**SFT221 - WS04**

**Student Name: Tsz Wa Wong**

**Student ID: 152181228**

**Student email: twwong9@myseneca.ca**

**Session : NBB**

**1st bug**

Line 12: return (n <= n) ? n \* factorial(n - 1) : 1;

Correct: return (n>=1)? n \* factorial(n - 1) : 1;

When I attempted to compile the program, the debugger provided by Visual Studio immediately prompted a stack overflow error, indicating a bug in line 12. As this type of bug falls under syntactic errors, it is easy to detect and report at compile time. The original checking condition was incorrect, leading to infinite recursion since the condition (n<=n) was always true. Therefore, I changed the checking condition to (n>=1), ensuring that when n becomes 1, the function will return 1 without making another recursive call.

**2nd bug**

Line 16: return n / n;

Correct:

int i = 0;

while(n > factorial(i))

{

i++;

}

return (i > 0) ? n / i : n;

When I attempted to compile the program, the debugger provided by Visual Studio immediately prompted an error of integer division by zero, indicating a bug in line 16. I am also aware that the passed argument is the factorial of an integer, requiring some calculation to obtain the value of that integer. After obtaining the value, we return the factorial of one less than the integer by performing the division

**3rd bug**

Line 4: #define NUM\_FACTS 100

Correct: #define NUM\_FACTS 13

When I compiled the program, the value of the result for each factorial sometimes appeared positive and sometimes negative. I identified this inconsistency as stemming from integer overflow. Consequently, I adjusted the value of NUM\_FACTS to a smaller size. Since 12! is the maximum value an integer can hold, passing 13! to the function 'reduceFactorial' would result in an integer overflow. Therefore, we can only output 11! when calling the 'reduceFactorial' function. To display 11!, we only need to iterate 13 times from 0 to 12, with 11! being shown alongside the number 12. Hence, I changed the value of NUM\_FACTS to 13.

**4th bug**

Line 36: printf("%5d %12f\n", i, results.results[i]);

Correct: printf("%5d %12d\n", i, results.results[i]);

While compiling the program, I observed a trailing decimal point. Upon closer inspection, I identified a bug associated with the printf statement. The initial format specifier %f is intended for printing floating-point numbers, but this is incorrect in this context. Since results.results[i] is an integer, the correct specifier to use is %d.

**5th bug**

Line 18: void computeFactorials(struct FactorialResults results, int numFactorials)

Line 23: results.results[i] = factorial(i);

Line 25: results.numResults = numFactorials;

Line 31: computeFactorials(results, NUM\_FACTS);

Correct: void computeFactorials(struct FactorialResults\* results, int numFactorials)

Correct: results->results[i] = factorial(i);

Correct: results->numResults = numFactorials;

Correct: computeFactorials(&results, NUM\_FACTS);

As I observed the value of results.results[i] consistently being 0, I realized there must be a bug related to passing by value. The original code passes the results structure by value, meaning a copy of the structure is passed to the function. Consequently, the function works with a local copy of the structure, and any modifications made inside the function do not affect the original structure. However, this is not the behavior we desire, as we want these functions to modify our results. Therefore, I modified the computeFactorials function to take a pointer to the struct FactorialResults

instead of the structure itself. When calling the function, the address of results is passed using the & operator.