Loops in C++

Introduction

Loops are used to repeat a block of code for a certain number of times or until a certain condition is met. Loops are useful for performing tasks that require iteration, such as processing data, generating sequences, validating input, etc.

There are four types of loops in C++:

- `for` loop: This loop executes a block of code for a fixed number of times based on an initial value, a condition, and an update expression.
- `while` loop: This loop executes a block of code repeatedly as long as a condition is true.
- `do-while` loop: This loop executes a block of code at least once and then repeats it as long as a condition is true.
- `range-based for` loop: This loop executes a block of code for each element in a container or range.

In this document, we will explain each type of loop in detail and give some examples of how to use them in C++.

`for` Loop

The `for` loop is used to execute a block of code for a fixed number of times based on an initial value, a condition, and an update expression. The syntax of the `for` loop is:

```
for (initialization; condition; update) {
    // code block to be executed
}
```

The initialization part is executed only once before the loop starts. It is usually used to declare and initialize a variable that controls the loop.

The condition part is evaluated before each iteration of the loop. It is usually a logical expression that determines whether the loop should continue or stop. If the condition is true, the code block is executed; if the condition is false, the loop is terminated.

The update part is executed after each iteration of the loop. It is usually used to modify the value of the variable that controls the loop.

The code block inside the curly braces `{}` is called the body of the loop. It can contain one or more statements, or be empty. The body of the loop is executed for each iteration as long as the condition is true.

For example, suppose we want to print the numbers from 1 to 10 using a `for` loop. We can write the following code:

```
// declare and initialize a variable i with 1
for (int i = 1; i <= 10; i++) {
    // print the value of i followed by a space
    cout << i << " ";
}
// print a new line after the loop ends
cout << endl;
```

The output of this code will be:

```
1 2 3 4 5 6 7 8 9 10
```

Here is how this code works:

- The initialization part sets `i` to 1 before the loop starts.
- The condition part checks if `i` is less than or equal to 10 before each iteration. If true, the body of the loop is executed; if false, the loop is terminated.
- The update part increases `i` by 1 after each iteration.
- The body of the loop prints the value of `i` followed by a space.
- After the loop ends, a new line is printed.

`while` Loop

The `while` loop is used to execute a block of code repeatedly as long as a condition is true. The syntax of the `while` loop is:

```
while (condition) {
    // code block to be executed
}
```

The condition part is evaluated before each iteration of the loop. It is usually a logical expression that determines whether the loop should continue or stop. If the condition is true, the code block is executed; if the condition is false, the loop is terminated.

The code block inside the curly braces `{}` is called the body of the loop. It can contain one or more statements, or be empty. The body of the loop is executed for each iteration as long as the condition is true.

For example, suppose we want to print “Hello World” five times using a `while` loop. We can write the following code:

```
// declare and initialize a variable count with 0
int count = 0;
// check if count is less than 5 before each iteration
while (count < 5) {
    // print "Hello World" followed by a new line
    cout << "Hello World" << endl;
    // increase count by 1 after each iteration
    count++;
}
```

The output of this code will be:

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

Here is how this code works:

- The variable `count` is initialized with 0 before the loop starts.
- The condition part checks if `count` is less than 5 before each iteration. If true, the body of the loop is executed; if false, the loop is terminated.
- The body of the loop prints “Hello World” followed by a new line and increases `count` by 1.
- The loop repeats until `count` becomes equal to 5, which makes the condition false.

`do-while` Loop

The `do-while` loop is used to execute a block of code at least once and then repeat it as long as a condition is true. The syntax of the `do-while` loop is:

```
do {
    // code block to be executed
} while (condition);
```

The code block inside the curly braces `{}` is called the body of the loop. It can contain one or more statements, or be empty. The body of the loop is executed for the first time without checking the condition.

The condition part is evaluated after each iteration of the loop. It is usually a logical expression that determines whether the loop should continue or stop. If the condition is true, the code block is executed again; if the condition is false, the loop is terminated.

The `do-while` loop differs from the `while` loop in terms of when the condition is checked. The `while` loop checks the condition before each iteration, while the `do-while` loop checks the condition after each iteration.

For example, suppose we want to ask the user to enter a positive number and print it using a `do-while` loop. We can write the following code:

```
// declare a variable num to store the user input
int num;
// execute the body of the loop at least once
do {
    // ask the user to enter a positive number
    cout << "Enter a positive number: ";
    // read the user input and store it in num
    cin >> num;
    // print the value of num followed by a new line
    cout << "You entered: " << num << endl;
// check if num is negative after each iteration
} while (num < 0);
```

The output of this code will be:

```
Enter a positive number: -5
You entered: -5
Enter a positive number: -3
You entered: -3
Enter a positive number: 10
You entered: 10
```

Here is how this code works:

- The variable `num` is declared before the loop starts.
- The body of the loop asks the user to enter a positive number, reads it, and prints it.
- The condition part checks if `num` is negative after each iteration. If true, the body of the loop is executed again; if false, the loop is terminated.
- The loop repeats until the user enters a non-negative number, which makes the condition false.

Range-based `for` Loop

The range-based `for` loop is used to execute a block of code for each element in a container or range. The syntax of the range-based `for` loop is:

```
for (type variable : container) {
    // code block to be executed
}
```

The type part specifies the data type of each element in the container or range. It can be auto or a specific type.

The variable part declares a variable that holds each element in turn.

The container part specifies the container or range to iterate over. It can be an array, a vector, a string, or any other data structure that supports range access.

The code block inside the curly braces `{}` is called the body of the loop. It can contain one or more statements, or be empty. The body of the loop is executed for each element in the container or range.

For example, suppose we want to print all elements in an array using a range-based `for` loop. We can write the following code:

```
// declare and initialize an array with 5 elements
int arr[5] = {10, 20, 30, 40, 50};
// iterate over each element in arr using auto type
for (auto x : arr) {
    // print x followed by a space
    cout << x << " ";
}
```

```
// print a new line after the loop ends
cout << endl;
```

The output of this code will be:

```
10 20 30 40 50
```

Here is how this code works:

- The array `arr` is declared and initialized with 5 elements before the loop starts.
- The range-based `for` loop uses auto type to infer the data type of each element in `arr`.
- The variable `x` holds each element in turn.
- The body of the loop prints `x` followed by a space.
- After the loop ends, a new line is printed.

Conclusion

In this document, we have learned about four types of loops in C++: `for`, `while`, `do-while`, and range-based `for`. We have seen how to use them to