C++ Chapter 4: Loops

Contents

Time required: 120 minutes

Online Tutorials

- C++ While Loop
- C++ Do/While Loop
- C++ For Loop
- C++ Break/Continue

Increment and Decrement Operators

For loops commonly use increment and decrement operators to help count through a loop.

increment operator: The increment operator (++) increases the value of a numeric variable by one.

decrement operator: The decrement operator (--) decreases the value of a numeric variable by one.

Pre increment

```
x = 1
// ++ happens first, then x value is assigned to y
y = ++x
y = 2
x = 2
```

Post increment is the most used operator in a for loop. It is used when we want the values to count upward by 1 at a time.

```
X = 1
// x value is assigned to y, then increment x++
y = x++
y = 1
x = 2
```

Pre decrement

```
x = 1
// -- happens first, then x value is assigned to y
y = --x
y = 0
x = 0
```

Post decrement is the most used operator in a for loop. It is used when we want the values to count down.

```
x = 1
// x value is assigned to y, then decrement --
y = x--
y = 1
x = 0
```

For Loop

In C++, a **for** loop is a control flow statement that allows you to repeatedly execute a block of code for a specified number of times. It's often used when you know in advance how many iterations you want to perform. The basic structure of a for loop consists of three parts:

1. **Initialization**: This part is executed only once at the beginning of the loop. It's typically used to set up a loop control variable or initialize any necessary values.
2. **Condition**: This part is a Boolean expression that is evaluated before each iteration of the loop. If the condition is true, the loop continues to execute; if it's false, the loop terminates.
3. **Iteration**: This part is usually responsible for updating the loop control variable. It's executed at the end of each loop iteration, just before reevaluating the condition.

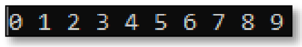
The syntax of a for loop.

```
for (Initialization; Condition; Iteration) {
    // Code to be executed in each iteration
}
```

For loop example.

```
#include <iostream>
int main()
{
    for (int i{0}; i < 10; i++){
        // The loop will execute as long as i is less than 10.
        std::cout << i << " ";
    } // end for loop
    return 0;
}
```

Example run:



```
0 1 2 3 4 5 6 7 8 9
```

This loop will print numbers from 0 to 9 because

1. **Initialization** sets **i** to 0
2. **Condition** checks if **i** is less than 10
3. **Iteration** updates **i** by incrementing it by 1 (**++i**) in each iteration.

While Loop

A while loop is typically used for a loop where you aren’t sure how many times you wish to loop.

Syntax of a while loop.

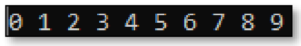
```
initialize counter
while(condition) {
    // Statements
    update counter
    // Statements
}
```

condition is a Boolean expression that is evaluated before each iteration of the loop. If the condition is true, the code inside the loop is executed; otherwise, the loop terminates.

A while loop is a pretest loop.

```
#include <iostream>
int main()
{
    // Counter variable
    int count{0};
    // While the condition is true, continue the loop
    while (count < 10)
    {
        std::cout << count << " ";
        // Increment counter variable
        count++;
    } // end while
    return 0;
}
```

Example run:



```
0 1 2 3 4 5 6 7 8 9
```

Do While

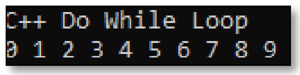
A do-while loop executes a block of code at least once and continues to do so as long as a given condition remains true.

```
initialize counter
do{
    // Statements
    update counter
    // Statements
}while(condition)
```

The do while loop is a posttest loop. It will run at least once before it tests the condition.

```
#include <iostream>
int main()
{
    int count{0};
    std::cout << "C++ Do While Loop" << std::endl;
    // posttest loop, runs at least once
    do
    {
        std::cout << count << " ";
        // Increment counter
        count++;
        // Continue loop while condition is true
    } while (count < 10);
    return 0;
} // end main
```

Example run:



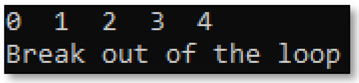
```
C++ Do While Loop
0 1 2 3 4 5 6 7 8 9
```

Break

Break causes the code to exit from the loop early. This is useful in a menu system.

```
#include <iostream>
int main()
{
    int count = 0;
    while (count < 10)
    {
        if (count == 5)
        {
            count++;
            std::cout << "\nBreak out of the loop" << std::endl;
            // break exits the loop
            break;
        }
        std::cout << count << "  ";
        count++;
    } // end while
} // end main
```

Example run:

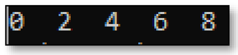


```
0  1  2  3  4
Break out of the loop
```

Continue

Continue causes the code to go directly to the beginning of the loop.

```
#include <iostream>
int main()
{
    int count = 0;
    while (count < 10)
    {
        // If % != 0, the number is even
        if (count % 2 != 0)
        {
            count++;
            // continue causes the current iteration to
            // go immediately to the beginning of the loop
            continue;
        }
        std::cout << count << "  ";
        count++;
    } // end while
} // end main
```



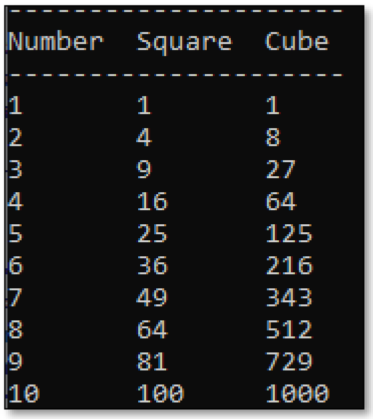
```
0  2  4  6  8
```

Tutorial 4.1: For Loop Squares and Cubes

Create a C++ program named: **squares_and_cubes.cpp**

```cpp
/*
 * Name: squares_and_cubes.cpp
 * Written by:
 * Written on:
 * Purpose: Use a for loop to print the squares
 * and cubes of the numbers 1 to 10
 */

#include <iostream>
int main()
{
    // Constant for dashes printed
    std::string DASHES = "---------------------";
    int square = 0;
    int cube = 0;

    // # Print the heading
    std::cout << DASHES << std::endl;
    std::cout << "Number\tSquare\tCube" << std::endl;
    std::cout << DASHES << std::endl;

    // A for loop from 1 - 10
    for (int i{1}; i < 11; i++)
    {
        // Calculate the square of the current number
        square = i * i;
        cube = i * i * i;
        std::cout << i << "\t"
                  << square << "\t" << cube << std::endl;
    }
    std::cout << DASHES << std::endl;
    return 0;
}
```

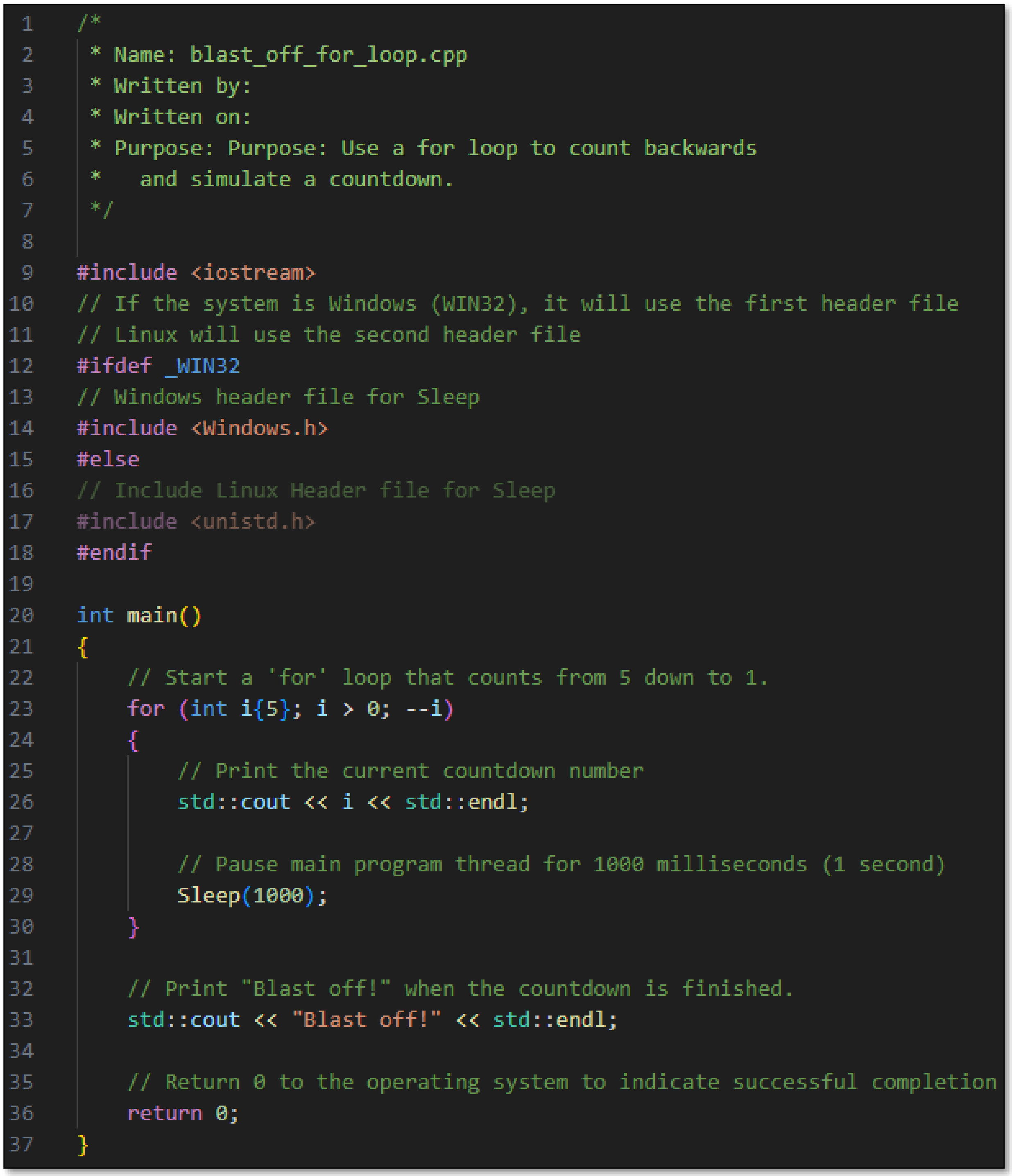
Example run:



```
---------------------
Number  Square  Cube
---------------------
1       1       1
2       4       8
3       9       27
4       16      64
5       25      125
6       36      216
7       49      343
8       64      512
9       81      729
10      100     1000
```

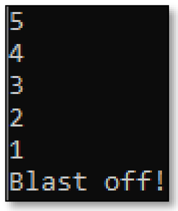
Tutorial 4.2: For Loop Blast Off!

Here is an example program that counts down from 5 and then prints a message. There are two header files needed for Sleep. One for Windows and one for Linux.



```cpp
/*
 * Name: blast_off_for_loop.cpp
 * Written by:
 * Written on:
 * Purpose: Purpose: Use a for loop to count backwards
 *    and simulate a countdown.
 */

#include <iostream>
// If the system is Windows (WIN32), it will use the first header file
// Linux will use the second header file
#ifdef _WIN32
// Windows header file for Sleep
#include <Windows.h>
#else
// Include Linux Header file for Sleep
#include <unistd.h>
#endif

int main()
{
    // Start a 'for' loop that counts from 5 down to 1.
    for (int i{5}; i > 0; --i)
    {
        // Print the current countdown number
        std::cout << i << std::endl;

        // Pause main program thread for 1000 milliseconds (1 second)
        Sleep(1000);
    }

    // Print "Blast off!" when the countdown is finished.
    std::cout << "Blast off!" << std::endl;

    // Return 0 to the operating system to indicate successful completion
    return 0;
}
```

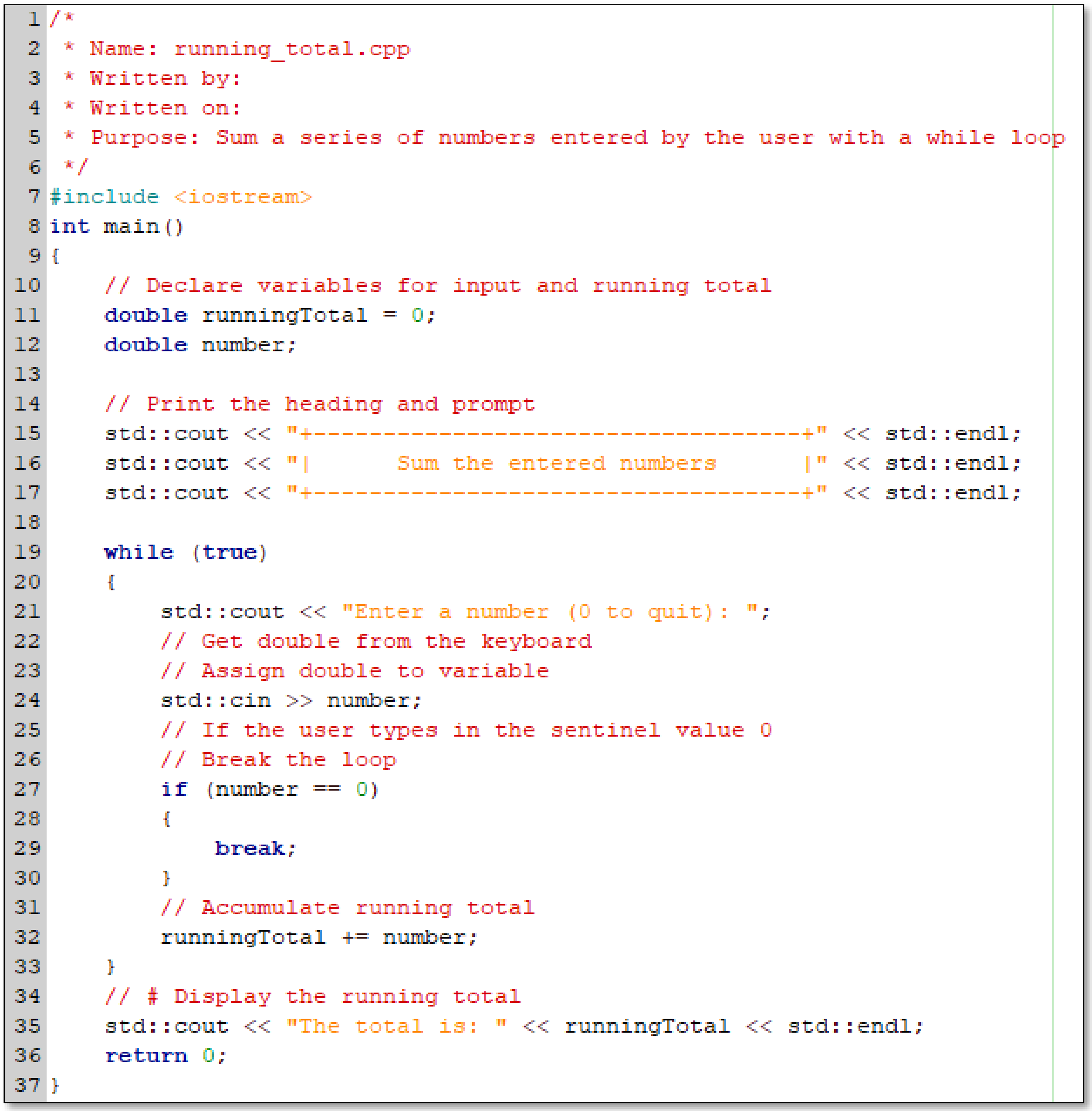
Example program run:


```
5
4
3
2
1
Blast off!
```

Tutorial 4.3: Running Total While Loop

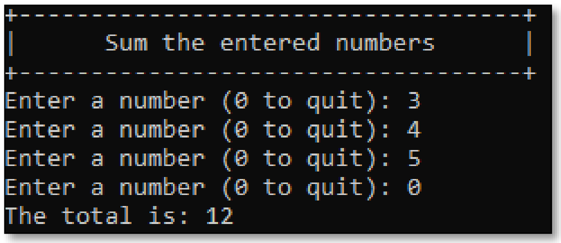
The following program demonstrates how to keep a running total of numbers entered by the user.

Create a C++ program named: **running_total.cpp**



```cpp
/*
 * Name: running_total.cpp
 * Written by:
 * Written on:
 * Purpose: Sum a series of numbers entered by the user with a while loop
 */
#include <iostream>
int main()
{
    // Declare variables for input and running total
    double runningTotal = 0;
    double number;

    // Print the heading and prompt
    std::cout << "+-----------------------------------+" << std::endl;
    std::cout << "|      Sum the entered numbers      |" << std::endl;
    std::cout << "+-----------------------------------+" << std::endl;

    while (true)
    {
        std::cout << "Enter a number (0 to quit): ";
        // Get double from the keyboard
        // Assign double to variable
        std::cin >> number;
        // If the user types in the sentinel value 0
        // Break the loop
        if (number == 0)
        {
            break;
        }
        // Accumulate running total
        runningTotal += number;
    }
    // # Display the running total
    std::cout << "The total is: " << runningTotal << std::endl;
    return 0;
}
```

Example run:

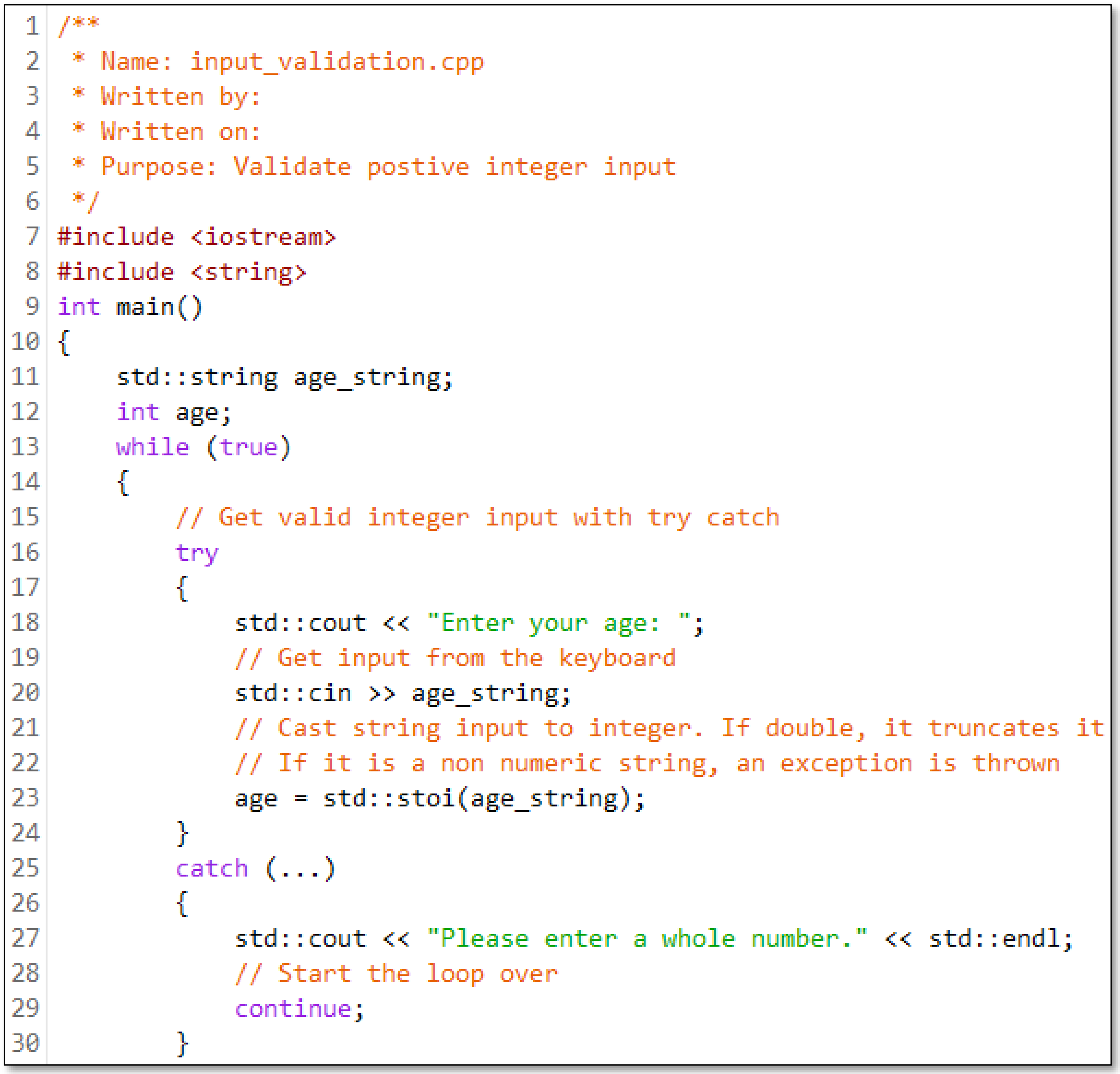


```
+-----------------------------------+
|      Sum the entered numbers      |
+-----------------------------------+
Enter a number (0 to quit): 3
Enter a number (0 to quit): 4
Enter a number (0 to quit): 5
Enter a number (0 to quit): 0
The total is: 12
```

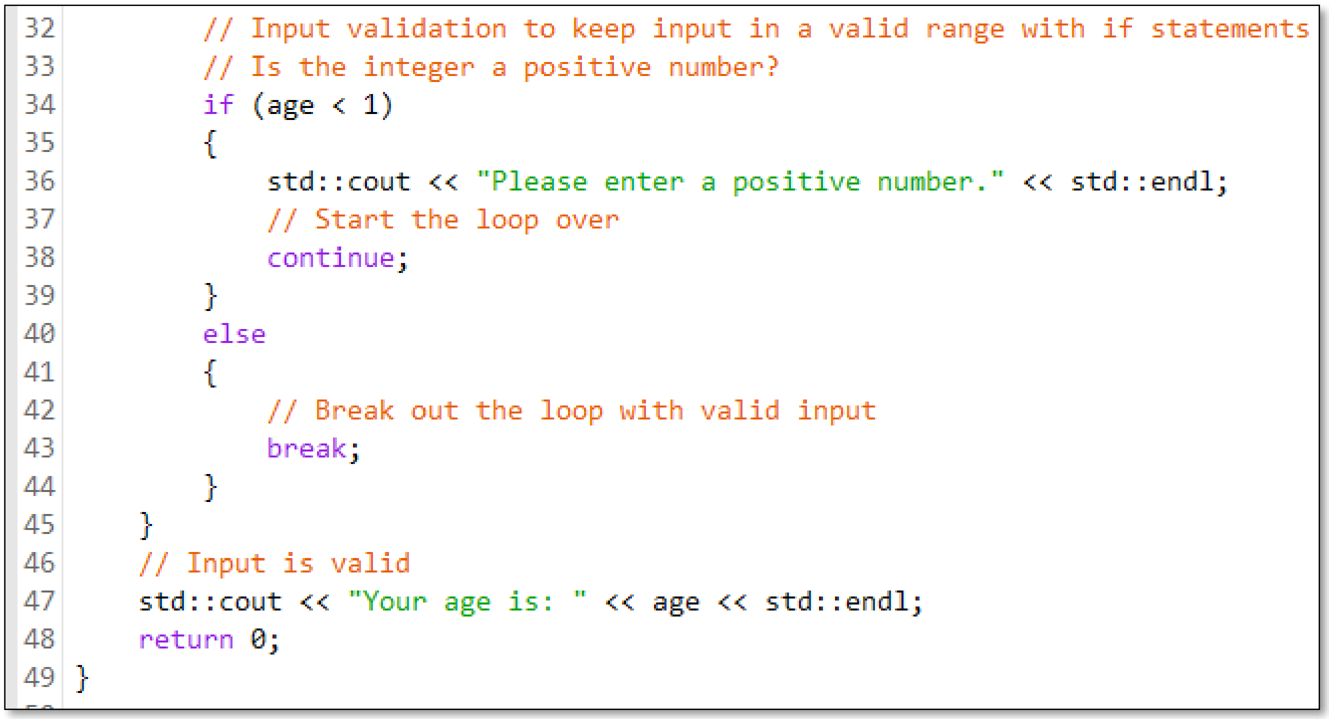
Tutorial 4.4: Input Validation and Try Catch

Part of the C++ string library are string parsing functions.

```
#include <string>
std::stoi(string)   // String to int
std::stof(string)   // String to float
std::stod(string)   // String to double
```

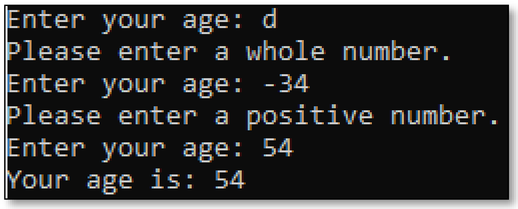


```cpp
/**
 * Name: input_validation.cpp
 * Written by:
 * Written on:
 * Purpose: Validate postive integer input
 */
#include <iostream>
#include <string>
int main()
{
    std::string age_string;
    int age;
    while (true)
    {
        // Get valid integer input with try catch
        try
        {
            std::cout << "Enter your age: ";
            // Get input from the keyboard
            std::cin >> age_string;
            // Cast string input to integer. If double, it truncates it
            // If it is a non numeric string, an exception is thrown
            age = std::stoi(age_string);
        }
        catch (...)
        {
            std::cout << "Please enter a whole number." << std::endl;
            // Start the loop over
            continue;
        }
```



```cpp
        // Input validation to keep input in a valid range with if statements
        // Is the integer a positive number?
        if (age < 1)
        {
            std::cout << "Please enter a positive number." << std::endl;
            // Start the loop over
            continue;
        }
        else
        {
            // Break out the loop with valid input
            break;
        }
    }
    // Input is valid
    std::cout << "Your age is: " << age << std::endl;
    return 0;
}
```

Example run:



```
Enter your age: d
Please enter a whole number.
Enter your age: -34
Please enter a positive number.
Enter your age: 54
Your age is: 54
```

When you run this code, you'll be prompted for your age until you enter valid data. This ensures that when the execution leaves the while loop, the age variable will contain a valid value that won't crash the program.

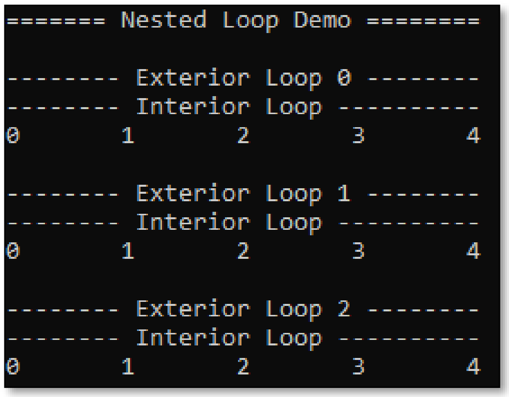
Tutorial 4.5: Nested Loops

You can place a loop inside a loop.

Create a C++ program named **nested_loops.cpp**

```cpp
/*
 * Name: nested_loops.cpp
 * Written by:
 * Written on:
 * Purpose: Use a nested loop
 */

#include <iostream>
int main()
{
   std::cout << "======= Nested Loop Demo ========" << std::endl;

   // Exterior loop
   for (int i{0}; i < 3; i++)
   {
      std::cout << "\n-------- Exterior Loop "
               << i << " --------" << std::endl;
      std::cout << "-------- Interior Loop ----------" << std::endl;

      // Interior loop
      for (int j{0}; j < 5; j++)
      {
         std::cout << j << "\t";
      }
      std::cout << "\n";
   }
   return 0;
}
```

Example run:



```
======= Nested Loop Demo ========

-------- Exterior Loop 0 --------
-------- Interior Loop ----------
0       1       2       3       4

-------- Exterior Loop 1 --------
-------- Interior Loop ----------
0       1       2       3       4

-------- Exterior Loop 2 --------
-------- Interior Loop ----------
0       1       2       3       4
```

Tutorial: C++ Random Number

This example program demonstrates how to generate random numbers in C++.

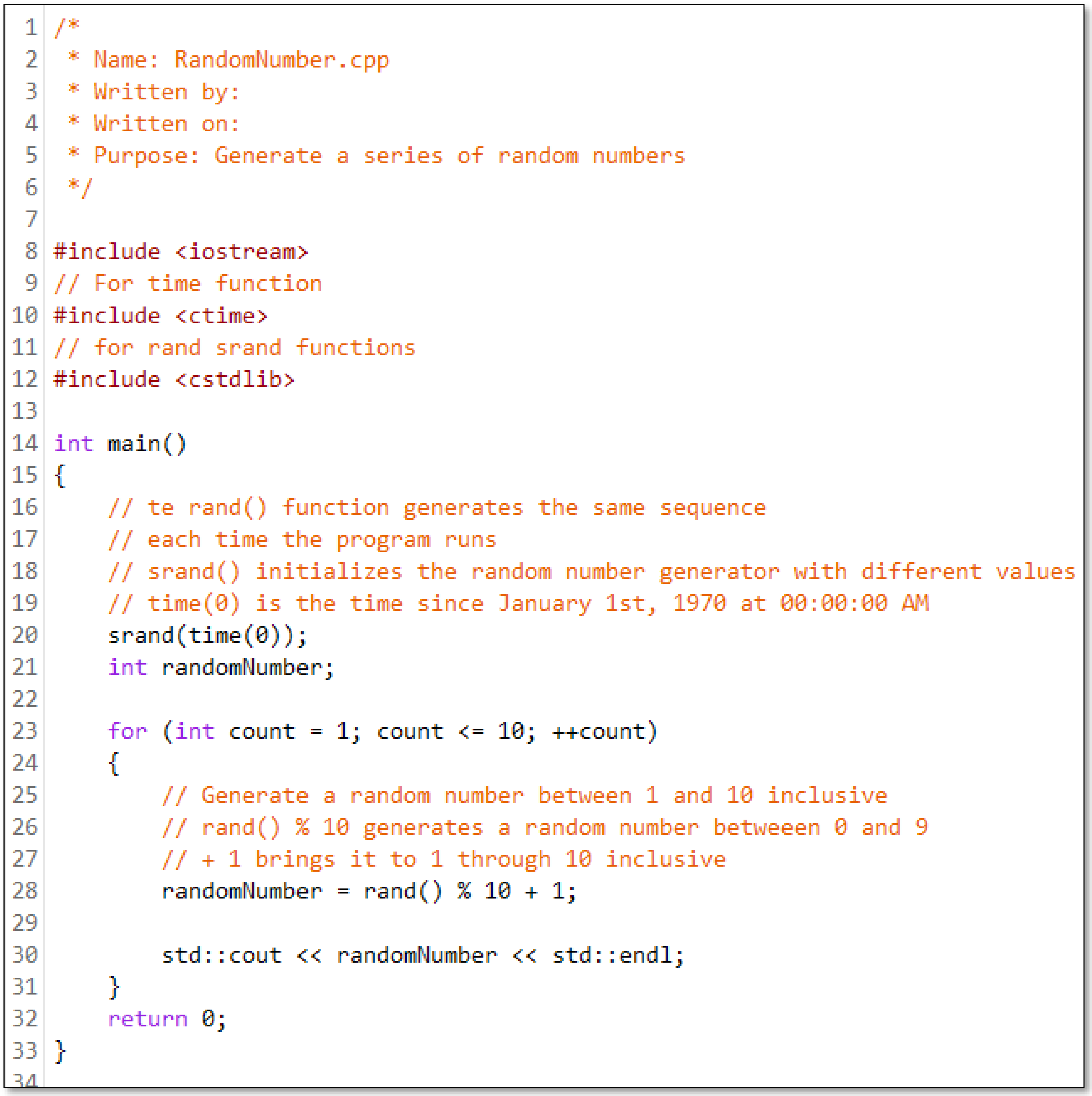
In C++, rand() is a function that generates a random integer between 0 and RAND_MAX (a large constant defined in <cstdlib>). However, to get a random number within a specific range, we use the modulus operator %.

Here's how rand() % 10 works:

1. rand() generates a random integer (e.g., 37429).
2. % 10 takes the remainder when dividing that number by 10.
3. This results in a value between 0 and 9.

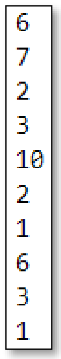
rand() % 10 will return a random number in the range [0, 9].

Create a C++ program named **RandomNumbers.cpp**



```cpp
/*
 * Name: RandomNumber.cpp
 * Written by:
 * Written on:
 * Purpose: Generate a series of random numbers
 */

#include <iostream>
// For time function
#include <ctime>
// for rand srand functions
#include <cstdlib>

int main()
{
    // te rand() function generates the same sequence
    // each time the program runs
    // srand() initializes the random number generator with different values
    // time(0) is the time since January 1st, 1970 at 00:00:00 AM
    srand(time(0));
    int randomNumber;

    for (int count = 1; count <= 10; ++count)
    {
        // Generate a random number between 1 and 10 inclusive
        // rand() % 10 generates a random number betweeen 0 and 9
        // + 1 brings it to 1 through 10 inclusive
        randomNumber = rand() % 10 + 1;

        std::cout << randomNumber << std::endl;
    }
    return 0;
}
```

Example run:



```
6
7
2
3
10
2
1
6
3
1
```

Assignment 1: C++ Guessing Game and the Temple of Doom

You guessed it, the world-famous Guessing Game program. The first game on your way to being a game software developer.

- Comment each line of code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

Pseudocode

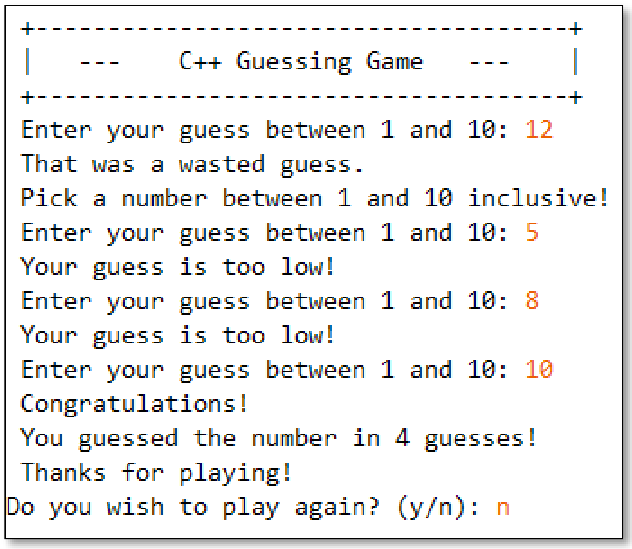
1. Write pseudocode or TODO for the exercise
2. Submit with the assignment

Minimum Requirements

Create the classic guessing game program.

- Print a nice title
- Generate a random number between 1 and 10
- Get the user guess
- Determine if the user guess is higher, lower, or equal to the random number
- Track number of tries per game
- Allow the user to play again or quit

Example run:



```
+-------------------------------------+
|   ---    C++ Guessing Game   ---    |
+-------------------------------------+
Enter your guess between 1 and 10: 12
That was a wasted guess.
Pick a number between 1 and 10 inclusive!
Enter your guess between 1 and 10: 5
Your guess is too low!
Enter your guess between 1 and 10: 8
Your guess is too low!
Enter your guess between 1 and 10: 10
Congratulations!
You guessed the number in 4 guesses!
Thanks for playing!
Do you wish to play again? (y/n): n
```

Assignment Submission

1. Attach the pseudocode.
2. Attach the program files.
3. Attach screenshots showing the successful operation of the program.
4. Submit in Blackboard.