# Assignment 4: Counting SLOCs

EE312 – The University of Texas at Austin – Fall 2016

Assigned: Monday, March 1st
Design Due: October 7th , at the **beginning** of your recitation
Program Due: October 7th, before 11:59PM

**Purpose**

- Learning how to do modular design
- Learning how to use file IO commands
- Learning about string manipulation

**Background**

A software engineering professional produces high quality solutions on time and on budget. Meeting this professional obligation requires not only the ability to execute the task, but also the ability to produce good estimates. While many factors enter into the amount of time required to develop software, two important aspects are size and complexity.

In this assignment you will write a tool that provides a very simple measure of size – the number of source lines of code (SLOC) associated with each C function contained in a given file on disk. In future assignments, you will use this tool to help determine the relationship between program size and development time.

**Deliverables**

There are both design (3 points) and code (17 points) deliverables for this assignment.
The design portion requires that you produce these three items on paper:

1. IPO diagram that identifies at a high level: the inputs to the program, the outputs from the program, and the general operations performed by the program.
2. Functional Block diagram identifying the main program and all subfunctions (not including library functions), showing who calls who, and what data is passed to and from each
3. Flow chart or pseudocode of your program's main logic - 1 page should be sufficient detail. Proper notation is required in either case.

Turn these in to your TA at recitation, with your name and unique section number written clearly on each page.

The code deliverable is a Standard C program submitted through GitHub as usual. It will be a file named ***assign4.c*** that contains all the source code and comments that meet the requirements for this assignment.

**Program Requirements**

The program shall print a list of all C functions contained in a given file and the number of semicolons contained within the function to the screen and also to an output file specified by the user. The total number of functions and statements are printed at the end of the list.
NOTE: For this program, by *definition* the number of lines of code in a function will equal the

number of semicolons contained in that function (see rules for recognizing semicolons below). The number of semicolons approximates the number of C statements.

**Output requirements**

For example, if your source code file for this assignment was processed by your program, it might produce the following sample output.

```
File Counting Summary Table

Filename:  assign4.c

Function Name       # Lines of Code

main                    21
readSource               9
detectFunction          25
countLOC                 8
printTable              11

Total Functions: 5
Total Lines of Code: 74
```

So, for each function in the file, you are to:
1. identify the function name
2. count the number of semi-colons contained in the function definition
3. however DO NOT include any semi-colons that are:
    - contained inside of comments (e.g. /* hi there;  */)
    - contained inside of a character or string literal (e.g. “Hello; World”)
    - that lie outside of a function definition (e.g. global variable declarations)

**Exceptions**: Your programs should handle and report on the following run-time exceptional conditions:
1. there was a problem opening the specified input file
2. no functions were found in the file
3. mis-matched begin-end of a function

**Required Assumptions**: To simplify the problem, you are required to create the following definitions at the beginning of your source code file for use in delimiting the beginning and ending of each function definition

```
#define BEGIN_FCN  {
#define END_FCN    }
```

With these definitions you may then assume that each function definition begins with its function header followed by a line that has only BEGIN_FCN on it.  You are further allowed to assume

that at the end of each function there will be a line with only END_FCN on it.

**For example, suppose the following function was included in your source code file**:

```
void fillArray(int a[])
BEGIN_FCN
 int i, j;
 fill = -3;  /* initialization; neg value */
 for( i = 0, j=2; i < sizeof(a); i++, j--)
   a[i] =  i + j * fill;    /* of course ; a
  comment may span multiple
  lines; pages */
 printf ("departing fill array function;");
 return;
END_FCN
```

***The above function would generate a SLOC count of 7***

**Input requirements**

The initial input to your program is the name of a file provided by the user when prompted. This is done for each file to be processed by your program until a sentinel of STOP is input by the user. After checking to make sure that the given file on disk can be opened for reading, your program will then input each line of text (one line at a time) to be processed for counting SLOCs. After outputting the File Counting Summary Table for that file, then ask the user for the next file to be processed.

**Design requirements**

Each line of the input file should be represented as a string. You are allowed to take advantage of any of the standard string manipulation functions contained defined in **<string.h>**.

You may use as many functions as you find necessary to improve modularity and understandability of your program. **You must include a minimum of two functions (of your own design)** besides the main function. These should be written with a view of reusing them in a future ee312 course assignment.

**Required Test Case(s):**

At a minimum, your program must successfully count the programs you wrote for Assignments 2, 3 and 4 (with appropriate modifications to support function delimiting). The program will also be tested with source code produced by the TA's.