

§ 1 Introduction

ENGG1111

Computer Programming and Applications

Dirk Schnieders

Outline

Knowing how to **use** a computer is important



Learning how to **program** a computer will empower you with extraordinary problem-solving skills

14/9/2017

ENGG1111

3

Operating System

- You do not normally talk to the computer directly
 - You communicate through an operating system
- We will be using Ubuntu in our class
 - Do not use any other operating system



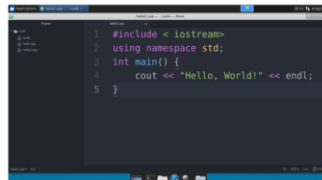
14/9/2017

ENGG1111

19

Syntax Error

- Sometimes, a program will not build
- This is often due to syntax error(s)
- Example:



14/9/2017

41

Computer

- It is not only referring to your personal computer (PC), it can be a ...

- fridge
- vending machine
- car
- quadcopter
- fan
- ...



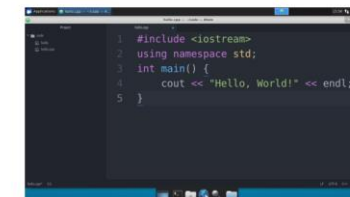
14/9/2017

ENGG1111

12

First Program

- Consider the following program written in C++

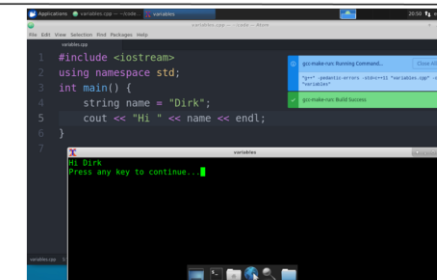


14/9/2017

ENGG1111

33

Sneak Peek: Variables



14/9/2017

ENGG1111

54

Knowing how to **use** a computer is important



Learning how to **program** a computer will empower you with extraordinary problem-solving skills

I don't want to major in computer science.

Why do I have to learn to program?



Learn to Code - Code to Learn

- In the process of learning to code (computer programming), you learn many other things
 - You are coding to learn
- In addition to learning mathematical and computational ideas you are also learning strategies for solving problems, designing projects, and communicating ideas
- These skills are useful not just for computer scientists but for everyone

Based on work by Mitchel Resnick

If you like solving problems,
there is a good chance that you will love programming

Coding

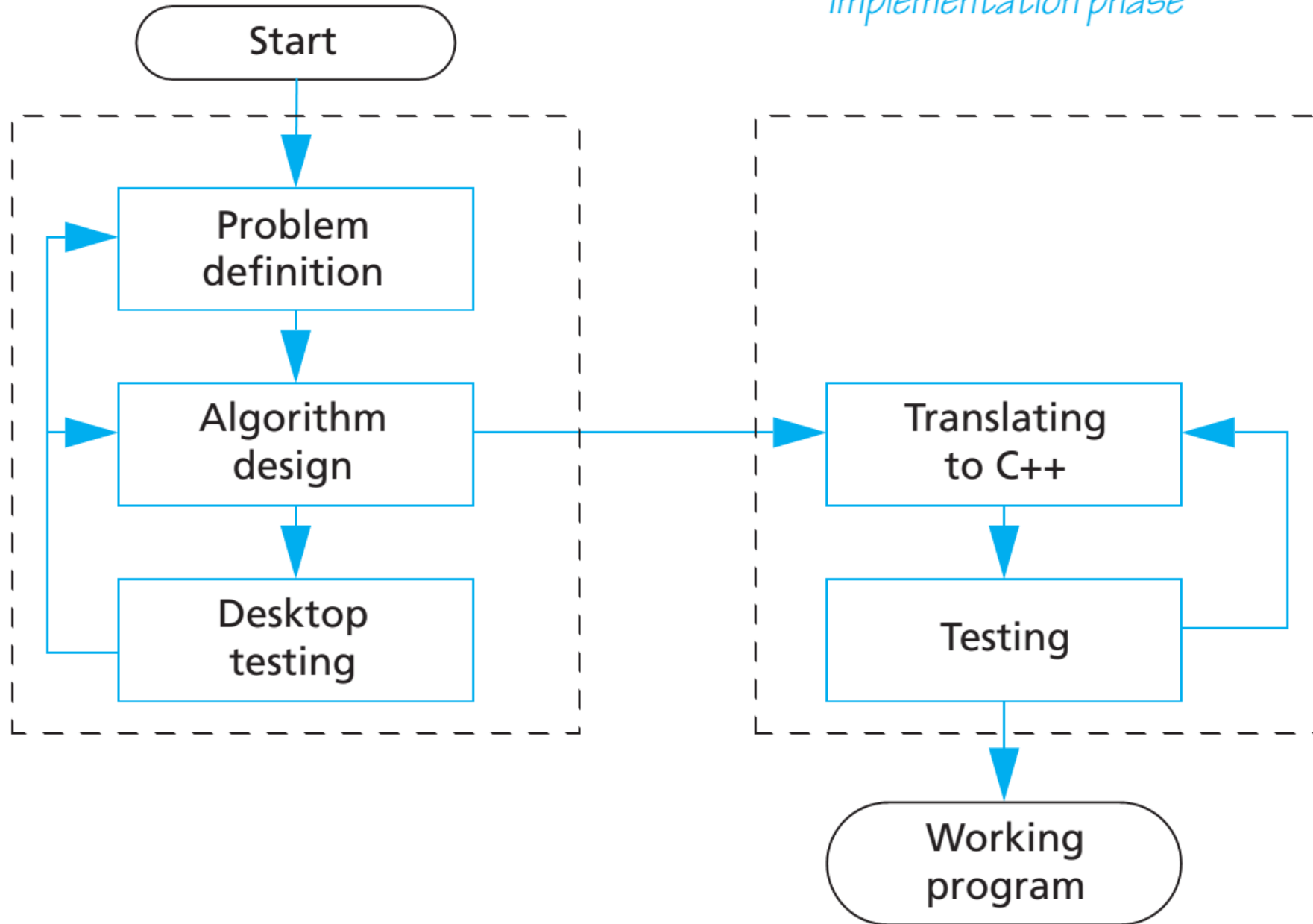
- Coding involves the following steps
 1. Problem definition
Examine, analyze and understand the problem
 2. Algorithm design
Devise a solution in the form of an algorithm
 3. Desktop testing
Check that the solution works under a range of conditions
 4. Translate to code
Translate the solution into a programming language and write the code
 5. Testing
Test your code

Programming and Problem-solving

When learning your first programming language, it is easy to get the impression that the hard part of solving a problem on a computer is translating your ideas into the specific language that will be fed into the computer. This definitely is not the case. The most difficult part of solving a problem on a computer is discovering the method of solution. After you come up with a method of solution, it is routine to translate your method into the required language, be it C++ or some other programming language. It is therefore helpful to temporarily ignore the programming language and to concentrate instead on formulating the steps of the solution and writing them down in plain English, as if the instructions were to be given to a human being rather than a computer. A sequence of instructions expressed in this way is frequently referred to as an *algorithm*.

Problem-solving phase

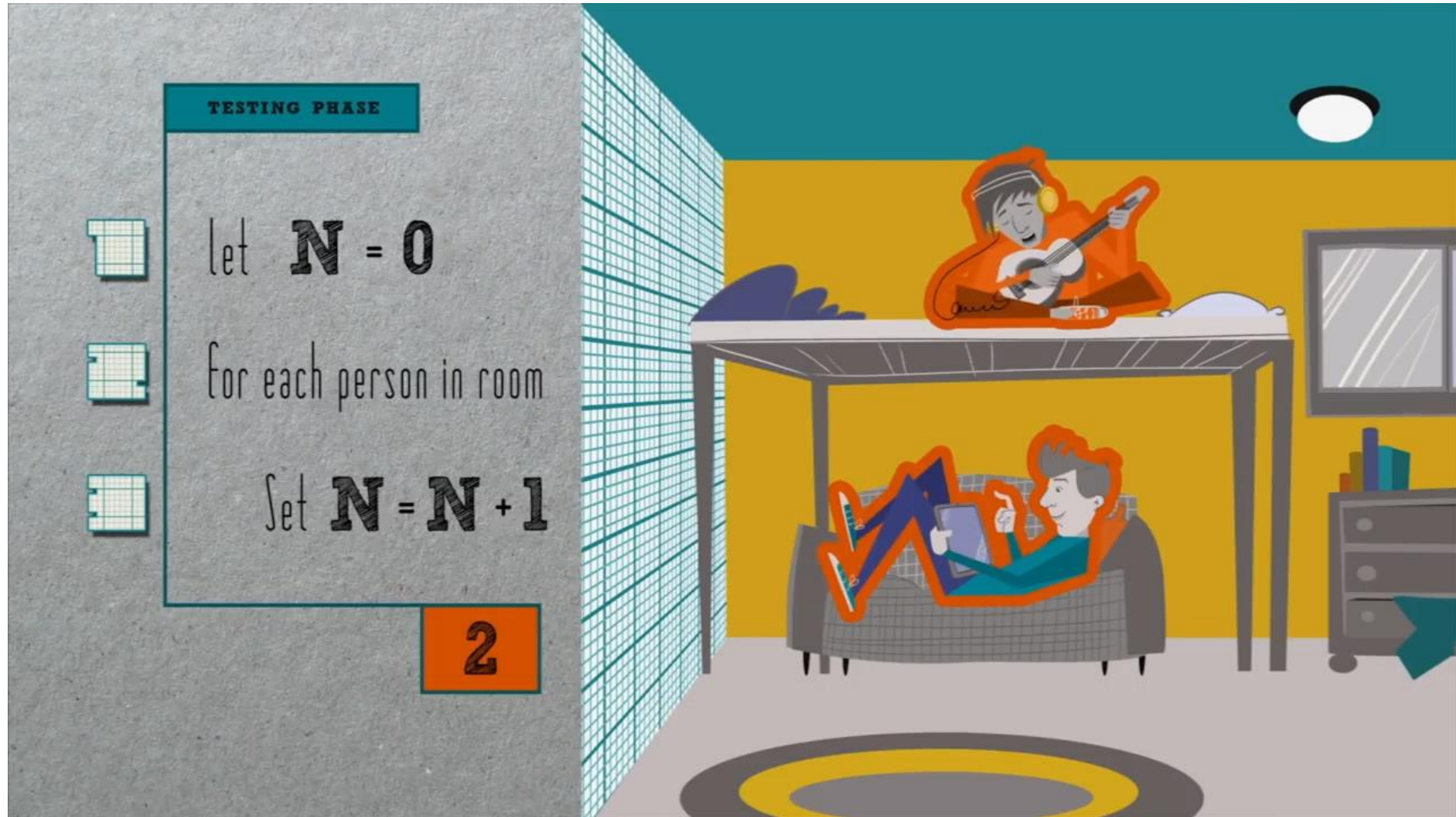
Implementation phase



Tip

- Do not skip any steps
 - Even if you initially feel you are solving a trivial problem

What is an Algorithm?



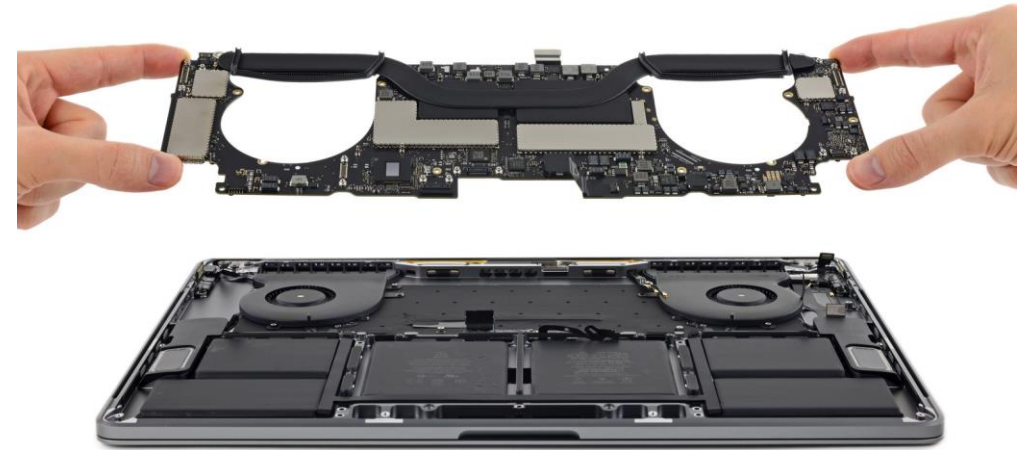
Computer

- It is not only referring to your personal computer (PC), it can be a ...
 - fridge
 - vending machine
 - car
 - quadcopter
 - fan
 - ...



Hardware

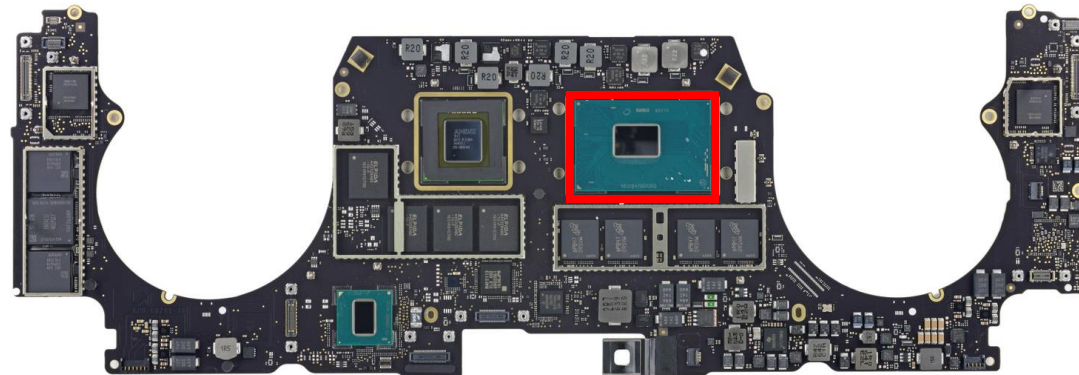
- The actual physical parts that make up a computer



2017 MacBook Pro 15"

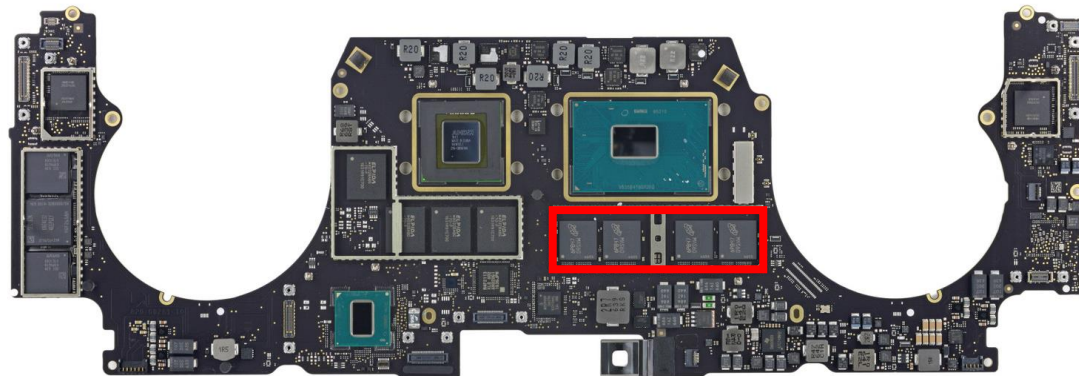
Central Processing Unit (CPU)

- Follows instructions in a program
- Performs simple tasks like
 - Addition, subtraction, multiplication and division
 - Move data from one memory location to another



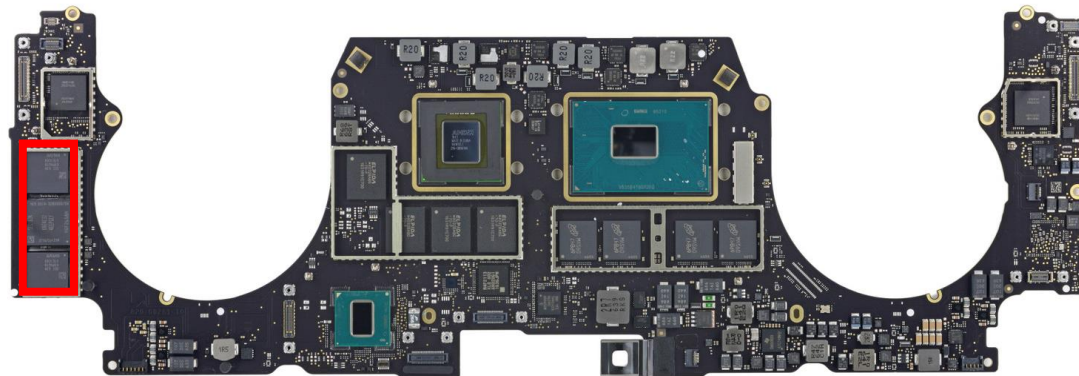
Main memory

- Long list of memory cells
- Each memory cell has an address (a number) that gives its position in the memory
- Each memory cell consists of 8 digits (1 byte)
- Volatile – information will be lost when the computer is switched off



Secondary memory

- Non-volatile – information will not be lost even after the computer is switched off
- For permanent storage of data in units of files
 - E.g., SSD, HDD



Input Device

- Any device that allows a user to input information



Output Device

- Any device that allows a computer to communicate information to the user



Operating System

- You do not normally talk to the computer directly
 - You communicate through an operating system
- We will be using Ubuntu in our class
 - Do not use any other operating system



Computer Program

- A computer program is a sequence of instructions for a computer to follow
 - Written to perform a specified task with a computer



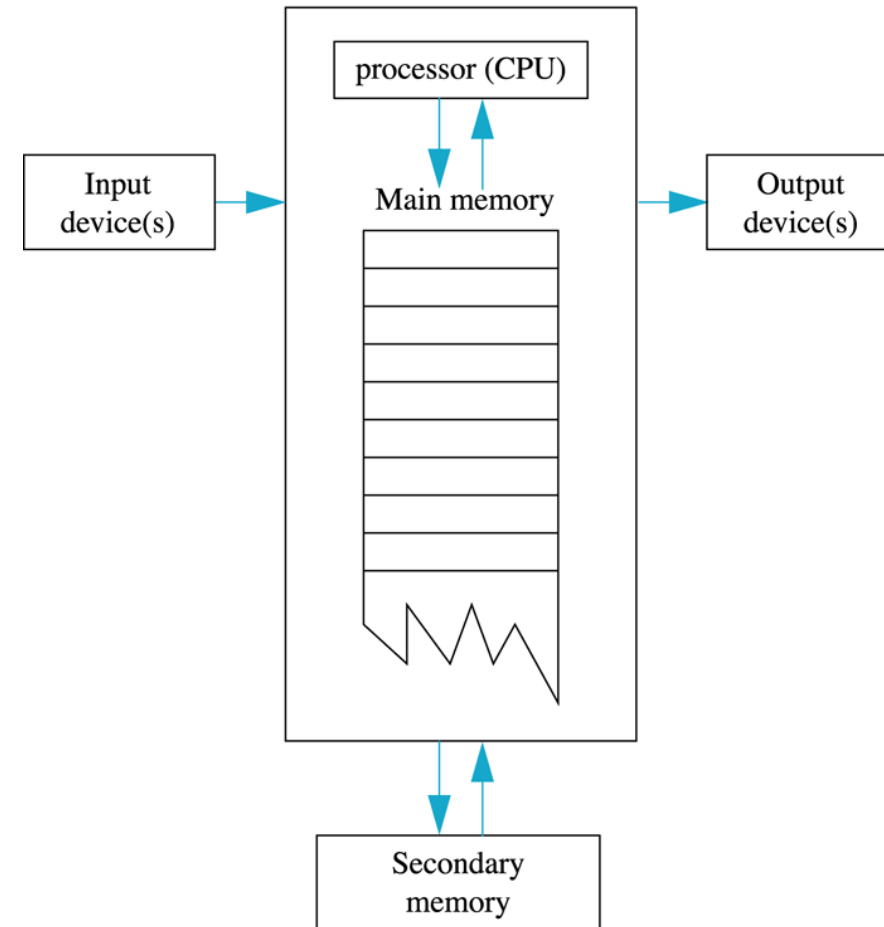
HKUmoodle



Computer Organization

- When a program is being executed
 - The program is loaded into the main memory
 - The CPU reads the program instructions and process the program data
 - The output of the program can be written to the main memory, secondary memory, or displayed through the output device(s)

Main Components of a Computer



Languages

Machine Language

01101001
10101011

Assembly Language

LOAD A
ADD B
STORE C

High-level
Programming Language

$C = A + B$

Human Language

Write a program that computes
the addition of two integers

Machine Language

- The CPU can only understand machine language (machine code)
- Low level instruction perform very specific tasks
 - E.g., a load, a jump, or an ALU operation
- Every processor or processor family has its own machine code instruction set
- All practical programs today are written in higher-level languages or assembly language

Assembly Language

- A low-level programming language
- Defined by the hardware manufacturer, so every kind of computer has its own unique assembler language
- Need to translate to machine code for CPU to execute
 - The translation program is called an assembler

- [illegible]

C++

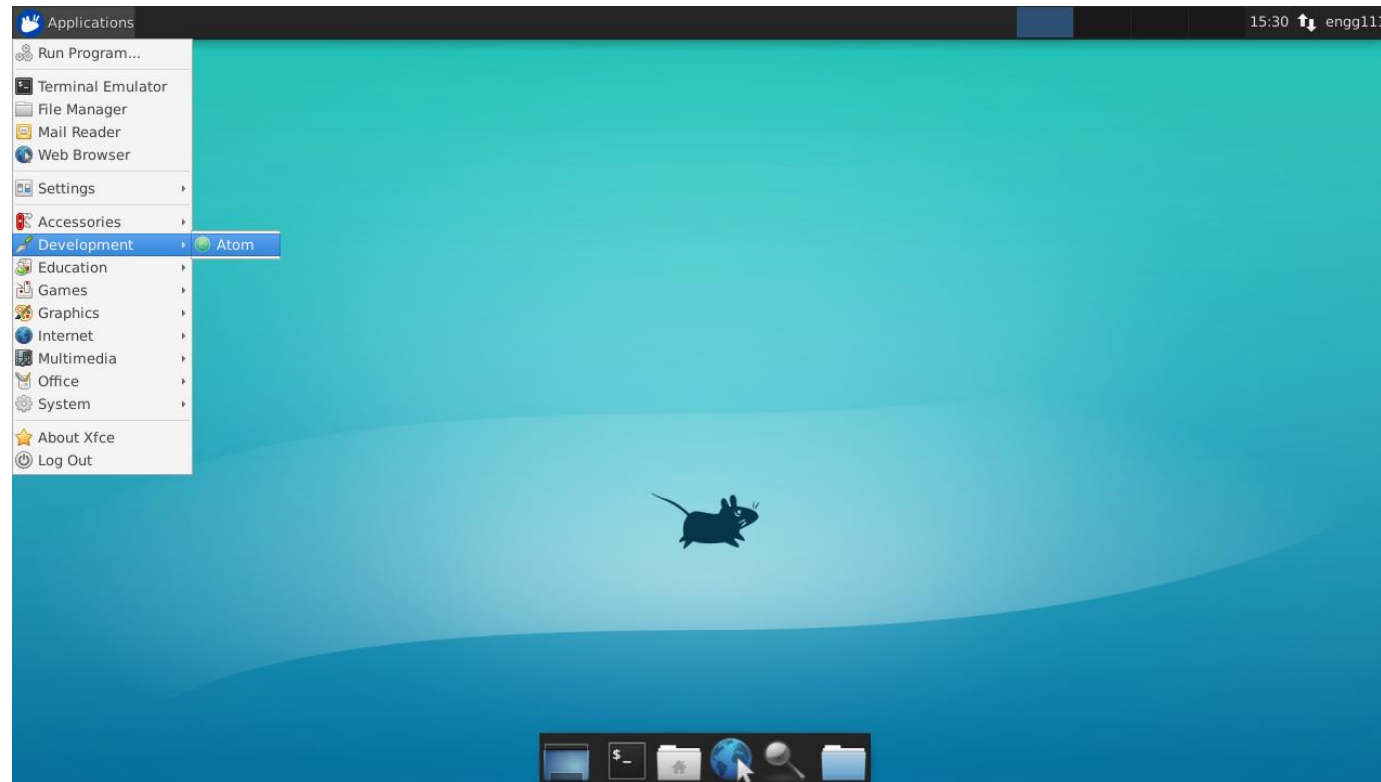
- In this course we will be using the C++ programming language
- C++ is a popular, industrial-strength language
- If you master general programming concepts using C++, you can apply them to many other languages

Text Editor

- A word-processor like program used for typing in and editing the source code of a program
 - E.g., Notepad, Emacs, vi, pico, Sublime Text, ATOM
- C++ files are saved as *.cpp
 - Always save your C++ file with the ending cpp

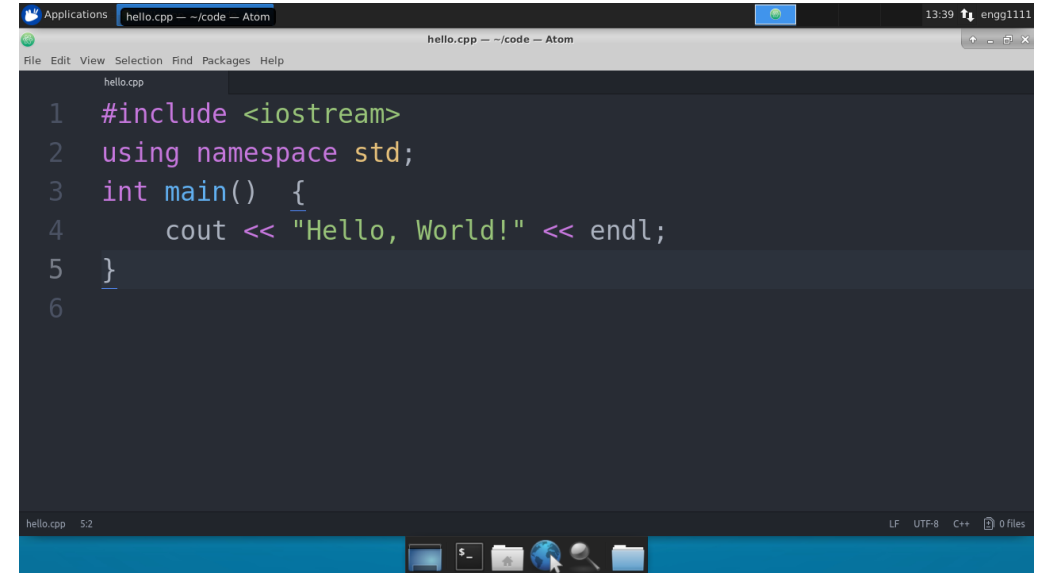
Text Editor

- In this course we are using ATOM
- In Ubuntu, you can start is as follows



Text Editor - Syntax Highlighting

- ATOM supports syntax highlighting for C++
 - I.e., different colours are assigned to keywords, literals, variables, etc.
- Note that the highlighting does not affect the meaning of the source itself and is not saved with the file
- If you save your file as *.cpp, ATOM create the syntax highlighting automatically



The screenshot shows the ATOM text editor interface. The title bar indicates the file is 'hello.cpp' and the editor is running on a Mac. The menu bar includes 'File', 'Edit', 'View', 'Selection', 'Find', 'Packages', and 'Help'. The code in the editor is as follows:

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     cout << "Hello, World!" << endl;
5 }
6
```

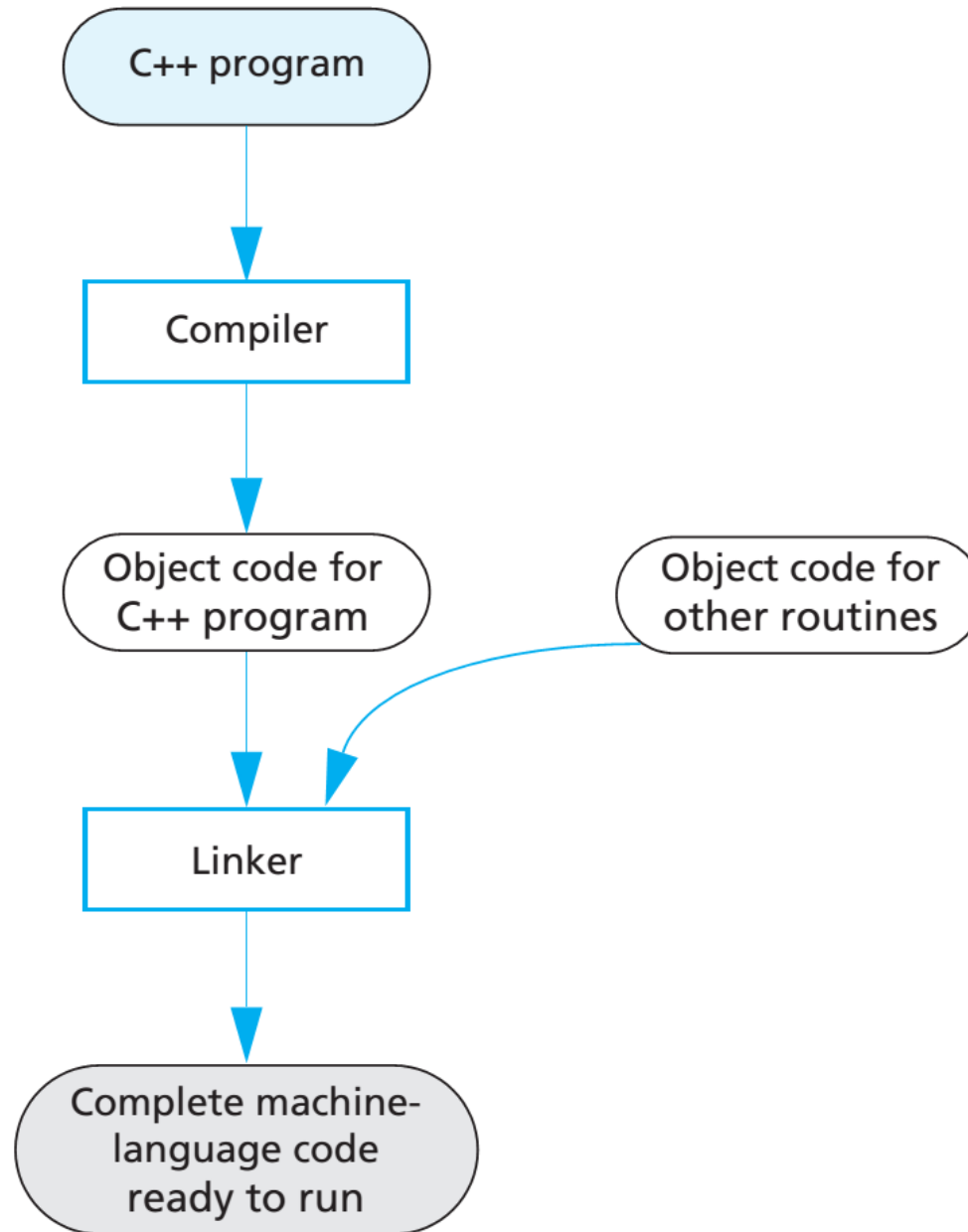
The code is syntax-highlighted: keywords like 'include', 'using', 'namespace', 'int', 'main', 'cout', and 'endl' are in blue; the string literal 'Hello, World!' is in green; and the header file '<iostream>' is in orange. The status bar at the bottom shows 'hello.cpp 5/2', 'LF UTF-8 C++', and '0 files'.

Compiler

- Computers can only understand programs written in low-level languages (simple instructions), which may differ from one kind of computers to another
- A compiler is a program that translates a high-level language program, such as a C++ program, into a machine-language program that a computer can directly understand and execute
- The input program is called the source program or source code, and the translated version is called the object program or object code

Linker

- When writing a program, we can make use of some routines that have been written by someone else (e.g., routines for handling I/O, trigonometric functions)
- Such routines have been compiled into object code and a collection of such pre-compiled routines is called a library
- A linker is a program that combines (links) the object code of your program with those of the routines from a library to produce an executable
- We are going to use g++ in this course as the driver for compiler and linker



First Program

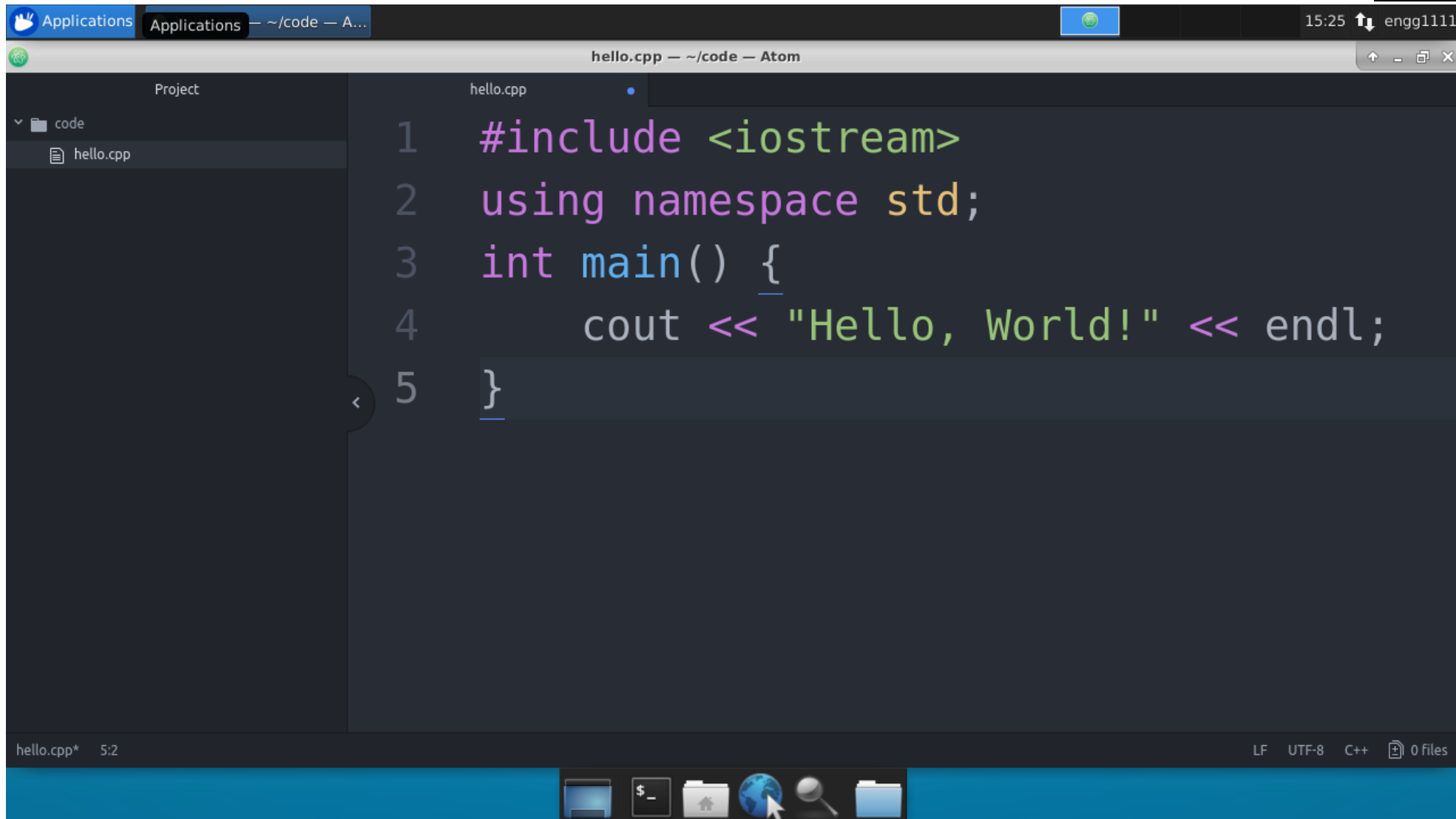
- Consider the following program written in C++

A screenshot of the Atom code editor interface. The window title is 'hello.cpp - ~/code - Atom'. The left sidebar shows a project view with a folder 'code' containing files 'hello' and 'hello.cpp'. The main editor area displays the following C++ code:

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      cout << "Hello, World!" << endl;
5  }
```

The status bar at the bottom indicates 'hello.cpp* 5:2', 'LF', 'UTF-8', 'C++', and '0 files'. The system tray at the bottom shows icons for a terminal, file manager, and other applications.

Building an Executable

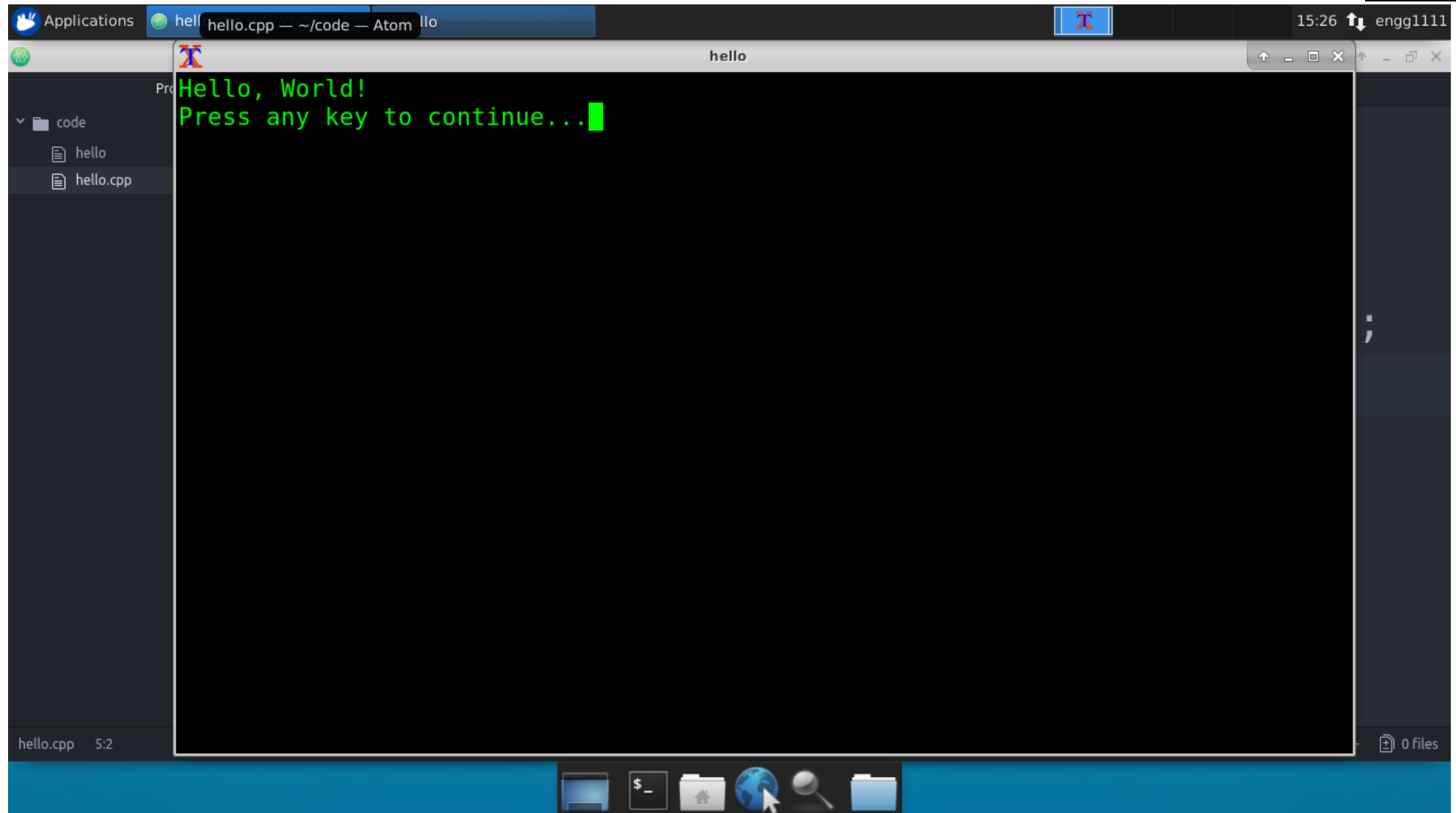


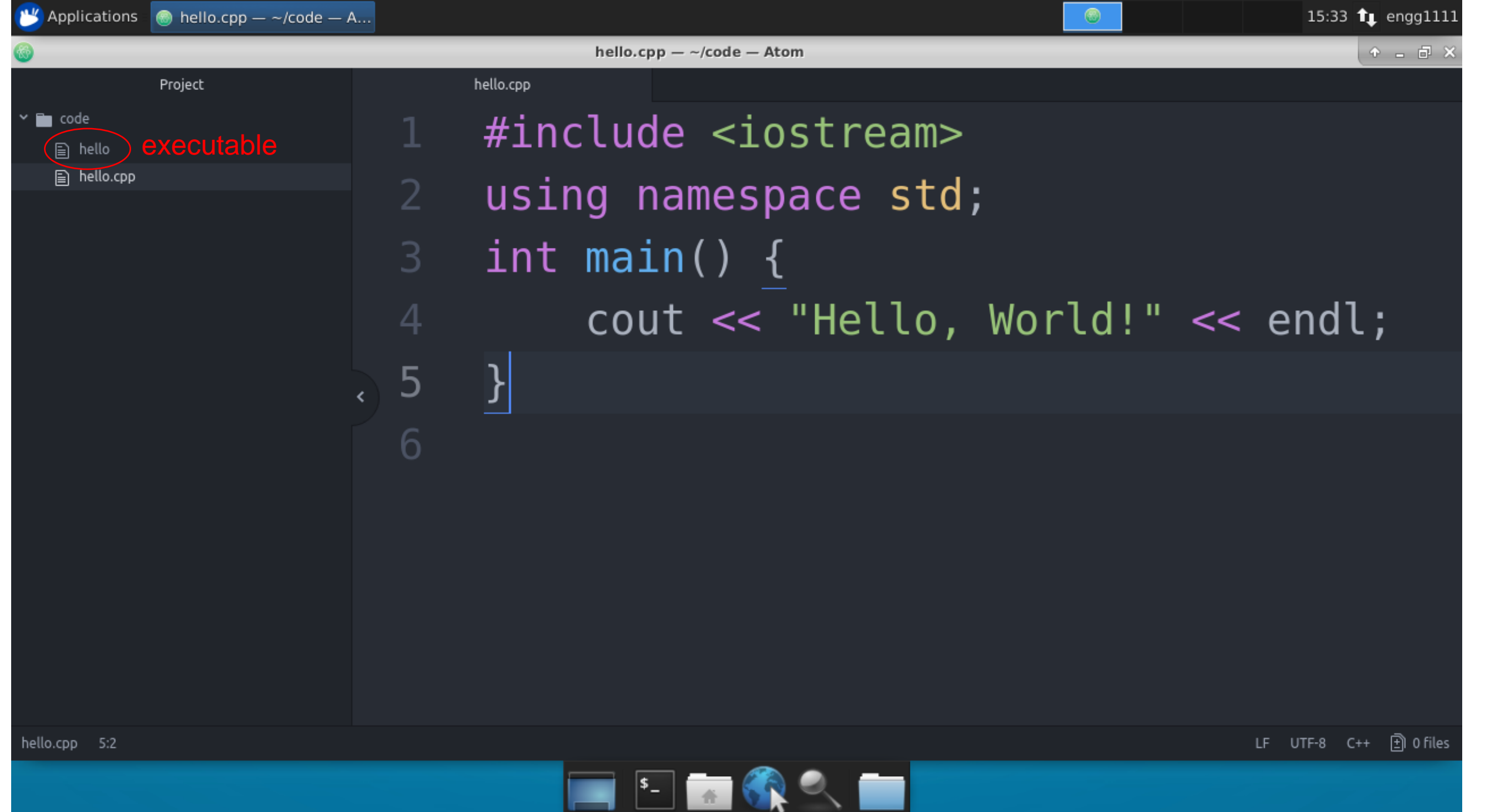
The screenshot shows the Atom code editor with a file named `hello.cpp` open. The code is as follows:

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      cout << "Hello, World!" << endl;
5  }
```

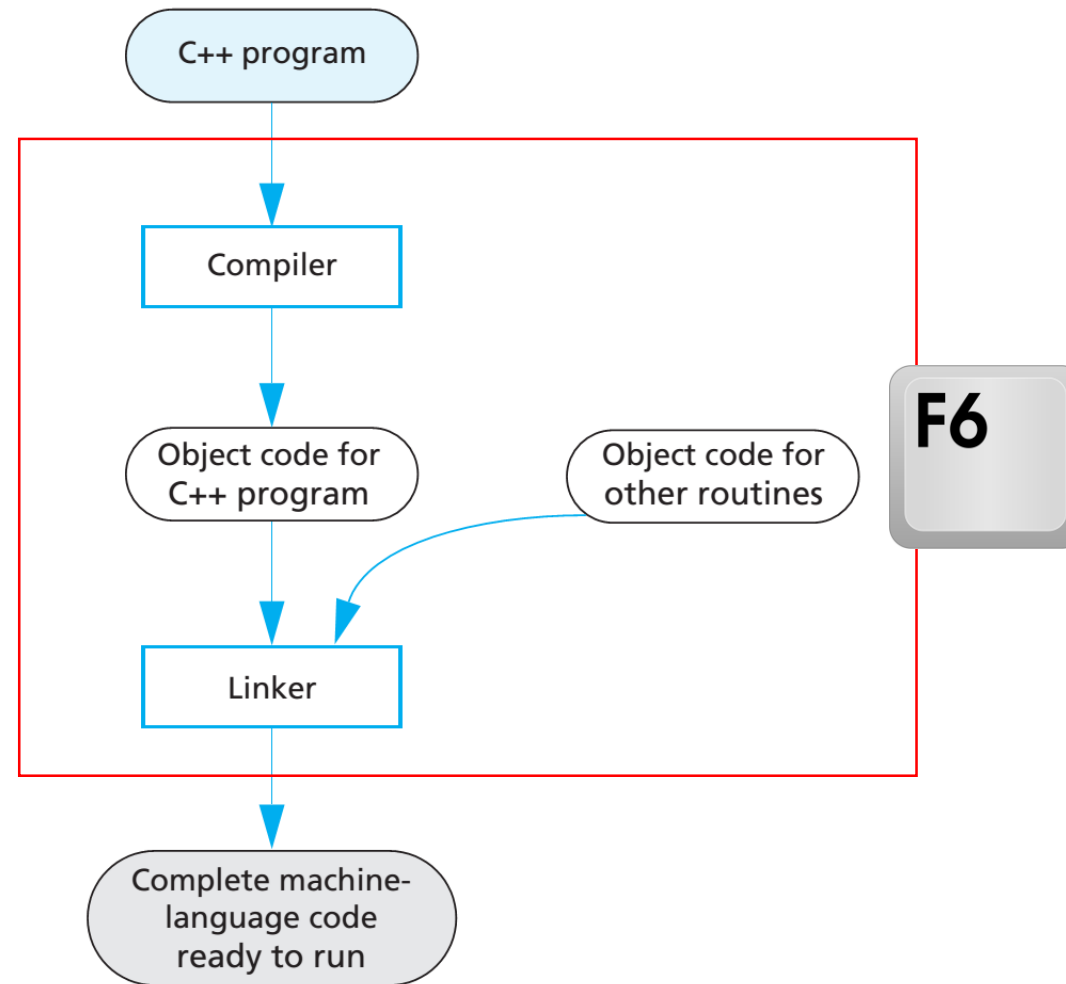
The editor interface includes a sidebar on the left showing the project structure with a folder named `code` and a file named `hello.cpp`. The status bar at the bottom indicates the file is `hello.cpp*` at line 5, column 2, with settings for `LF`, `UTF-8`, `C++`, and `0 files`. The Windows taskbar is visible at the very bottom.

Building an Executable



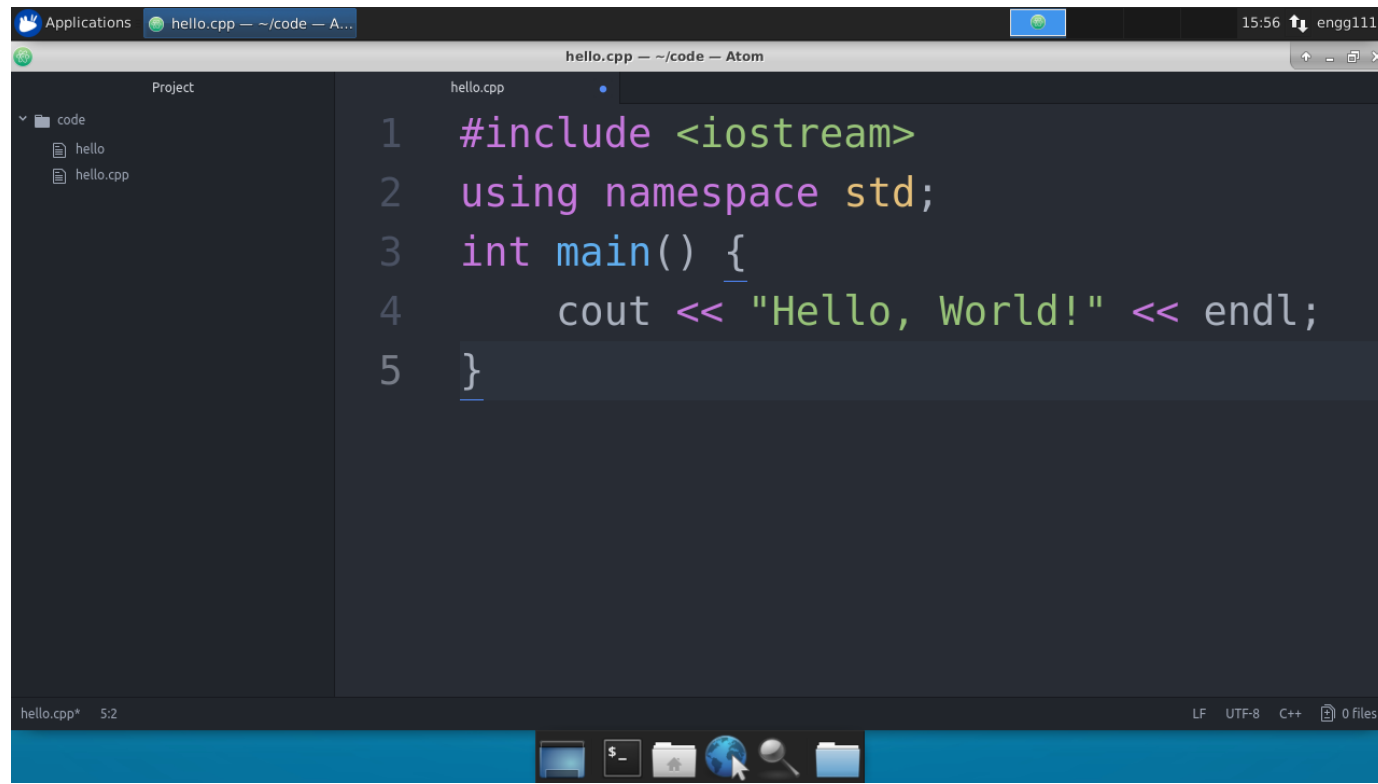


Building an Executable



First Program

- For now we will consider lines 1, 2 and 3 to be rather complicated way of saying “The program starts here.”

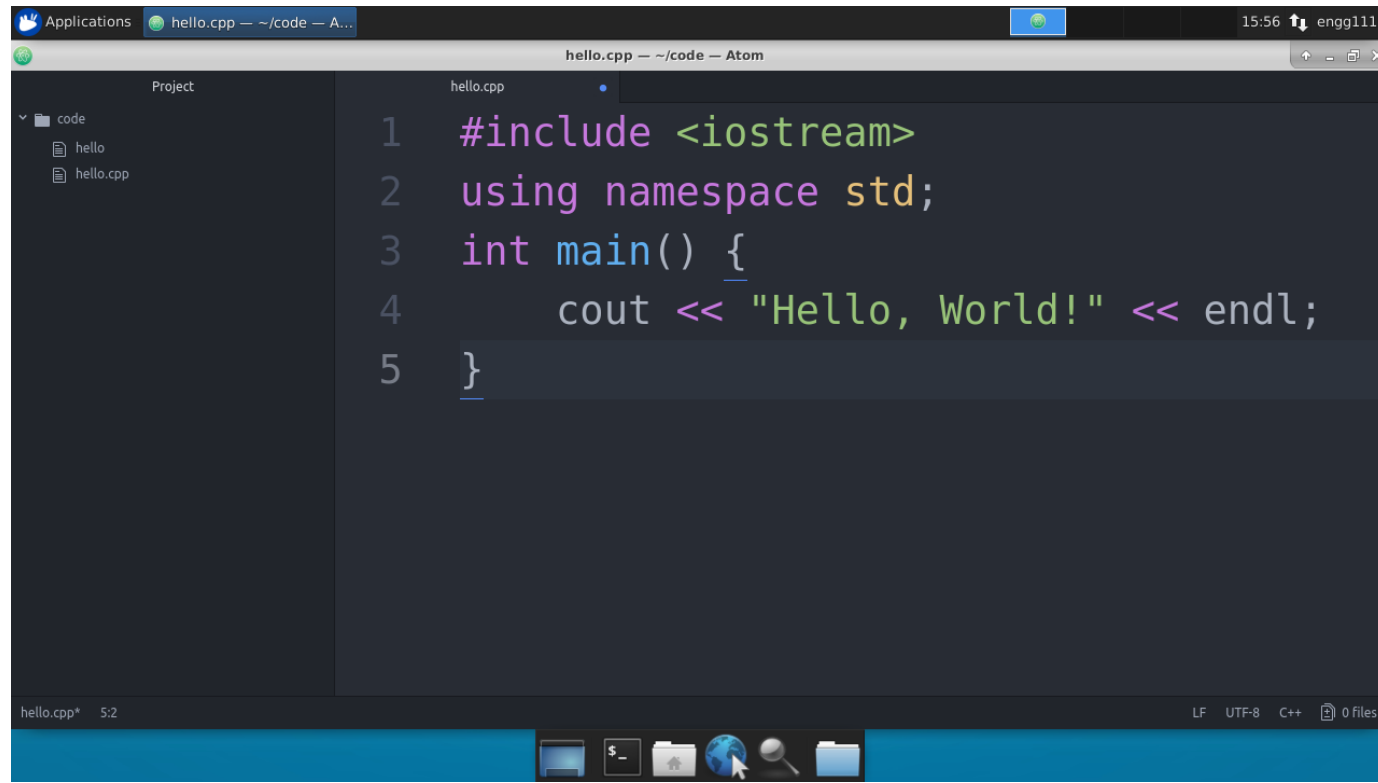
A screenshot of the Atom code editor interface. The window title is 'hello.cpp - ~/code - Atom'. The left sidebar shows a project structure with a folder 'code' containing files 'hello' and 'hello.cpp'. The main editor area displays the following C++ code:

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      cout << "Hello, World!" << endl;
5  }
```

The status bar at the bottom indicates 'hello.cpp* 5:2', 'LF', 'UTF-8', 'C++', and '0 files'. The system tray at the very bottom shows various icons including a terminal, file manager, and network status.

First Program

- Line 5 simply means “The program ends here.”

A screenshot of the Atom code editor interface. The window title is 'hello.cpp - ~/code - Atom'. The left sidebar shows a project view with a folder 'code' containing files 'hello' and 'hello.cpp'. The main editor area displays the following C++ code:

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      cout << "Hello, World!" << endl;
5  }
```

The status bar at the bottom indicates 'hello.cpp* 5:2', 'LF UTF-8 C++', and '0 files'. The system tray at the very bottom shows various icons including a terminal, file manager, and network status.

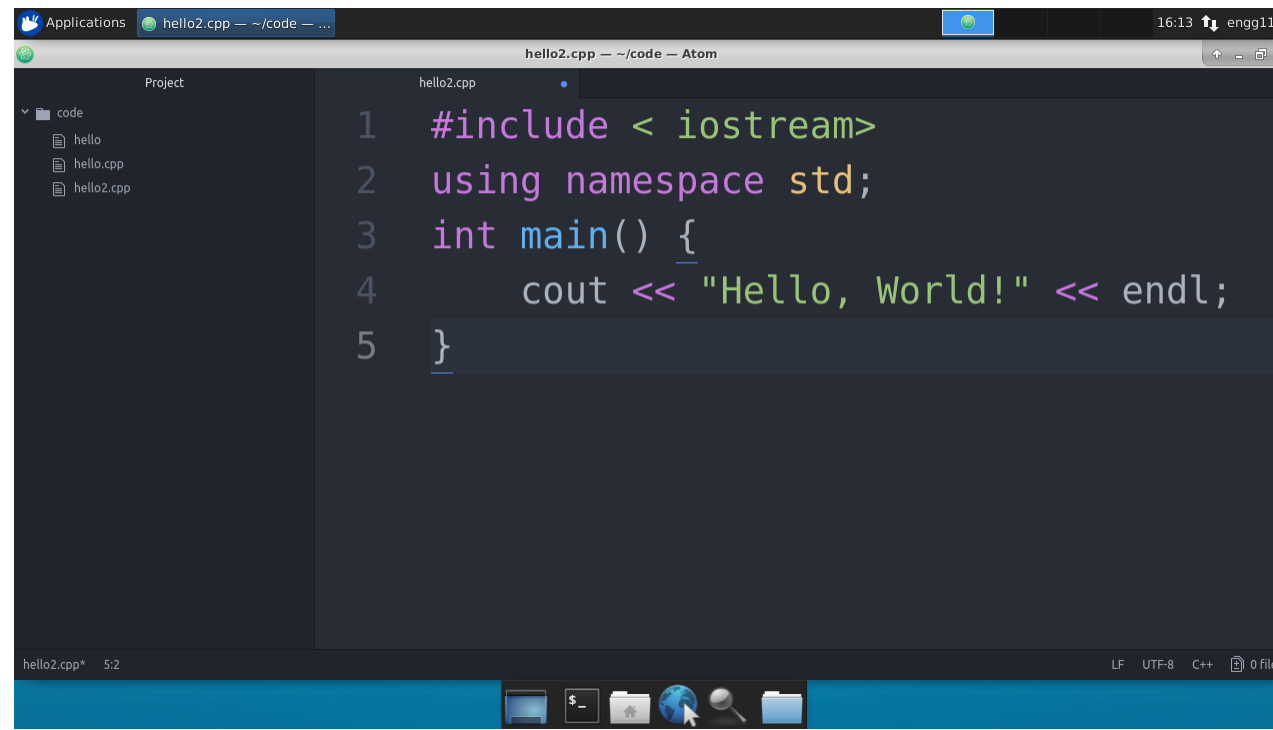
First Program

- Line 4 is a single statement
 - Usually statements fit in exactly one line
 - Each statement ends with a semicolon
 - Multiple statements are executed sequentially
- `cout <<` is used to output on the screen
- A sequence of characters enclosed by a pair of double quotes forms a string literal, e.g., "Hello, World!"
- `<< endl` is used to produce a newline
- The complete statement will print Hello, World! to the screen followed by a newline

```
4      cout << "Hello, World!" << endl;
```


Syntax Error

- Sometimes, a program will not build
- This is often due to syntax error(s)
- Example:



The screenshot shows the Atom code editor with a file named `hello2.cpp` open. The code is as follows:

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     cout << "Hello, World!" << endl;
5 }
```

The code is syntactically correct. However, the status bar at the bottom indicates a syntax error: `hello2.cpp* 5:2`. This suggests that the error is not in the code itself but in the file's encoding or line endings. The status bar also shows `LF UTF-8 C++ 0 files`.

Project

- code
 - hello
 - hello.cpp
 - hello2.cpp

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     cout << "Hello"
5 }
6
```

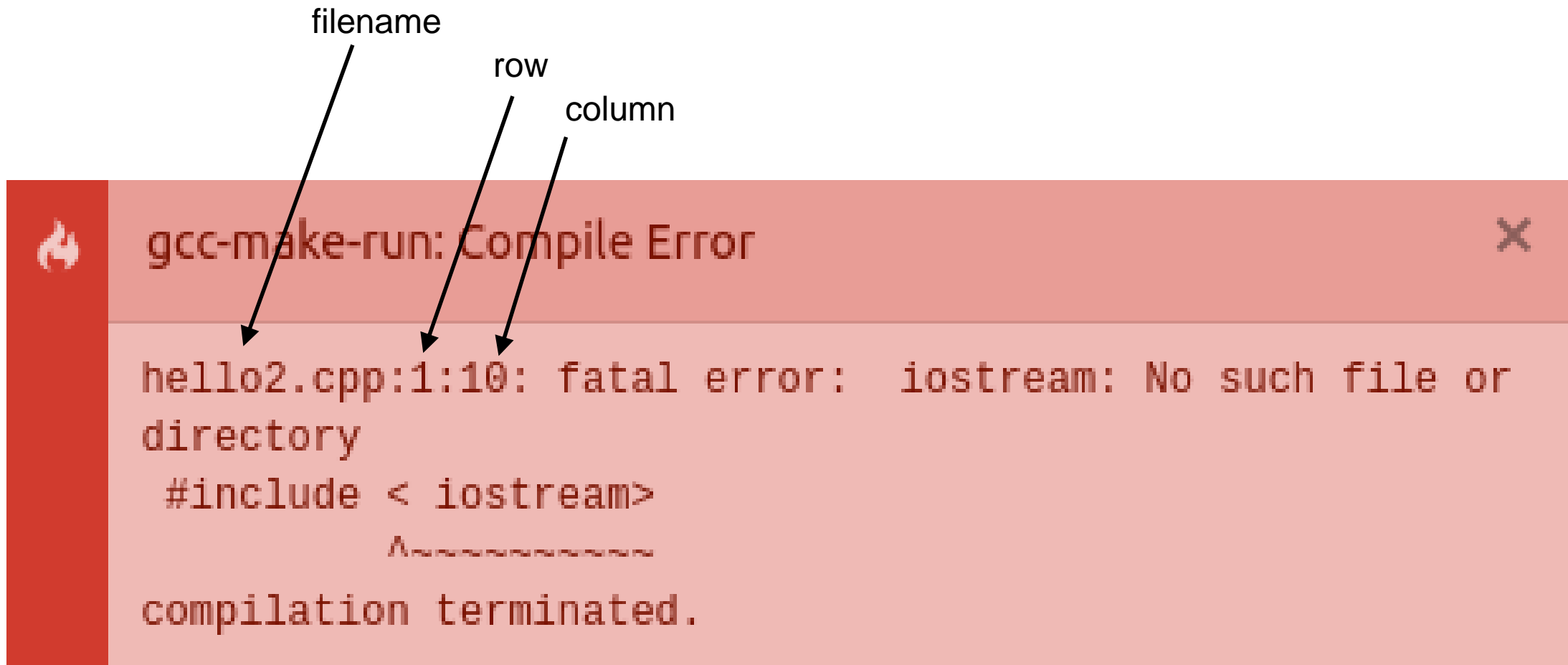
gcc-make-run: Running Command... Close All

"g++" -pedantic-errors -std=c++11 "hello2.cpp" -o "hello2"

gcc-make-run: Compile Error ×

hello2.cpp:1:10: fatal error: iostream: No such file or directory
#include <iostream>
~~~~~  
compilation terminated.

# Understanding Compiler Error

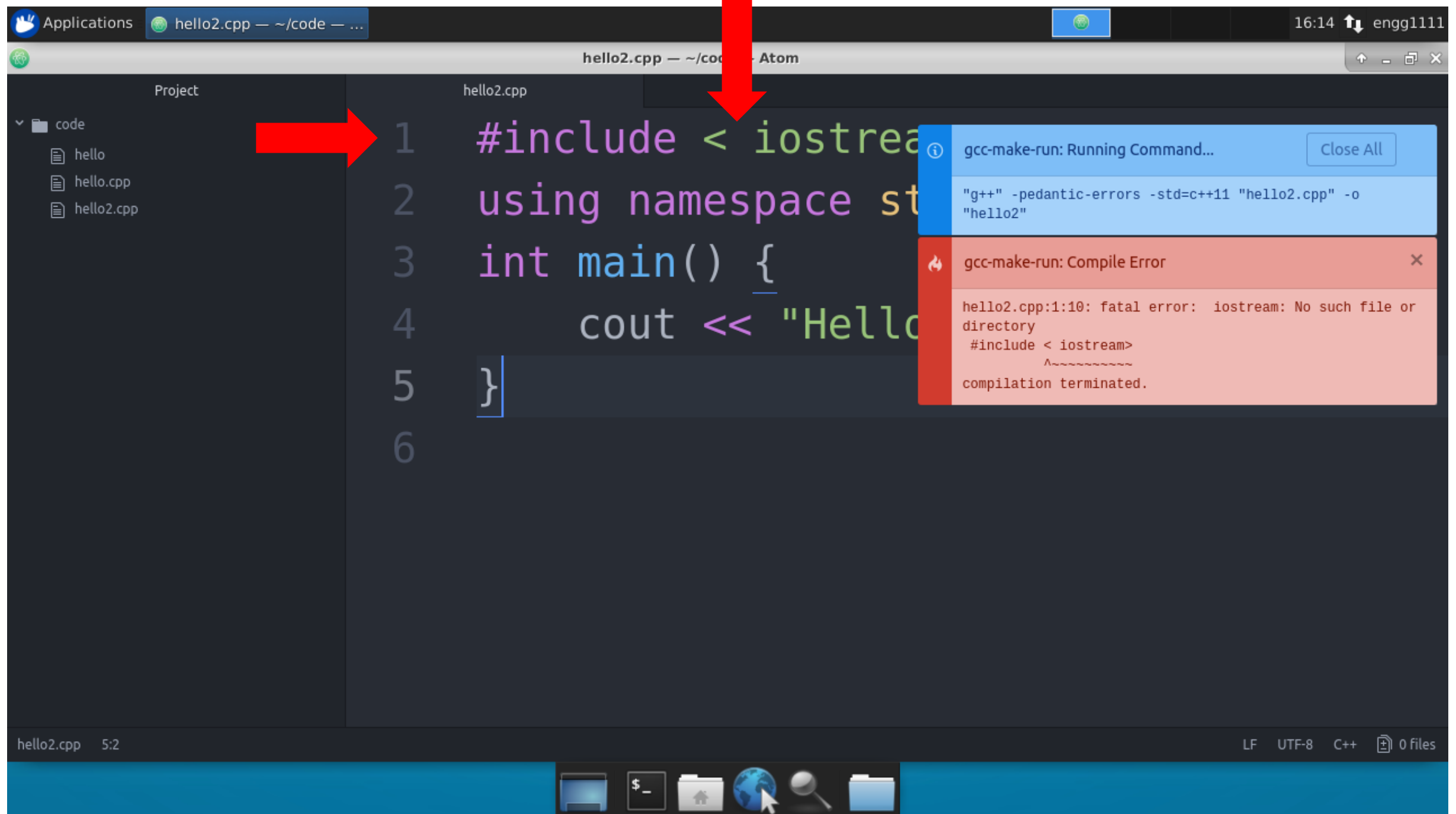


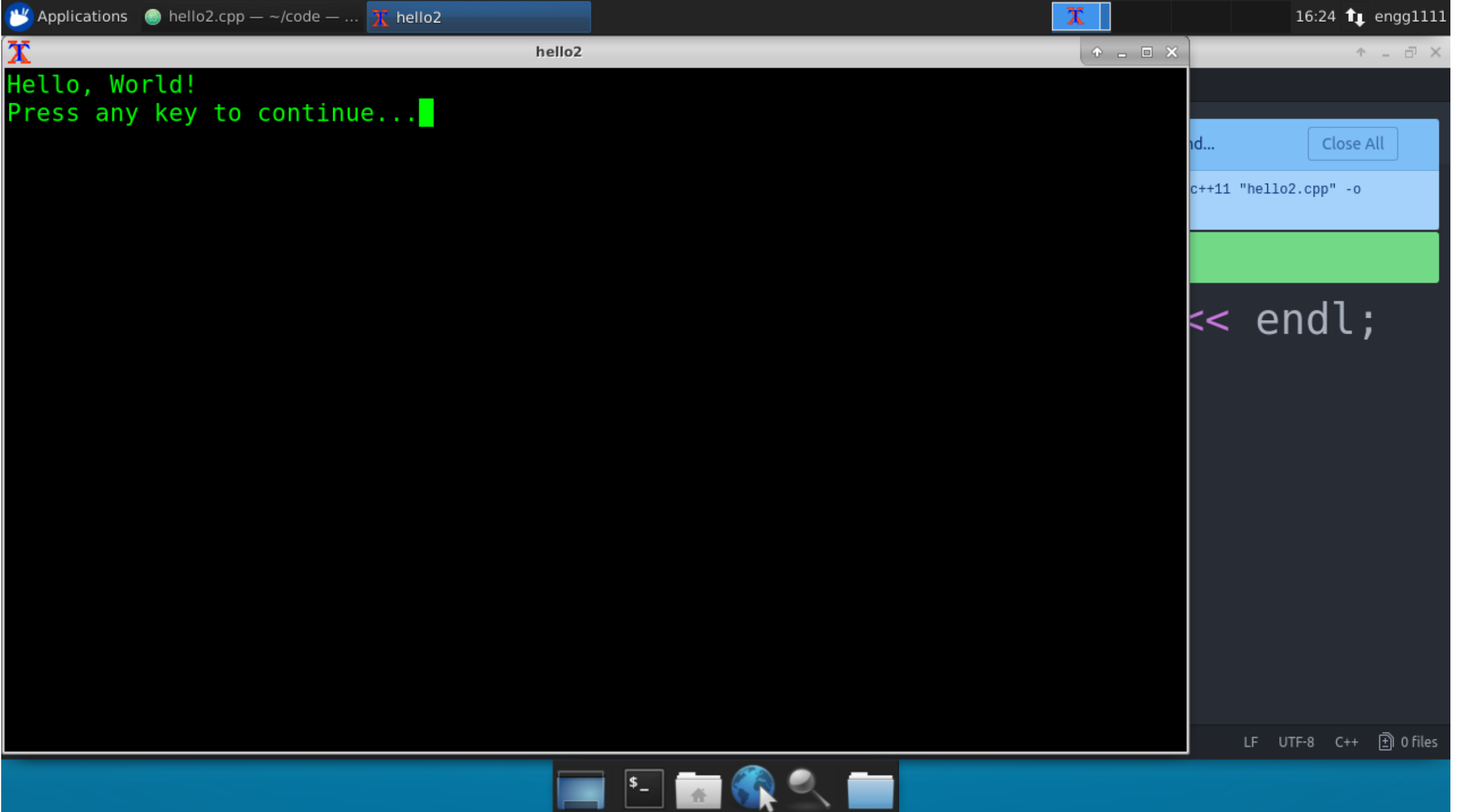
The image shows a terminal window with a red header bar. The header bar contains a flame icon on the left, the text "gcc-make-run: Compile Error" in the center, and a close button (X) on the right. Below the header bar, the terminal displays the following text:

```
hello2.cpp:1:10: fatal error:  iostream: No such file or
directory
#include < iostream>
           ^~~~~~
compilation terminated.
```

Three arrows point from labels above the terminal to specific parts of the error message:

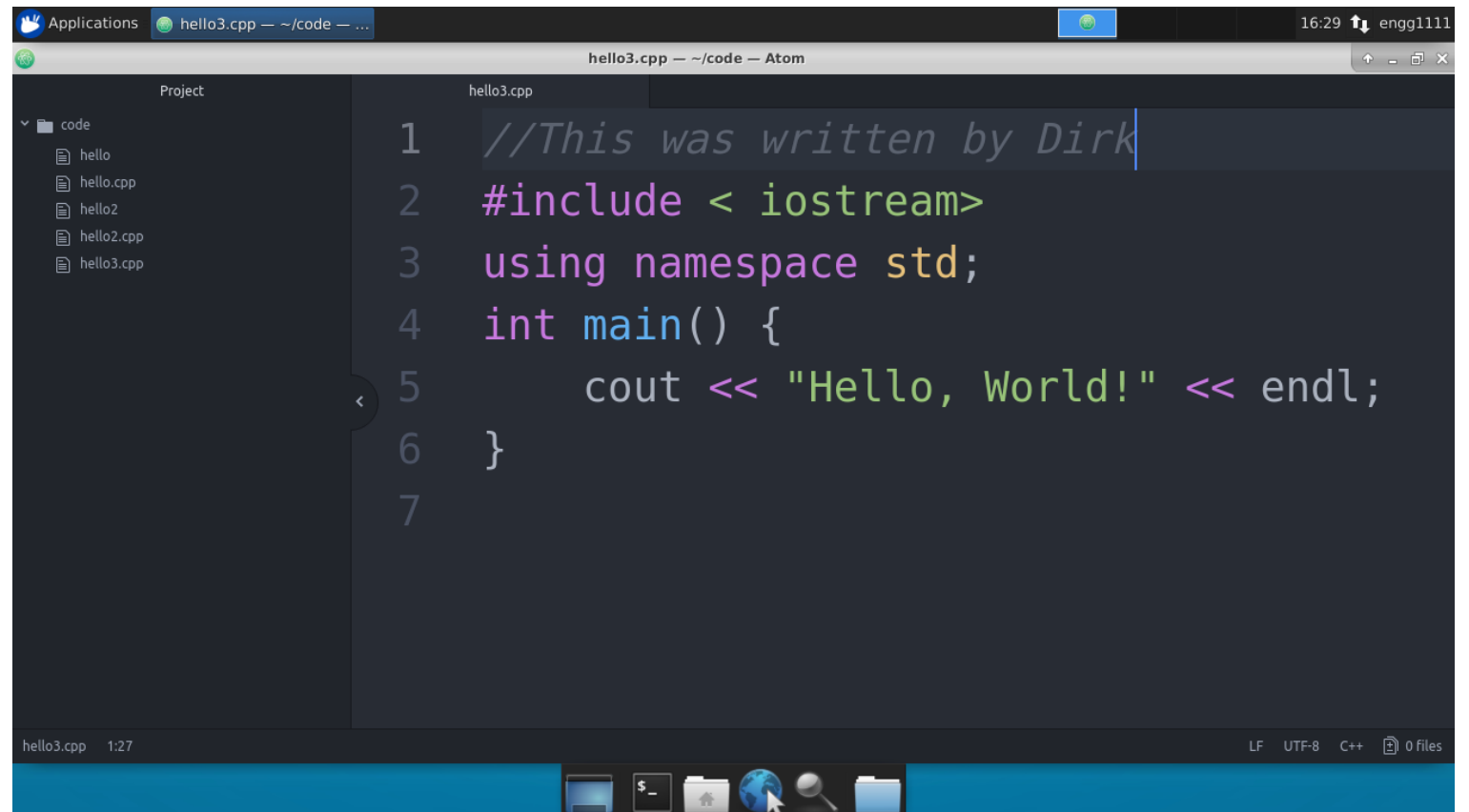
- The arrow labeled "filename" points to "hello2.cpp" in the first line of the error message.
- The arrow labeled "row" points to "1" in the first line of the error message.
- The arrow labeled "column" points to "10" in the first line of the error message.





# Comment

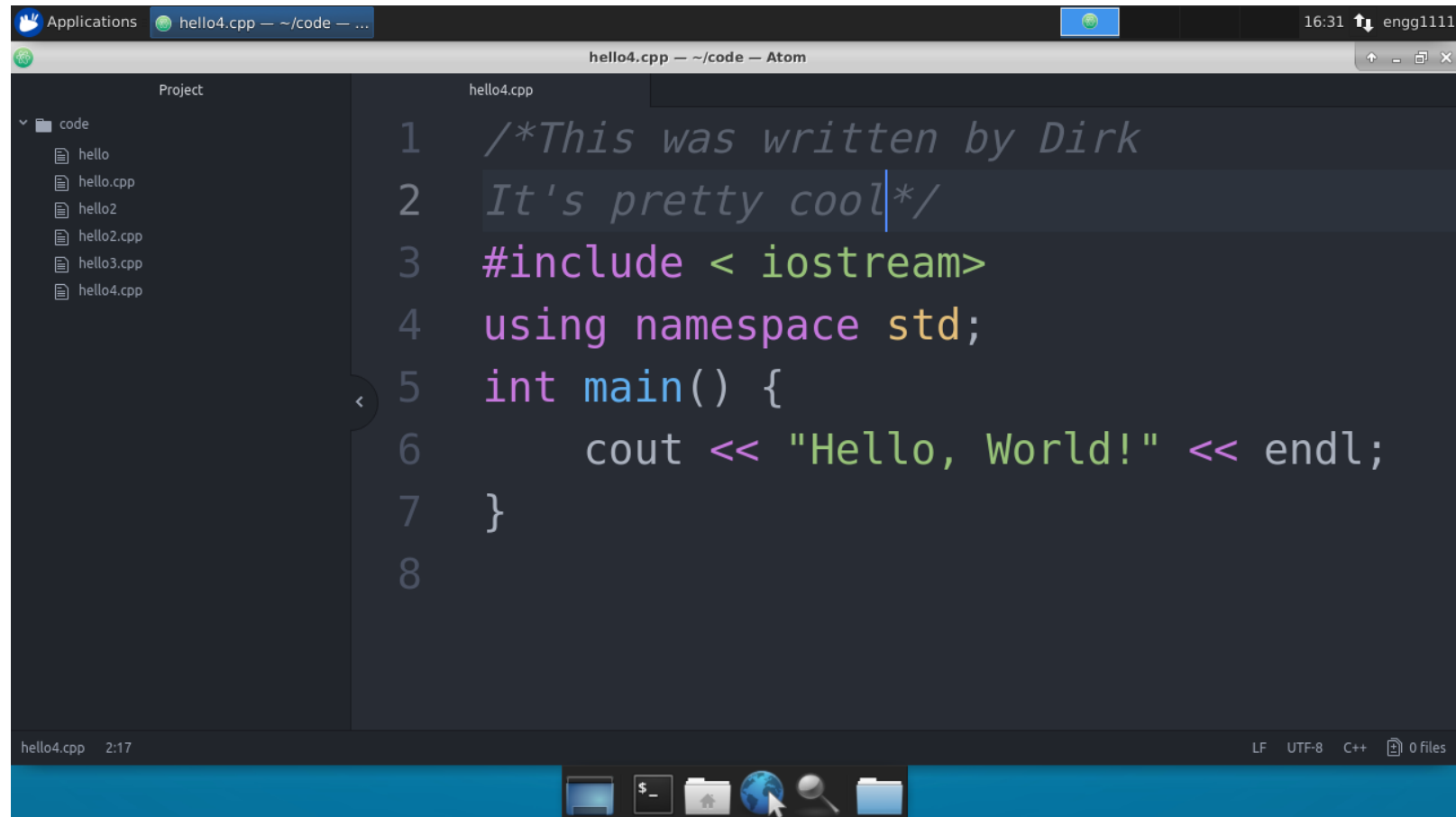
- A double slash indicates a comment line
  - Any text after `//` till the end of the line will be ignored by the compiler
- Example:



```
1 //This was written by Dirk
2 #include <iostream>
3 using namespace std;
4 int main() {
5     cout << "Hello, World!" << endl;
6 }
7
```

# Comment

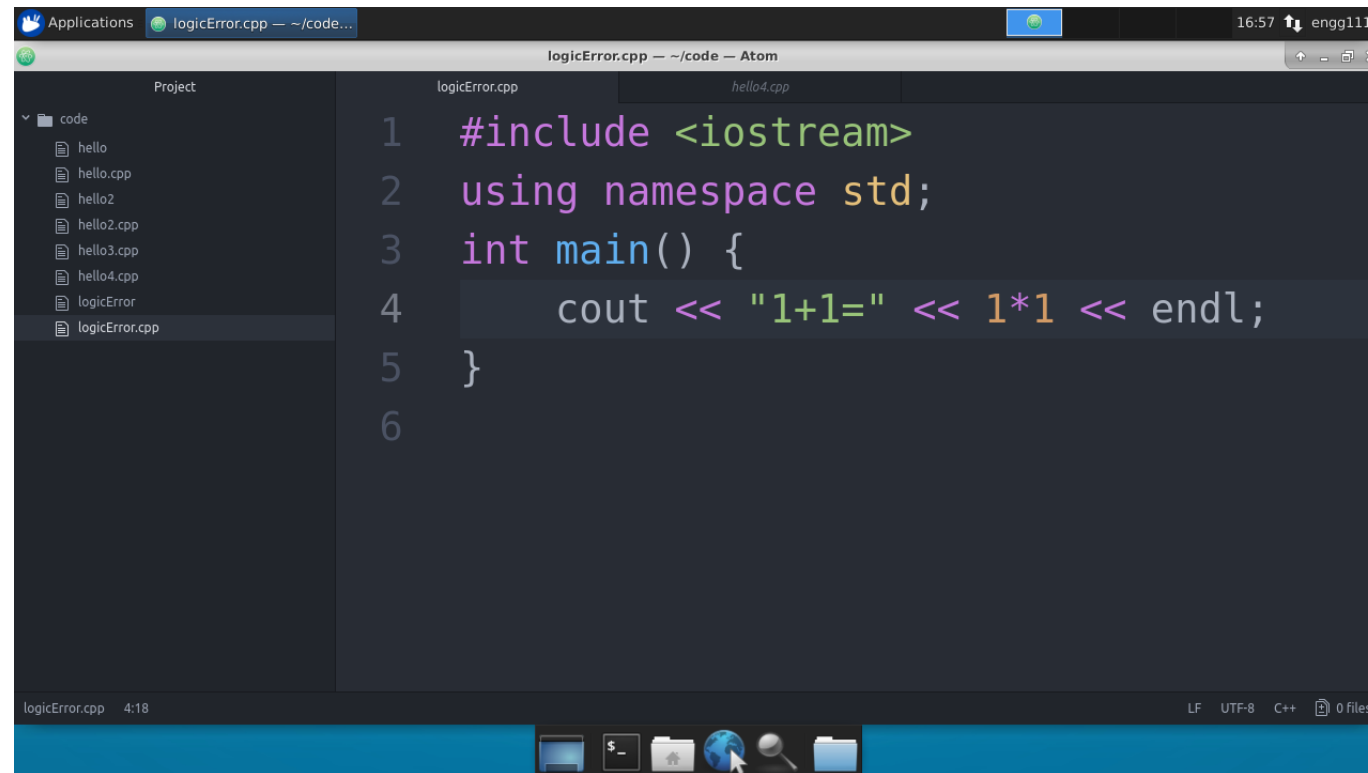
- A multi-line comment is bounded by `/* */`
- Example:



```
1  /*This was written by Dirk  
2  It's pretty cool*/  
3  #include <iostream>  
4  using namespace std;  
5  int main() {  
6      cout << "Hello, World!" << endl;  
7  }  
8
```

# Logic Error

- Sometimes, a program will compile and run successfully but has logic error(s)
- Example:



The screenshot shows the Atom code editor with a project named 'code' containing several files. The file 'logicError.cpp' is open and displays the following C++ code:

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      cout << "1+1=" << 1*1 << endl;
5  }
6
```

The code is syntactically correct and will compile and run successfully. However, it contains a logic error: the output of the program will be "1+1=" followed by the result of 1\*1 (which is 1), resulting in "1+1=1". This is incorrect because 1+1 should equal 2.



# Runtime Error

- Sometimes, a program will compile successful (with or without warning ) but does not run
- Example:

The screenshot shows the Atom code editor with a file named `runtimeError.cpp`. The code is as follows:

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     cout << "1+1=" << 1 / 0 << endl;
5 }
6
```

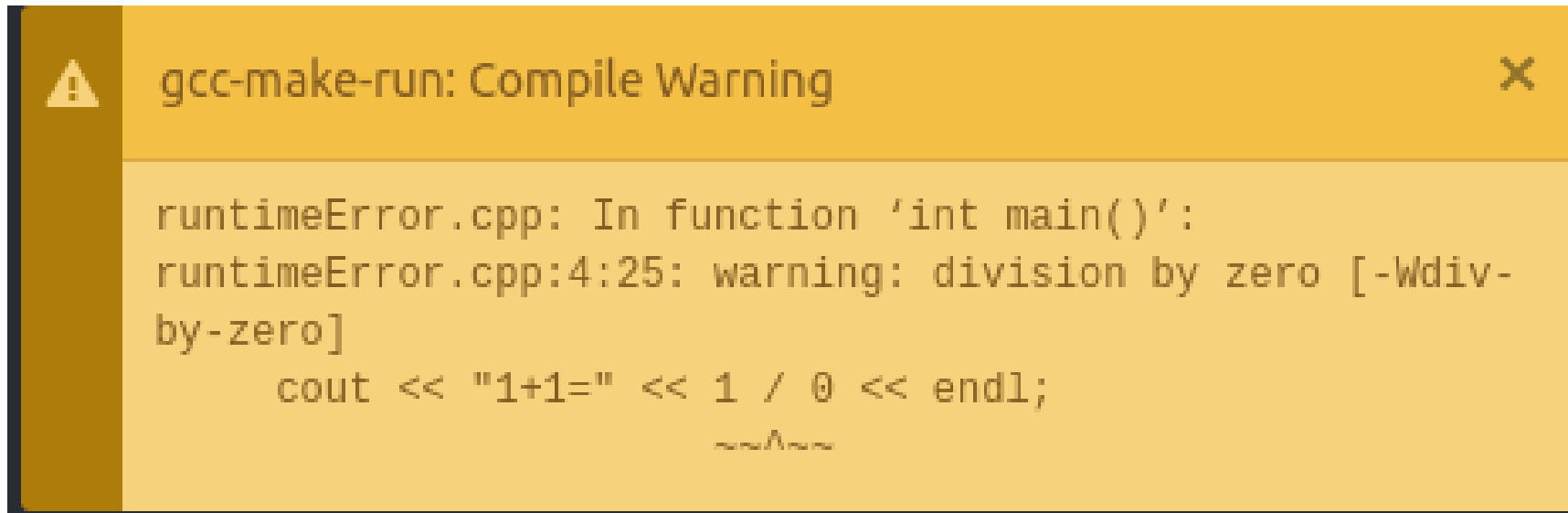
On the right side of the editor, there are three notification panels:

- A blue panel titled "gcc-make-run: Running Command..." with a "Close All" button. It shows the command: `"g++" -pedantic-errors -std=c++11 "runtimeError.cpp" -o "runtimeError"`.
- An orange panel titled "gcc-make-run: Compile Warning" with a close button. It shows a warning: `runtimeError.cpp: In function 'int main()': runtimeError.cpp:4:25: warning: division by zero [-Wdiv-by-zero]` for the line `cout << "1+1=" << 1 / 0 << endl;`.
- A green panel titled "gcc-make-run: Build Success" with a checkmark icon.

Below the editor, a terminal window titled `runtimeError` shows the output of running the program:

```
bash: line 1: 20307 Floating point exception(core dumped) "./runtimeError"
Press any key to continue...
```

# Compiler Warning



gcc-make-run: Compile Warning

runtimeError.cpp: In function 'int main()':  
runtimeError.cpp:4:25: warning: division by zero [-Wdiv-by-zero]  
 cout << "1+1=" << 1 / 0 << endl;  
 ~~~~^~~~~

Warning vs. Error

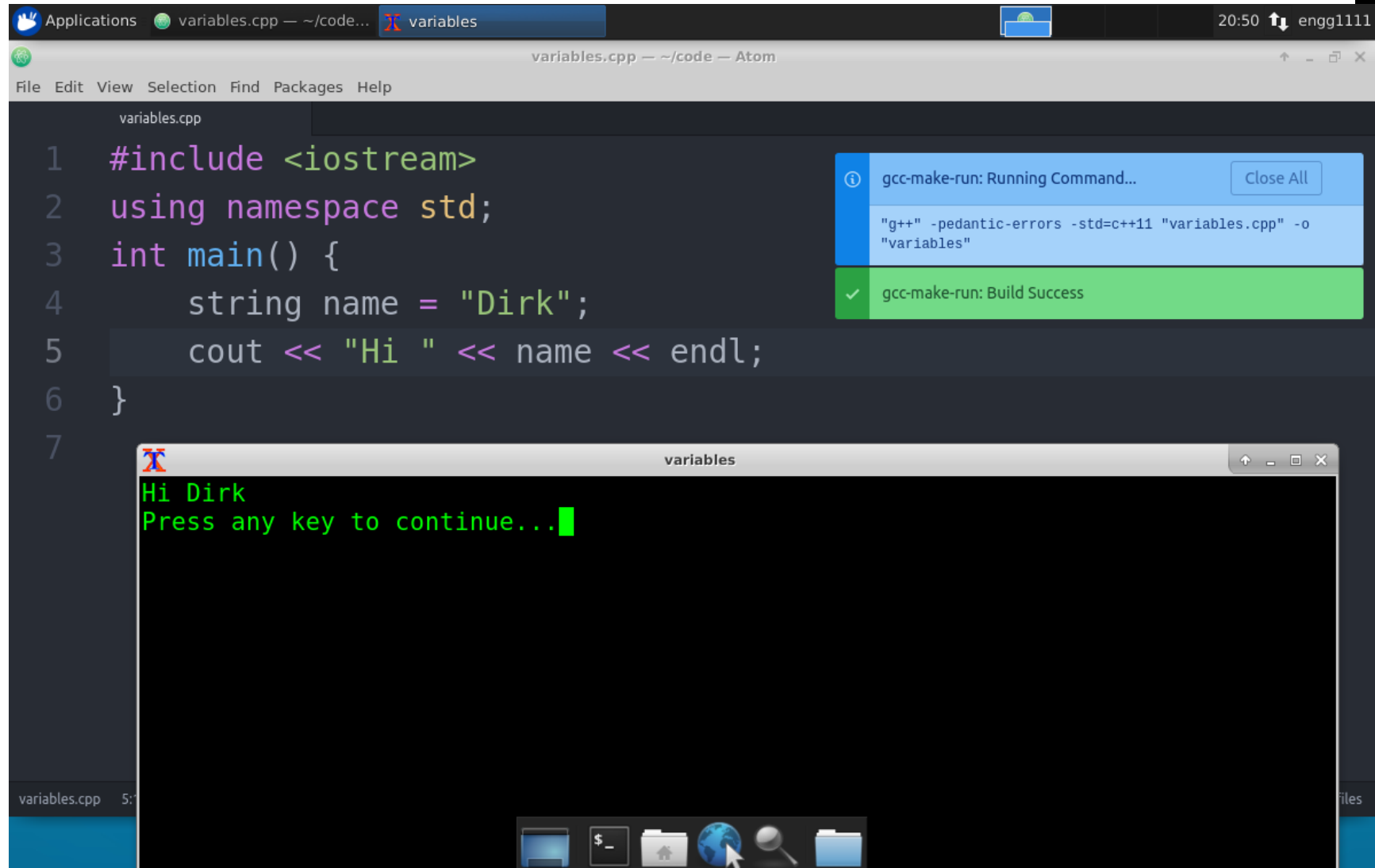
- A direct violation of the syntax rule will result in a compiler error
- A compiler warning usually indicates a likely mistake but is not a violation of the syntax rule
- At this point, you should treat every warning as if it was an error

Do not assume your
untested program is correct

Testing & Debugging

- A mistake in a program is called a bug, and the process of eliminating bugs is called debugging
- Syntax errors can be discovered relatively easily based on the error messages reported by the compiler during compilation
- Runtime and logic errors can only be discovered during program execution
- Carefully designed test cases (inputs with expected outputs) are used to catch any possible runtime and logic errors
 - We will use various test cases to test your assignment submissions

Sneak Peek: Variables



The screenshot shows the Atom code editor with a file named `variables.cpp` open. The code is as follows:

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     string name = "Dirk";
5     cout << "Hi " << name << endl;
6 }
7
```

On the right side of the editor, there is a panel showing the output of the `gcc-make-run` command. It displays the command used to compile the program:

```
"g++" -pedantic-errors -std=c++11 "variables.cpp" -o "variables"
```

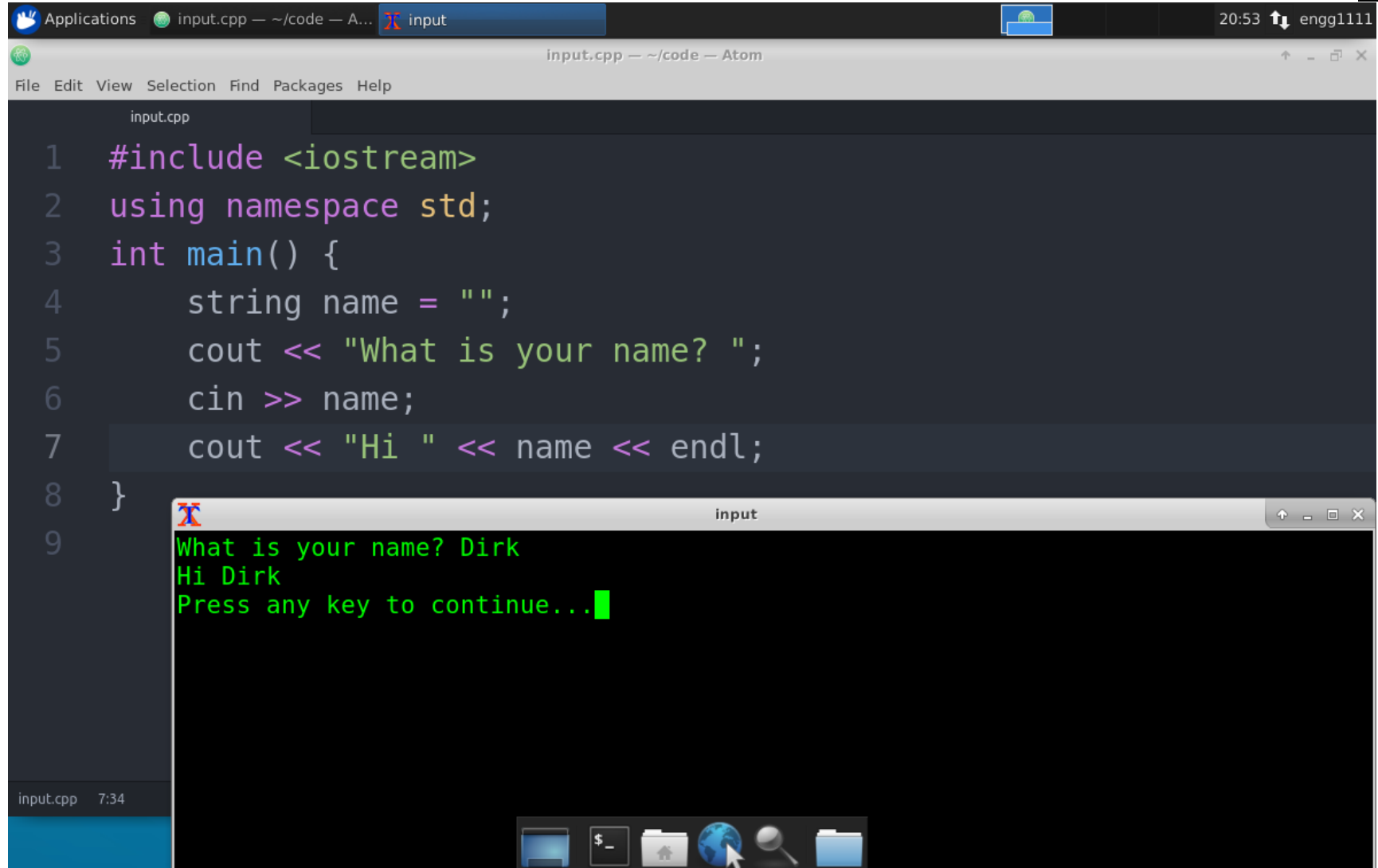
Below the command, it shows the result: `gcc-make-run: Build Success`.

At the bottom of the editor, there is a terminal window titled `variables` showing the output of the program:

```
Hi Dirk
Press any key to continue...
```

The terminal window is currently showing the output of the program, which is `Hi Dirk`, followed by a prompt to press any key to continue.

Sneak Peek: Input



The screenshot displays a development environment with two windows. The top window is the Atom code editor, showing a C++ file named `input.cpp`. The code is as follows:

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      string name = "";
5      cout << "What is your name? ";
6      cin >> name;
7      cout << "Hi " << name << endl;
8  }
9
```

The bottom window is a terminal titled `input`. It shows the output of the program: the prompt "What is your name? ", the user input "Dirk", and the response "Hi Dirk". It also shows the prompt "Press any key to continue..." with a cursor.