SFT 221

Workshop 4

Name:	Gyeongrok oh
ID:	119140226
Email:	goh3@myseneca.ca
Section:	NBB

Authenticity Declaration:
I declare this submission is the result of my own work and has not been shared with any other student or 3rd party content provider. This submitted piece of work is entirely of my own creation.

Updated code is below

```
#include <stdio.h>

#define MAX_FACTORIALS 10000
#define NUM_FACTS 100

struct FactorialResults {
   int results[MAX_FACTORIALS];
   int numResults;
};

int factorial(const int n) {
   return (n > 1) ? n * factorial(n - 1) : 1;
//    //prenious code: return (n <= n) ? n * factorial(n - 1) : 1;
//    //The previous code, unlike this code, compares 'n' with 'n' and checks if it is less than or
equal to 'n'.
//    //However, this code gets stuck in an infinite loop because it never encounters a false
condition.In other words,
//    //it continuously outputs true, and 'n' keeps decreasing endlessly by 1
}

int reduceFactorial(const int n) {

   return n;
// The code you provided performs division of n by n.
// However, this is integer division, and it will result in an error if n is 0.
// Therefore, this operation always leads to an exceptional situation.
// To obtain the correct result, a different approach should be used.

}
```

What was the most useful technique you used to find the bugs? Why was it more useful tha nother techniques you tried?

Unexpected behaviors that may lead to bugs. In addition, logging plays a vital role in the debugging process. By strategically inserting log statements, developers can gain valuable insights into the program's internal state and pinpoint the exact location of an issue. Moreover, unit testing provides an organized approach to verify the functionality of specific code units, aiding in detecting bugs within isolated sections. The collaborative effort of code review also helps catch potential bugs before they manifest in production. With its comprehensive approach and the ability to delve deep into the code, debugging stands out as an essential technique in the quest for bug-free software."

Look up answers to the following questions and report your findings:
a. What are the largest integer and double values you can store?
b. Why is there a limit on the maximum value you can store in a variable?
c. If you exceed the maximum value an integer can hold, what happens? Explain why the format causes this to happen.
d. What is the format for the storage of a floating point variable? How does this differ from the way an integer is stored?

a. an enormous integer value counts on the data type used. For a signed 32-bit integer, it's 2,147,483,647. For a double, it's about 1.79769 × 10^308.

b. There is a limitation on the maximum value to provide predictable conduct and prevent memory overflow.

c. Surpassing the maximum value of integer results in wraparound behavior, where the value wraps around to the minimum value Because of. The finite number of bits used for storage.

d. Floating-point variables use the IEEE 754 standard, including a sign bit, exponent, and significand. Integers are stored as whole numbers without a separate exponent or fractional part.

What is the default amount of stack memory that is given to a program when Visual Studio starts a C or C++ program? What is the default heap size? Did you hit any of the limits? If so, which one(s)? If you hit a limit, would increasing the amount of memory allocated to the program fix the problem? Justify your answer. Why do they limit the stack and heap size for a program?

The default stack memory and heap size in Visual Studio for C or C++ programs are not fixed. They can vary depending on the operating system, compiler settings, and program configuration. Exceeding these limits can lead to stack overflow or heap exhaustion, causing program crashes or unexpected behavior. While increasing the memory allocation may alleviate the issue, it is subject to system constraints. Stack and heap size limits are imposed to ensure efficient memory management, prevent resource abuse, and maintain system stability by allocating memory resources effectively. These limitations help strike a balance between program requirements and overall system performance.