Tishk International University
Cybersecurity Department
Course Code: CBS 113

Programming Fundamentals

Lecture 5

Loops

Fall 2024

Hemin Ibrahim, PhD
hemin.ibrahim@tiu.edu.iq

Outline

- Increment and Decrement
- Intro to loops
- The while loop
- The for loop
- Break statement

Objectives

By the end of this lesson, you will be able to

- Apply increment/decrement operations and loop structures through hands-on coding exercises.
- Write efficient and optimized code using the most suitable loop construct and judicious use of increment/decrement operations.
- Enhance problem-solving skills by tackling diverse challenges with loops and control flow mechanisms.

The Increment and Decrement Operators

"**++**" and "**--**" are operators that add and subtract 1 from their operands. To increment a value means to increase it by one, and to decrement a value means to decrease it by one.

These are two ways to increment the variable num

num = num + 1;

num += 1;

And num is decremented in both of the following statements:

```
num = num - 1;
num -= 1;
```

The Increment and Decrement Operators (cont.)

C++ has operators dedicated to increasing (++) and decreasing (--) variables.

The following statement uses the ++ operator to increment num:

```
num++;    // Increment by One
num--;    // Decrement by One
```

Postfix and Prefix Modes

Postfix Mode: (x++)

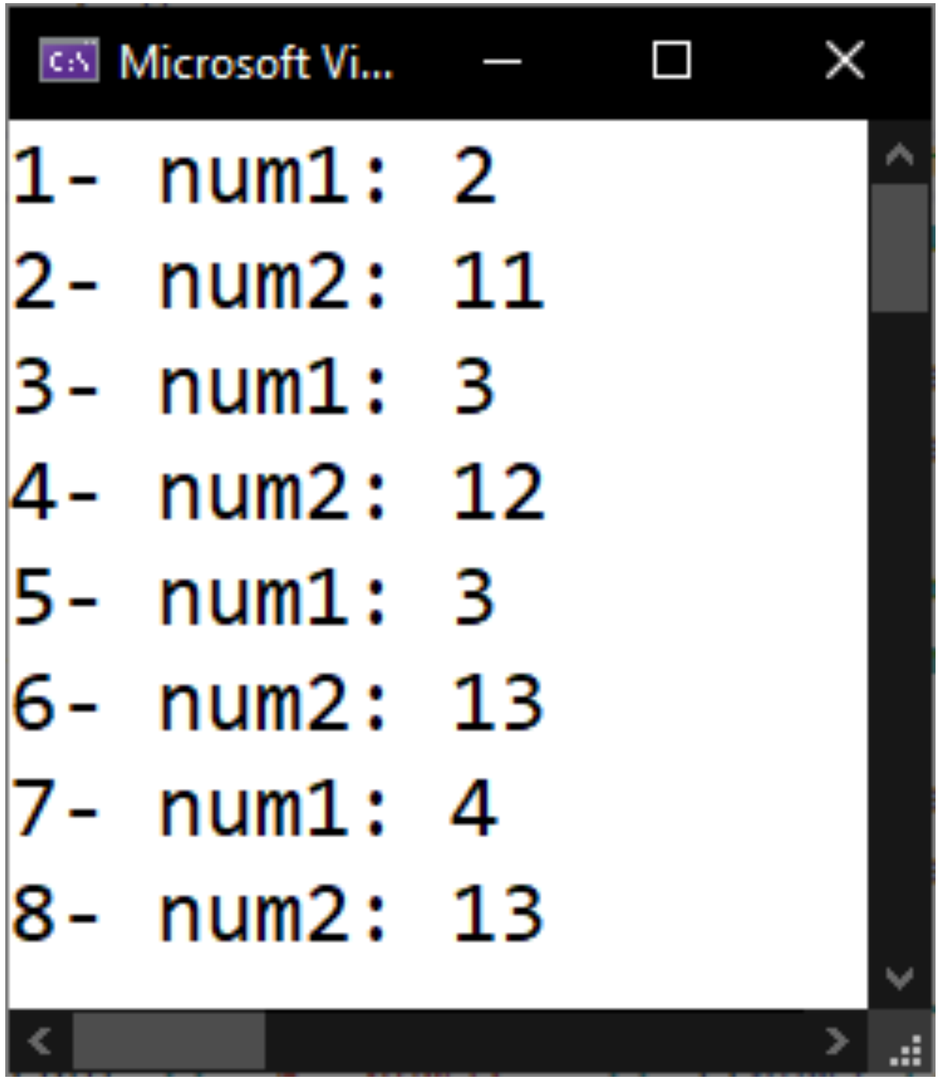
- Operator comes after the operand.
- Operand's value is used first, then it's incremented or decremented.
- The syntax for the postfix **increment** and **decrement** operators is **x++** and **x--**, respectively.

Prefix Mode: (++x)

- Operator comes before the operand.
- Operand is incremented or decremented first, then its updated value is used.
- The syntax for the prefix increment and decrement operators is **++x** and **--x**, respectively

The Difference Between Postfix and Prefix Modes

```cpp
#include <iostream>
using namespace std;
int main(){
int num1 = 2; // num1 starts out with 2
int num2 = 11; // num2 starts out with 11

cout << "1- num1: " << num1 << endl;
cout << "2- num2: " << num2 << endl;

num1++; // Use postfix ++ to increment
++num2; // Use prefix ++ to increment

cout << "3- num1: " << num1 << endl;
cout << "4- num2: " << num2 << endl;

cout << "5- num1: " << num1++ << endl;
cout << "6- num2: " << ++num2 << endl;

cout << "7- num1: " << num1 << endl;
cout << "8- num2: " << num2 << endl;

return 0;
}
```



Combined Assignment

- Combined Assignment, also known as compound assignment, involves combining an arithmetic operation with an assignment.
- It allows you to perform an operation (such as addition, subtraction, multiplication, etc.) and assignment in a single statement.

Operator	Example Usage	Equivalent to
+=	`x += 5;`	`x = x + 5;`
-=	`y -= 2;`	`y = y - 2;`
*=	`z *= 10;`	`z = z * 10;`
/=	`a /= b;`	`a = a / b;`
%=	`c %= 3;`	`c = c % 3;`

Combined Assignment -Example

```cpp
#include <iostream>
using namespace std;
int main() {
    int total = 0;
    total += 10;
    total += 5;
    total -= 3;

    cout << "Total: " << total << endl;

    return 0;
}
```

Introduction to Loops

A **loop** is a control structure that causes a statement or group of statements to repeat.

C++ has three looping control structures:

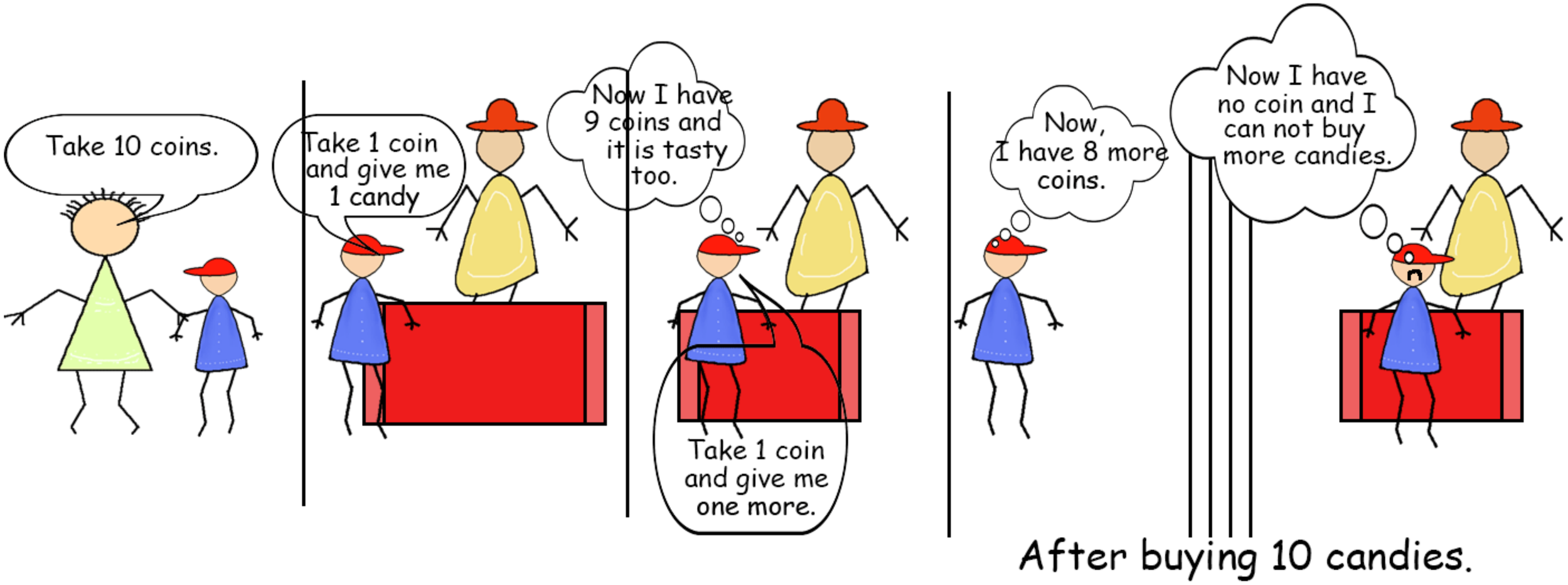
- `while` loop,
- `do-while` loop, and
- `for` loop.

The difference between these structures is how they control the repetition.

Introduction to Loops

- There are two main types of loops: **condition-based** and **number-based** loops.
- A **condition-based loop** runs as long as a specific condition is true, and the number of iterations is uncertain. (While Loop)
- In contrast, a **number-based** loop repeats a fixed number of times. (For Loop)

Introduction to Loops

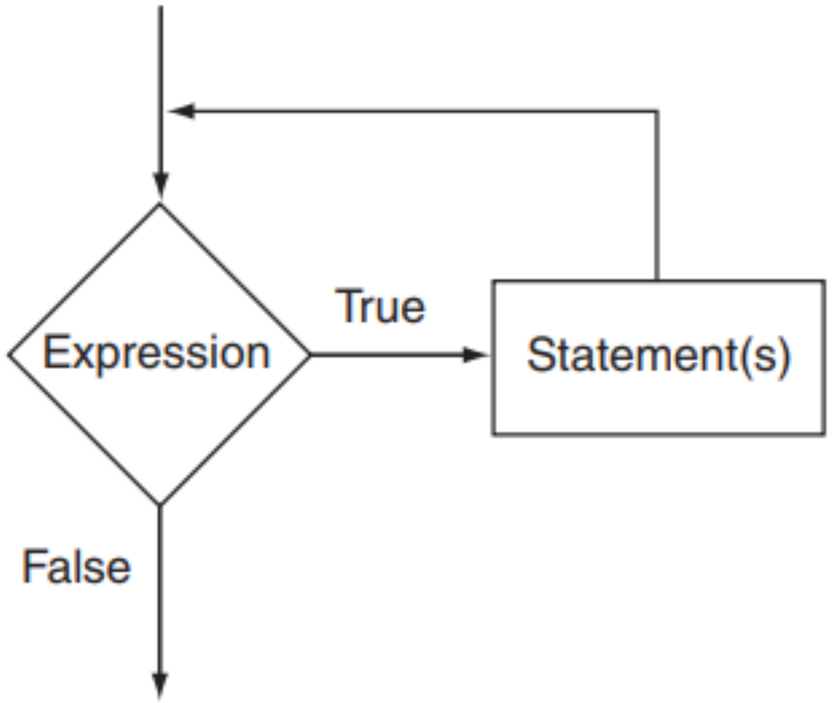


The `while` Loop

The **while loop** has two important parts:

1- an expression that is tested for a true or false value.

2- a statement or block that is repeated as long as the expression is true.



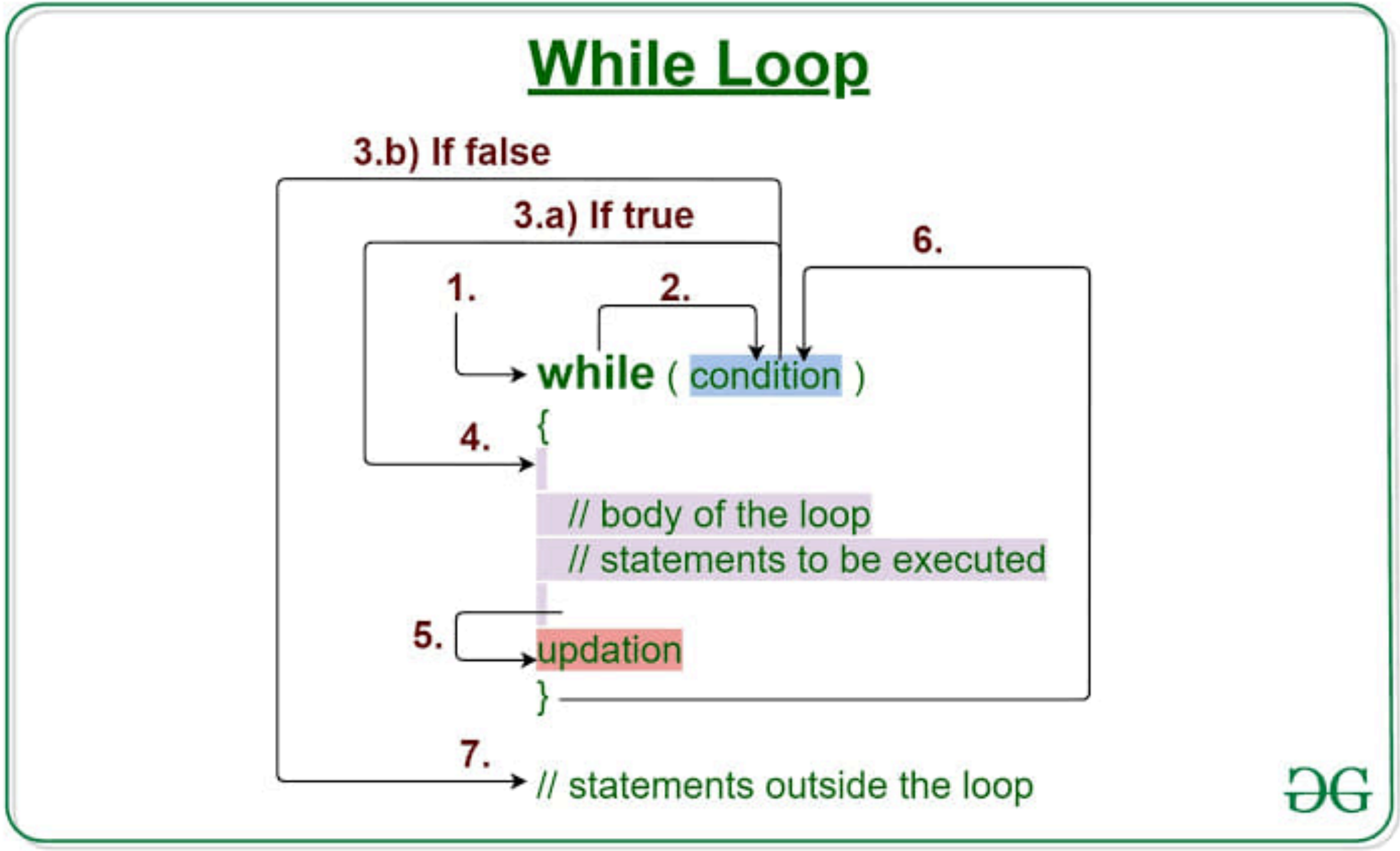

```
while (expression)
{
    statement;
    statement;
    // Place as many statements here
    // as necessary.
}
```

The `while` Loop

```
int a = 1;
while ( a < 4 )
{
cout << "Hello World" << endl;
a ++;
}
```

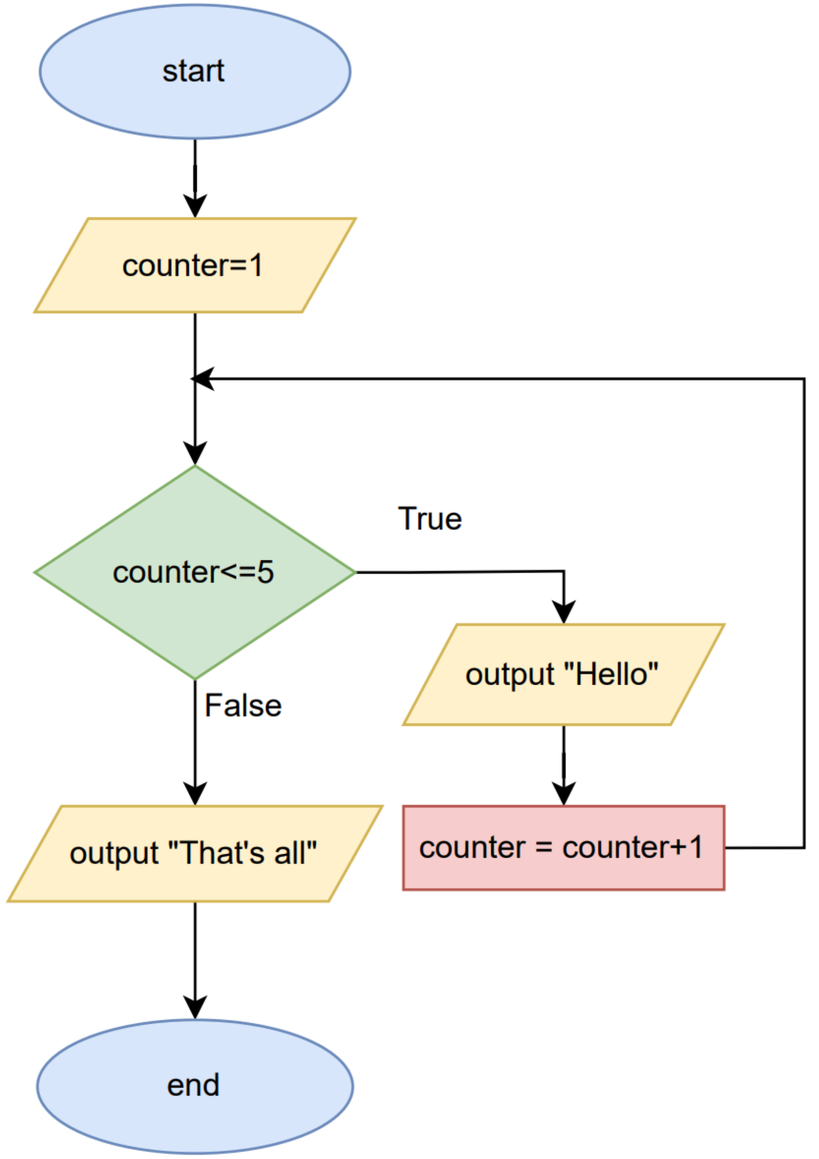
Output

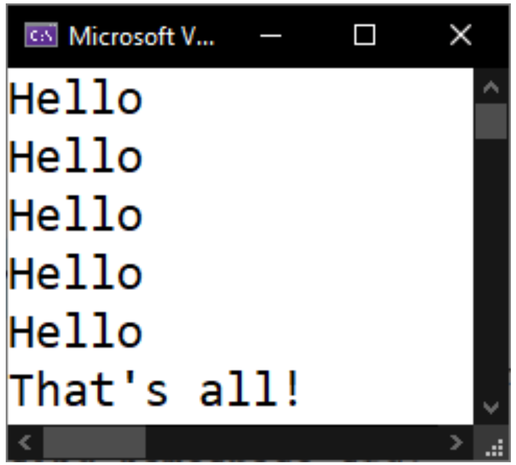
The `while` Loop



The while Loop - Example #1

```
#include <iostream>
using namespace std;
int main(){
    int counter = 1;

    while (counter <= 5){
        cout << "Hello" << endl;
        counter++;
    }

    cout << "That's all!";

    return 0;
}
```





The while Loop - Example #2

Write C++ code that prints numbers from 1 to 10 and finds the square for each.

```
#include <iostream>
using namespace std;
int main() {
    int i = 1;
    cout << "Number \t\t Square" << endl;
    while (i <= 10) {
        cout << i << "\t\t\t\t" << i * i << endl;
        i++;
    }

    return 0;
}
```

Output

```
Number          Square
1                  1
2                  4
3                  9
4                  16
5                  25
6                  36
7                  49
8                  64
9                  81
10                 100
```

Infinite Loops

In most situations, loops need a way to stop. This means that something inside the loop must eventually make the condition false.

The below loop goes on forever because it lacks a statement to modify the number variable. With each test of the expression counter <= 5, the number variable remains at 1, causing an **infinite** loop.

```
#include <iostream>
using namespace std;
int main(){
    int counter = 1;

    while (counter <= 5){
        cout << "Hello" << endl;
    }

    cout << "That's all!";

    return 0;
}
```

Using the while Loop for Input Validation - Example #1

```cpp
#include <iostream>
using namespace std;

int main() {
    int num;

    cout << "Enter a positive number: ";
    cin >> num;

    while (num <= 0) {
        cout << "Invalid input. Please enter a positive number: ";
        cin >> num;
    }

    cout << "You entered a positive number: " << num << endl;

    return 0;
}
```

Output

```
Enter a positive number: 0
Invalid input. Please enter a positive number: -2
Invalid input. Please enter a positive number: -19
Invalid input. Please enter a positive number: 5
You entered a positive number: 5
```

Using the while Loop for Input Validation - Example #2

```cpp
#include <iostream>
using namespace std;

int main() {
    int num;

    while (true) {
        cout << "Enter a positive number: ";
        cin >> num;

        if (num > 0) {
            break; // Exit the loop if the user enters a positive number
        }

        cout << "Invalid input. ";
    }

    cout << "You entered a positive number: " << num << endl;

    return 0;
}
```

Output

```
Enter a positive number: 0
Invalid input. Enter a positive number: -6
Invalid input. Enter a positive number: 8
You entered a positive number: 8
```

Using the while Loop for Input Validation

```cpp
#include <iostream>
using namespace std;

int main() {
    int num;

    while (true) {
        cout << "Enter a positive number: ";
        cin >> num;

        if (num > 0) {
            break; // Exit the loop if the user enters a positive number
        }

        cout << "Invalid input. ";
    }

    cout << "You entered a positive number: " << num << endl;

    return 0;
}
```

```cpp
#include <iostream>
using namespace std;

int main() {
    int num;

    cout << "Enter a positive number: ";
    cin >> num;

    while (num <= 0) {
        cout << "Invalid input. Please enter a positive number: ";
        cin >> num;
    }

    cout << "You entered a positive number: " << num << endl;

    return 0;
}
```

Using the while Loop for Input Validation - Example #2

```cpp
#include <iostream>
using namespace std;
int main(){
    int grade;
    cout << "Enter a grade: ";
    cin >> grade;
    while (grade < 0 || grade > 100){
        cout << "Invalid grade\nEnter valid grade: ";
        cin >> grade;
    }

    if (grade >= 50){
        cout << "Passed" << endl;
    } else {
        cout << "Failed" << endl;
    }
    return 0;
}
```

Output

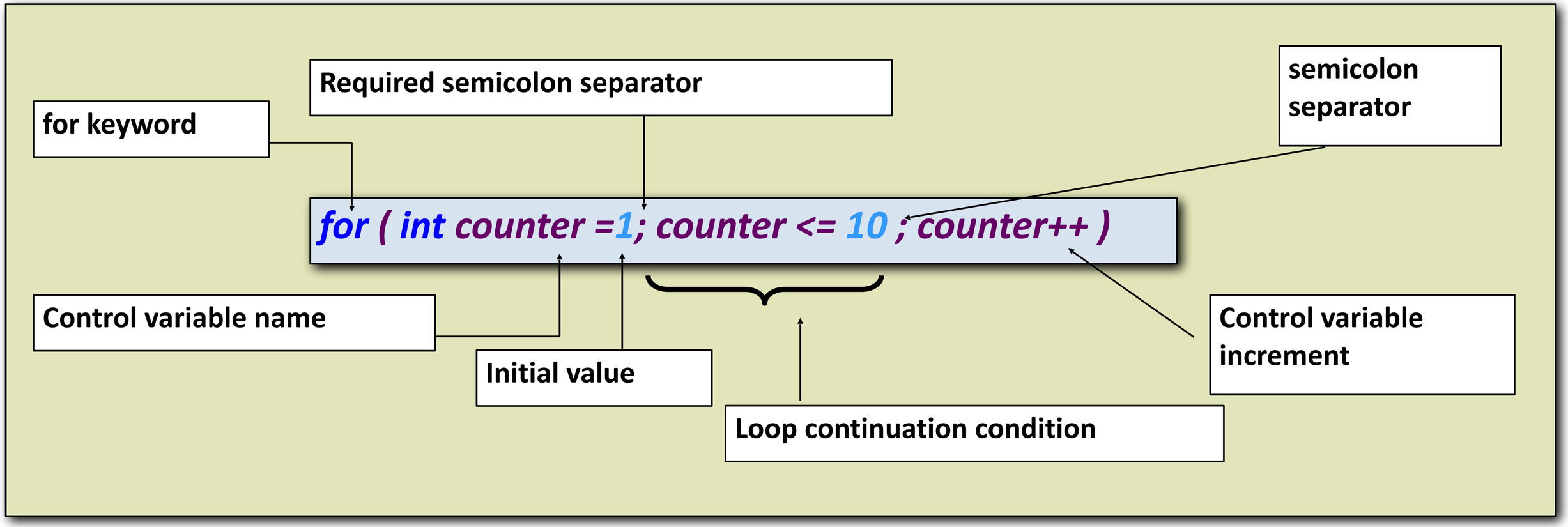
```
Enter a grade: -5
Invalid grade
Enter valid grade: 101
Invalid grade
Enter valid grade: 78
Passed
```

The for Loop

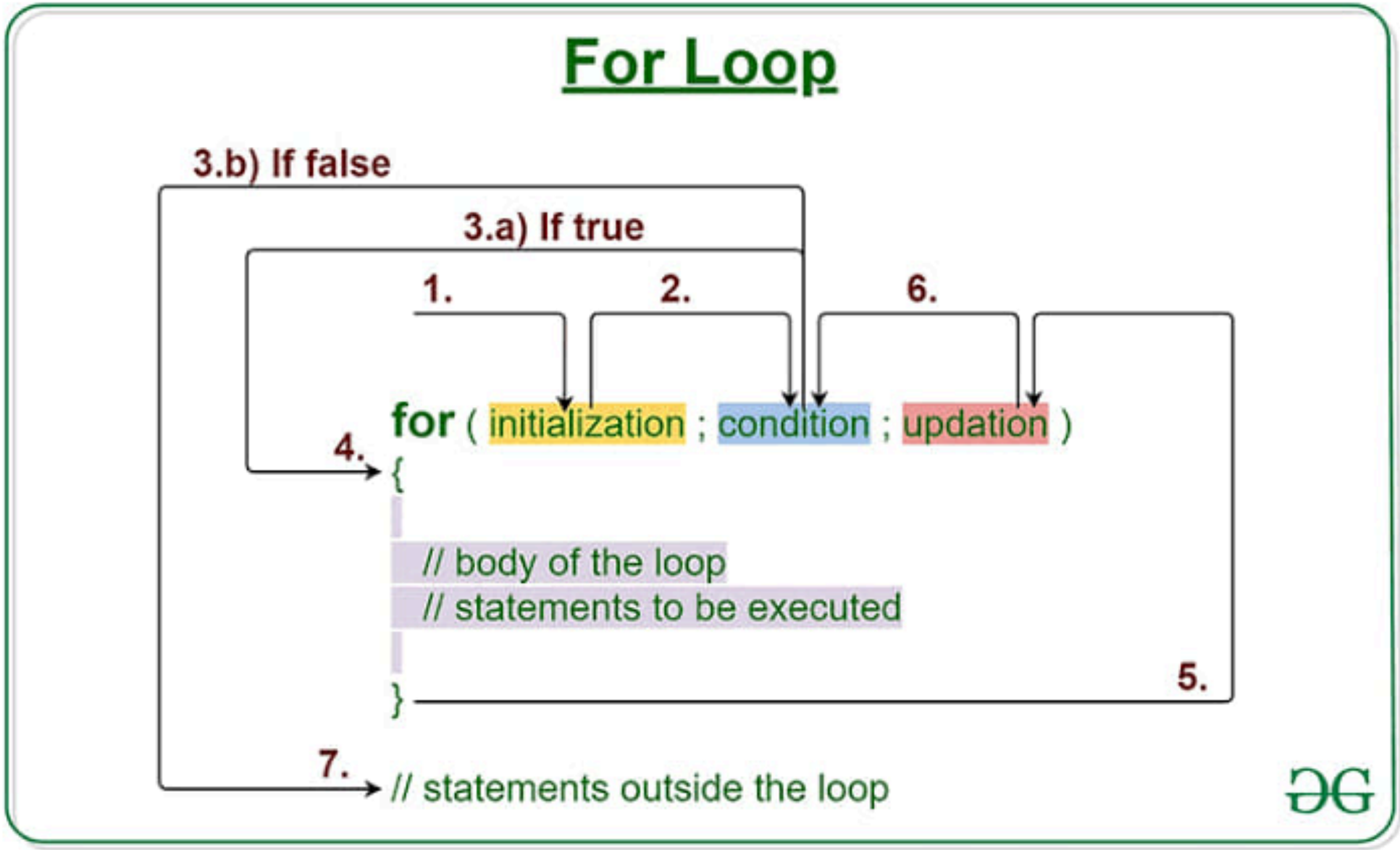
- Count-controlled loops are so common that C++ provides a type of loop specifically for them. It is known as the **`for`** loop.
- The **for** loop is suitable when a known number of iterations is required.
- Three essential elements define a **count-controlled loop**:
  - **Initialization**: It starts with setting a counter variable to an initial value.
  - **Termination condition**: The loop runs while the counter variable is less than or equal to a maximum value; when false, the loop ends.
  - **Update**: The counter variable is modified during each iteration, typically through incrementing.

```
for (initialization; test; update)
{
   statement;
   statement;
   // Place as many statements here
   // as necessary.
}
```

The for Loop


for keyword
Required semicolon separator
semicolon separator
for (int counter =1; counter <= 10 ; counter++)
Control variable name
Initial value
Loop continuation condition
Control variable increment

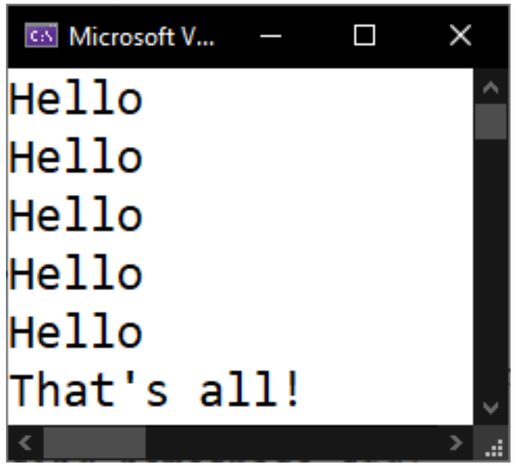
The for Loop


For Loop
3.b) If false
3.a) If true
1.
2.
6.
for (initialization ; condition ; updation)
4.
{
// body of the loop
// statements to be executed
}
5.
7.
// statements outside the loop

The for Loop - Example #1

```cpp
#include <iostream>
using namespace std;
int main() {

    for(int i=0;i<5;i++){
        cout<<"Hello"<<endl;
    }
    cout<<"That's All";

    return 0;
}
```

Output



The for Loop - Example #2

```cpp
#include <iostream>
using namespace std;
int main() {

    cout<<"Number\t\tSquare"<<endl;
    for(int i=1;i<=10;i++){
        cout<<i<<"\t\t\t\t"<<i*i<<endl;
    }

    return 0;
}
```

Output

```
Number          Square
1                   1
2                   4
3                   9
4                   16
5                   25
6                   36
7                   49
8                   64
9                   81
10                  100
```

Other Forms of the Update Expression

- Vary the control variable from 1 to 100 in increments of 1.

```
for ( int i = 1; i <= 100; i++ )
```

- Vary the control variable from 100 down to 1 in increments of -1 (decrements of 1).

```
for ( int i = 100; i >= 1; i-- )
```

- Vary the control variable from 7 to 77 in steps of 7.

```
for ( int i = 7; i <= 77; i += 7 )
```

- Vary the control variable from 20 down to 2 in steps of -2.

```
for ( int i = 20; i >= 2; i -= 2 )
```

- Vary the control variable over the following sequence of values: 2, 5, 8, 11, 14, 17, 20.

```
for ( int i = 2; i <= 20; i += 3 )
```

Other Forms of the Update Expression

```cpp
#include <iostream>
using namespace std;
int main() {
    // 1. Increasing increments
    for (int i = 1; i <= 10; i+=2) {
        cout << i << " ";
    }
    cout << "\n";

    // 2. Decreasing increments
    for (int i = 10; i >= 1; i-=2) {
        cout << i << " ";
    }
    cout << "\n";

    // 3. Increments of other amounts
    for (int i = 5; i <= 20; i+=5) {
        cout << i << " ";
    }
    cout << "\n";

    return 0;
}
```

```cpp
#include <iostream>
using namespace std;
int main() {
    cout << "Generating the sequence: 1, 4, 9, 16, 25\n";

    for (int i = 1; i <= 5; i++) {
     int squared = i * i;
     cout << squared << " ";
    }
    cout << "\n";

    cout << "Generating the sequence: 10, 8, 6, 4, 2\n";

    for (int i = 10; i >= 2; i-=2){
       cout << i << " ";
    }
    cout << "\n";

    cout << "Generating the sequence: 3, 6, 9, 12, 15\n";
    for (int i = 3; i <= 15; i+=3){
      cout << i << " ";
    }

   cout << "\n";

    return 0;
}
```

Creating a User Controlled for Loop

Write a C++ program that asks the user to input two numbers and print the numbers between them.

```cpp
#include <iostream>
using namespace std;
int main() {

    int firstNumber, secondNumber;

    cout << "Enter the first number: ";
    cin >> firstNumber;

    cout << "Enter the second number: ";
    cin >> secondNumber;

    for (int i = firstNumber+1; i < secondNumber; i++) {
        cout << i << " ";
    }

    return 0;
}
```

Output

```
Enter the first number: 5
Enter the second number: 9
6 7 8
```

“break" Statement

- The break statement is used to prematurely exit a loop (such as a for loop and while loop) when a certain condition is met.
- When encountered, the break statement terminates the nearest enclosing loop.
- It's useful for avoiding unnecessary iterations and improving code efficiency.

“break" Statement

```
#include <iostream>
using namespace std;
int main() {
    for (int i = 0; i < 10; i++) {
    if (i == 5) {
        cout << "Breaking the loop at i = " << i << endl;
        break;
    }
    cout << "Currently at i = " << i << endl;
}
    return 0;
}
```

Thank You