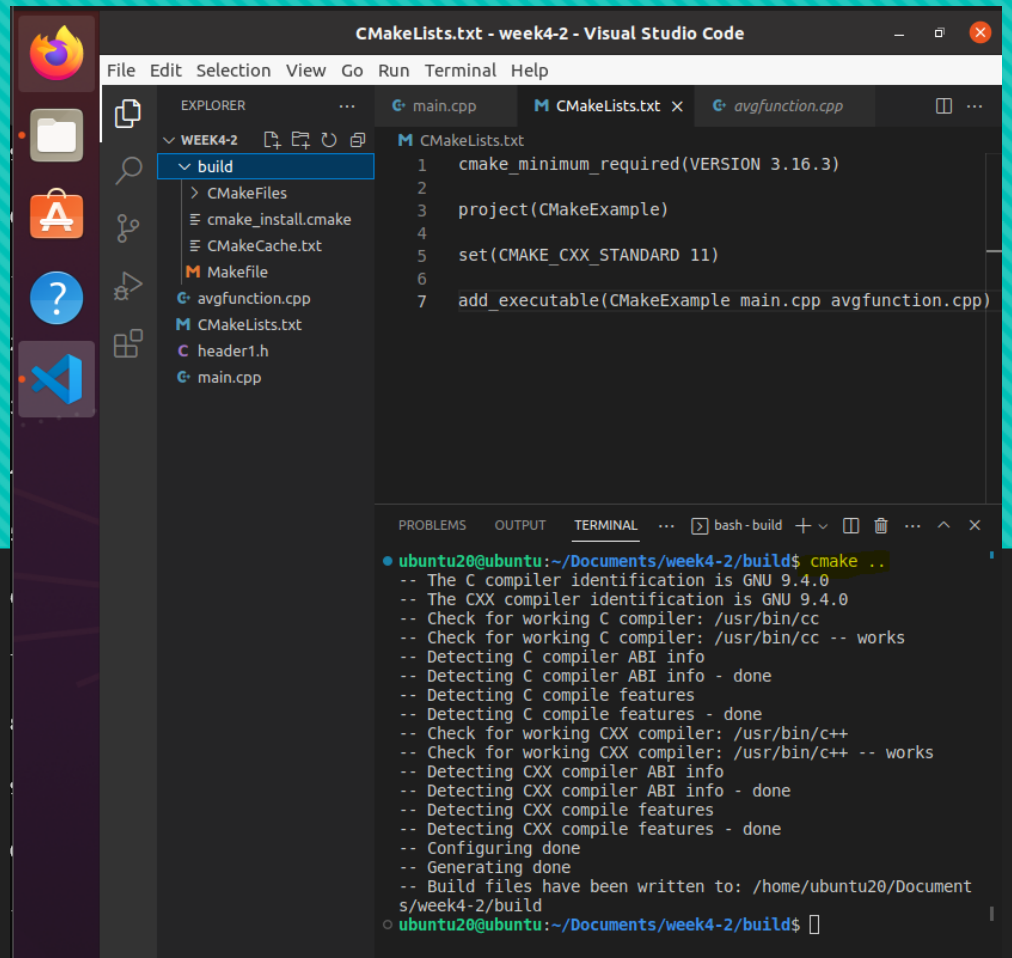
UNLV Howard R. Hughes College of ENGINEERING

DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

3. CMake

- Type in the command : "cmake .."



UNLV Howard R. Hughes College of ENGINEERING

DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

3. CMake

- Now, to generate executable file, type in the command : "make".
- Executable file "CMakeExample" is generated.