SFT 221

Workshop 4

Name:	patel vrundaben vijaykumar
ID:	158605220
Email:	vvpatel20@myseneca.ca
Section:	NGG

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

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#define MAX_FACTORIALS 10000
#define NUM_FACTS 10
struct FactorialResults
{
int results[MAX_FACTORIALS];
int numResults;
};
int factorial(const int n)
{
return (n > 1) ? n * factorial(n - 1) : 1;
}
int reduceFactorial(const int n)
{
return (n > 1) ? n : 1;
}
void computeFactorials(struct FactorialResults  *results, int numFactorials)
{
if (numFactorials > MAX_FACTORIALS) {
numFactorials = MAX_FACTORIALS;
}

int i;
for (i = 0; i < numFactorials; i++)
{
results->results[i] = factorial(i);
}
results->numResults = numFactorials;
}
int main(void)
{
struct FactorialResults results = { {0}, 0 };
int i;
computeFactorials(&results, NUM_FACTS);
for (i = 0; i < results.numResults; i++)
{
results.results[i] = reduceFactorial(results.results[i]);
printf("%5d %12d\n", i, results.results[i]);
}

return 0;
}
```

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

The most helpful way to find and fix the problems in the code was to read it carefully and go through it step by step. This worked better than other methods for several reasons. Firstly, reading the code carefully allowed me to understand how it was supposed to work and spot where it was not doing what it was supposed to do. I could see if the logic was wrong or if it was using the wrong values. Secondly, going step by step through the code helped me catch problems early on. I could see if a function was called with the wrong arguments or if variables were used incorrectly. This helped prevent bigger issues down the line. Thirdly, understanding what the code was supposed to do was crucial. By comparing it to the desired output, I could see where it was going wrong. Other techniques like automated tools might find some errors, but they can't understand the code's purpose like a human can.

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. The largest integer value you can store depends on the data type used. For a 32-bit signed integer (common on many systems), the maximum value is 2,147,483,647. The largest double value depends on the system but is approximately 1.8 x 10^308.
b. There's a limit on the biggest number we can save because our computer's memory is not infinite. It can only hold a certain amount of data, so numbers have to fit within that limit.
c. If we exceed the maximum value an integer can hold, it causes an overflow. This happens because the binary representation of the number cannot fit within the allocated memory for the variable. Overflow can result in unpredictable and incorrect results.
d. Floating-point variables are stored using a format that includes a sign, an exponent, and a fraction part. This allows them to represent numbers with decimals. while , integers are stored as whole numbers without a fractional part, using a fixed number of bits. So, floating-point numbers can handle a wider range of values with decimals, while integers store whole numbers more directly.

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?

In Visual Studio, a typical C or C++ program starts with around 1MB of stack memory by default. The heap size, on the other hand, depends on how much memory our computer has. If our program runs into stack or heap size limits, giving it more memory might help sometimes, but it won't always solve the problem. They put these limits on stack and heap sizes to make sure programs don't use up too much memory. If the stack (used for functions and local stuff) gets too big, it can quickly run out of memory. If the heap (used for flexible memory needs) gets too big, it can waste memory and even make our computer unstable. So, these limits try to keep a good balance between memory and keeping our program working well.