Lab 3: Structs and Pointers to Functions

CPSC 1021 – Spring 2020

CLEMSON School of COMPUTING

Introduction

During this lab you will:

1. Practice reading the existing code.
2. Use structs and pointers to functions.
3. Create and submit a tarball.

This lab will integrate and build on concepts and code from Lab 2 to produce a complete understanding of the following topics.

Part I – Review of structs

A *struct* (short for structure) is a unique structure. It is an *aggregate* data type, that is, it consists of several variable members that can have different types. For example:

```
struct Employee{
      int age;
      String name;
      boolean cashier;
};
```

In this case, the Employee struct has 3 members within it. In order to access a variable within a data type, you will use the *dot (.) operator*. For example, if you want to set *Employee's* name then you would simply type:

```
Employee fred;        // declare a struct named fred
fred.name = "Fred";   // set fred's name to Fred
```

If you have a pointer to a struct in your program, you can access struct members via an *arrow (->) operator*, as shown below.

```
Employee* ePtr;        // declare a pointer to a struct
ePtr->name = "Fred";   // set the name to Fred
```

Part II – Review of pointers to functions

You are already familiar with pointers and functions. A pointer is an address of the variable it is pointing to. A function is a collection of operations that solve a problem or a part of a problem. Functions also live in memory, so you can also declare a pointer to a function. Function pointer would point to a location where function (the actual function code) resides in memory. You have already covered them in the lecture portion of the class.

For example, below you have a function called *switchFunction* and a pointer to it.

```
int switchFunction(int a, int b){
      a += b;
      return a;
}

 //function pointer that points to switchFunction
 int (*funcPoint)(int, int) = switchFunction;
```

Please notice that the pointer to this function has the same return type as the function it points to (an integer in this case), and the same number and type of parameters (two integers). In correct syntax is to place an * next to the name of the function pointer inside the parenthesis.

You can now use the pointer to call the function, and you will pass the parameters to it, just like you would do to a function.

```
int result;
int num1 = 3, num2 = 5;
result = funcPointer(num1, num2);
```

Part III – Putting it all together

During this lab you will use the provided starter files. Download the tarball from the lab home page on canvas and unzip it to get started. You can check previous labs for help with that, consult the online manual *man*, or ask a TA. The first thing you would need to do is examine the code to figure out exactly what it does. (Yes, this is very similar to the code you have written in lab2).

It is always more difficult to read code written by someone else, so please take the time and go over the code in detail. Once you have done that, you can start implementing the modifications. Please remember to make changes one at a time, and compile and run it.

If you work in a modular fashion, that is, making one change at a time, and compiling and running it, it would be much easier to find and correct errors that if you wrote an entire page of code and then started to debug it.

1. In the header file
   a. Create a struct named “Stats”. This struct should have 2 members, a float and a pointer to a function that can point to any of the calculating functions (calStd, calMean, etc.). Remember that the return type and parameters of the pointer to a function will be the same as in the function it is pointing to, so carefully examine the return type and parameters of those 3 functions. Comments in the

file show you where to add your code.

b. Add two function prototypes to the header file. See comments in the file.

1). The *construct* function should return a float and take the following parameters: a Stats pointer, a float pointer, an int, and a function pointer (able to point to calStdev, calMean, or calVariance).

2). The *output* function will be a void function that takes the following parameters: 3 Stats pointers and a function pointer (able to point to max or min function).

2. In the stddev.cpp file

a. Write the implementation of the functions *construct* and *output*. In the *construct* method your Stats struct function pointer should be set to point to the passed function pointer. The float member of the Stats struct should be set to the value returned by the function call made via the passed pointer.

b. The *output* method should pass in the 3 Stats pointers to the function pointer within the method.

3. In the main.cpp file

a. Right after the for-loop, create function pointers for mean, variance, and stddev of the Stats struct type. You can allocate memory using either the C-style *malloc* or *calloc*, or a C++ style *new*. Do not forget to free that memory when you are done using it. Comments in the file show you where to add code.

b. Complete the output statements by calling the correct data member to display from the Stats struct. You can use either the C-style *printf*, or the C++ style *cout.*

4. Testing and creating a tarball

Thoroughly test your program to ensure it works and create a tarball. Submit it to canvas.

NOTE: If you submit the archive that cannot be open, or is corrupt, you will not be given the opportunity to redo the work and resubmit. You have one shot to get it right. Please use your TAs' help with this step, if you need to.

What/How to submit

That's it, you are done! You can now submit your zipped tarball to canvas to be graded.