

Tishk International University
IT Department
Course Code: IT-117

Programming I

Lecture 5



Loops

Fall 2023

Hemin Ibrahim

hemin.ibrahim@tiu.edu.iq



Outline



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

Objectives



- 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.

Both of the following statements 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.

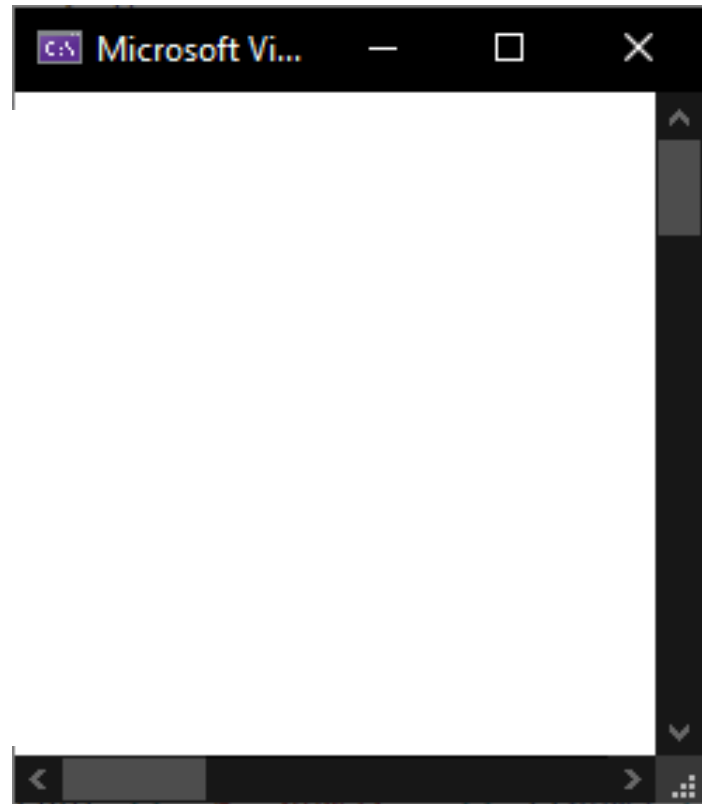
Prefix Mode: (**++x**)

- 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



```
1  #include <iostream>
2  using namespace std;
3  int main(){
4  int num1 = 2; // num1 starts out with 2
5  int num2 = 11; // num2 starts out with 11
6
7  cout << "1- num1: " << num1 << endl;
8  cout << "2- num2: " << num2 << endl;
9
10 num1++; // Use postfix ++ to increment
11 ++num2; // Use prefix ++ to increment
12
13 cout << "3- num1: " << num1 << endl;
14 cout << "4- num2: " << num2 << endl;
15
16 cout << "5- num1: " << num1++ << endl;
17 cout << "6- num2: " << ++num2 << endl;
18
19 cout << "7- num1: " << num1 << endl;
20 cout << "8- num2: " << num2 << endl;
21
22 return 0;
23 }
```



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
<code>+=</code>	<code>x += 5;</code>	<code>x = x + 5;</code>
<code>-=</code>	<code>y -= 2;</code>	<code>y = y - 2;</code>
<code>*=</code>	<code>z *= 10;</code>	<code>z = z * 10;</code>
<code>/=</code>	<code>a /= b;</code>	<code>a = a / b;</code>
<code>%=</code>	<code>c %= 3;</code>	<code>c = c % 3;</code>



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.

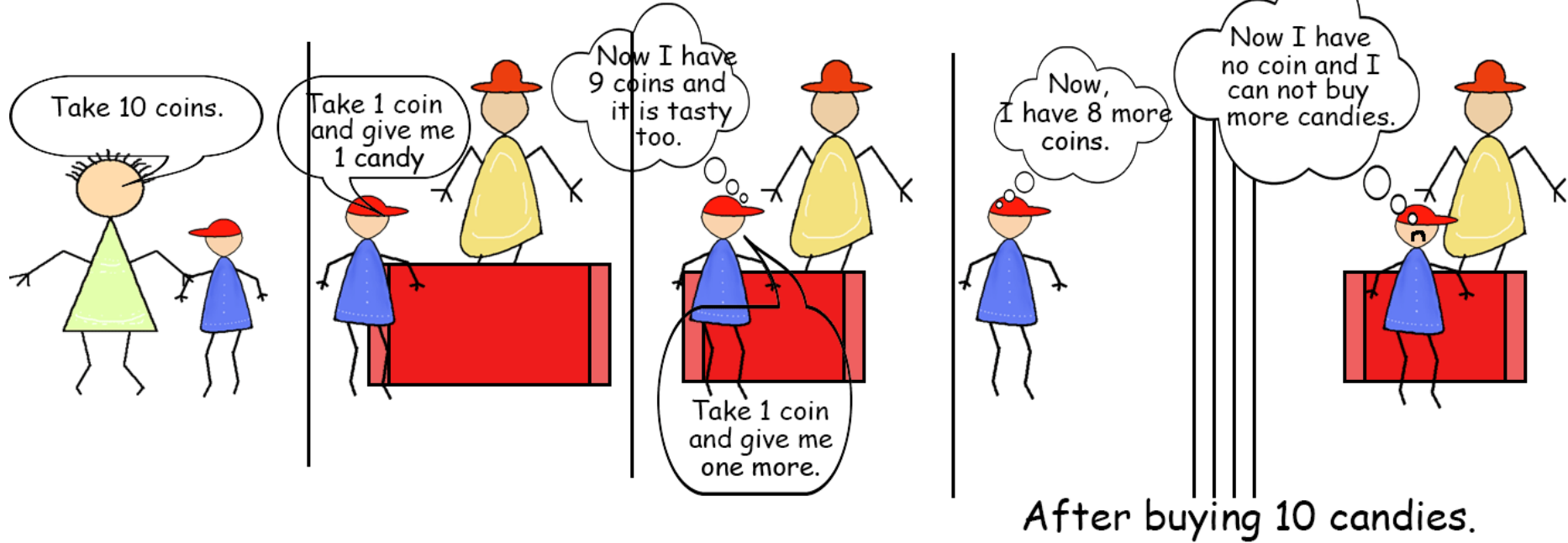
The difference between these structures is how they control the repetition.



Introduction to Loops

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

Introduction to Loops

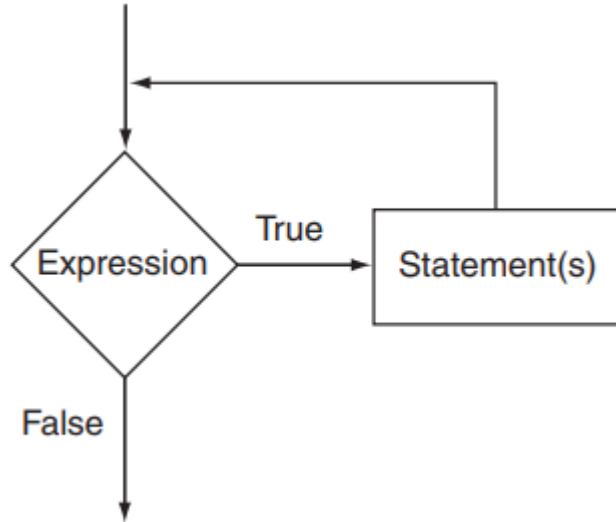


The while Loop

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

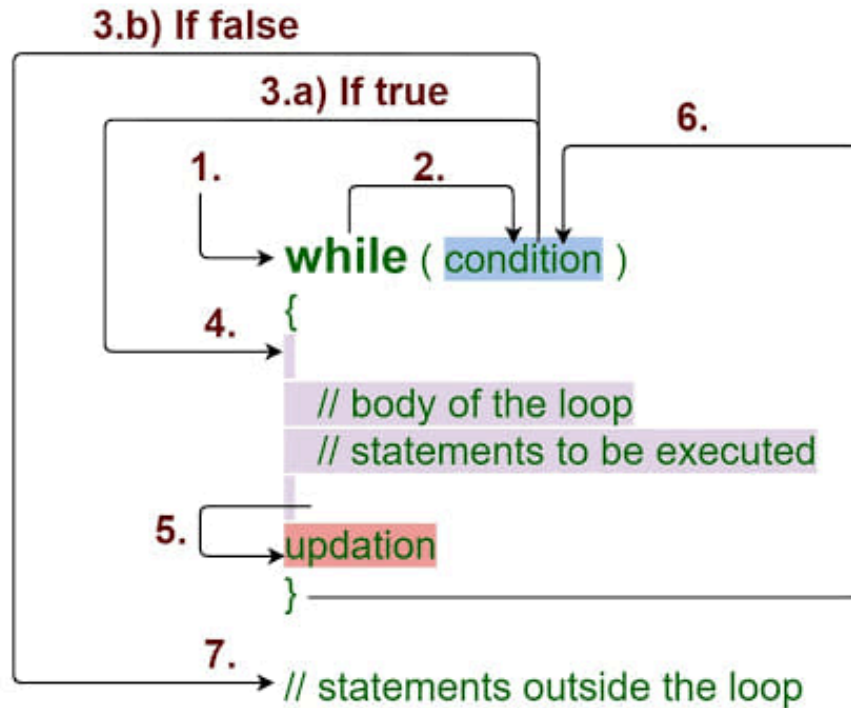


```
int a = 1;
while ( a < 4 )
{
cout << "Hello World" << endl;
a ++;
}
```

Output

The while Loop

While Loop



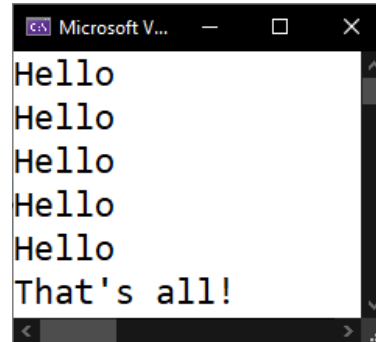
The while Loop - Example #1

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

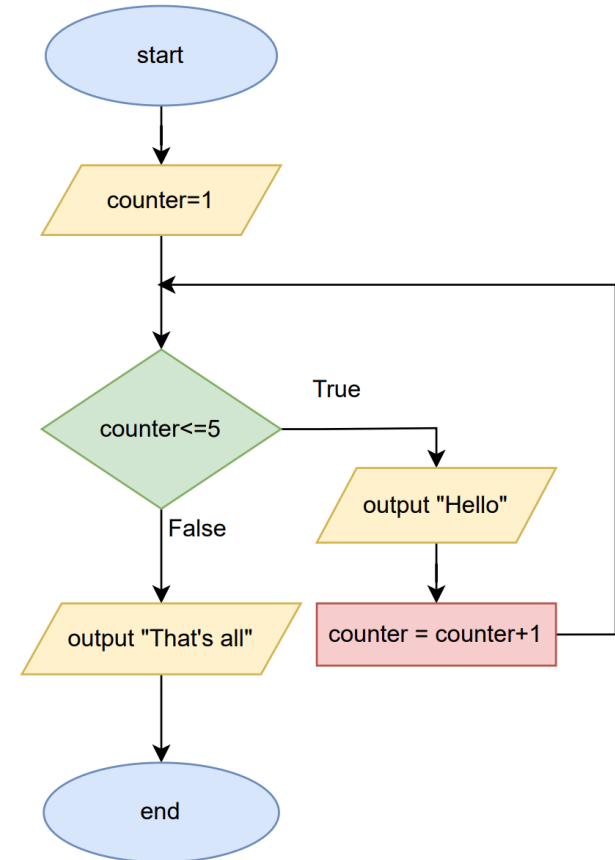
    while (counter <= 5){
        cout << "Hello" << endl;
        counter++;
    }

    cout << "That's all!";

    return 0;
}
```

A screenshot of a Windows command prompt window titled 'Microsoft V...'. The window displays the output of the C++ program: five lines of 'Hello' followed by 'That's all!' on the sixth line.

```
Microsoft V...
Hello
Hello
Hello
Hello
Hello
That's all!
```



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

```
#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

```
#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

```
#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;
}
```

```
#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

```
#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
```

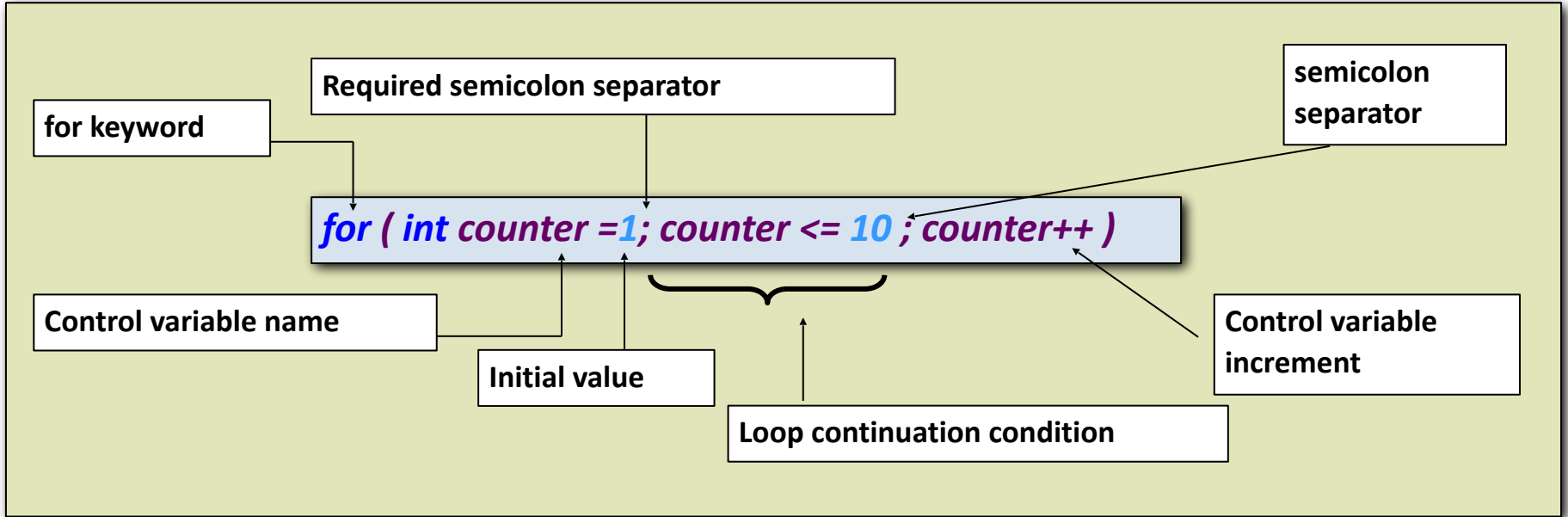
The for Loop



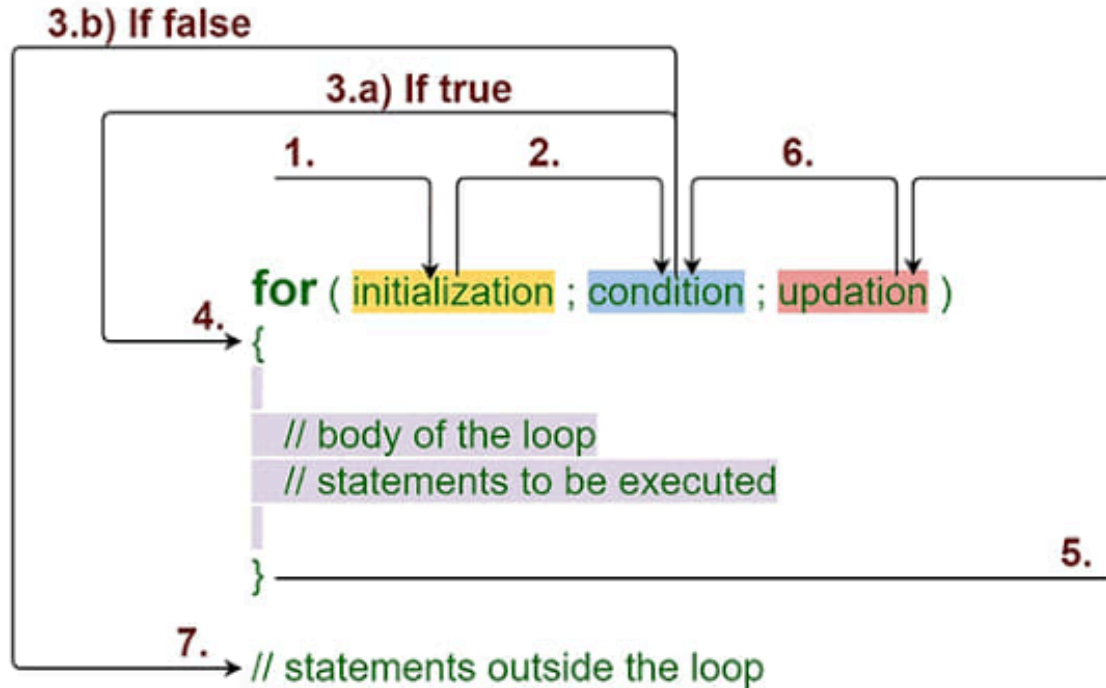
- Count-controlled loops are so common that C++ provides a type of loop specifically for them. It is known as the **for** loop.
- The **for** loop is suitable when a known number of iterations is required.
- Three essential elements define a **count-controlled loop**:
 - **Initialization**: It starts 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 Loop



The for Loop - Example #1



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

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

    return 0;
}
```

Output

A screenshot of a Microsoft Visual Studio console window. The window title is "Microsoft V...". The console output shows five lines of "Hello" followed by a line that says "That's all!".

```
Microsoft V...
Hello
Hello
Hello
Hello
Hello
That's all!
```

The for Loop - Example #2

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

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



Creating a User Controlled for Loop

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

```
#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

