Discussion Session
Week 2

Basics of Programming/Early Practice

Learning To Code vs Learning To Program

- Anyone can learn to code
- While learning to program, you will learn to code
- **Programming** is the process of solving a problem that's been given to you
    - This is the key point to remember while working/learning
- **Coding** is writing the code to support your solution to a problem in a given syntax

The best way to learn to program is to do it….A lot

- LeetCode, Kattis, HackerRank, and Programiz are great websites to use for practice problems

C vs C++

- You can think of C++ as just “C with Libraries”
- C code can be used in C++ programs, C++ cannot be used in C programs
- C has 32 keywords, C++ has 52
    - You don’t need to memorize them but you should be able to recognize when they come up
- C++ is more object oriented, C is function-driven
- In most of your classes, you’ll be working in C++ at URI
    - 212 is typically taught in C++, 412 uses C early in the semester and C++ later on
    - Electives like 415 will also have you using C++

Refreshers

- C and C++ are strictly typed
  - You must explicitly declare the type of a variable(or function) when creating it
  - If a function does not return anything (yes this is possible), then it must be declared as a **void** function
  - Main must always be declared as an int
- **Comments are your friend**
  - Get used to writing them and reading them
  - Yes I will be mentioning this just about every week :)
- All C/C++ programs must have a main function
  - We can get away with never writing a main function in Python, but a C/C++ program wouldn't compile without it

Sample Program

See if you can figure out what this program is doing by reading the comments

```cpp
/**
 * Sample C++ program showing some syntax and what code should look like
 **/

/***********************
 *      Includes       *
 ***********************/
#include <iostream>
#include <stdio.h>

/***********************
 *      Defines        *
 ***********************/
#define MAX_INPUT 20

//Factorial function computes the factorial of a given number
int fact(int num);

int main(int argc, char* argv[])
{
    int input = 0;
    int sumOdds = 0;

    //Take in inputs from the user repeatedly until the user enters something greater than 20
    for(std::cin >> input; input < MAX_INPUT; std::cin >> input)
    {
        //Print factorial of input if even, add to sum otherwise
        (input % 2 == 0) ? printf("Factorial of %d is %d\n", input, fact(input)) : sumOdds += input;
    }

    std::cout << "Sum of odd inputs: " << sumOdds << std::endl;

    return 0;
}

int fact(int n)
{
    if(n <= 1) return n;
    return n * fact(n-1);
}
```

Types of Problems

- Decision
- Sorting
- Search
- Optimization

Functions

- In general, you want as little code in Main as possible
  - Think of how we utilized main in 110 and how much work was being done
  - Typically you'll use main to handle some basic I/O, and call helper functions to solve problems
- Functions are essentially blocks of code that can be called repeatedly
- Function syntax
  - functionReturnType functionName(param1Type param1Name, param2Type param2Name……)

    {

        Function body

        Return statement

    }

- Parameters are arguments to your functions
  - Think of parameters like x in the math expression f(x)

Best Practice

- Some things are mandatory (we will receive compiler errors if they're missing or wrong)
  - For instance, if a function is called in Main before it is declared, that's an error
- camelCase, PascalCase
  - Gives readability to your programs
  - Everyone will choose what suits them best, but I use
    - camelCase for variableNames
    - PascalCase for ClassNames and ConstructorNames
- Common cpp file to get used to
  - Includes, defines, and other preprocessors are at the top
  - Local function declarations and contracts are written next
  - The main function follows those function declarations
  - The local functions are then defined below main

Exercise One

- Write a program that does the following
  - Creates a function that does not return anything
    - In that function, print “Hello World”
    - Hint: Think about the kind of function definitions we’ve talked about
  - Creates another function that adds two integers together and returns that sum
  - Call both of these functions in main
    - Print out the result of your sum function

Solution

Note that main can come before or go after your functions

```cpp
#include <iostream>

void printHello()
{
    std::cout << "Hello World" << std::endl;
    return;
}

int sum(int a, int b)
{
    return a+b;
}

int main(int argc, char* argv[])
{
    printHello();

    std::cout << sum(5, 10) << std::endl;

    return 0;
}
```

Reading Documentation

- Cppreference is the goto resource for C++ programming
  - This website will give you just about everything you'd ever need to know about the language
  - It looks pretty antiquated, but it's kept up to date and things are grouped together meaningfully
- When you understand this site, you will outperform others who cannot understand it

Example of the site

Let’s take a look at std::min

Note the anatomy of the page

- Where this function originates
- Links to other reference pages (such as std::initalizer_list)
- What the function does
- Parameters
- Examples

Preprocessors

#include

- Access a user-defined or standard library
- More or less “Copy and paste a file at this location”

#define

- Create a variable or function (different use cases)
- Typically comes before any functions and just after your includes
- #define PI 3.14

More niche preprocessors

#ifndef

- Allows us to check if something is defined prior to compilation
  - If not, execute a block of code
- There's some discourse about choosing #if vs #ifdef vs #ifndef

#endif

- Basically, these expressions occur before compilation

```c
#define VERBOSE 1
#define USE_REPLACEMENT_MANAGER 1
#if USE_REPLACEMENT_MANAGER
  #include "replacementManager.h"
#else
#include <stdlib.h>
#endif
#include <stdio.h>

void test();

int main() {

  // initalize space if we're using the replacement memory manager
  #if USE_REPLACEMENT_MANAGER
    init(1000);
  #endif


  // just a call to set the verbose mode to true
  if (VERBOSE) {
    verbose_mode();
  }

  return 0;
}
```

Include

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

#include "../Utilities/Map.h"
#include "../Utilities/Log.h"
#include "Dispatcher.h"
#include "ChildProc.h"
```

- This is where the “++” part of C++ comes in
- There are many functions, classes, structs, and other artifacts predefined for you
- If you want to use a function defined in a library, you must first include that library
  - #include <libraryName>
  - #include “localFileHeaderName”

Caveats

- Although many utility functions are written for you in libraries, be careful when completing assignments
  - Something like std::accumulate only takes a few lines to write on your own and will give you some much-needed practice
- The vast majority of the time, you will need to write your own functions (with some exceptions)
  - Sometimes you'll be given starter code or work on similar problems in lab
  - **Use these resources to help you**
- Many beginner level programs are solved widely online
  - Don't copy and paste a solution….you'll get caught
  - Even if you're able to pass, you won't learn anything and be in big trouble when you enter 212

Exercise Two

Using the cmath/math.h library, write a program that does the following

- Takes in two numbers on standard input
- Prints out the greater of the two numbers
- Prints out the result of raising the larger number to the power of 6

Compiling

- Exposure to g++ is really important (and will not become any less so as time goes on)
  - There are many other things that can be done besides simply compiling a program
- Default executable name is a.out
- -no-pie -g -O0 -Wall -Wextra -Werror -Wfatal-errors -std=c++11 -pedantic
  - And many, many other compiler flags
- Each flag has its own purpose
- Eventually (beyond scope of this course), you will learn about linking while compiling
  - Basically a means of including libraries that are not standard with C++
    - Libraries you have defined or downloaded locally
- Learning some flags and using them will make you a better programmer
  - Werror treats warnings as errors and forces you to be more careful
  - Wall, Wextra, Weverything adds more warnings to your code
  - std=c++11 specifies that this program uses C++11
  - etc