

## Final Examination Programming Project: Exception

Topics: Exceptions, Pre- and Post-conditions

CPSC-298-6 Programming in C++

jbonang@chapman.edu

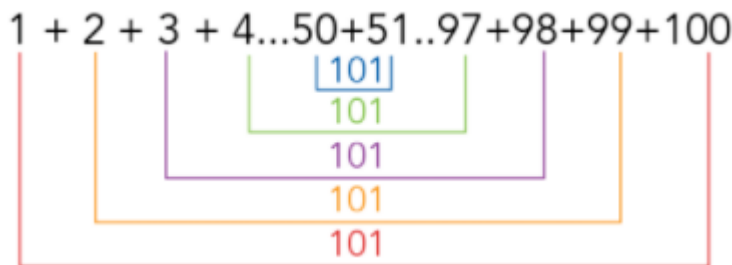
## Introduction

According to legend, a very young Carl Friedrich Gauss was asked by his grade school teacher to sum the numbers from 1 to 100, inclusive. In other words,  $1 + 2 + 3 + 4 + \dots + 100$ , or, more succinctly:

$$S = \sum_{i=1}^{100} i$$

The teacher assumed this would take the class a while; however, Gauss produced the answer, 5050, almost instantly. How did he do it?

He realized that you could add the pairs of numbers  $100 + 1$ ,  $99 + 2$ ,  $98 + 3$ , and so on up to  $51 + 50$ . Each pair adds up to 101 and there are 50 pairs so the total is  $101 \times 50 = 5050$ .



This can be generalized to:

$$(\text{number of pairs}) \times (\text{"last number in the sum"} + 1) = 50 (100 + 1) = 5050$$

For succinctness, we'll call the "last number in the sum"  $n$ .

The number of pairs, 50, is equal to  $n/2$ , so our formula becomes  $(n/2)(n+1)$ .

$$\frac{n(n+1)}{2} = \sum_{i=1}^n i$$

It works if  $n$  is an odd number, too.

Suppose we choose  $n$  as 67. The sum of  $1 + 2 + 3 + \dots + 67 = 2278$ .

If we substitute  $n = 67$  in to our formula, we have.

$$67(67 + 1)/2 = 67(68)/2 = 67(34) = 2278$$

What about  $n = 0$ . Nope - not a valid value for  $n$ . We're summing the values from 1 to  $n$  and constrain the values of  $n$  to  $n \geq 1$ . That means that negative integer values are also not allowed.

## ***The Assignment***

For this Final Examination programming project, you'll write a function called `sigma` that accepts  $n$ , an unsigned integer, as a parameter and computes the sum of the numbers from 1 to  $n$ . The function prototype for `sigma` is shown in the code listing below.

```
// Sum the integers from 1 to n. For example, if n is 5, then
// sigma returns the sum 1 + 2 + 3 + 4 + 5 == 15.
// @param n integer upper bound of summation
// @pre n >= 1
// @post return value is the sum of the integers from 1 to n.
// @return sum of the integers from 1 to n, where n is
//         specified as an argument or 0 if an error occurred.
unsigned long sigma(unsigned long n)
```

By declaring the argument  $n$  to be an unsigned long, the negative integers are excluded, but zero isn't. It's possible a user might not read the function preamble comments and pass a zero in as the actual argument to parameter  $n$ ; that's something you'll need to handle.

You'll compute the sum the hard way, by writing a for loop and summing the numbers from 1 to  $n$ .

Function `sigma` will be similar to the outline provided in the code listing below.

```

// Sum the integers from 1 to n. For example, if n is 5, then
// sigma returns the sum 1 + 2 + 3 + 4 + 5 == 15.
// @param n unsigned integer upper bound of summation
// @pre n >= 1
// @post return value is the sum of the integers from 1 to n.
// @return sum of the integers from 1 to n, where n is
//         specified as an argument or 0 if an error occurred.
unsigned long sigma(unsigned long n)
{
    unsigned long sum = 0;

    try
    {
        // Check preconditions
        //     Raise a std::runtime_error exception if the precondition (n >= 1) is not met.

        // Function body
        //     Implement a for loop to compute sum of integers from 1 to n

        // Check postconditions
        //     Raise a std::runtime_error exception if the postcondition (sum == (n(n+1)/2))
        //     is not met.
    }
    catch (std::runtime_error & ex)
    {
        std::cout << "Exception: " << ex.what() << std::endl;
        std::cout << "Cannot compute sum; returning 0" << std::endl;
        sum = 0; // 0 is returned to indicate an error occurred.
    }

    return sum;
}

```

You'll check that the input argument is valid ( $n \geq 1$ ) and report an error if it isn't by raising a `std::runtime_exception` exception. The exception should be raised in an if block. The conditional expression of the if block should check if the precondition ( $n \geq 1$ ) is **not** true.

When raising the `std::runtime_exception` in the case of a precondition violation, pass the following message string to the `std::runtime_exception` constructor as an argument:

```

std::string strMessage = "Precondition n>=1 violated; invalid value for argument n: " + std::to_string(n) +
    " (Loc: " + __FILE__ + ", " + std::to_string(__LINE__) + ")";

```

The string uses the `__FILE__` and `__LINE__` macros to report the name of the file and the line number within the file where the exception was raised. It will also print out the erroneous value of `n`.

Use a for statement to compute the sum of the integers from 1 to n, inclusive. Remember that the condition expression of the for loop should use the less-than-or-equal-to operator:

```
i <= n
```

For the post-condition, you'll check that the sum computed in your for loop is identical to the sum expected from the formula  $n(n+1)/2$ .

If the post-condition is not satisfied, raise a `std::runtime_exception` exception, as you did for the precondition. Pass the following string as the argument to the `std::runtime_exception` constructor:

```
std::string strMessage = "Postcondition sum == (n(n+1))/2 violated: sum: " + std::to_string(sum) + "; n(n+1)/2: " + std::to_string((n * (n + 1)) / 2) +  
    " (Loc: " + __FILE__ + ", " + std::to_string(__LINE__) + ")";;
```

Remember to include the following header files:

```
#include <iostream>  
#include <stdexcept>  
#include <string>
```

## Running the Program

In the main program, call the sigma function twice, once with the actual argument 5 (which is valid) and once with actual argument 0, which is invalid and should trigger the precondition exception.

```
int main()  
{  
    int sum = sigma(5);  
    std::cout << "sigma(5) = " << sum << std::endl;  
    sum = sigma(0);  
    std::cout << "sigma(0) = " << sum << std::endl;  
  
    return 0;  
}
```

The output of the program will appear similar to that shown in the figure below.

```
sigma(5) = 15  
Exception: Precondition n>=1 violated; invalid value for argument n: 0 (Loc: C:\Users\Jim\source\repos\PrePostConditions\PrePostConditions\PrePostConditions.cpp, 59)  
Cannot compute sum; returning 0  
sigma(0) = 0
```

Note that the file and line number where the exception was thrown is printed out when the precondition is violated.

## ***Injecting a Fault and Running the Program Again***

Modify your program so that the for loop does not correctly compute the sum. For example, change the for loop condition from  $i \leq n$  to  $i < n$ , a common error.

Build the program again and run it. The post-condition is not satisfied in this case and an exception should be raised.

The output of the program should appear similar to that shown in the figure below.

```
Exception: Postcondition  $\text{sum} == (n(n+1))/2$  violated: sum: 10;  $n(n+1)/2$ : 15 (Loc: C:\Users\Jim\source\repos\PrePostConditions\PrePostConditions\PrePostConditions.cpp, 72)
Cannot compute sum; returning 0
sigma(5) = 0
Exception: Precondition  $n \geq 1$  violated; invalid value for argument n: 0 (Loc: C:\Users\Jim\source\repos\PrePostConditions\PrePostConditions\PrePostConditions.cpp, 59)
Cannot compute sum; returning 0
sigma(0) = 0
```

The post-condition exception should be raised.

Restore the for loop condition to the correct expression,  $i \leq n$ , and submit the assignment.