Introduction

Most books are top-down which makes it hard for high school students, ie. Object-oriented programming. So, I will be teaching bottom-up, procedural programming. This is typical of older languages such as BASIC and scripting languages. The problem with top-down is like trying to teach you calculus without teaching you algebra first!

So why learn programming? Yes, it can be a career, but more importantly, it will teach you problem solving; organization; writing instructions explicitly and in the correct order; expanding your working memory; and improving your math. Yes, it will do all that for you; only if you persist!

A basic component of computers is memory. Two things that go into memory are data and code, together we call them a program. By the way, data is the plural of datum. Code and data can be mixed in memory, but most languages will separate code and data. Most modern operating systems are multi-tasking and to ensure security, the OS will give a program a chunk of memory and will forbid access outside of this chunk.

Each piece of code and each piece of data will be put into its own memory location (or address). Where it goes depends on the compiler and linker. If you have a global data, it generally has a fixed memory location for the life of the program. Code almost never moves in memory while executing.

The essence of any program is to manipulate data using some logic. We calculate some values, store them or send them to the OS to be outputted to some hardware interface or device. More complicated programs simply have more code, logic, and data. So learn the basics well!

In the beginning, you will be exposed to some advanced code. Just use it and be aware that you will fully understand it later in the course.

Chapter 1 – Printing and Simple Methods

The basics of any program are to display information whether it is text or graphics. So we start with displaying text. Displaying text can show calculated information or just to let you know where you are in the code and the order that it is executing. This is important because the code may not be running in the order that you expect.

Note: For ALL assignments in this course, comment your code with your name, the assignment, and the date. Use "//" to start a comment. For example:

```
// Mr. Amory KC Wong
// Exercise 1.1
// 2015-08-16
```

Exercise 1.1: Modify Example_1_1 to print: "*your name* is now programming!"

Some prettier printing can be achieved with escape codes. You also need escape codes to generate characters, like backslash and quotes. Fortunately, the main ones are common between C++, Java, and Objective-C. The tab escape code can be useful to generate prints in column format.

Exercise 1.2: Use tabs and new lines to print the following (use Example_1_2 as a template):

```
Num    Squared
1      1
2      4
3      9
4      16
5      25
```

Simple methods can make doing repetitive tasks with fewer errors. The goal is to make a working method that can be used repeatedly; useful methods can be re-used in future programs to save coding time. In fact, you have already been using the library print method!

Exercise 1.3: Use methods (use Example_1_3 as a template) to print the following (exactly):

```
**********
*        *
*        *
*        *
**********
*        *
*        *
*        *
**********
```

Exercise 1.4: Use your judgment to create a solution to print the following:

```
*
**
***
****
*****
******
*******
```

Exercise 1.5: Use the ASCII character set bitmap to write your enlarged name. Example:

```
*   *                   *   *
*   *                   *   *
** **                   *   *
* * * * **              *   *  ***  * **   ***
* * * **  *             *   * *   * **  * *   *
*   * *                 * * * *   * *   * *   *
*   * *                 * * * *   * *   * *   *
*   * *       *         ** ** *   * *   *  ****
*   * *      ***        *   *  ***  *   *     *
              *                           *   *
                                           ***
```

Exercise 1.6: Do ASCII art minimum 30 x 30 characters.

The above problems are all considered static because nothing changes – it always does the same thing. So we now look at some dynamic programs that change based on user input. The dynamic items are stored in variables until the variables go out of scope.

Exercise 1.7: Modify example_1_4 to do the following:

```
What is your name?  Amory
Amory, you are a programmer!
What is the name of the person sitting beside you? Joe
Joe is an artist!
Amory and Joe make a good team!
```

Exercise 1.8: Make up your own input and output program using 1.7 as an example. Need at least 2 input strings.

Email me the exercise source files (not the whole folder) when you are done.