STATEMENT OF WORK

3

Assignment 3

CSE 6329 – Spring, 2018

**Cyclomatic Complexity**

**Summary**

The purpose of this assignment is for you to learn how to draw flowgraphs of programs and how to compute cyclomatic complexity. The assignment will involve two phases:

**Phase 1**

1. **Produce a flowgraph** and **determine the cyclomatic complexity** of a “main” program that will be provided to you. This program calls a function.
2. **Develop the function** (in the “C” programming language), **produce a flowgraph of the function** and **determine its cyclomatic complexity**.

**Phase 2**

**Produce flowgraphs** and **determine cyclomatic complexity for three functions that other people develop** -- you will analyze three small C functions written by others, and determine their flowgraphs and cyclomatic complexities.

## Reference Material

The blackboard contains the following “main” program, which you are to use in phase 1a): **A3 CSE6329 2018sp Main Program.txt (or maybe .c)**

This is a “main” program that calls the function. It may help you understand how the function is used. You may also use it, if you wish, to test your function, although that is not required.

The blackboard contains the following template for the function you are to write in phase 1b): **A3 CSE6329 2018sp template.txt (or maybe .c)**

This template consists of the shell of the function. You are to edit this so as to provide the actual function.

Appendix A of this SOW contains the requirements for the function. This is a C function and should take a few pages of C code.

The class notes for modules 19-20 should be sufficient to enable you to do the complexity analysis and flowgraphs for this assignment, although you may need material from earlier modules to actually write the function. Further details of flow diagrams and cyclomatic complexity can be found in chapter 9 of Fenton.

**Requirements for flowgraphs:**

1. **Label all arcs that leave predicate nodes** (nodes that make a decision) so it is easy to determine which arc represents which decision (such as “yes” or “no”).
2. **Number all nodes** with the line numbers of the corresponding lines of C code, so it is easy to tell which nodes correspond to which lines of code in the function or program being graphed.
3. **Use a notation for your flowgraphs that is similar** to the one used in class (module 20) or else provide an explanation of your notation.
4. You may do this in any tool you wish but must **turn in the flowgraph as part of a Word document**. You can draw it in Word or, if you prefer, draw it in some other tool and paste into the Word document (I recommend pasting it as a bitmap to avoid problems).

**1.0 Work to be performed for this assignment**

* 1. **Phase 1**
     1. **Main Program**
        1. **Produce a flowgraph of the main program.** (See the requirements for flowgraphs, above)
        2. **Compute the cyclomatic complexity of the main program.**
     2. **Function**
        1. **Write a C function that satisfies the requirements indicated in appendix A.** The template mentioned above provides the shell of the function, so that would be a good place to start. There is no requirement to compile, build or run your function, but you may do so if you choose. Should you wish to test your function, you may use the “main” program (see file name, above). You will submit your function, as a text file.
        2. **Produce a flowgraph of the function.** (See requirements for flowgraphs, above.)
        3. **Compute the cyclomatic complexity of the function.**

The next-to-last page of this SOW contains a **phase 1 cover sheet**, which should be placed at the front of the Word document containing the two phase-1 flowgraphs. The cover sheet has a place for you to show the calculation of the Cyclomatic complexity for the “main” program and the function. Your grade for phases 1 and 2 will be recorded on this cover sheet after both phases have been graded.

**1.1.3 Hint**

When developing the C function, you may find it easier to draw the flowgraph first, before producing your C program. I recommend doing the flowgraph by hand until you have it right, then draw it on the computer.

* 1. **Phase 2 – Review and evaluate three programs provided to you by the instructor or grader.**
     1. **You will receive three text (or c) files**, each with a different file name, and each containing a C function similar to the one you wrote in phase 1, but written by another student (or team) during phase 1 of this assignment.
     2. **For each function (a total of three), do the following**:
        1. Review the function and **note any errors**.
        2. **Produce a flowgraph** of the function. (See requirements for flowgraphs, above.)
        3. **Compute the cyclomatic complexity** of the function.
        4. **Turn in a Word document** containing the flowgraph, using the phase 2 cover sheet.
        5. On the **phase 2 cover sheet** you should show the following:
           1. The file name of the function you evaluated.
           2. A description of any errors you found and how to correct them.
           3. The complexity calculation information (same as what you did in phase 1 for your own function).

**3.0 Deliverables**

**3.1 Phase 1.**  You will turn in two files, as follows:

**3.1.1 A text file containing your C function.** The file name will be:

**A3 CSE6329 2018sp Function last first.txt (or .c)**

**Note**: your name(s) should not appear within the function – only in the file name.

**Suggestion**: you should start with the template file:

**A3 CSE6329 2018sp template.txt (or .c)**

**3.1.2 A Word document containing the phase 1 cover sheet** (see cover sheets at the end of this SOW) and the **flowgraphs** of the main program and your C function**.** The cover sheet will show your computation of the cyclomatic complexity for each of these.The file name will be:

**A3 CSE6329 2018sp Flowgraphs last first.docx**

**Make sure your last name and first initial are included on the cover sheet** and in the file name! For teams, include both names (first initials only – otherwise the file name will be too long).

* 1. **Phase 2.**  You will turn in three files, as follows:
     1. You will receive **three** text files with different names.

For example, a file might have the name: **BLUE12.txt (or .c)**. Each file will contain a version of the C function, as written by some other student or student team.

**You will do the analysis described above and turn in three Word documents**, one for each of the three text files, as follows:

**3.2.2 Turn in a Word document containing the phase 2 coversheet and your flowgraph.**

(If you find an error, include a discussion of what was done incorrectly and how to correct it). Your error discussion and the computation of the cyclomatic complexity will appear on the phase 2 cover sheet.The file name will be:

**A3 CSE6329 2018sp <text file name> last first.docx**

For example, if the text file was **BLUE12.txt**, your file should be:

**A3 CSE6329 2018sp BLUE12 last first.docx**

**Make sure your last name and first initial are included** on the cover sheet and in the file name! For teams, include both last names and first initials, in alphabetical order by last name.

***REMEMBER TO PUT THE APPROPRIATE COVER SHEET [last two pages of this SOW] ON THE FRONT OF EACH WORD FILE THAT YOU SUBMIT.***

Appendix A

**Requirements Specification for C Function**

1. C Program Functional Requirements – Sort an array of integer numbers and calculate Mode, Mean, Median, Variance, and Standard Deviation.
2. Program Purpose:

The program will be a C function *Stats ()* which calculates various statistics on an array of integer numbers, and returns a validity status on the input data. The function will comply with the following prototype:

\_Bool *Stats* (int *size*, int *inArray[]*, int *outArray[]*, int *\*mode*, float *\*mean*, float *\*median,* float *\*variance*, float *\*stdev*);

The function has two inputs and seven outputs as detailed below.

**Program Inputs:**

The function input is comprised of the first two parameters in the *Stats*() argument list:

1. *size* is of type **int** and equals the number of elements in the input array (*inArray[]*).
2. *inArray[]* is an unsorted array of integer numbers of type **int**. The numbers may be positive or negative. There is no limit on the size of the numbers. The values may be as large or as small as is allowed by the **int** type on the particular computer that executes the function. The input data will be stored in range *inArray [0]* to *inArray[size-1].*

**Program Outputs:**

The function will produce seven outputs consisting of the latter six parameters in the *Stats*() argument list plus the *Stats*() \_**Bool** return parameter:

1. *outArray[]* (type **int**) contains the result of an ascending (lowest-to-highest) order sort of the numbers in the input array (*inArray[]).* The array may be sorted using any method chosen by the programmer. The lowest value will be in *outArray[0].* Highest value will be in *outArray [size-1].*
2. *mode* (type \* **int)** is a pointer to the most frequently occurring value in the sorted array (*outArray[]*). If there is no mode, provide the minimum value of the array. If there is more than one mode, the one that occurs most frequently should be returned. If there are two or more modes that occur equally frequently, the lowest one should be returned.
3. *mean* (type \* **float**) is a pointer to the average (mean) of the values of the elements in the sorted array, as follows):

*mean* = / *size*

1. *median* (type \* **float**) is a pointer to the center value of the sorted array as follows:

If *size* is odd, *median* is *outArray* [*size*/2] (the decimal portion of the result of size/2 is truncated)

If *size* is even, *median* is (*outArray*[*size*/2] + *outArray*[*size*/2 - 1]) / 2

1. *variance* (type \* **float**) is a pointer to the variance of the values in the sorted array (see note about computing variance, below)
2. *stdev* (type \* **float**) is a pointer to standard deviation of the values in the sorted array (the standard deviation is the square root of the variance)
3. *Stats()*(type \_**Bool**) returns 1 if *size* >0 and returns 0 if *size* <=0.

**Note about Calculating the Variance**

It turns out that computing the variance is not always easy. The basic formula for the variance of an array X of values is simple:

Var(X) = 2 = (xi-)2

**N**

where N is the size of the array, xi’s are the elements of the array, and is the mean or average value of the array. Note also that is the standard deviation, so if you can calculate the variance it is easy to calculate the standard deviation (and vice versa). However, because of the squares involved in the calculation, this computation can result in overflow or inaccurate calculations if the numbers in the array are large or N is large. For this assignment, you can assume that such problems will not happen and use relatively straightforward calculations. If you are curious, you may want to search the web for methods of calculating the variance in the C language. If you find an interesting way to do it, be sure to provide comments that give a link to the appropriate web site so someone reviewing your program can understand what you did and why.

If you wish to test your program, use no more than 20 values in your arrays, and make sure the numbers are relatively small (between -100 and +100). This will avoid such overflow problems.

Examples

Example 1 -- Valid input (even size, mean = median):

*size* = 6

*inArray[]* = {3, 6, 2, 1, 4, 5}

Example 1 -- Function output:

1. return = 1
2. *outArray* [] = {1, 2, 3, 4, 5, 6}
3. *mode* = 1
4. *mean* = 3.5
5. *median* =3.5
6. *variance =* 2.92
7. standard deviation = 1.71

Example 2 -- Valid input (odd size; mean not equal to median):

*size* = 7

*inArray[]* ={3, 6, 2, 1, 4, 5, 6}

Example 2 -- Function output:

1. return = 1
2. *outArray[]* = {1, 2, 3, 4, 5, 6,6}
3. *mode* = 6
4. *mean* = 3.86
5. *median* =4.00
6. *variance =* 3.27
7. standard deviation = 1.81

Example 3 -- Invalid input (invalid size):

*size* = 0

*inArray* = don’t care

Example 3 -- Function output:

1. return = False or 0
2. *outArray[]* = don’t care
3. *mode* = don’t care
4. *mean* = don’t care
5. *median* = don’t care
6. *variance = don’t care*
7. standard deviation = don’t care

Example 4 – Mode = Mean = Median:

*size* = 5

*inArray* = {6,16,10,8,10}

Example 4 -- Function output:

1. return = 1
2. *outArray[]* = {6,8,10,10,16}
3. *mode* = 10
4. *mean* = 10.0
5. *median* = 10.0
6. *variance =* 11.2
7. standard deviation = 3.35

Example 5 – Two Modes:

*size* = 7

*inArray* = {5, 6, 2, 4, 1, 2, 6}

Example 5 -- Function output:

1. return = 1
2. *outArray[]* = {1, 2, 2, 4, 5, 6, 6}
3. *mode* = 2
4. *mean* = 3.71
5. *median* = 4.0
6. *variance =* 3.63
7. standard deviation = 1.91

1. C Function Non-Functional Requirements

By adhering to the instructions in this section, your function will automatically meet the non-functional requirements, which are:

* 1. Your function shall satisfy **all** the functional requirements as specified in section **A-1** and **only** the requirements specified in **A-1**. Do not include additional inputs or outputs. The goal of this assignment is for you to make flowgraphs and compute cyclomatic complexity, not to make elaborate C programs.
  2. Your function shall satisfy the following:
     1. The function shall **not** include variables that are **struct**, **global**, **static** or **extern**.
     2. Function inputs shall be passed to the function only via the argument list and the output will be provided only by the “return” and via the function’s argument list. (See function prototype, below.)
     3. Input and output array data shall be stored beginning with array index 0.
     4. No testing functionality shall be included. This means no I/O. Your function shall not display, print, or import anything to or from a terminal, file, keyboard, printer or other external device.
  3. Your function shall use only the following constructs of the C language:
     1. The function shall use only the following data types: **\_Bool**, **float** and **int**.
     2. The function shall not include types “**char**”,“**long**”, “**double**”, “**short**” , or “**unsigned**”. You do not need them for this assignment and their use will unnecessarily complicate your function.
  4. The function shall adhere to the following prototype:

**\_Bool** Stats (**int** size, **int** inArray[], **int** outArray[],

**int** \*mode, **float** \*mean, **float \***median, **float** \*variance,

**float** \*stdev);

* 1. To support the readability of your function, the following guidelines are **recommended** (but NOT required) for naming the **internal** elements of your function:
     1. Unless otherwise indicated, use lower case values for non-numeric portions of element names.
     2. Variable names (including array names): Use non-numeric lower case values for the first letter of the variable name. Use upper case non-numeric values to for the first letter of each subsequent word in the variable name. Avoid using underscores. Examples:

arrayOne[];

sum;

* + 1. Function names: Use non-numeric, uppercase values to begin each word in a function name. Do not use underscores to separate the words. Example:

NameThisFunction();

[cover sheet, phase 1 and full assignment]

3

ASSIGNMENT 3

**CYCLOMATIC COMPLEXITY, PHASE 1**

CSE 6329 -- SOFTWARE MEASUREMENT AND QUALITY ENGINEERING

Professor Dennis J. Frailey

**Spring, 2018**

|  |  |
| --- | --- |
| NAME | STUDENT ID NUMBER |
| **<your name goes here>** | **<ID number goes here>** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | **Cyclomatic Complexity Calculation** | | | | |
| **Program** | **Arcs** | | **Nodes** | **C (Number of Separate Flowgraphs)** | **Arcs - Nodes** | **Arcs – Nodes + 2C (Cyclomatic Complexity)** |
| Main |  | |  |  |  |  |
| Function |  | |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Grading Comments (student – do not write inside this box)** | | | | |
| **Phase 1** | | | | |
| **<total goes here>** | **/10** | **Flowgraph (main)** | | **Comments from Grader** |
| **/5** | **Complexity (main)** | |
| **/10** | **Program (function)** | |
| **/10** | **Flowgraph (function)** | |
| **/ 5** | **Complexity (function)** | |
| **Phase 2** | | | | |
| **<total goes here>** | **<name>** | | First Program Reviewed | |
| **/ 5** | | **Errors** |  |
| **/ 5** | | **Cyclomatic Complexity** |
| **/ 10** | | **Flowgraph** |
| **<total goes here>** | **<name>** | | Second Program Reviewed | |
| **/ 5** | | **Errors** |  |
| **/ 5** | | **Cyclomatic Complexity** |
| **/ 10** | | **Flowgraph** |
| **<total goes here>** | **<name>** | | Third Program Reviewed | |
| **/ 5** | | **Errors** |  |
| **/ 5** | | **Cyclomatic Complexity** |
| **/ 10** | | **Flowgraph** |
| **<total>** | **Grand Total** | | |  |

**Place flowgraphs on next page(s)**[cover sheet – phase 2 (one for each program evaluated)]

3

ASSIGNMENT 3

**CYCLOMATIC COMPLEXITY, PHASE 2**

CSE 6329 -- SOFTWARE MEASUREMENT AND QUALITY ENGINEERING

Professor Dennis J. Frailey

**Spring, 2018**

|  |  |
| --- | --- |
| NAME | STUDENT ID NUMBER |
| **<your name goes here>** | **<ID number goes here>** |

**Turn this in separately for each program you evaluate!**

|  |  |
| --- | --- |
| **Name of Program Evaluated** | **Discussion of Any Errors Found and How to Correct Them** |
|  | (expand as needed) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Cyclomatic Complexity Calculation** | | | | |
| **Arcs** | **Nodes** | **C (Number of Separate Flowgraphs)** | **Arcs - Nodes** | **Arcs – Nodes + 2C**  **(Cyclomatic Complexity)** |
|  |  |  |  |  |

**Place flowgraph on next page.**