CSCI 2270 – Data Structures

Recitation 1, Summer 2020

Objectives:

1. Setup moodle account
2. VS Code editor and g++ installation
3. Run code using VS Code command line
4. Explore functions
5. Read and write files
6. Introduction to Pointers
7. Address-of Operator
8. Dereferencing Operator

1. Moodle Account

All students in CSCI 2270 this semester will be accessing course materials through the Computer Science Moodle:

http://moodle.cs.colorado.edu

To use Moodle, you need to login using your CU identikey and password, and then enroll in your class. Once you've logged in, select the **'CSCI 2270 – Ashraf, Godley - CS2: Data Structures'** class and enter the enrollment key: **"summer2270"** to enroll. Once you're enrolled in the class, you should see the course materials that have been uploaded so far, organized by weeks.

Piazza

We don't have Piazza for this course this semester.

2. Installation of Visual Studio Code editor

You will install the VS Development Environment and use it to write programs this semester.

Log into our Moodle course shell and, under the Recitation 1 section, locate and complete the instructions within the document titled **VS Code Setup Guide.**

3. Programming exercise to run code using VS Code, Passing and handling command-line arguments

3a. Programming exercise to run code using VS Code

- Create a folder (for eg. lab-1) before starting in your home directory.
  Steps:
  1. Open File tab
  2. Select Open Project
  3. A File explorer window opens
  4. Create a folder "lab-1"
  5. Click OK
  6. A folder will be opened in the Project window in VS Code editor.

- Creating a single .cpp file
  1. Right click on the folder in your VS Code Editor
  2. Select New File option
  3. Name the file "hello.cpp"
  4. Type the following contents in the file:

File: hello.cpp

```
#include <iostream>
int main ()
{
    std :: cout << "Hello World!"<< std :: endl ;
}
```

- Building the .cpp file using Command line
  - ❖ Open a terminal (Mac/Linux) or command prompt (Windows)

- ❖ Use the 'cd' command to move to the directory where hello.cpp is present.
  1. The terminal shows you which directory you're currently in.
  2. Use the command 'cd <name>' to move to the directory with that name.
- ❖ Run the command '**g++ -std=c++11 hello.cpp -o hello**'.
  1. 'g++' is the name of the compiler program.
  2. The '-std=c++11' option tells the compiler to use the 2011 version of C++.
  3. 'hello.cpp' is the file to be compiled.

  '-o hello' tells the compiler to write its output to a file named 'hello' ('hello.exe' on Windows). If this is missing, the output file will be named 'a.out' or 'a' by default.

  4. If the last command was successful, there should now be another file named 'hello' ('hello.exe' on Windows). To run the program, run the command './hello' (or simply hello on Windows).

3b. Passing and handling command line arguments

When you run your program it starts by running the main function in your source code. Main function can also receive arguments if it is declared like this:

```
int main (int argc, char const *argv[])
```

Notice that there are two parameters passed to main function from the terminal. The first one, **argc**, is the total number of arguments you passed to the main function when you're running your program on the terminal. The second one, **argv**, is an array storing all the arguments you passed. Change your program to the following code:

File: commandLine.cpp

```
#include <iostream>
int main ( int argc, char const *argv[])
{
   std :: cout << "Number of arguments: " ;
   std :: cout << argc << std :: endl ;
   std :: cout << "Program arguments: " << std :: endl ;
   for ( int i = 0 ; i < argc; i++) {
      std :: cout << "Argument #" << i << ": " ;
      std :: cout << argv[i] << std :: endl ;
   }
}
```

<u>**Example1 : No arguments**</u>

The first example is a straightforward one. Recompile and run your command using the same steps are before. The main function only receives one argument, which is the name of the program itself. Thus, argc is one, and argv is an array of length 1, where the only element in this array is a string “./commandLine”.

Example2 : More arguments

We can pass multiple arguments by typing each one after the function name, separated by spaces. So if we run the program using the command:

```
./commandLine arg1 arg2 arg3
```

Now argc is 4 and argv is an array of length 4. The first string in the array is the program name “./commandLine”, and the rest of them are the strings we typed on the terminal, delimited by spaces or tab.

4. Functions

Having a separate header file to declare function is useful when we need to reuse the function in multiple source files.

Function declaration in C++ or prototype.

File: function.h

```
int add ( int a, int b);
```

Function definition

File: funcdef.cpp

```
#include "function.h"
int add ( int a, int b)
{
  return a + b;
}
```

Calling a declared function

File: main.cpp

```
#include <iostream>
#include "function.h"
```

```
using namespace std ;
int main ()
{
  cout << "2+3=" << add( 2 , 3 ) << endl ;
  return 0 ;
}
```

Compiling multiple files

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

5. File I/O

File I/O is reading or writing from files. C++ uses ifstream and ofstream for reading and writing respectively.

Declaring an instance of file input:

```
ifstream iFile ( "filename" );
```

Similarly for file output :

```
ofstream oFile ( "filename" );
```

File operation modes:

```
ios::app -- Append to the file
ios::ate -- Set the current position to the end
ios::trunc -- Delete everything in the file
```

File mode example:

```
ofstream ofile ( "test.txt" , ios::app );
```

File output example - oFile.cpp

```
#include <fstream>
#include <iostream>

using namespace std ;

int main ()
{
  // File Writing
  //Creates instance of ofstream and opens the file
  ofstream oFile ( "filename.txt" );
  // Outputs to filename.txt through oFile
  oFile<< "Inserted this text into filename.txt" ;
  // Close the file stream
  oFile.close();
}
```

File input example - iFile.cpp

```
#include <fstream>
#include <iostream>

using namespace std ;
int main ()
{
  // File Reading
  char str[ 10 ];
  //Opens the file for reading
  // Ensure that filename.txt is present in the same directory
// as that of the source file
  ifstream iFile ( "filename.txt" );
  //Reads one string from the file
  iFile>> str;
  //Outputs the file contents
  cout << str << "\n" ;
  // waits for a keypress
  cin.get();
  // iFile is closed
  iFile.close();
}
```

6. Pointers

Every variable we declare in our code grabs a space in the memory. So it has an address in memory where it resides. A pointer is a variable that stores a memory address. By means of a pointer variable, we can access the address and the value at that address.

7. Address-of Operator

Consider the following lines of code:

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

int main()
{
	int a = 10;
	cout<< a << endl;
	cout<< &a << endl;
	return 0;
}
```

Output:

```
10
0x7ffccbbcd804
```

The first cout is quite straightforward. It is simply printing the value of a. But what is being printed by the second cout? It is the address of the variable a. The operator '&' is used to get the address of the variable a. Hence it is a referencing operator or address-of operator.

8. Dereferencing operator

Consider the following lines of code

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

int main()
{
       int a = 10;

       // * here denote p is pointer variable
        int *p = &a;
        cout<< a <<endl;
        cout<< p <<endl;

       //* is used to dereference p
        cout<< *p << endl;
        return 0;
}
```

Output:

```
10
0x7ffccbbcd804
10
```

This code is a bit trickier than the previous one. Let's see what is happening in the code. Note that we are storing the address-of *a* in another variable p. Note the type of p. It is int* instead of just int. **Int *p** tells that p is a pointer variable (* denote it as a pointer) and it will store the address of an int variable.

The second cout prints p, the address of variable a. Now, what is the third cout doing? Remember p has the address of a. By putting a '*' before p, we are accessing the value stored at the position addressed by p. Hence, * is dereferencing the address p.

CSCI 2270 – Data Structures

Recitation 1, Summer 2020

Quiz

1. Consider an array **int a[] = {1, 2, 7}.** What is the output for the following?
   a. cout << a+2;
   b. cout << *(a+2);
   c. cout << *a;
   d. cout << *a[0];

Exercise

Your zipped folder for this Lab will have a main.cpp file. Follow the TODOs in that file to complete your Recitation exercise!