| Previous Lecture | lect02 Before Slides | lect02 Annotated Slides | Next Lecture |
|---|---|---|---|

<markdown=”1”>

## Code from lecture

https://github.com/ucsb-cs16-f18-mirza/cs16-f18-lectures/tree/master/lec-02

## Topics

- Programming in the unix environment
- The vim (editor) survival skills
- Writing, compiling and running a C++ program (hello world) program
- Breaking down the hello world program
- Code anywhere and everywhere with git

## Programming in the unix environment

- Unix is an operating system just like Windows and Mac OS
- All your data and programs are stored in files, within the unix filesystem
- File vs. directory
- Files are organized within the *unix filesystem*
- You can navigate the filesytem with some simple commands inside a terminal:
    - ls
    - mv
    - cp
    - pwd
    - mkdir
    - cd
- Relative path vs. absolute path (important for mv, cp, mkdir, cd)

---

---

## vim Editor

- We will use vim for the first few weeks this quarter.
- Important to be comfortable with a Unix-based command-line text editor.
- Be sure to understand how to do the **basic eight** functions.

# Writing, compiling and running a C++ program (hello world) program

```
// hello.cpp
#include <iostream>

using namespace std;

int main() {
        cout << "Hello CS 16!" << endl;
        return 0;
}
```

- Compile and execute the program

```
$ g++ -o hello hello.cpp
$ ./hello
Hello CS 16!
$
```

- `g++` is one of several C++ compilers
    - Compilers translate "source code" (i.e. the contents in the .cpp file) into a lower-level representation that is easier for computer system hardware to understand.
- `-o` is a "flag" that instructs the g++ compiler to produce an executable file called `hello`
- `hello.cpp` is the source file for g++ to use when producing the executable file.
- In order to actually run an executable file in Unix, `./[filename]` is used.

# Breaking down the Hello World Program

```
// hello.cpp
#include <iostream>

using namespace std;

int main() {
        cout << "Hello CS 16!" << endl;
        return 0;
}
```

```
#include <iostream>
```

- This line (also known as an include directive) tells our C++ program to include a library dealing with Input/Output (I/O) functionality.
    - We need the library `<iostream>` to print stuff to our terminal.

```
using namespace std;
```

- This line allows us to use parts of the iostream library without having to prepend `std::`.
    - For more context, `std` is short for “standard”.
    - including libraries between angle brackets (< >) imply that this is part of the C++ Standard Library, which is part of the C++ language specification.

```
int main() { ... }
```

- The main function. Every C++ program needs to have one main function as its “starting point”.

```
cout << "Hello CS 16!" << endl;
```

- `cout << [some_value]` tells the program to display some_value to the terminal.
- `<< endl;` tells the program to insert a *newline* at the end.
    - This places the next values to be written on the next line in the terminal.

```
return 0;
```

- Since main must be declared to return a value of “int” type, we are simply returning 0.
    - May get more into the relevance of this later.

# Comments

- Any commented text will be ignored by the compiler.
- Important to comment code for communication with others working with your code!
- `//` denotes a single-line comment.
- `/* */` denotes a multi-line comment.

## Example

```
// single line comment

/* multi
line
comment
*/
```

# Code anywhere and everywhere with git

## What is git?

- Git is a version control system (VCS). A VCS allows you to keep track of changes in a file (or groups of files) over time
- Git allows you to store code on different computers and keep all these different copies in sync

## Why are we learning git in this class?

- Collaborate
- Share code ownership
- Work on larger projects
- Provide feedback on work in progress
- Learn professional software development tools

## Git concepts

- repo (short for repository): a place where all your code and its history is stored

- remote repo: The repo created on github.com. You can access a remote repo via a browser (we'll explore other ways later)

- In class demo:
    - creating a repo on github.com
    - adding collaborators to the repo
    - adding files to the repo
    - Updating files in a remote repo using a web browser
    - Viewing the version history
- cloning a repo: Once you have created a repo, you (and your collaborators) can copy (or clone) it on different computers. Each instance is now a local repo, because it exists on your (local) computer rather than the web.
- Using git command line tools, we can sync up different repos. This means multiple collaborators can work on the same code on different machines.