COEN 79L - Object-Oriented Programming and Advanced Data Structures

Lab 1: Getting familiar with C++

1) Count the number of alphanumeric characters and non-alphanumeric characters from the standard input (`cin`). Spaces (' ') should not be counted toward either type of character.

- Example:
    - `"Hello, World!" has 10 alphanumeric characters and 2 non-alphanumeric characters.`
    - `"Santa Clara University" has 20 alphanumeric characters and 0 non-alphanumeric characters.`

2) Display the following pattern using C++ `string` and `setw()`

```
0123456789
 9876543210          0123456789
   9876543210            0123456789
     9876543210              0123456789
       9876543210              0123456789
         9876543210
```

3) Convert to uppercase and display all the words from an input file with length of at least 10 characters. All punctuation marks are removed and do not contribute to the words' length. The name of the file to be read should be read in as a command line argument.

- Example:
  Here is the output with the Gettysburg Address as the input:

```
PROPOSITION
BATTLEFIELD
ALTOGETHER
CONSECRATE
CONSECRATED
UNFINISHED
GOVERNMENT
```

Lab 1: Getting familiar with C++

Compiling with gcc (or g++) and using gdb

Suppose you have a file called main.cpp containing your c++ code. You should compile it with the following command:

```
g++ main.cpp -o main
```

In order to compile with all warnings enabled and to produce ANSI C compatible code, use these flags:

```
-Wall
```

You also need to put a -g flag to tell the compiler that you may want to debug your program later.

The final command turns into:

```
g++ main.cpp -g -Wall -Werror -o main
```

Provided you've compiled your program with the debugging symbols enabled, you're ready to start debugging. Any time there is text you should replace, I've put it in <angle brackets>.

Starting GDB
To start GDB, in the terminal,

```
gdb <executable name>
```

For the above example with a program named main, the command becomes

```
gdb main
```

Setting Breakpoints
You'll probably want you program to stop at some point so that you can review the condition of your program. The line at which you want the program to temporarily stop is called the breakpoint.

```
break <source code line number>
```

COEN 79L - Object-Oriented Programming and Advanced Data Structures

Lab 1: Getting familiar with C++

Running your program
To run your program, the command is,

```
run
```

Looking at the code
When the program is stopped, you can do a number of important things, but most importantly you need to see which part of the code you've stopped. The command for this purpose is "`list`". It shows you the neighboring 10 lines of code.

`Next` **and** `Step`
Just starting and stopping isn't much of a control. GDB also lets you to run the program line-by-line by the commands 'next' and 'step'. There is a little difference between the two, though. Next keeps the control strictly in the current scope whereas step follows the execution through function calls.

Suppose you have a line in the code like

```
1. value = f1()
2. f2();
```

If you use the `next` command, the line gets executed and the control advances to the next line, `f2()`, where you can perhaps examine `'value'` to get an idea of how `f1()` worked.
But if you use the `step` command, you get to follow what `f1()` does directly, and the control advances to the first line of `f1()`, wherever it is.

Examining your Variables
When you want to find the misbehaving portion of your program, it often helps to examine local variables to see if anything unexpected has occurred. To examine a variable, just use

```
print <var name to print>
```

Note: You can also modify variables' values by

```
set <var> = <value>
```

You can modify variables to see if an issue is resolved if the variable has another value or to force the program to follow a particular path to see if the reason for a bug was due to a variable having the wrong value.

Setting Watchpoints
Setting watchpoints is like asking the debugger to provide you with a running commentary of any changes that happen to the variables. Whenever a change occurs, the program pauses and provides you with the details of the change.
The command to set a simple watchpoint is

Lab 1: Getting familiar with C++

```
watch <var>
```

Here's some example output when GDB pauses due to a change in <var>:

```
Continuing.
Hardware watchpoint 2: variable

Old value = 0
New value = 1
0x08048754 at main.cpp:31
31        variable=isalpha(ch)
```

Note: You can only set watchpoints for a variable when it is in scope. So, to watch something within another function or an inner block, first set a breakpoint inside that scope and then when the program pauses there, set the watchpoint.

`Quit`
To stop your program, when it is paused, use kill and to quit GDB itself, use quit.

The TA will show you examples of how to use gdb.

COEN 79L - Object-Oriented Programming and Advanced Data Structures

Lab 1: Getting familiar with C++

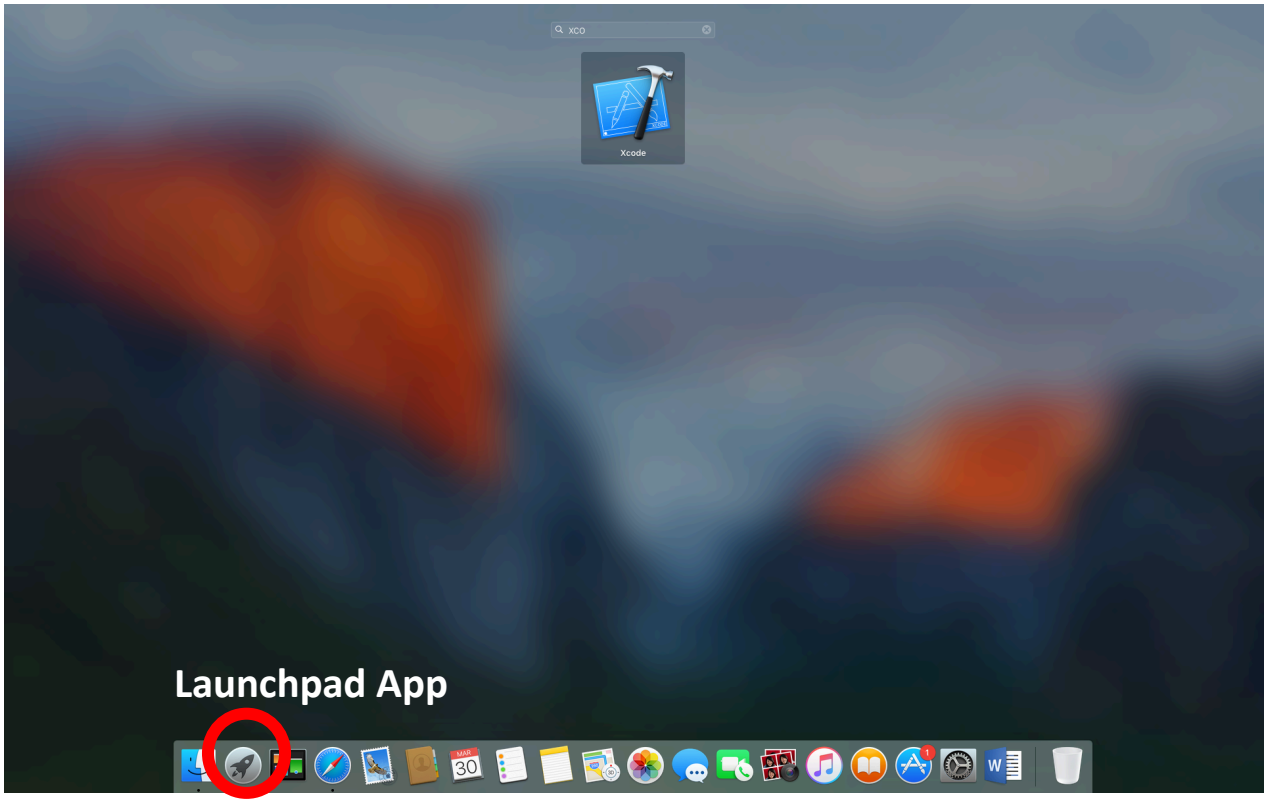
XCode Manual

Introduction

This manual was designed to guide you to navigate the Xcode IDE. By the end of this tutorial you should be able to create a new project, add and delete files to that project, and have an understanding of using the debugging tools such as setting breakpoints and examining variables.

Launching Xcode

You can find Xcode using the Launchpad app.

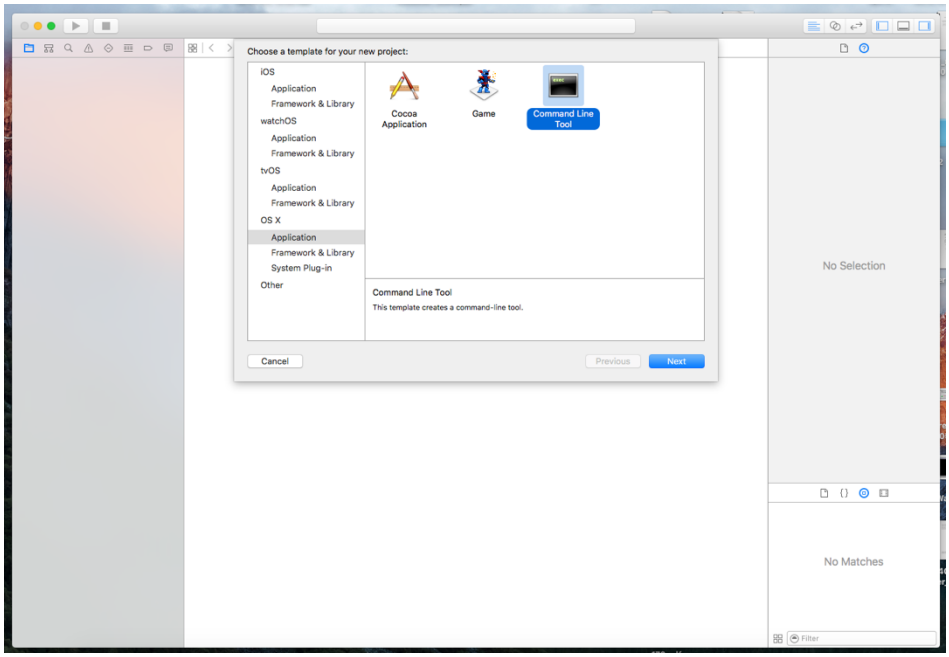


Creating a Project

When you open Xcode, this splash screen shows up. To create a new project, click on the "Create a new Xcode Project" button. Alternatively, you can create a new project by pressing `CMD+Shift+N` or by clicking `File->New->Project` in the toolbar on top.

Lab 1: Getting familiar with C++

Now, click the *Command Line Tool* option.



The next step is to configure your project's options. For product name you would put the name of your project, i.e., "Hello World", "Lab 1", etc.

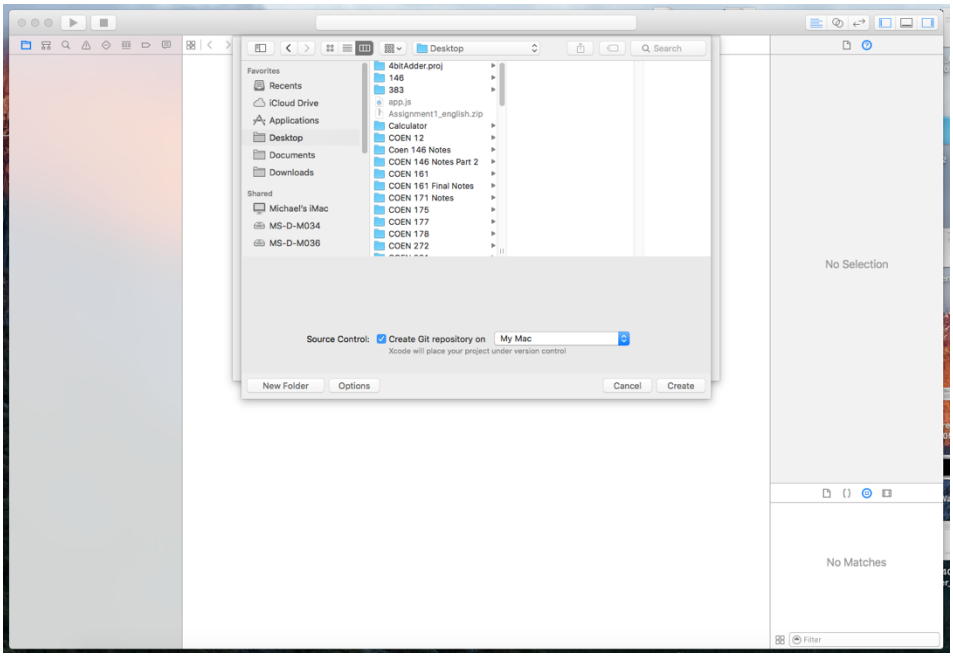
For organization name, put "SCU".

Lab 1: Getting familiar with C++

For Organization Identifier, you can put in a random string, for our case it serves no purpose, but is required to go onto the next step.

Since we are doing this course in C++, select C++ for the language.

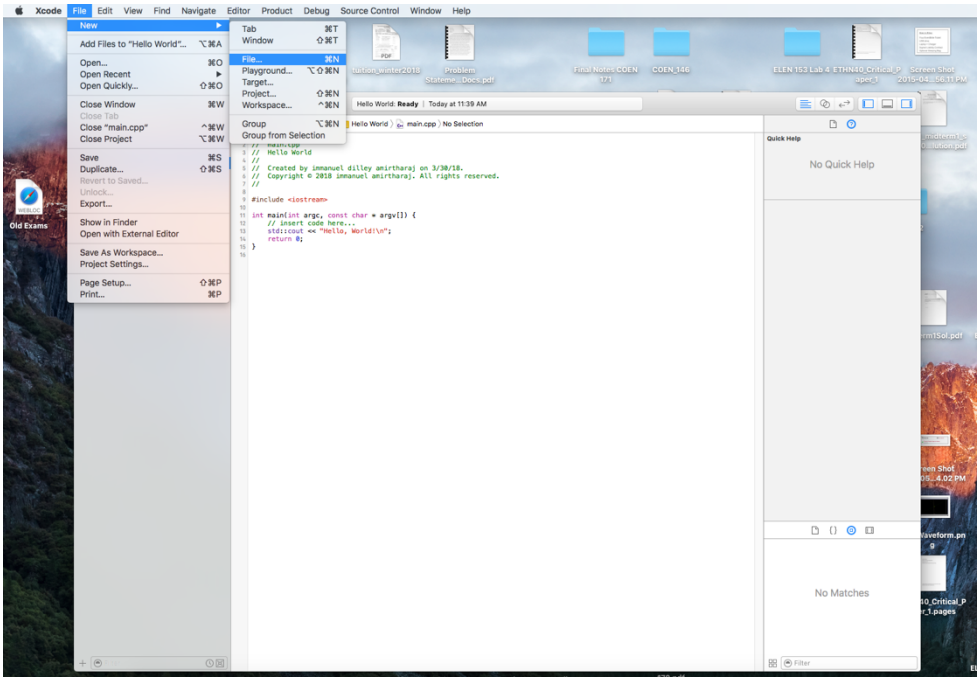
The next step will be to select the filesystem to save your new project. Ideally you would want to create a "*COEN 79 labs*" folder and save all your Xcode projects in there.

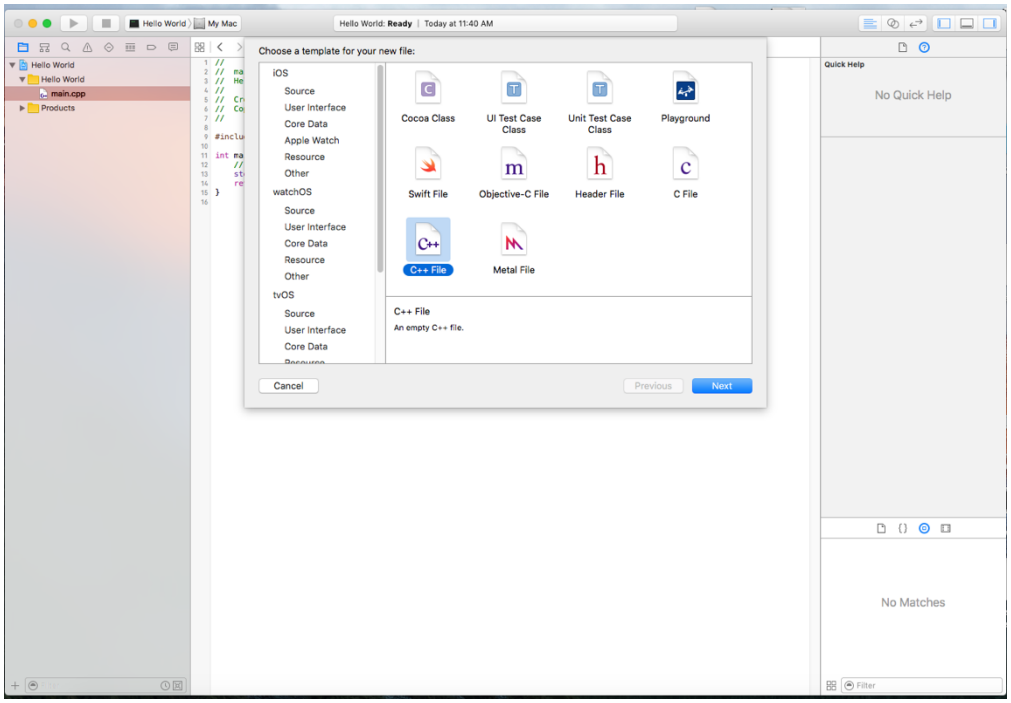


Adding New Files to your project

To add new files to your project, go to `File->New->File` or press `CMD+N`.
Now, select the C++ file option.

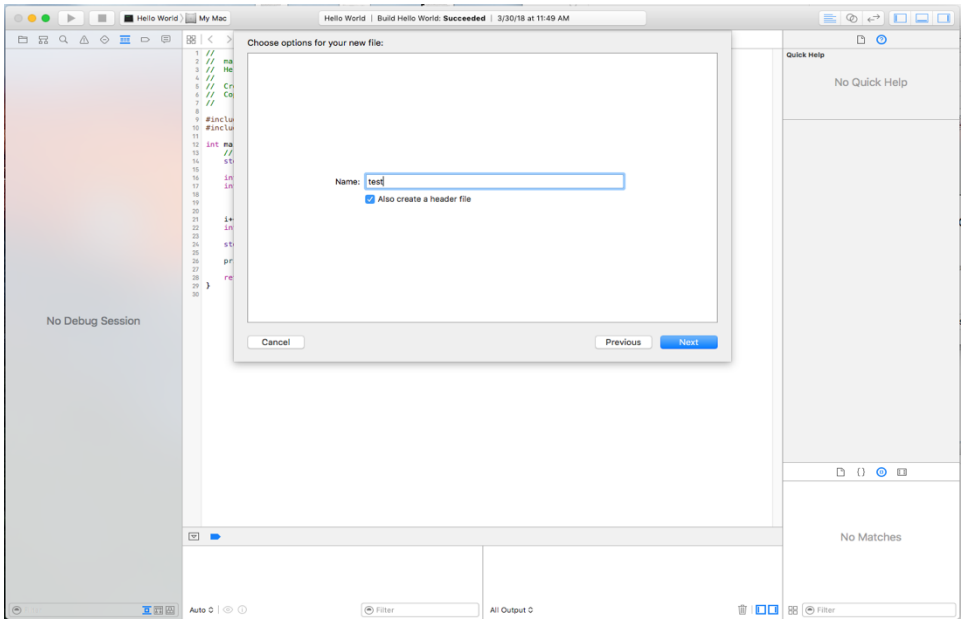
Lab 1: Getting familiar with C++



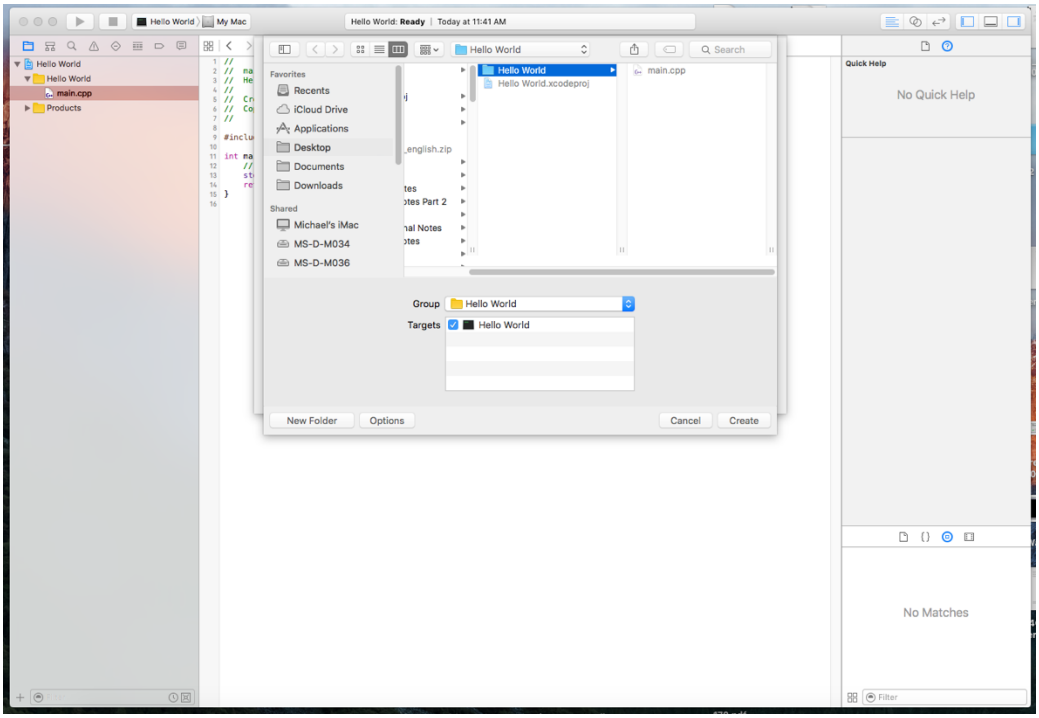


Create a name for the file and then select the create a header file option. In this case we will call the file 'test'. This step will create a test.cpp and test.hpp file.

Lab 1: Getting familiar with C++



Now select the destination to save the files. New files should be saved in the same directory as main.cpp. Make *group* and *targets* settings stay the same.



COEN 79L - Object-Oriented Programming and Advanced Data Structures

Lab 1: Getting familiar with C++

Sample Code

In `test.cpp` enter the following:

```
8
9  #include "test.hpp"
10
11
12 void print_hello() {
13     printf("print_hello just got called\n");
14 }
```

In `test.hpp` enter the following:

```
8
9  #ifndef test_hpp
10 #define test_hpp
11
12 #include <stdio.h>
13
14 void print_hello();
15
16 #endif /* test_hpp */
17
```

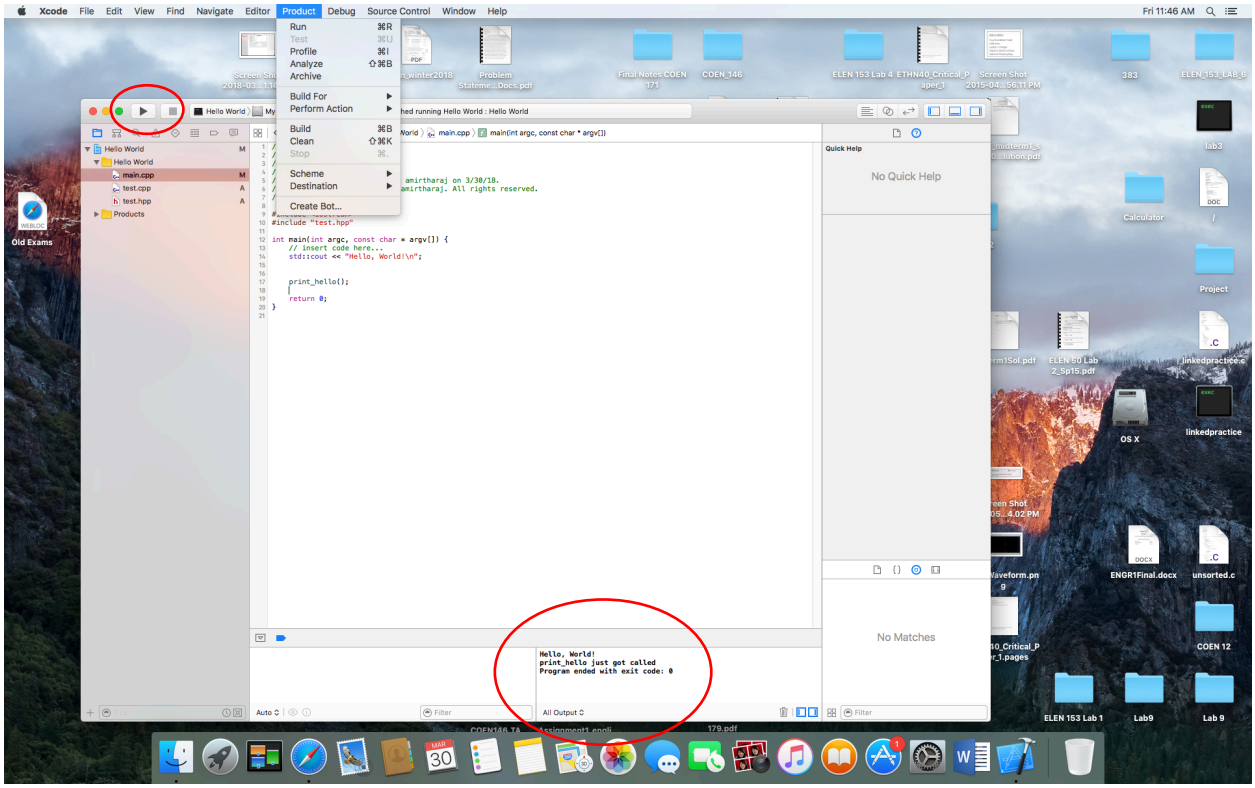
Your main.cpp file.

Lab 1: Getting familiar with C++

Running your Project

To run your project, press the Play button on the top left side. Alternatively, you can go to Product and then Run, or press CMD+R.

You should now be able to see your output in the bottom of the screen.



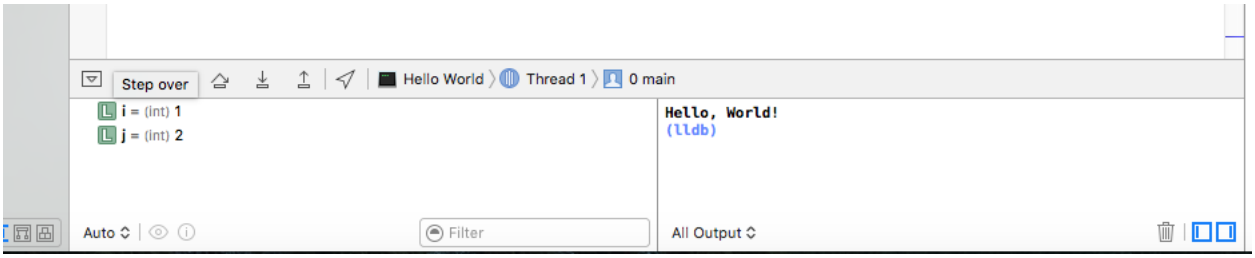
Lab 1: Getting familiar with C++

Debugging

The benefit of using an IDE such as Xcode is that it gives users the ability to inspect their code at runtime. This can be helpful in catching unexpected behavior that occurs in your program.

To your existing code, add the following line to your code. Then click on the numbers on lines 20 and 23 to add breakpoints. On line 20 we will be able to see the variables `i` and `j` and on line 23 we should be able to see the new values of `i` and `j`, as well as the new addition of `k`.

Now run your code. If everything worked fine, you should be seeing the following on the bottom of your screen. Since the program has paused on line 20, Hello World is printed and you only see variables i and j which have been initialized to 1 and 2 respectively.

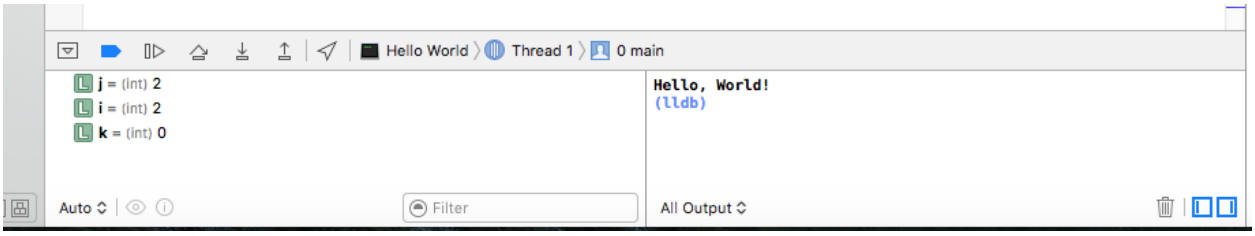


Now press the play button to resume execution. This should now stop at the second breakpoint. Now you will see that i has incremented to 2 and that k has been initialized to 4.

Lab 1: Getting familiar with C++

Press the play button again to finish execution of your program.

Apart from setting breakpoints, Xcode provides a couple other tools to navigate through and inspect your program while it is paused.



1) The play button allows you to resume execution.
2) The step over button allows you to execute the next line and stop.
3) If you are currently stopped before calling a function, the step into button allows you to go inside a function and stops at the beginning.
4) When you are inside a called function the step out button allows you to exit the function and stops at the next line.

Lab 1: Getting familiar with C++

LLDB Manual

Introduction

You can also debug your program on the command line using lldb. lldb is unique only to Apple machines. By the end of this section, you should be able to set breakpoints and run your program using lldb.

Setup and Compilation

Use the `-g` flag when you compile your program. The `-g` flag tells the compiler to enable lldb when debugging. The `-o` flag tells the compiler to create an executable called 'hello'.

```
dcmac12:Hello World iamirtha$ g++ -g -o hello main.cpp test.cpp
dcmac12:Hello World iamirtha$
```

The LLDB Interface

Type 'lldb hello' to open up the lldb program and set the default executable to 'hello'.

```
dcmac12:Hello World iamirtha$ g++ -g -o hello main.cpp test.cpp
dcmac12:Hello World iamirtha$ lldb hello
(lldb) target create "hello"
2018-04-07 17:44:45.587 lldb[16831:1604301] Metadata.framework [Error]: couldn't get the client port
Current executable set to 'hello' (x86_64).
(lldb)
```

Setting Breakpoints

Type '`b main.cpp:20`' to set a breakpoint at line 20 in main.cpp. Type '`b main.cpp:23`' to set a breakpoint at line 23 in main.cpp.

```
(lldb) b main.cpp:20
Breakpoint 1: where = hello`main + 75 at main.cpp:21, address = 0x0000000100000efb
(lldb) b main.cpp:23
Breakpoint 2: where = hello`main + 93 at main.cpp:24, address = 0x0000000100000f0d
(lldb)
```

Running and Inspecting

In the lldb shell, type the command 'run'. This should start the execution of your program.

```
(lldb) run
Process 17265 launched: '/DCNFS/users/student/iamirtha/Desktop/Hello World/Hello World/hello' (x86_64)
Hello, World!
k is 4
print_hello just got called
Process 17265 exited with status = 0 (0x00000000)
(lldb)
```

When you are at the breakpoint, you can now inspect variables. Type '`print i`' to print the value of i.

```
(lldb) print i
(int) $0 = 1
(lldb)
```

To continue running the program to the second breakpoint, type '`next`'. Typing '`continue`' also works.

```
(lldb) continue
Process 17271 resuming
Process 17271 stopped
* thread #1: tid = 0x187d41, 0x0000000100000f0d hello`main(argc=1, argv=0x00007fff5fbffac0) + 93 at main.cpp:24, queue = 'com.apple.main-thread', stop reason = breakpoint 2.1
    frame #0: 0x0000000100000f0d hello`main(argc=1, argv=0x00007fff5fbffac0) + 93 at main.cpp:24
   21       i++;
   22       int k = i+j;
   23
-> 24       std::cout << "k is " << k << std::endl;
   25
   26       print_hello();
   27
(lldb)
```

Type '`continue`' to finish running the program.

```
(lldb) continue
Process 17271 resuming
k is 4
print_hello just got called
Process 17271 exited with status = 0 (0x00000000)
(lldb)
```

More LLDB Commands

- "`br list`" : lists all breakpoints numbered from 1 to n
- "`br del <breakpoint number>`": deletes a breakpoint
- More commands can be found at https://lldb.llvm.org/tutorial.html
- The equivalent to lldb for linux is gdb. If you are using the linux machines, you have to use gdb to debug your code.