COSC 151: Intro to Programming: C++

Introduction &
Chapter 1

Overview

- **Syllabus**
  - Homework is important, keep up with it
  - Attendance is mandatory
    - You won't keep up otherwise
  - Quizzes
    - Will use these from time to time to judge your progress
  - Final (Capstone) Project
    - Will be a large project of code you write

Why Programming???

- **Software Engineering**
  - One of the fastest growing fields
  - Consistently high ranks in pay, job satisfaction
- **Computers are everywhere!**
  - Every business uses computers at some level
  - Desktop PCs, phones, watches, gaming systems, embedded devices (Nest, etc.)

Making Your Life Easier

- **Important concepts are in RED.**
- **Code/Commands are `fixed font and purple`**
- **When discussing C++**
  - Normal code is `fixed width, dark gray`
  - Keywords are `fixed width, blue`
  - Comments are `fixed width, green`
- **stackoverflow.com**
  - Don't be afraid to ask questions

Making You Life Easier, contd

- **In your code, use consistent indentation!**

```
int main() { std::cout << “Hello, World”
<< std::endl; return
0; }
```

  - Is significantly harder to read than (they have the same meening!!!!!)

```
int main()
{
   std::cout << “Hello, World” << std::endl;
   return 0;
}
```

- **In your code, use braces around conditional and iteration statements, even if they aren't technically required!**

```
if(x == 0)
   print_stuff();
```

  - While legal (and correct), prefer (it's easier to follow and understand!!!)

```
if(x == 0)
{
   print_stuff();
}
```

Chapter 1: Getting Started

- **Objectives**
  - We should be able to
    - Define variables
    - Perform input and output
    - Use control blocks
      - Conditionals
      - Iteration
    - Use a data structure to hold program data

C1: Writing a Simple Program

- **All C++ programs are made up of one or more functions, one of which must be called “main”**

```
int main()
{
   return 0;
}
```

- **Function definitions have 4 elements**
  - Return type (in this case `int`)
  - Function name (in this case `main`)
  - A (possibly empty) parameter list (in this case `()`)
  - A function body (in this case the lines including the braces)

C1: Input and Output

- **Input and output typically done through streams**
- **Streams provided as part of the standard library, our examples use `iostream`**
- **Input is done through an input stream (`istream`)**
- **Output is done through an output stream (`ostream`)**

C1: Input and Output - contd

- **Standard IO Objects**
  - There are four standard IO objects we can use in our programs
    - `std::cout` – standard output
    - `std::cin` – standard input
    - `std::cerr` – standard error
    - `std::clog` – standard log
    - By default, these all print to the same output window

C1: Using the Standard Library

- **Whenever we use a type or object from the standard library, we prefix it's usage with `std::`.**
  - This tells the compiler to use the type or object as defined in the `std` namespace.
  - Namespaces allow us to use names we define without colliding with other potential uses of those same names.
  - We will learn more about namespaces in the coming weeks.

C1: Writing to a Stream

- **We write to a stream using the “stream insertion operator” (`operator<<`)**

```
std::cout << “Hello, World!” << std::endl;
```

- **Calls to the stream insertion operator can “chain” making the above possible, similar to this:**

```
std::cout << “Hello, World!”;
std::cout << std::endl;
```

- **Output statements can be very helpful for debugging your programs**

C1: Reading from a Stream

- **We read from a stream using the “stream extraction operator” (`operator>>`)**

```
int x = 0, y = 0;
std::cin >> x >> y;
```

- **Calls to the stream extraction operator can “chain” making the above possible, similar to this:**

```
int x = 0, y = 0;
std::cin >> x;
std::cin >> y;
```

C1: Comments

- **Comments allow us to add information to our source code that isn't read by the compiler**
  - Line Comments, all text from `//` to the end of line are ignored.
  - Comment pairs, all text between `/*` and `*/` are ignored.
    - Comment pairs do not nest!

C1: Include Directives

- **When we want to use code that isn't defined locally in our source file, we need to tell the compiler that we want to use it.**
- **We do that with an "include directive".**
- **An include directive tells the compiler to find the requested file and make the declarations and definitions in that file available to our local source file.**
- **Example:**

```
#include <iostream>
```

  - This tells the compiler to find the iostream header file and make the "stuff" in that file available for use.

C1: A Program using IO Streams

```
#include <iostream> // Necessary to use the IO stream library
int main()
{
   // Use the standard output stream to prompt the user.
   std::cout << “Enter two numbers: “ << std::endl;

   // Define two int variables, and read them in
   // using the standard input stream
   int v1 = 0, v2 = 0;
   std::cin >>v1 >> v2;

   // print the output
   std::cout << “The sum of “ << v1 << “ and “ << v2
      << “ is “ << v1 + v2 << std::endl;

   return 0;
}
```

C1: Flow of Control

- **Almost all programs require us to control logic flow, the basic patterns available are**
  - Iteration – iterate over a range of values and take action
  - Conditional – determine an appropriate course of action at runtime based on inputs or program state

C1: Flow of Control: Iteration (`while`)

- A **while statement** (or while loop) executes a section of code as long as a given condition is `true`.
- Looping from 1 to 10:

```
int sum = 0, val = 1;
while(val <= 10) // keeps executing until val > 10
{
   sum += val;  // equivalent to sum = sum + value;
   ++val;       // equivalent to val = val + 1;
}
```

C1: Flow of Control: Iteration (`for`)

- **A for statement (or for loop) executes a section of code as long as a given condition is `true`.**
- **A for statement captures the control variables at the top, where it's less likely they will be missed.**
- **Looping from 1 to 10:**

```
int sum = 0;  // no need to define val here
              // sum defined here because it needs to be used
              // outside of the for statement's scope

// val is defined, the condition is checked, and val incremented
// all within the top of the for statement
for(int val = 1; val <= 10; ++val)
{
   sum += val;  // equivalent to sum = sum + val;
}
```

C1: Flow of Control: Conditional (`if`)

- The **if statement** supports conditional execution.
- An if statement can be standalone

```
if(grade > 70)
{
   std::cout << “You're passing.” << std::endl;
}
```

- Or it can have an else condition

```
if(grade > 70)
{
   std::cout << “You're passing.” << std::endl;
}
else
{
   std::cout << “Oh, no!” << std::endl;
}
```

- Or it can have (one or more) else if conditions

```
if(grade > 98)
{
   std::cout << “Wow, A+” << std::endl;
}
else if(grade > 93)
{
   std::cout << “Excellent, A” << std::endl;
}
else if(grade > 90)
{
   std::cout << “Alright, A-” << std::endl;
}
else
{
   std::cout << “Not, an A :(“ << std::endl;
}
```

C1: Assignment vs. Equality

- **C++ (and a significant number of other modern language) differentiate between assignment (`operator=`) and equality (`operator==`).**
  - This can be confusing, especially since there are places where assignment is allowed, but typically not expected (by the programmer). **A common source of errors:**

```
int x = 10;

if(x = 20)  // Assigns x the value of 20!
{
   std::cout << “x is 20!” << std::endl;  // Will print!
}

if(x == 10) // Compares x to 10, but x is 20 (from above)
{
   std::cout << “x is 10!” << std::endl; // Will not print!!
}
```

C1: Reading multiple Inputs

- The input stream supports an **implicit conversion** that allows it to be checked for validity
- We can use this to read in an unknown number of inputs

```
int sum = 0, val = 0;

while(std::cin >> val)  // keep reading until end-of-file, or failure
{
   sum += value;
}
```

- On Linux (your koding.com virtual machine), you stop entering inputs using `<CTRL> + d`

C1: Introducing Classes

- **C++ (like most modern languages) allows users to create their own data types**
  - The goal is to define class types that behave as naturally as the built-in types.
- **A class defines a type along with a collection of data values and operations that are related to that type.**
- **The data values are specific to an instance of the class. There will be distinct copies of the data members for each class instance defined.**

C1: Classes Define Behavior

- **A Class defines what operations are allowed and what those operations do**
- **For example**
  - Adding two class objects – is it allowed, what does it mean?
  - Subtracting?
  - Other operations?
  - We will learn more about what operations we can overload in the coming weeks

C1: Reading and Writing Classes

- **The stream insertion and stream extraction operators can be defined for a class, allowing reading from and writing to a stream.**

```
Sales_item book;  // we define our book item, like any other type

std::cin >> book; // read it from a stream
std::cout << book // write it to a stream
   << std::endl;
```

C1: Adding Classes

- **The Sales_item class defines the addition operator (operator+) to aggregate its data members**

```
Sales_item item1, item2;

std::cin >> item1 >> item2; // read in two books
std::cout << item1 + item2  // write out their “sum”
   << std::endl;
```

C1: Class Member Access

- **You can access the members of a class using the dot operator (“.” `operator`).**

```
Sales_item item1;

std::cin >> item1 >> item2; // read in two books
if(item1.isbn() == item2.isbn()) // if they have the same ISBN
{
   std::cout << item1 + item2  // write out their “sum”
      << std::endl;
}
else
{
   std::cout << “Books have different ISBN” << std::endl;
}
```

- **Here we've called a member function named `isbn()`.**
  - Member functions are sometimes called methods.

Final Thoughts

- **Setup your environment on koding.com**
- **We have to define variables before we can use them**
- **Print output using `std::cout`**
  - Helpful for debugging
- **Read input using `std::cin`**
- **Use `while` or `for` for iteration**
- **Use `if/else if/else` for conditional checking**
- **Classes are user defined types that group data and operations**

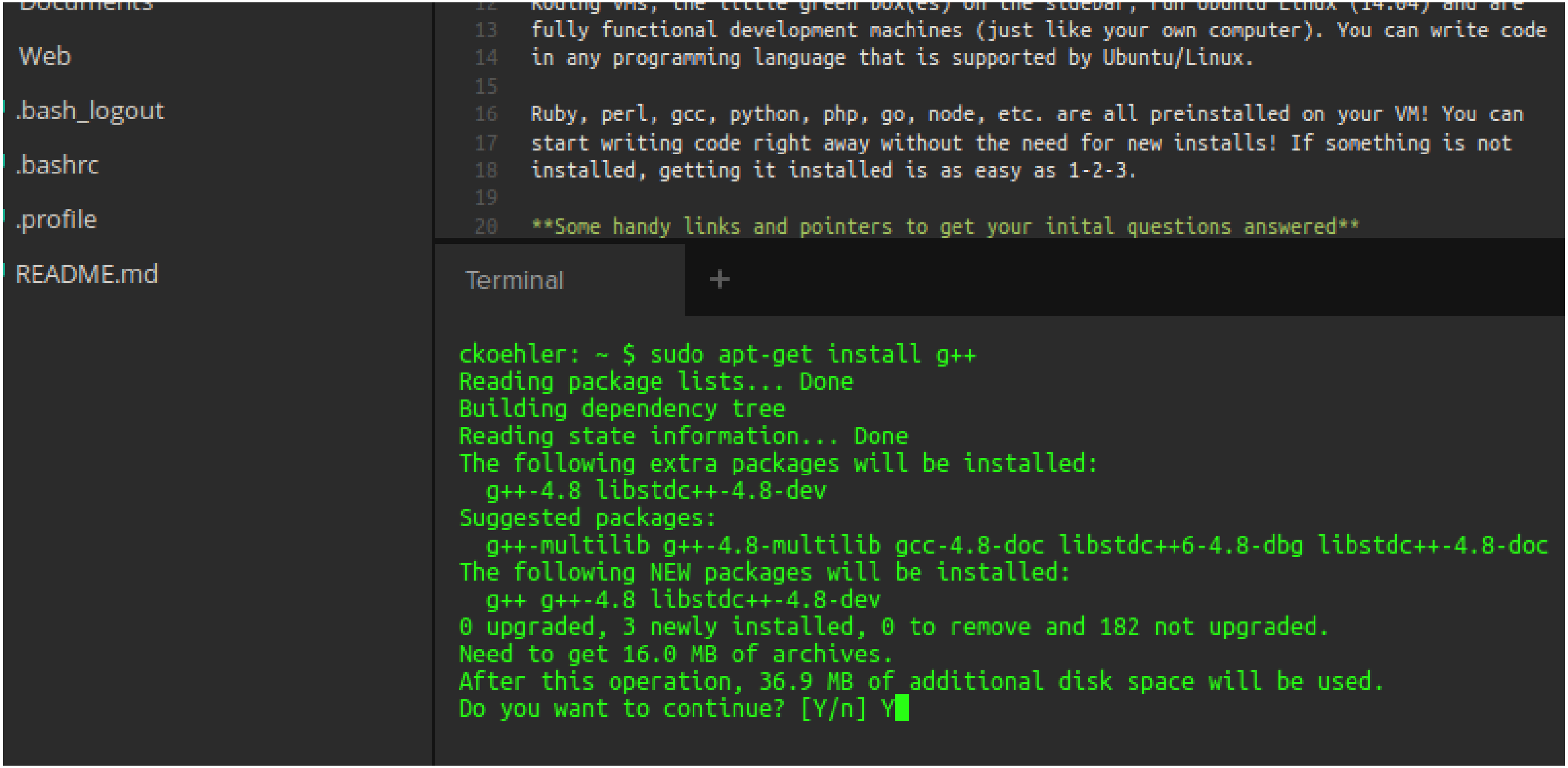
Environment Setup

- **Register for an account on koding.com**
  - Once registered and verified, a VM will be created for you on the site, this will be our main environment for development
- **Install the necessary software**
  - In the terminal tab, type the following then hit <Enter>

    ```
    sudo apt-get install g++
    ```

  - When prompted to continue, press Y then <Enter>

Environment Setup - contd

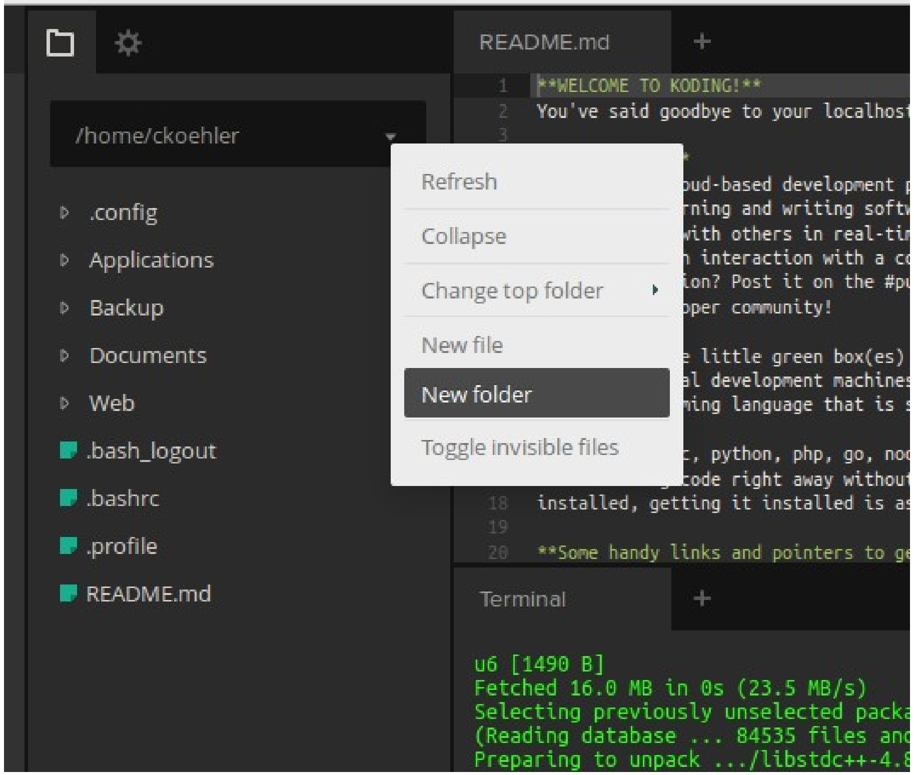


```
Koding VMs, the little green box(es) on the sidebar, run Ubuntu Linux (14.04) and are
fully functional development machines (just like your own computer). You can write code
in any programming language that is supported by Ubuntu/Linux.

Ruby, perl, gcc, python, php, go, node, etc. are all preinstalled on your VM! You can
start writing code right away without the need for new installs! If something is not
installed, getting it installed is as easy as 1-2-3.

**Some handy links and pointers to get your inital questions answered**
```

```
ckoehler: ~ $ sudo apt-get install g++
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  g++-4.8 libstdc++-4.8-dev
Suggested packages:
  g++-multilib g++-4.8-multilib gcc-4.8-doc libstdc++6-4.8-dbg libstdc++-4.8-doc
The following NEW packages will be installed:
  g++ g++-4.8 libstdc++-4.8-dev
0 upgraded, 3 newly installed, 0 to remove and 182 not upgraded.
Need to get 16.0 MB of archives.
After this operation, 36.9 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
```

Environment Setup - contd

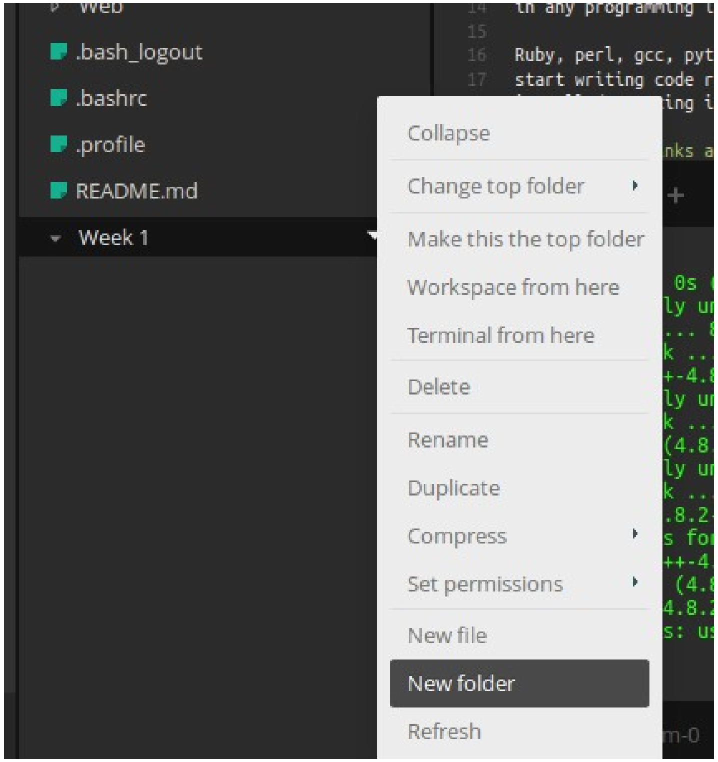
- **Create folders for each week, and each assignment:**



```
**WELCOME TO KODING!**
You've said goodbye to your localhost
```

```
installed, getting it installed is as

**Some handy links and pointers to ge
```

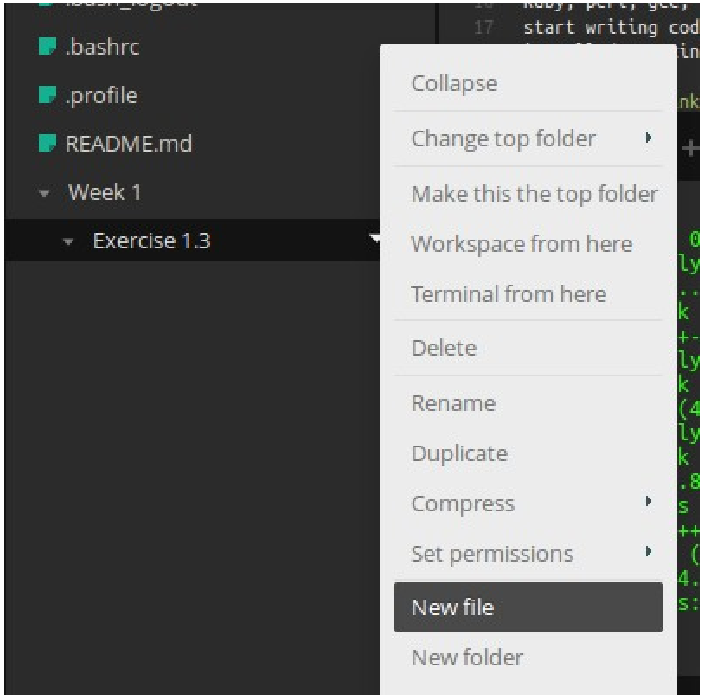
```
u6 [1490 B]
Fetched 16.0 MB in 0s (23.5 MB/s)
Selecting previously unselected packa
(Reading database ... 84535 files and
Preparing to unpack .../libstdc++-4.8
```



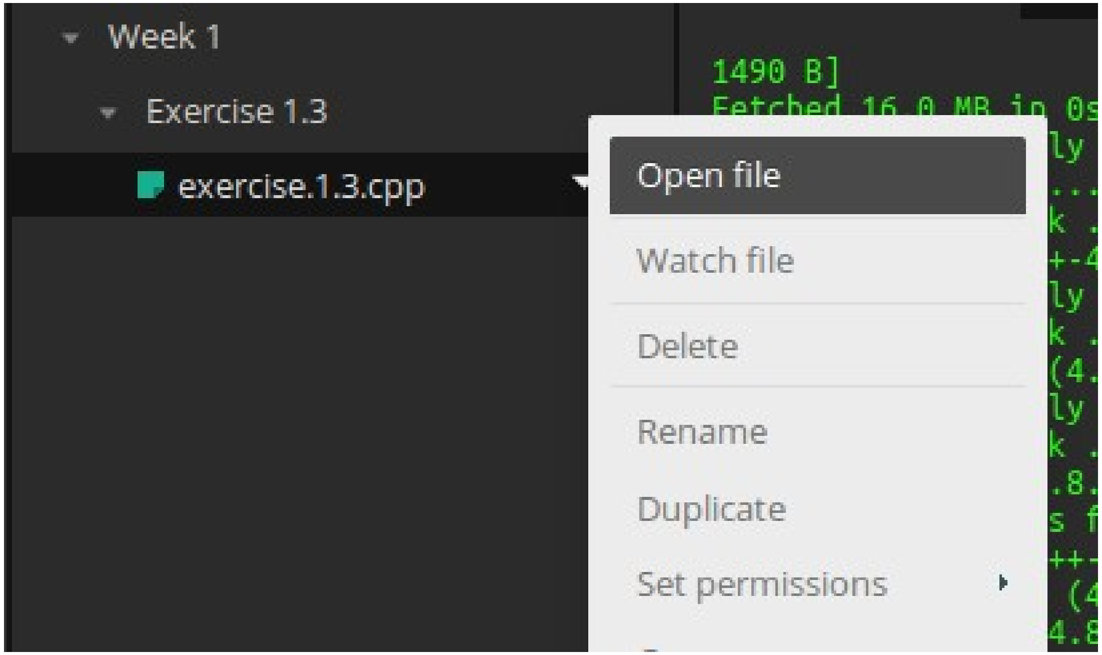
```
in any programming l

Ruby, perl, gcc, pyt
start writing code r
```

Environment Setup - contd

- **When working on an assignment, create a file in the assingment's folder**
  - Name the file `exercise.[chapter].[exercise].cpp`
  - For example: `exercise.1.3.cpp`.
  - Click on “Open File” to begin editing



```
17  start writing cod
```



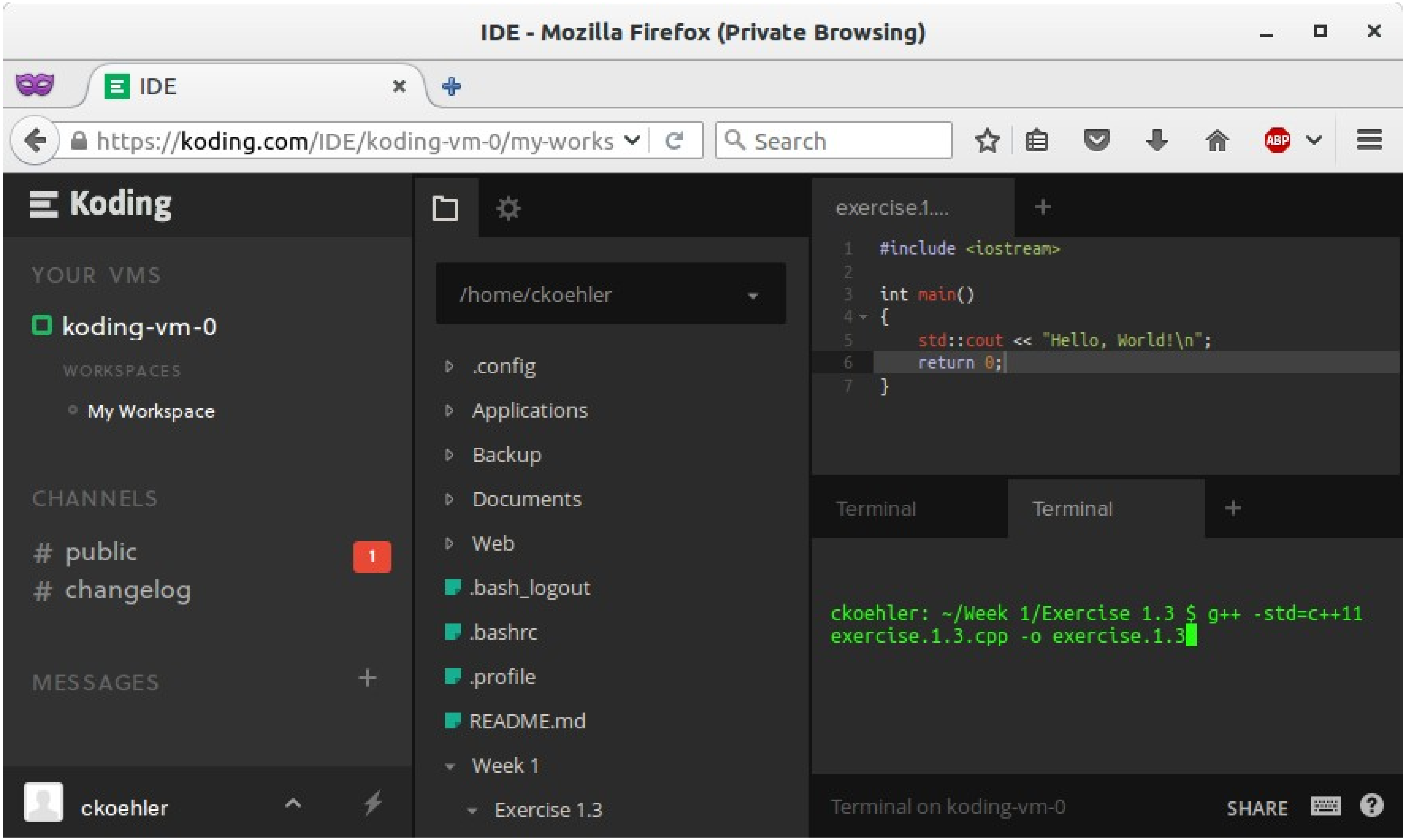
```
1490 B]
Fetched 16.0 MB in 0s
```

Environment Setup – contd

- **Edit your file, hit `<CTRL> + S` to save.**
- **To build:**
  - right click the assignment's folder, then select “Terminal from Here.”
    - This will open a new terminal tab in your assignments working directory.
  - Invoke the Compiler, with the following command:

```
g++       -std=c++11    <input file(s)> -o exercise.[chapter].[exercise]

compiler  C++ standard  input file(s)      output file name
```

Environment Setup – Full Example



```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, World!\n";
    return 0;
}
```

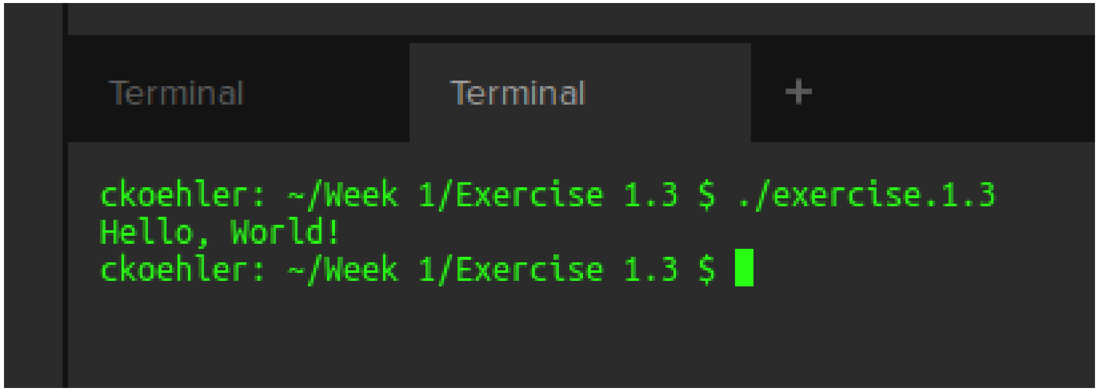
```
ckoehler: ~/Week 1/Exercise 1.3 $ g++ -std=c++11
exercise.1.3.cpp -o exercise.1.3
```

Environment Setup - Running

- **To run your code, from a terminal in your working directory, type the following:**

  `./exercise.[chapter].[exercise]`

  - Example:



```
ckoehler: ~/Week 1/Exercise 1.3 $ ./exercise.1.3
Hello, World!
ckoehler: ~/Week 1/Exercise 1.3 $ 
```