Unit 4d

Compiling and Debugging

EDITING AND COMPILING MANUALLY

Review: Print Statements / Narration

- When we first discussed debugging we said an easy way to find an error is to add print statements that will "narrate" where you are and what the variable values are
- Be a **detective** by narrowing down where the error is
  - Put a print statement in each 'for', 'while', 'if' or 'else' block…this will make sure you are getting to the expected areas of your code
  - Then print variable values so you can see what data your program is producing

Editors

- "Real" developers use editors designed for writing code
  - No word processors!!
- You need a text editor to write your code
  - Eclipse, Sublime, MS Visual Code, Emacs, Atom, and many others
- These often have handy functions for commenting, indenting, checking matching braces ({..}) etc.

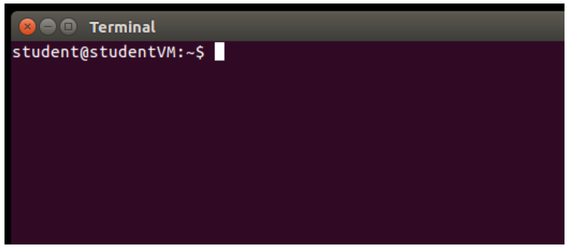
Compilers

- Several free and commercial compilers are available
  - g++:
  - clang++
  - XCode
  - MS Visual Studio
- Several have "integrated" editors, debuggers and other tools and thus are called IDE's (Integrated Development Environments)

Using the Command Line

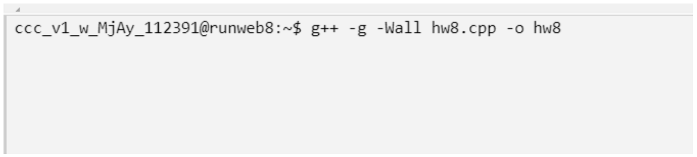
- While GUIs are nice, we often have more control when using the command line interface (i.e. the terminal)
- Linux (the OS used by Codio and in CS 103, 104, etc.) has a rich set of command line utilities (Mac & Windows do too, though Windows uses different names for the utilities)
- We can navigate the file system (like you would with Explorer or Finder), start programs (double-clicking an icon), and much more by simply typing commands

Terminal Icon



```
student@studentVM:~$
```

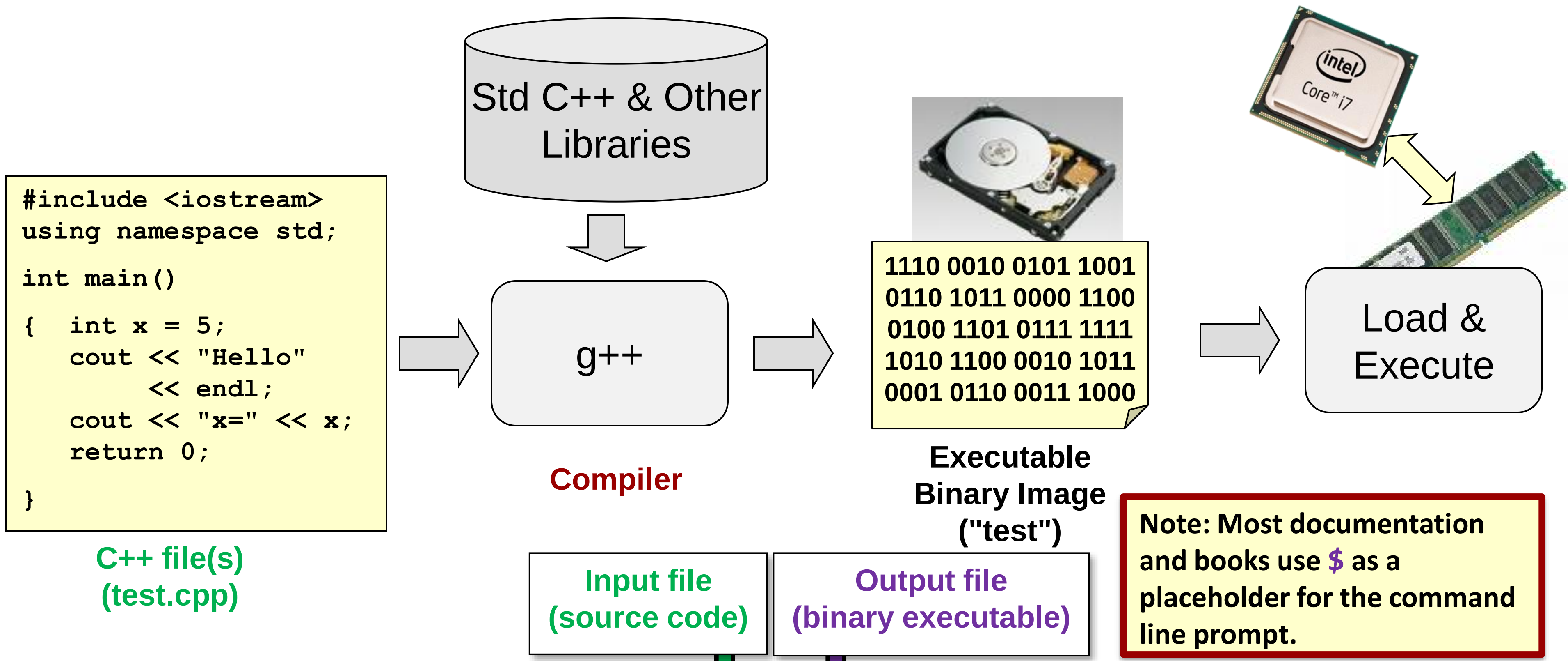
Linux Terminal View



```
ccc_v1_w_MjAy_112391@runweb8:~$ g++ -g -Wall hw8.cpp -o hw8
```

Vocareum Terminal View

Software Process


Std C++ & Other Libraries
g++
Compiler
Load & Execute
Executable Binary Image ("test")
C++ file(s) (test.cpp)
Input file (source code)
Output file (binary executable)

```
#include <iostream>
using namespace std;

int main()

{   int x = 5;
    cout << "Hello"
         << endl;
    cout << "x=" << x;
    return 0;

}
```

```
1110 0010 0101 1001
0110 1011 0000 1100
0100 1101 0111 1111
1010 1100 0010 1011
0001 0110 0011 1000
```

Note: Most documentation and books use $ as a placeholder for the command line prompt.

```
$ code test.cpp &
```

1 Edit & write code

```
$ code test.cpp &

$ g++ -g -Wall test.cpp -o test
or
$ make test
```

2 Compile & fix compiler errors

```
$ code test.cpp &

$ g++ -g -Wall test.cpp -o test

$ ./test
```

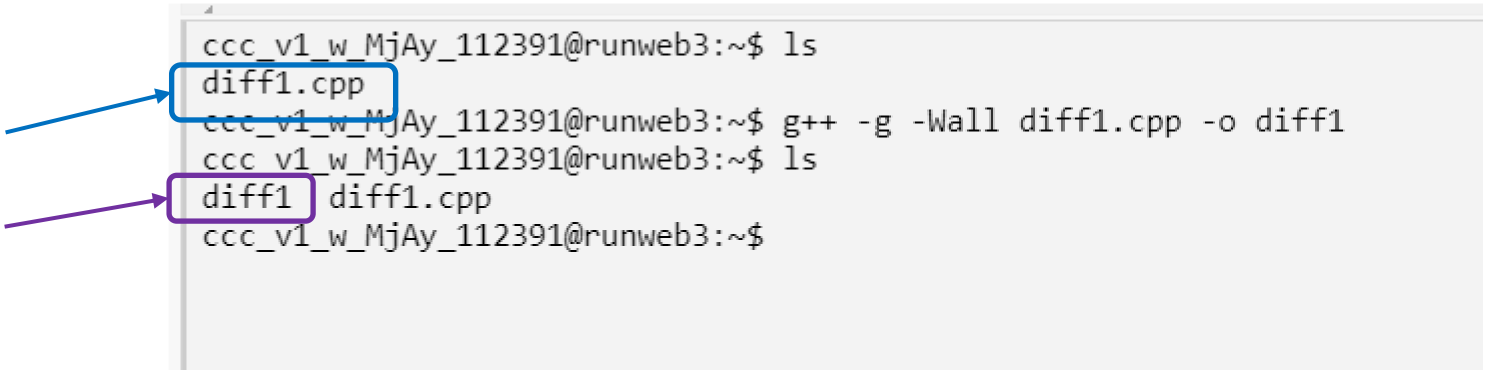
3 Load & run the executable program

g++ Options

- Most basic usage
  - `g++` *`cpp_filenames`*
  - Creates an executable `a.out`
- Options
  - `-o` => Specifies output executable name (other than default a.out)
  - `-g` => Include info needed by debuggers like gdb, kdbg, etc.
  - `-Wall` => show all warnings
- Most common usage form:
  - `$ g++ -g -Wall kiosks.cpp -o kiosks`

Listing Files (Folder Contents)

- In Mac/Linux, to view the files in a folder, just type `ls` (stands for list)



```
ccc_v1_w_MjAy_112391@runweb3:~$ ls
diff1.cpp
ccc_v1_w_MjAy_112391@runweb3:~$ g++ -g -Wall diff1.cpp -o diff1
ccc_v1_w_MjAy_112391@runweb3:~$ ls
diff1  diff1.cpp
ccc_v1_w_MjAy_112391@runweb3:~$
```

Running the Program

- First ensure the program compiles
  - `$ g++ -g -Wall kiosks.cpp -o kiosks`
- Then run the program by preceding the executable name with `./`
  - `$ ./kiosks`
  - Remember NOT to type the `$`

DEBUGGERS

Step 4: Using a Debugger

- Allows you to
  - Set a breakpoint (the code will run and then stop when it reaches a certain line of code)
  - Step through your code line by line so you can see where it goes
  - Print variable values when you are stopped at a certain line of code

GDB and Other Debuggers

- gdb is a simple text-based debugger that comes with many Linux/Unix based system
  - We'll focus on this debugger
- Other development environment have built in debuggers
  - MS Visual Studio
  - Apple Xcode
  - Eclipse
  - Even Codio has a graphical interface for GDB
- Online compiler + GUI based GDB tool
  - https://www.onlinegdb.com/

Running the Debugger

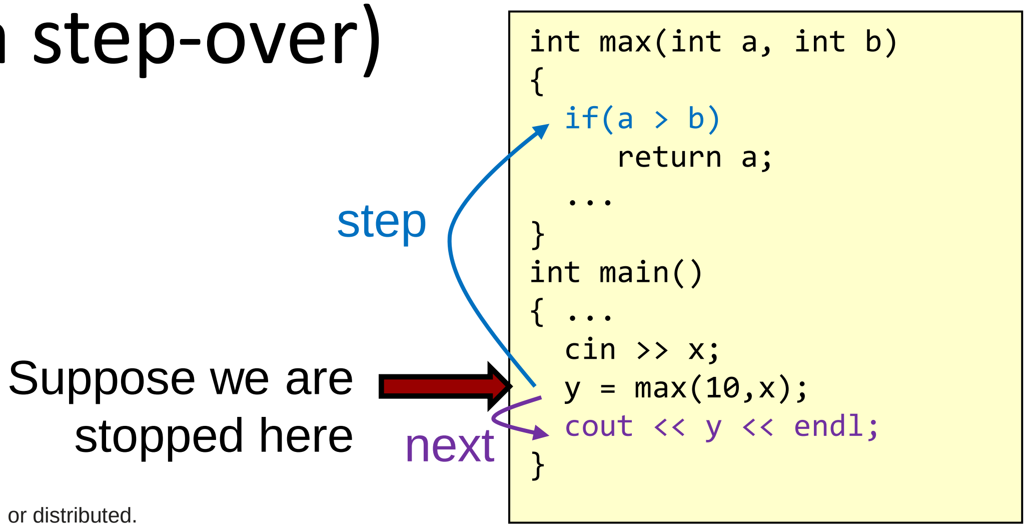
- Compile your program and include the -g flag
  - `$ g++ -g search.cpp -o search`
- Then start the debugger giving it the program you want to debug (not the source code)
  - `$ gdb ./search`

Breakpoints

- Stop program execution at a given line number
- Set before you start the program in the debugger
  - `(gdb) break 36`
- Get the program running
  - `(gdb) run`
- It will run and then stop when it reaches the code on line 36

Stepping

- Type step to execute one line and then stop (aka single-step)
  - (gdb) `step`
- If you get to a line with a function call that you don't want to go into but just have execute fully, type next (aka step-over)
  - (gdb) `next`



```
int max(int a, int b)
{
  if(a > b)
     return a;
  ...
}
int main()
{ ...
  cin >> x;
  y = max(10,x);
  cout << y << endl;
}
```

Printing values

- At any point in time you can print a variable or an expressions
  - `(gdb) print size`
    - Would print the value of the size variable
  - `(gdb) print nums[i]`
    - Would print the value in nums[i]
  - `(gdb) print nums[i] == target`
    - Would print true or false

gdb TUI / GUI

- Gdb has a text-based user interface (aka "TUI") that you can enable if desired:
- To see both the source code and the debugger command area, type:
  - `(gdb) layout src    (turns on src display)`
  - `(gdb) tui disable  (turns off src display)`
- You can also toggle between the two display modes using the key sequence: `Ctrl-X, a`
- To move your cursor between scrolling in the source window and entering commands in the gdb command terminal, use: `Ctrl-X, o` to toggle back and forth
- A web-based GUI for gdb: https://www.onlinegdb.com/

Exercises

- Practice with gdb in Codio
  - search
  - sumpairs
  - count
  - primes

Codio Exercises 2

- Practice using a debugger
  - histogram (break at line 8)