Using C++

**For Loop:** For loop is a programming language conditional iterative statement which is used to check for certain conditions and then repeatedly executes a block of code as long as those conditions are met. The for loop is distinguished from other looping statements through an explicit loop counter or loop variable which allows the body of the loop to know the exact sequencing of each iteration.

**Syntax:**

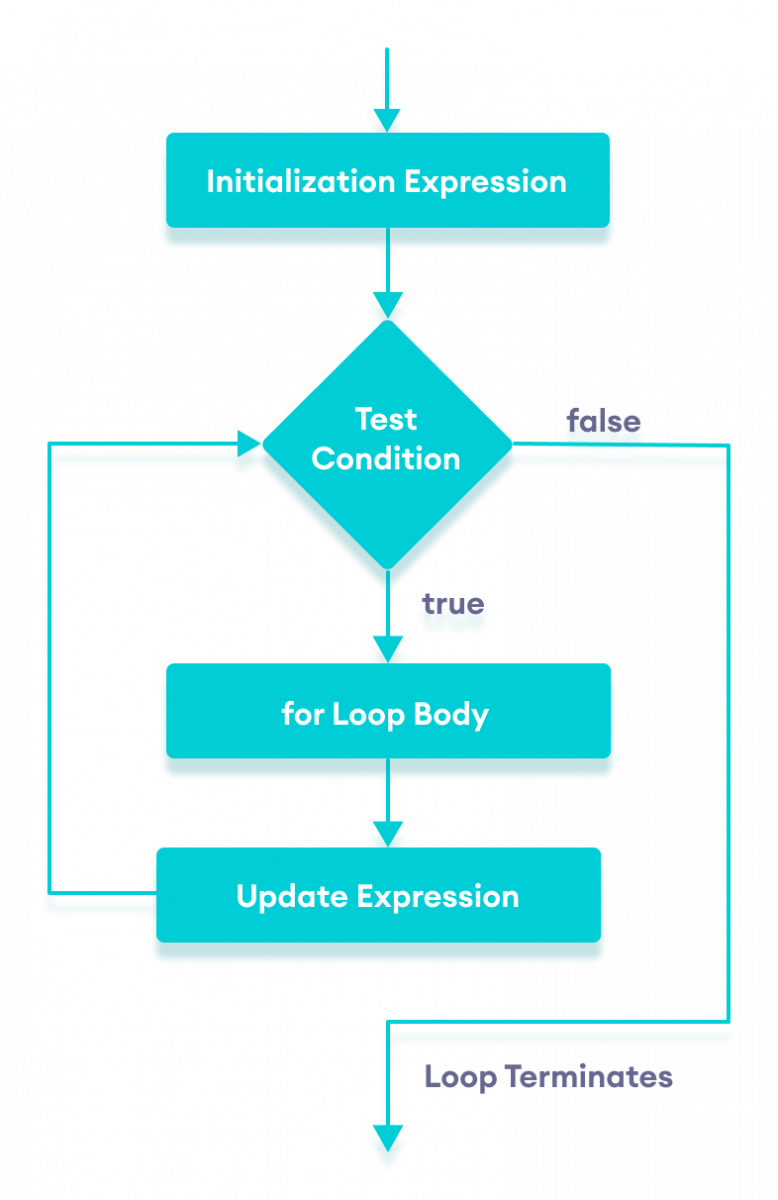
for (initialization; condition; update (increment/decrement)) {

// body of-loop

}

* Initialization – In this portion variable is initialized and is executed once.
* Condition – In this portion the condition is checked if the condition is satisfied (true) then the loop is executed else loop is not executed i.e. loop is terminated.
* update - updates the value of initialized variables and again checks the condition

Flow Chart of For Loop



Example:

#include <iostream>

using namespace std;

int main() {

for (int i = 1; i <= 3; ++i) {

cout << "Mahak Bhansali " << endl;

}

return 0;

}

Explanation: The above code will print the text 3 times

**Output:**

Mahak Bhansali

Mahak Bhansali

Mahak Bhansali

**While Loop:** It is an entry-controlled loop that is best suited for cases where you are not sure about the exact number of iterations. The while loop works on a boolean condition that regulates the iteration of the loop. It terminates the loop once it violates the condition. Since it is an entry-controlled loop, this loop checks if the statement is satisfied or not at the entry of the loop.

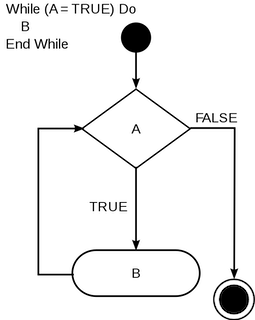
Syntax:

while (condition) {

   // Executable loop body

}

Flow of While Loop:



Example:

#include <iostream>

using namespace std;

int main()

{

int i=1;

while(i<=3)

{cout<<"Mahak Bhansali"<<endl;

i++;

}

return 0;

}

Explanation: The above code will print the text 3 times.

**Output:**

Mahak Bhansali

Mahak Bhansali

Mahak Bhansali

**Continue:** In computer programming, the continue statement is used to skip the current iteration of the loop and the control of the program goes to the next iteration.

Example:

#include <iostream>

using namespace std;

int main() {

for (int i = 1; i <= 4; i++) {

// condition to continue

if (i = = 3) {

continue;

}

cout << i << endl;

}

return 0;

}

Explanation: When i is equal to 3, the continue statement skips the current iteration and starts the next iteration. Then, i become 4**,** and the condition is evaluated again. Hence, 4 and 5 are printed in the next two iterations.

**Output:**

1

2

4

**If Else:** The if...else statement is used to run one block of code under certain conditions and another block of code under different conditions.

There are three forms of if...else statements in C++.

1. if statement
2. if...else statement
3. if...else if...else statement

1) **if statement:** The if statement evaluates the condition inside the parentheses ( ).If the condition evaluates to true, the code inside the body of if is executed. If the condition evaluates to false, the code inside the body of if is skipped.

Syntax:

if (condition) {

// body of if statement

}

2) **if else :** The if statement can have an optional else clause. If the condition evaluates true,

* the code inside the body of if is executed
* the code inside the body of else is skipped from execution

If the condition evaluates false,

* the code inside the body of else is executed
* the code inside the body of if is skipped from execution

Syntax:

if (condition) {

// block of code if condition is true

}

else {

// block of code if condition is false

}

3) **if else…else if:** The if...else statement is used to execute a block of code among two alternatives. However, if we need to make a choice between more than two alternatives, we use the if...else if...else statement. If condition1 evaluates to true, the code block 1 is executed.

* If condition1 evaluates to false, then condition2 is evaluated.
* If condition2 is true, the code block 2 is executed.
* If condition2 is false, the code block 3 is executed.

**Syntax:**

if (condition1) {

// code block 1

}

else if (condition2){

// code block 2

}

else {

// code block 3

}

Example:

#include <iostream>

using namespace std;

int main() {

int number;

cout << "Enter an integer: ";

cin >> number;

if (number > 0) {

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

}

else if (number < 0) {

cout << "You entered a negative integer: " << number << endl;

}

else {

cout << "You entered 0." << endl;

}

cout << "This line is always printed.";

return 0;

}

Output:

Enter an integer: 1

You entered a positive integer: 1.

This line is always printed.

Enter an integer: -2

You entered a negative integer: -2.

This line is always printed.

Enter an integer: 0

You entered 0.

This line is always printed.

**Switch:** The switch statement allows us to execute a block of code among many alternatives. The expression is evaluated once and compared with the values of each case label.

* If there is a match, the corresponding code after the matching label is executed. For example, if the value of the variable is equal to constant2, the code after case constant2: is executed until the break statement is encountered.
* If there is no match, the code after default: is executed.

Syntax:

switch (expression) {

case constant1:

// code to be executed if

// expression is equal to constant1;

break;

case constant2:

// code to be executed if

// expression is equal to constant2;

break;

.

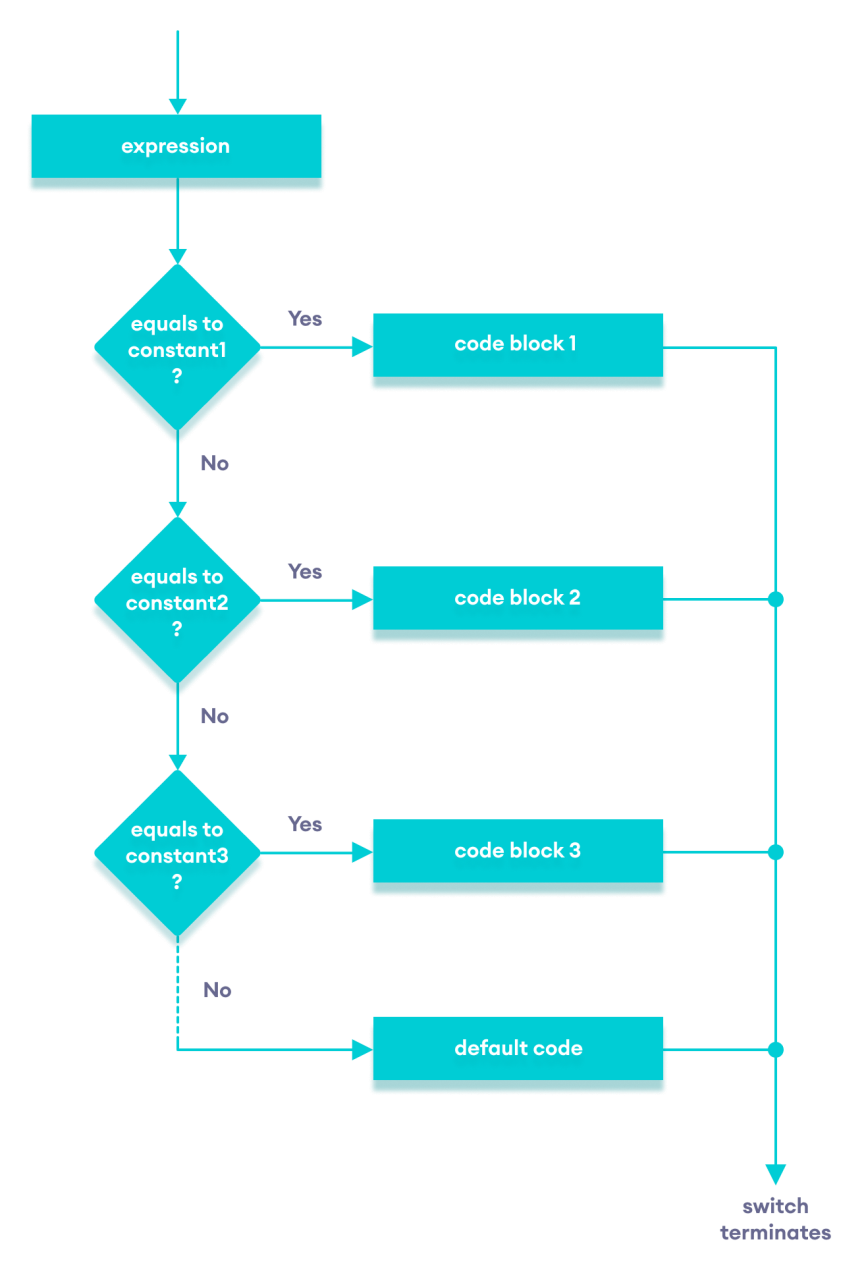
default:

// code to be executed if

// expression doesn't match any constant

}

Flow of Switch:



Example:

// Program to build a simple calculator using switch Statement

#include <iostream>

using namespace std;

int main() {

char oper;

int num1, num2;

cout << "Enter an operator (+, -, \*, /): ";

cin >> oper;

cout << "Enter two numbers: " << endl;

cin >> num1 >> num2;

switch (oper) {

case '+':

cout << num1 << " + " << num2 << " = " << num1 + num2;

break;

case '-':

cout << num1 << " - " << num2 << " = " << num1 - num2;

break;

case '\*':

cout << num1 << " \* " << num2 << " = " << num1 \* num2;

break;

case '/':

cout << num1 << " / " << num2 << " = " << num1 / num2;

break;

default:

// operator is doesn't match any case constant (+, -, \*, /)

cout << "Error! The operator is not correct";

break;

}

return 0;

}

Output:

Enter an operator (+, -, \*, /): +

Enter two numbers:

2

4

2 + 4 = 6

Enter an operator (+, -, \*, /): \*

Enter two numbers:

2

4

2 \* 4 = 8

**Recursion:** A function that calls itself is known as a recursive function. And, this technique is known as recursion.

Syntax:

void recurse()

{

recurse();

}

int main()

{

recurse();

}

Example:

#include <iostream>

using namespace std;

int factorial(int);

int main() {

int n, result;

cout << "Enter a non-negative number: ";

cin >> n;

result = factorial(n);

cout << "Factorial of " << n << " = " << result;

return 0;

}

int factorial(int n) {

if (n > 1) {

return n \* factorial(n - 1);

} else {

return 1;

}

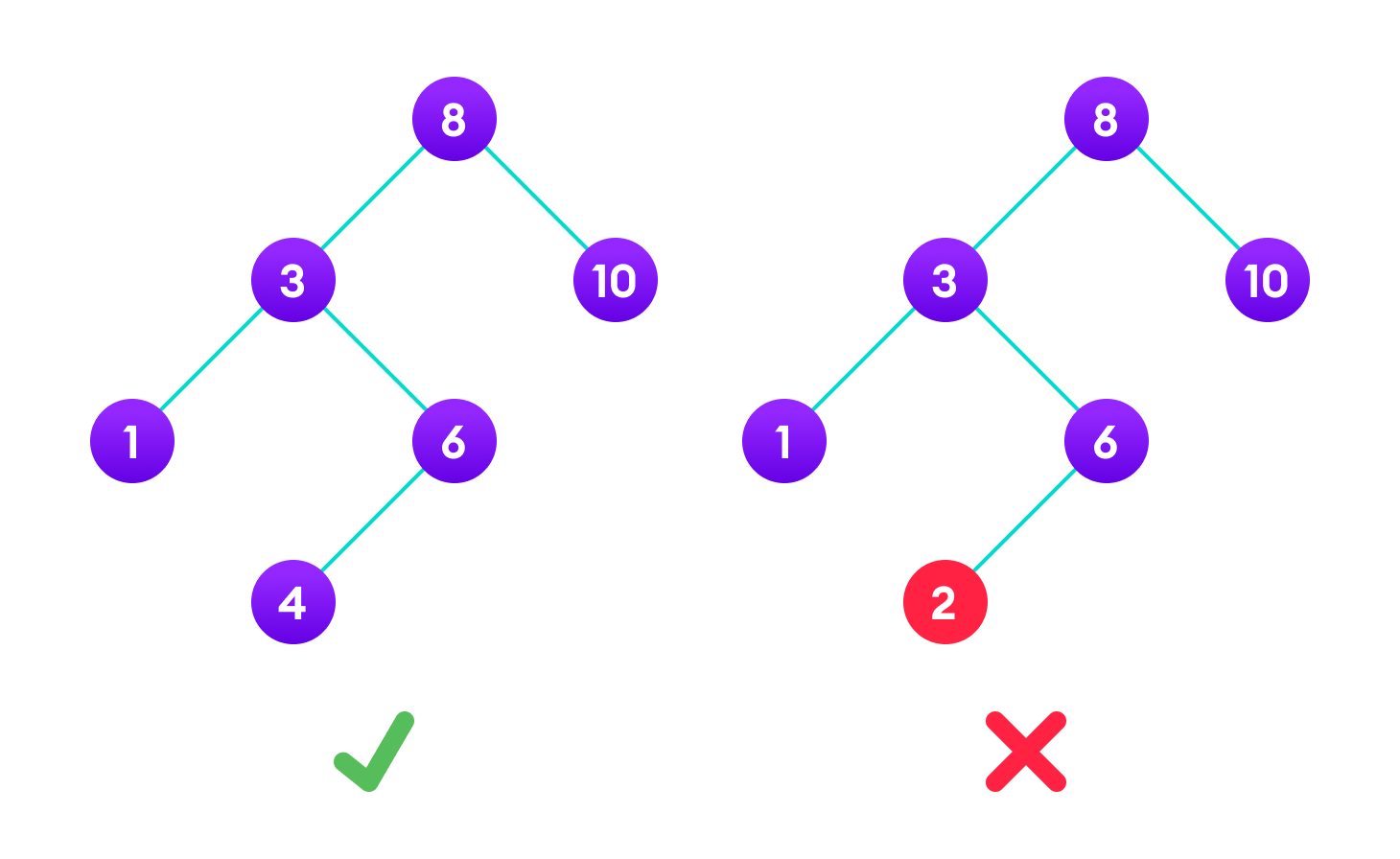
}

Output:

Enter a non-negative number: 3

Factorial of 3 = 6

**Binary Search Tree:** Binary search tree is a data structure that quickly allows us to maintain a sorted list of numbers. It is called a binary tree because each tree node has a maximum of two children. It is called a search tree because it can be used to search for the presence of a number in O(log(n)) time.



Closure: A closure is a technique for implementing lexically scoped name binding in a language with first-class functions. Operationally, a closure is a record storing a functiontogether with an environment.