Loop Statement

- C++ provides a set of **looping statements** that allow a block of code to be **executed repeatedly** as long as a specified condition remains true.
- Looping is commonly used to **traverse arrays**, **repeat tasks**, or **perform operations** until a desired condition is met.

C++ provides the following types of loops:

- **do-while loop**
- **while loop**
- **for loop**

do-while Loop in C++

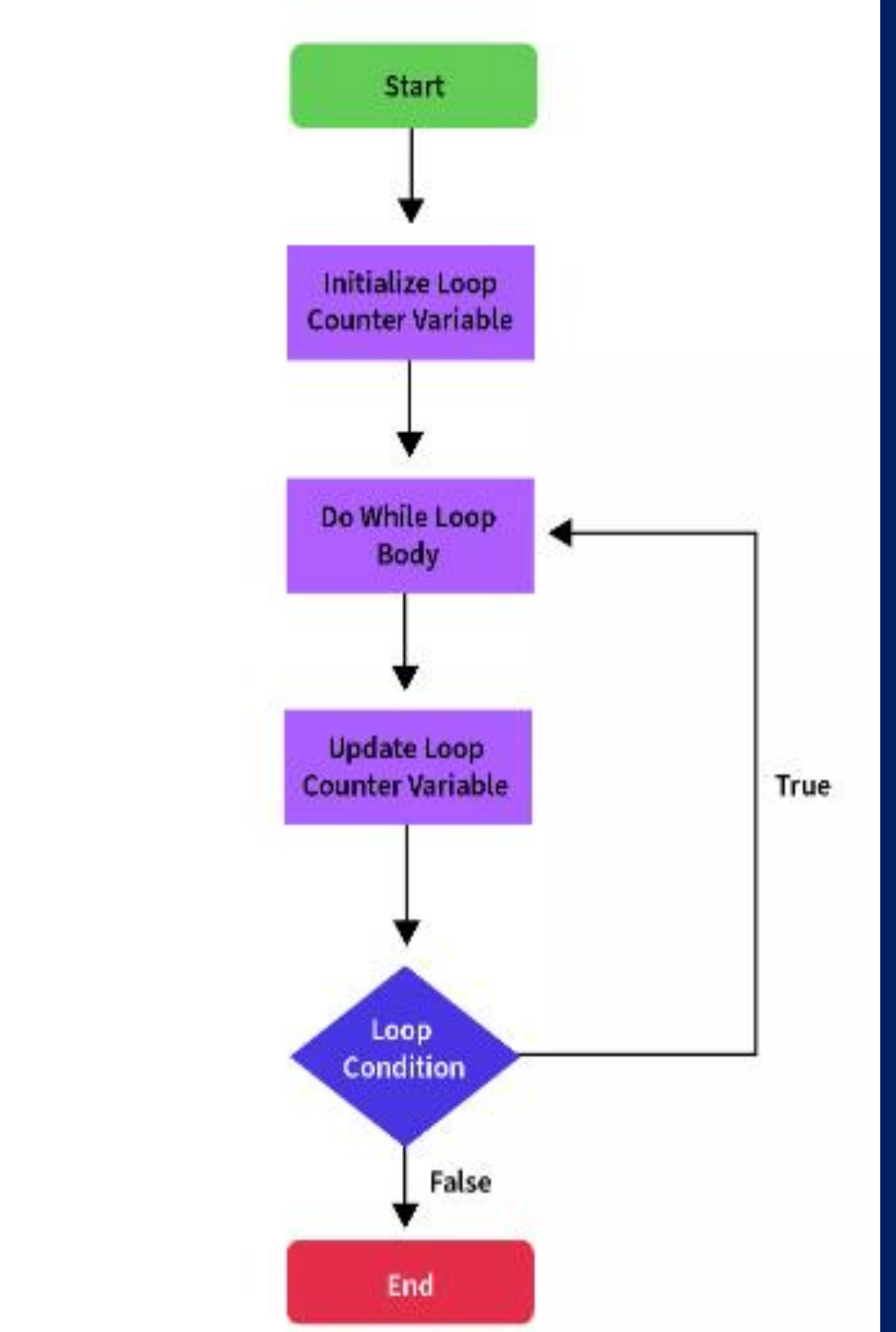
Key Points

- Executes a block of code **at least once**, no matter what the condition is.
- The **condition is checked after** executing the loop body.
- Best used when the loop **must run at least once** before checking the condition.
- Uses the **do { } while(condition);** syntax.
- Ends with a **semicolon** after the while(condition) part.

Syntax:

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

do-while Loop in C++



do-while Loop Examples:

1: Print numbers from 1 to 5
2: Keep asking for password until correct

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

int main() {
    int i = 1;
    do {
        cout << i << " ";
        i++;
    } while (i <= 5);
    return 0;
}
```

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

int main() {
    int password;
    do {
        cout << "Enter password: ";
        cin >> password;
    } while (password != 1234);

    cout << "Access Granted!" << endl;
    return 0;
}
```

while Loop in C++

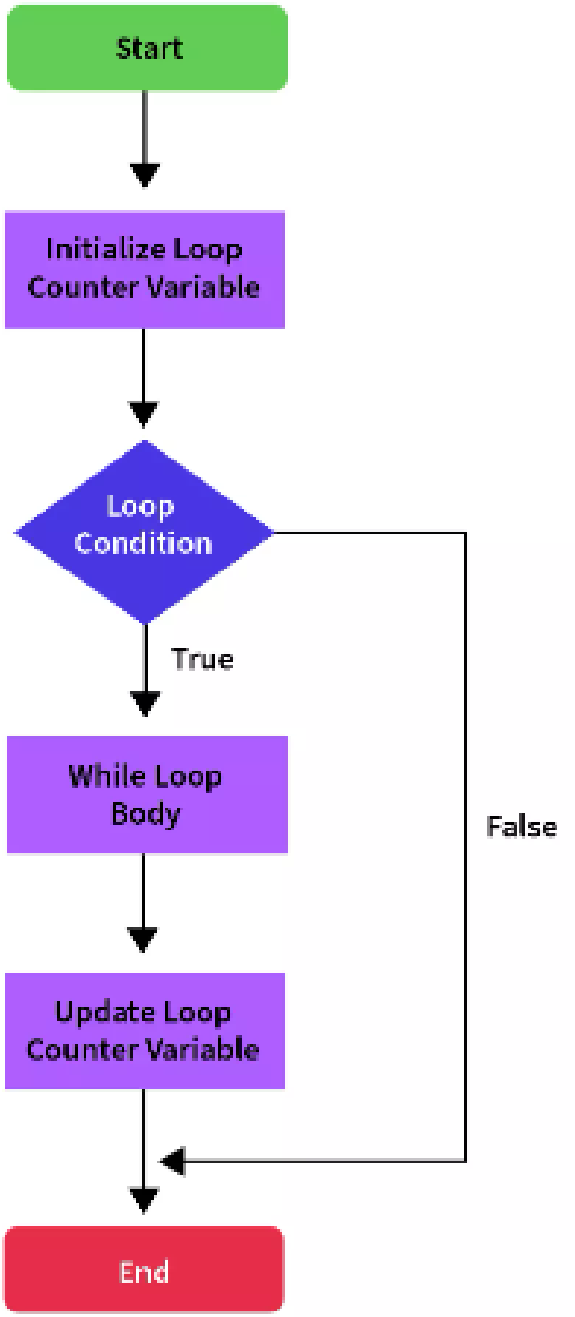
Key Points

- Executes a block of code repeatedly as long as the condition is true.
- The condition is checked before executing the loop body.
- If the condition is false at the beginning, the loop body does not run at all.
- Best used when the number of iterations is not known in advance.
- Uses the while (condition) { } syntax.
- No semicolon after the while(condition) line (unless loop is empty).

Syntax:

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

while Loop in C++



while Loop Examples:

1: Countdown from 5 to 1
2: Sum of first 5 natural numbers

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

int main() {
    int i = 5;
    while (i >= 1) {
        cout << i << " ";
        i--;
    }
    return 0;
}
```

```
#include <iostream>
using namespace std;
int main() {
    int i = 1, sum = 0;
    while (i <= 5) {
        sum += i;
        i++;
    }
    cout << "Sum = " << sum;
    return 0;
}
```

For Loop in C++

Key Points

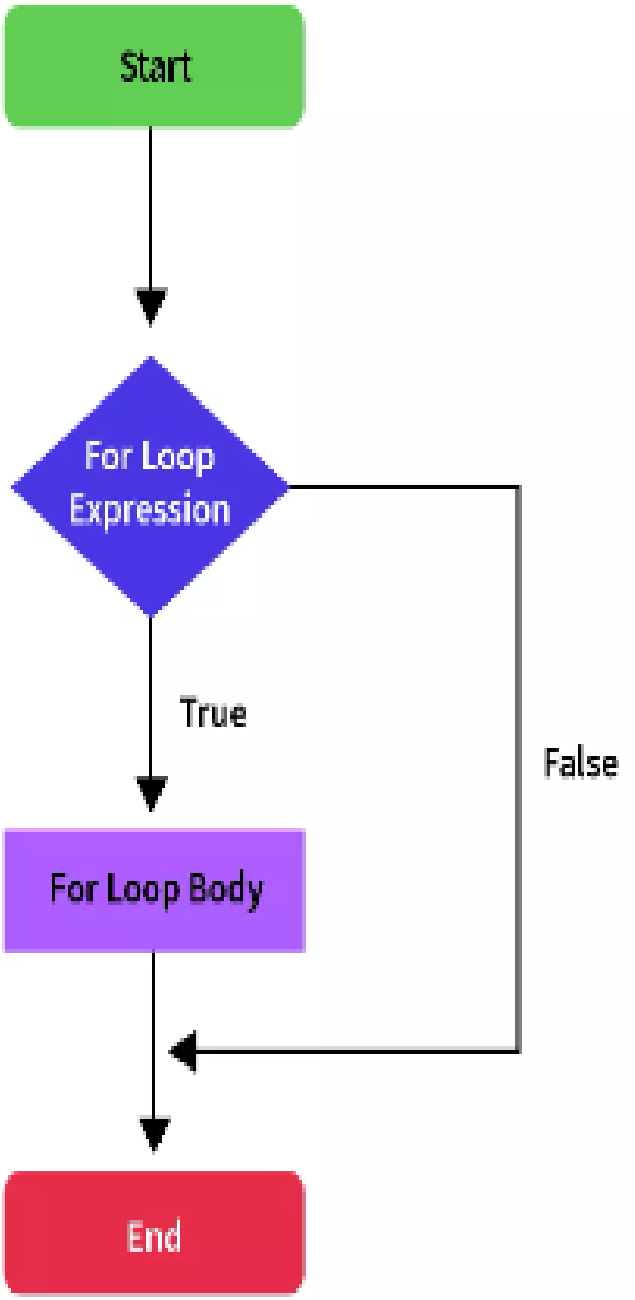
- Executes a block of code repeatedly for a specified number of times.
- The condition is checked before each iteration.
- The loop has initialization, condition, and update in one line.
- Best used when the number of iterations is known in advance.
- Uses the syntax:

  for (initialization; condition; update) { }
- No semicolon after the for line (unless loop is empty).

Syntax:

```
for (initialization;    condition;    increment/decrement)
{
   // code block to be executed if condition is true
}
```

while Loop in C++



while Loop Examples:

1: Print even numbers from 2 to 10
2: Calculate factorial of 5

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

int main() {
    for (int i = 2; i <= 10; i += 2) {
        cout << i << " ";
    }
    return 0;
}
```

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

int main() {
    int fact = 1;
    for (int i = 1; i <= 5; i++) {
        fact *= i;
    }
    cout << "Factorial of 5 is " << fact;
    return 0;
}
```

Jumping Statements

- Jumping statements are used to change the flow of execution by jumping from one part of the program to another.
- These are helpful in controlling loops or exiting functions based on certain conditions.
- They override the default top-to-bottom control flow in a C++ program.

Types of Jumping Statements

1. break
2. continue
3. return
4. goto

Jumping Statements

1 - break

- Used to **exit** from a loop (for, while, do-while) or switch statement immediately.
- Control moves to the **next statement after the loop or switch**.

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

int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3) {
            cout << "Breaking the loop at i = " << i << endl;
            break; // exits the loop
        }
        cout << "i = " << i << endl;
    }
    cout << "Loop exited using break." << endl;
    return 0;
}
```

Jumping Statements

2 - continue

- Used to **skip the current iteration** of a loop.
- The loop continues from the **next iteration**, skipping remaining code inside the loop for that cycle.

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

int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3) {
            cout << "Skipping i = " << i << endl;
            continue; // skip the rest and go to next iteration
        }
        cout << "i = " << i << endl;
    }
    cout << "Loop completed using continue." << endl;
    return 0;
}
```

Jumping Statements

3 - return

- Used to **exit a function** and optionally **return a value**.
- In main(), it is often used to return 0 to indicate successful program execution.

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

int add(int a, int b) {
    return a + b; // exits function and returns result
}

int main() {
    int result = add(10, 20);
    cout << "Sum = " << result << endl;
    return 0; // exits main function
}
```

Jumping Statements

4 - goto

- Used to **jump to a labeled statement** in the same function.
- Can make code confusing and harder to maintain.

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

int main() {
    int x = 5;
    if (x == 5) {
        goto jump; // jumps to the label named jump
    }
    cout << "This line is skipped." << endl;

 jump:
    cout << "This is the jump label." << endl;
    return 0;
}
```

Questions :

1. Print the multiplication table of any number (e.g., 7) from 1 to 10.
2. Print only odd numbers from 15 to 1 in reverse order.
3. Write a program to check whether a number is a prime number.
4. Write a program to find the sum of digits of a given number.
5. Write a program to reverse a number and check if it is a palindrome.
6. Write a program to find the factorial of a number using a loop.
7. Write a program to print all Armstrong numbers between 1 and 1000.
8. Write a program to check whether a number is a perfect number.
9. Write a program to count how many times digit 3 appears in a number.
10. Write a program to print the sum of all even numbers between 1 and 50.