Lab-1

To write and run C++ programs, you need to set up the local environment on your computer. Refer to the complete article Setting up C++ Development Environment. If you do not want to set up the local environment on your computer, you can also use online IDE to write and run your C++ programs.

```
// C++ program to display "Hello World"


// Header file for input output functions
#include <iostream>
using namespace std;

// Main() function: where the execution of
// program begins
int main()
{
	// Prints hello world
	cout << "Hello World";

	return 0;
}
```

Important Points

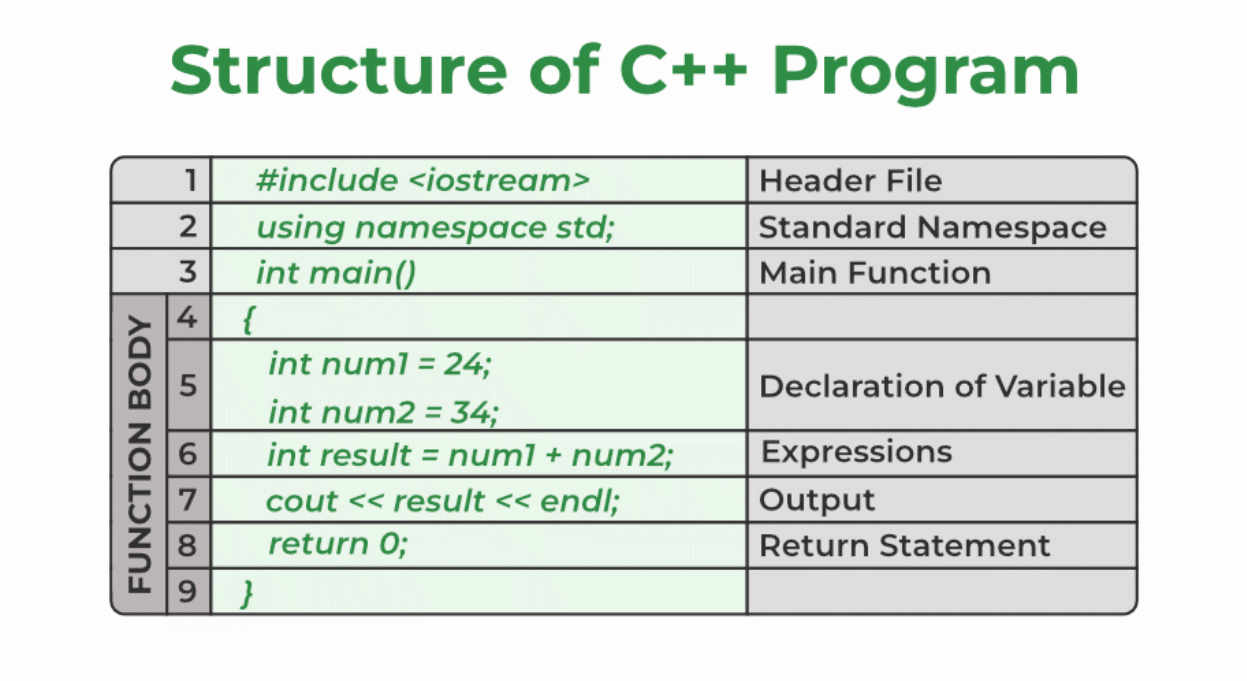
1. Always include the necessary header files for the smooth execution of functions. For example, **<iostream>** must be included to use **std::cin** and **std::cout**.
2. The execution of code begins from the **main()** function.
3. It is a good practice to use **Indentation** and **comments** in programs for easy understanding.
4. **cout** is used to print statements and **cin** is used to take inputs.

What is Syntax?

Syntax refers to the rules and regulations for writing statements in a programming language. They can also be viewed as the grammatical rules defining the structure of a programming language.

Basic Syntax of a C++ Program

We can learn about basic C++ Syntax using the following program.



1. Header File

The header files contain the definition of the functions and macros we are using in our program. They are defined on the top of the C++ program.

In **line #1**, we used the **#include <iostream>** statement to tell the compiler to include an **iostream** header file library which stores the definition of the cin and cout methods that we have used for input and output. **#include** is a preprocessor directive using which we import header files.

Syntax:

#include <*library_name*>

2. Namespace

A namespace in C++ is used to provide a scope or a region where we define identifiers. It is used to avoid name conflicts between two identifiers as only unique names can be used as identifiers.

In **line #2**, we have used the **using namespace std** statement for specifying that we will be the standard namespace where all the standard library functions are defined.
Syntax:
using namespace std;

3. Main Function

Functions are basic building blocks of a C++ program that contains the instructions for performing some specific task. Apart from the instructions present in its body, a function definition also contains information about its return type and parameters. To know more about C++ functions, please refer to the article Functions in C++.
In **line #3**, we defined the main function as **int main().** The main function is the most important part of any C++ program. The program execution always starts from the main function. All the other functions are called from the main function. In C++, the main function is required to return some value indicating the execution status.

Syntax:

```
int main() {


   ... code ....
   return 0;
}
```

4. Blocks

Blocks are the group of statements that are enclosed within **{ }** braces. They define the scope of the identifiers and are generally used to enclose the body of functions and control statements. The body of the main function is from **line #4** to **line #9** enclosed within **{ }.**
Syntax:

```
{

  // Body of the Function

   return 0;
}
```

5. Semicolons

As you may have noticed by now, each statement in the above code is followed by a (**;**) semicolon symbol. It is used to terminate each line of the statement of the program. When the compiler sees this semicolon, it terminates the operation of that line and moves to the next line.

Syntax:
any_statement **;**

6. Identifiers

We use identifiers for the naming of variables, functions, and other user-defined data types. An identifier may consist of uppercase and lowercase alphabetical characters, underscore, and digits. The first letter must be an underscore or an alphabet.

Example:
int **num1** = 24;
int **num2** = 34;
num1 & num2 are the identifiers and int is the data type.

7. Keywords

In the C++ programming language, there are some reserved words that are used for some special meaning in the C++ program. It can't be used for identifiers.

For example, the words **int, return, and using** are some keywords used in our program. These all have some predefined meaning in the C++ language.

There are total 95 keywords in C++. These are some keywords.

int void if while for auto bool break

this static new true false case char class

8. Basic Output cout

In **line #7,** we have used the **cout** method which is the basic output method in C++ to output the sum of two numbers in the standard output stream (stdout).
Syntax:
cout << result << endl;

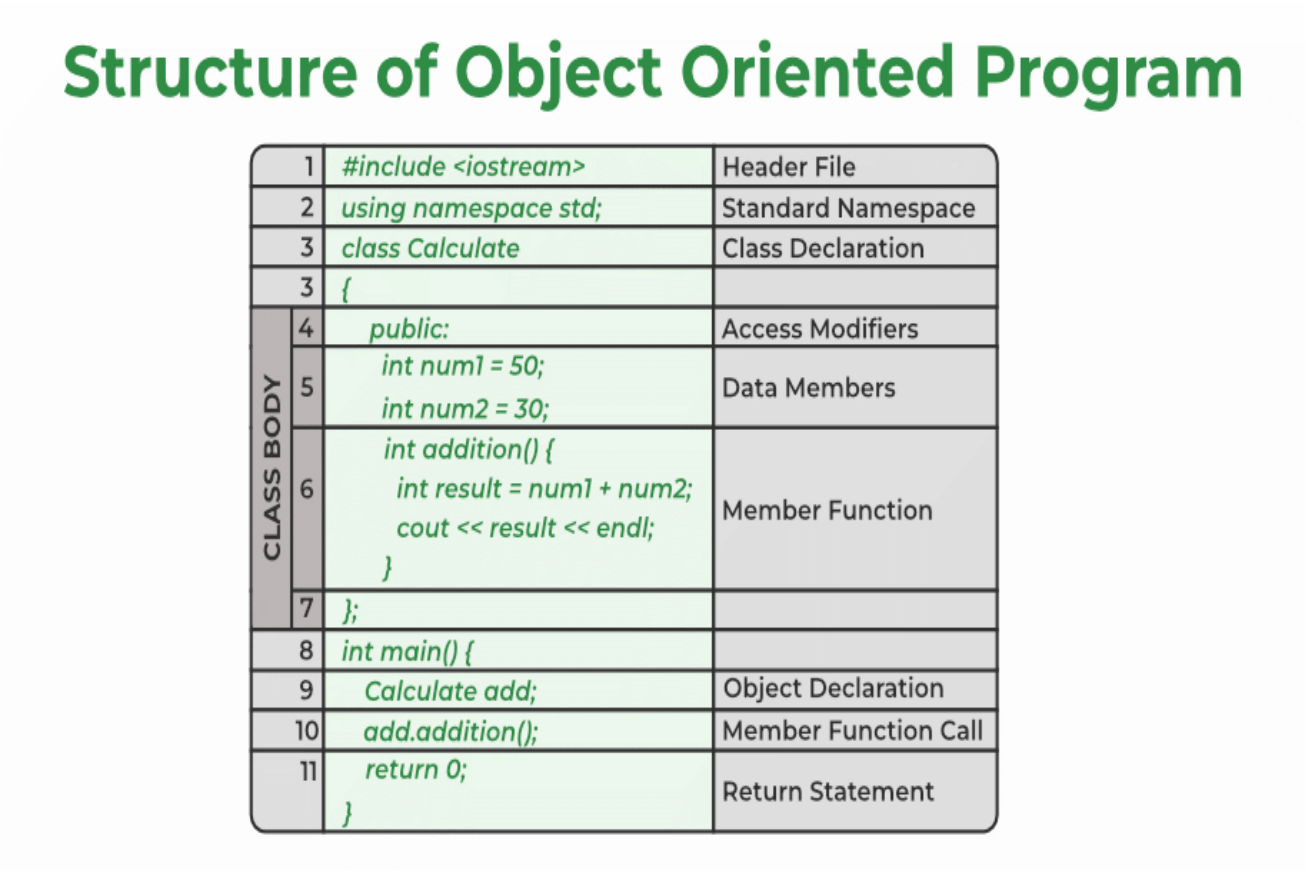
Now, we have a better understanding of the basic syntax structure of the above C++ program. Let's try to execute this program and see if it works correctly.

```
// C++ program to demonstrate the basic syntax
// Header File Library
#include <iostream>
```

```
// Standard Namespace
using namespace std;

// Main Function
int main()
{

	// Body of the Function

	// Declaration of Variable
	int num1 = 24;
	int num2 = 34;

	int result = num1 + num2;

	// Output
	cout << result << endl;

	// Return Statement
	return 0;
}
```

Object-Oriented Programming in C++

C++ programming language supports both procedural-oriented and object-oriented programming. The above example is based on the procedural-oriented programming paradigm.



1. Class

A class is a template of an object. For example, the animal is a class & dog is the object of the animal class. It is a user-defined data type. A class has its own attributes (data members) and behavior (member functions). The first letter of the class name is always capitalized & use the ***class*** keyword for creating the class.

Syntax:

class *class_name*{

// class body

};

2. Data Members & Member Functions

The attributes or data in the class are defined by the data members & the functions that work on these data members are called the member functions.

Example:

```
class Calculate{

   public:
      int num1 = 50;    // data member
      int num2 = 30;    // data member

      // member function
      int addition() {
         int result = num1 + num2;
         cout << result << endl;
      }
};
```

In the above example, num1 and num2 are the data member & addition() is a member function that is working on these two data members. There is a keyword here **public** that is **access modifiers**. The access modifier decides who has access to these data members & member functions. **public** access modifier means these data members & member functions can get access by anyone.

3. Object

The object is an instance of a class. The class itself is just a template that is not allocated any memory. To use the data and methods defined in the class, we have to create an object of that class.

They are declared in the similar way we declare variables in C++.

Syntax:
class_name object_name;

We use dot operator (**.**) for accessing data and methods of an object.

Now, let's execute our code to see the output of the program.

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

class Calculate{

        // Access Modifiers
        public:
                // data member
                int num1 = 50;
                int num2 = 30;
```

```
                // memeber function
                int addition() {
                        int result = num1 + num2;
                        cout << result << endl;
                }
};

int main() {

        // object declaration
        Calculate add;
        // member function calling
        add.addition();

        return 0;
}
```

C++ Comments

Comments in C++ are meant to explain the code as well as to make it more readable. When testing alternative code, it can also be used to prevent execution. The purpose of the comments is to provide information about code lines. Programmers commonly use comments to document their work.

Why Comments are used in C++?

Comments in C++ are used to summarize an algorithm, identify a variable's purpose, or clarify a code segment that appears unclear. Comments are also used for:

- Comments are used for easier debugging.
- It makes a program more readable and gives an overall description of the code.
- Comments are helpful in skipping the execution of some parts of the code.
- Every time a program or code is reused after long periods of time, the comment recaps all the information of the code quickly.

Types of Comments in C++

In C++ there are two types of comments:

- **Single-line comment**
- **Multi-line comment**

C++ Keywords

Token

When the compiler is processing the source code of a C++ program, each group of characters separated by white space is called a token. Tokens are the smallest individual units in a program. A C++ program is written using tokens. It has the following tokens:

- Keywords
- Identifiers
- Constants
- Strings
- Operators

C++ Keyword			
asm	double	new	switch
auto	else	operator	template
break	enum	private	this
case	extern	protected	throw
catch	float	public	try
char	for	register	typedef
class	friend	return	union

const	goto	short	unsigned
continue	if	signed	virtual
default	inline	sizeof	void
delete	int	static	volatile
do	long	struct	while

C++ Variables

Variables in C++ is a name given to a memory location. It is the basic unit of storage in a program.

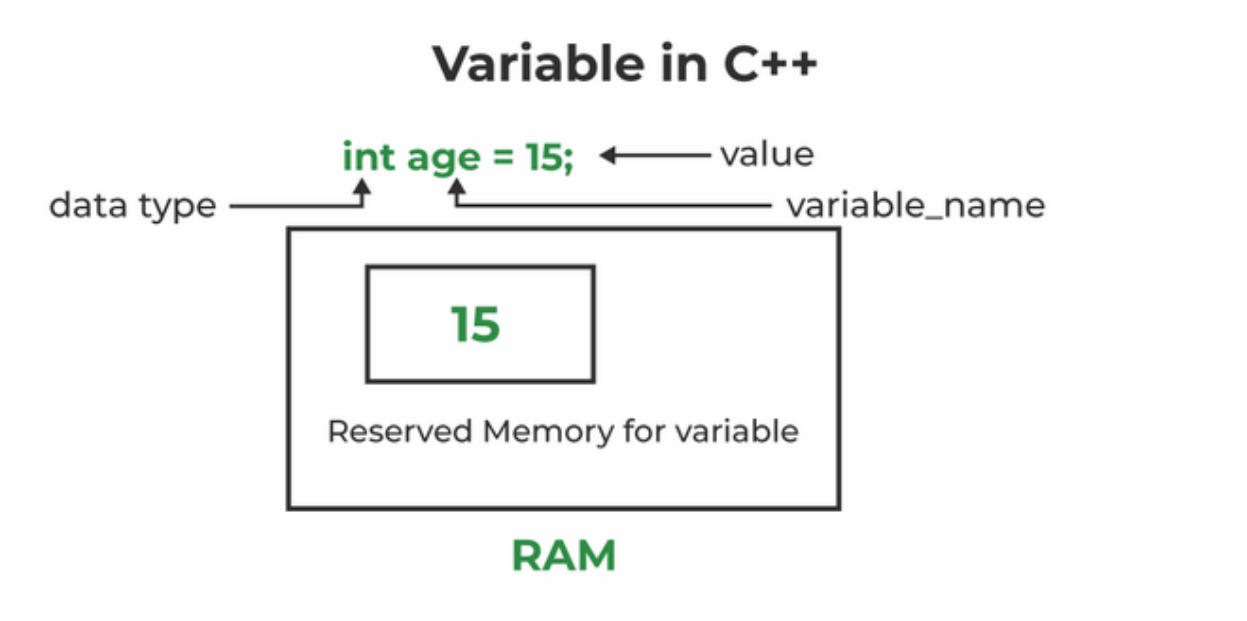
- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
- In C++, all the variables must be declared before use.

How to Declare Variables?

A typical variable declaration is of the form:

```
// Declaring a single variable

type variable_name;


// Declaring multiple variables:

type variable1_name, variable2_name, variable3_name;
```

A variable name can consist of alphabets (both upper and lower case), numbers, and the underscore '_' character. However, the name must not start with a number.



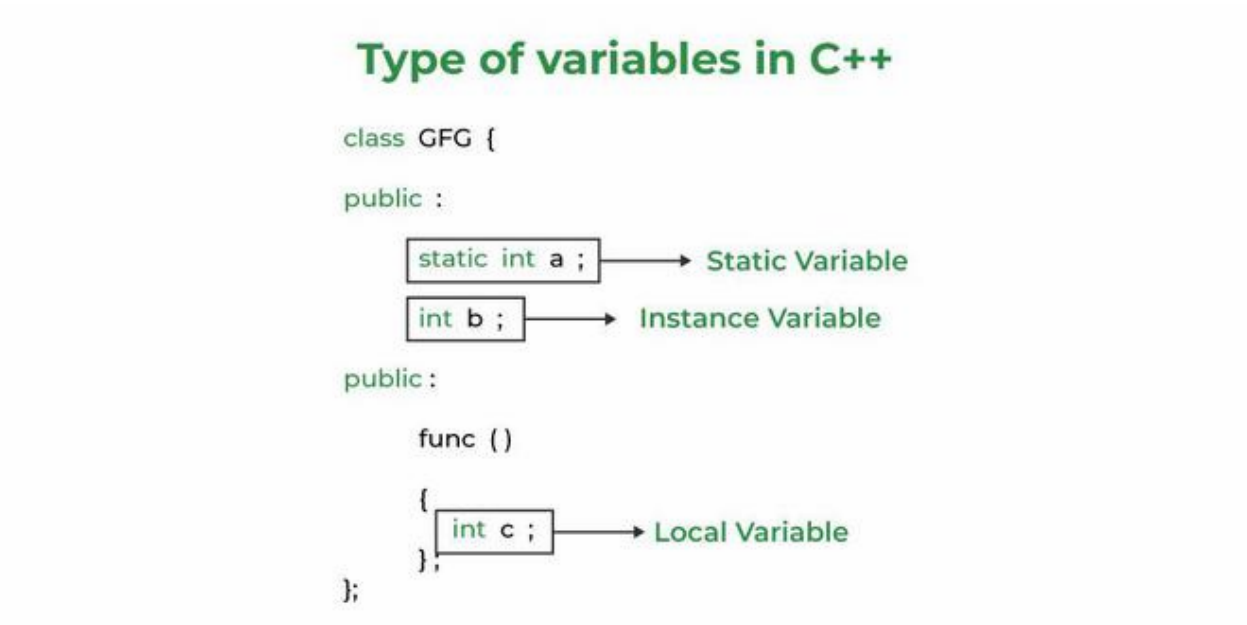
Rules For Declaring Variable

- The name of the variable contains letters, digits, and underscores.
- The name of the variable is case sensitive (ex Arr and arr both are different variables).
- The name of the variable does not contain any whitespace and special characters (ex #,$,%,*, etc).
- All the variable names must begin with a letter of the alphabet or an underscore(_).
- We cannot used C++ keyword(ex float,double,class)as a variable name.

Types of Variables

There are three types of variables based on the scope of variables in C++

- **Local Variables**
- **Instance Variables**
- **Static Variables**



Scope of Variables in C++

In general, the scope is defined as the extent up to which something can be worked with. In programming also the scope of a variable is defined as the extent of the program code within which the variable can be accessed or declared or worked with. There are mainly two types of variable scopes:

1. Local Variables
2. Global Variables

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

// global variable
int global = 5;

// main function
int main()
{
    // local variable with same
    // name as that of global variable
    int global = 2;

    cout << global << endl;
}
```

Local variable

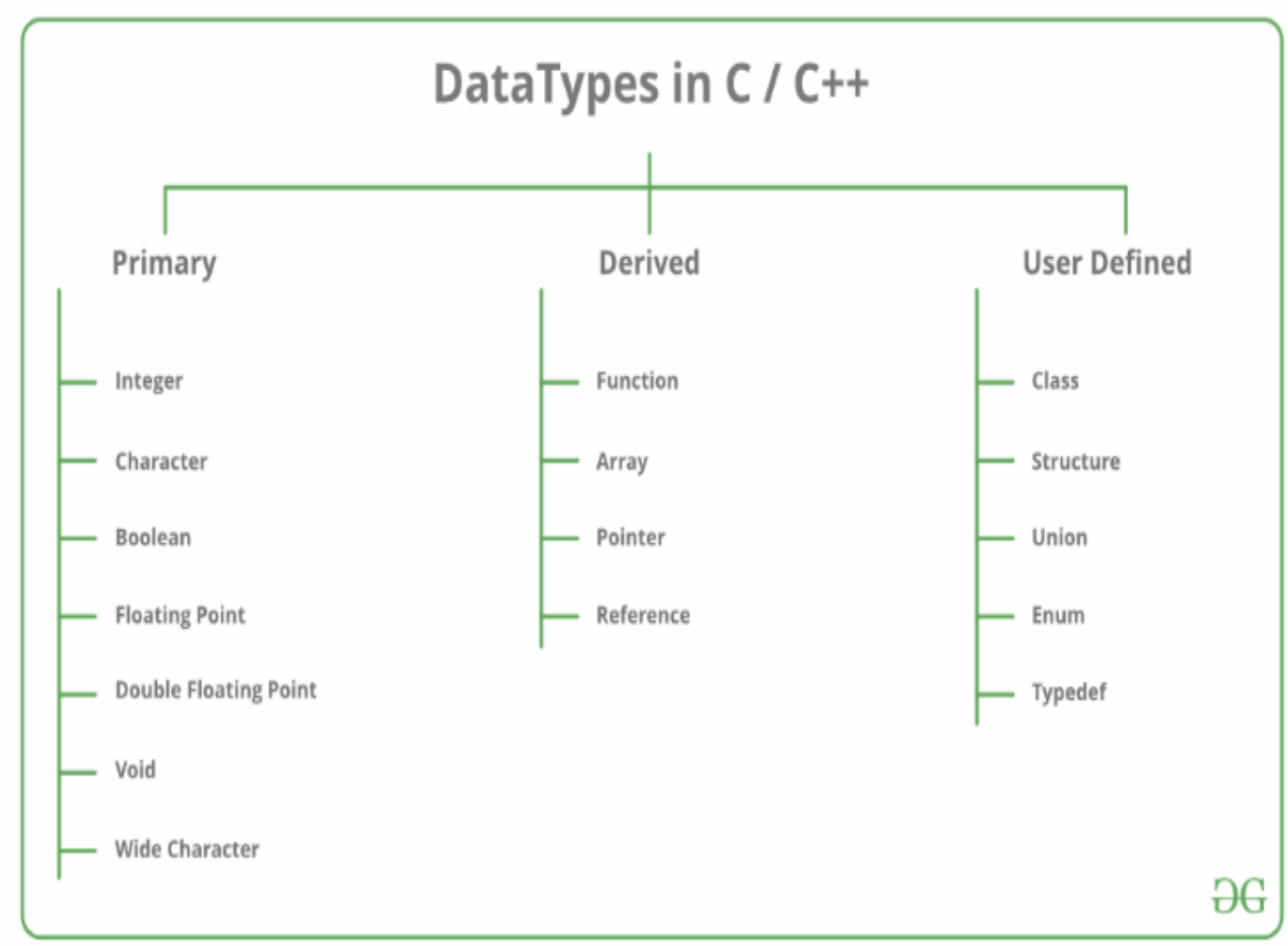
// CPP program to illustrate
// usage of local variables
#include<iostream>

```
using namespace std;

void func()
{
	// this variable is local to the
	// function func() and cannot be
	// accessed outside this function
	int age=18;
	cout<<age;
}

int main()
{
	cout<<"Age is: ";
	func();

	return 0;
}


// CPP program to illustrate
// usage of global variables
#include<iostream>
using namespace std;

// global variable
int global = 5;

// global variable accessed from
// within a function
void display()
{
	cout<<global<<endl;
}

// main function
int main()
{
	display();

	// changing value of global
	// variable from main function
	global = 10;
	display();
}
```

C++ Data Types

C++ supports the following data types:

1. **Primary** or **Built-in** or **Fundamental data type**
1. **Derived data types**
1. **User-defined data types**



Operators in C++

An **operator** is a symbol that operates on a value to perform specific mathematical or logical computations. They form the foundation of any programming language. In C++, we have built-in operators to provide the required functionality.

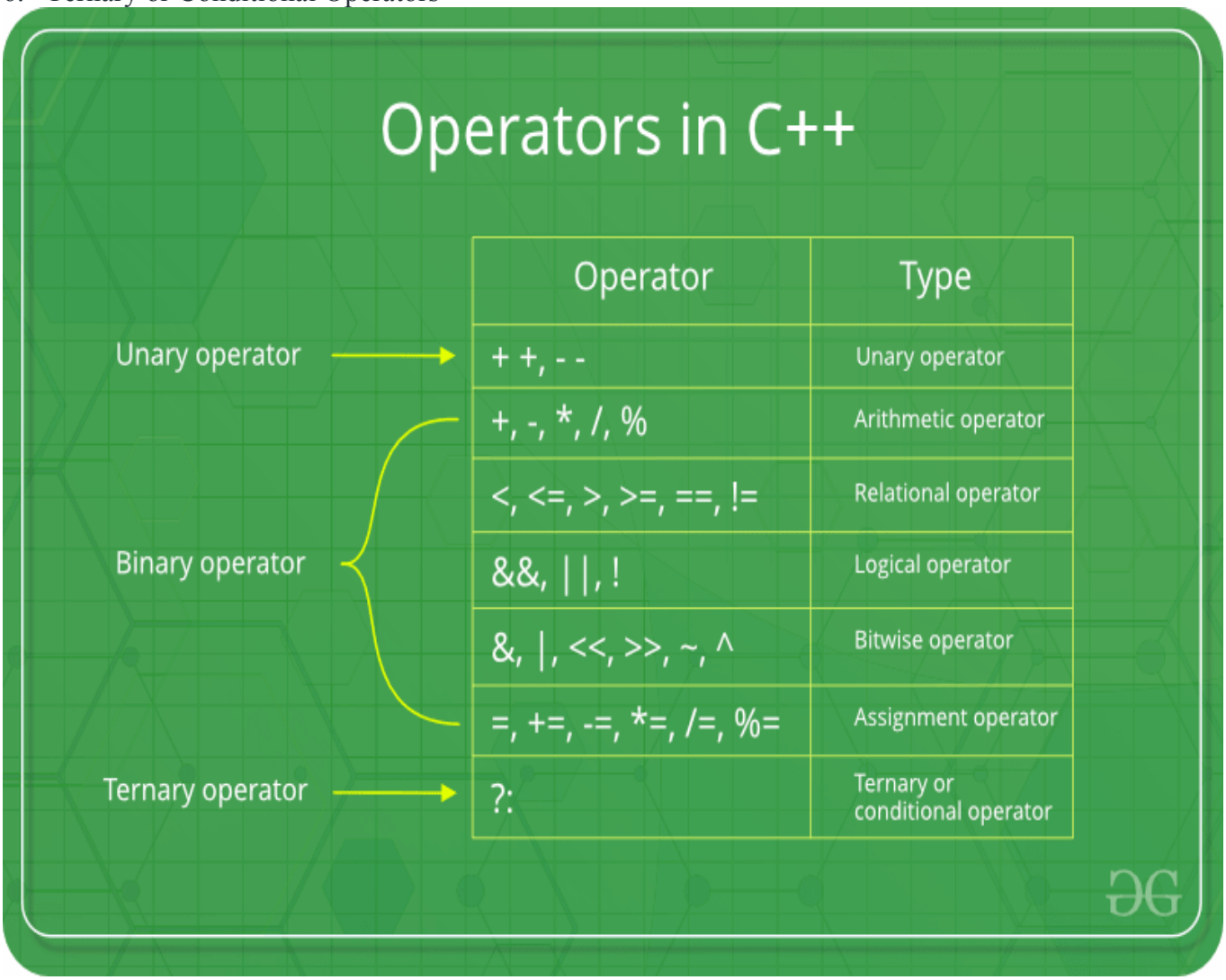
An operator operates the **operands**. For example,

int c = a + b;

Here, '+' is the addition operator. 'a' and 'b' are the operands that are being 'added'.

Operators in C++ can be classified into 6 types:

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Ternary or Conditional Operators




```
// CPP Program to demonstrate the Binary Operators
#include <iostream>
using namespace std;

int main()
{
        int a = 8, b = 3;
```

```
        // Addition operator
        cout << "a + b = " << (a + b) << endl;

        // Subtraction operator
        cout << "a - b = " << (a - b) << endl;

        // Multiplication operator
        cout << "a * b = " << (a * b) << endl;

        // Division operator
        cout << "a / b = " << (a / b) << endl;

        // Modulo operator
        cout << "a % b = " << (a % b) << endl;

        return 0;
}
```

```
// CPP Program to demonstrate the Relational Operators
#include <iostream>
using namespace std;

int main()
{
        int a = 6, b = 4;

        // Equal to operator
        cout << "a == b is " << (a == b) << endl;

        // Greater than operator
        cout << "a > b is " << (a > b) << endl;

        // Greater than or Equal to operator
        cout << "a >= b is " << (a >= b) << endl;

        // Lesser than operator
        cout << "a < b is " << (a < b) << endl;

        // Lesser than or Equal to operator
        cout << "a <= b is " << (a <= b) << endl;

        // true
```

```
        cout << "a != b is " << (a != b) << endl;

        return 0;
}

// CPP Program to demonstrate the Logical Operators
#include <iostream>
using namespace std;

int main()
{
        int a = 6, b = 4;

        // Logical AND operator
        cout << "a && b is " << (a && b) << endl;

        // Logical OR operator
        cout << "a ! b is " << (a > b) << endl;

        // Logical NOT operator
        cout << "!b is " << (!b) << endl;

        return 0;
}

// CPP Program to demonstrate the Bitwise Operators
#include <iostream>
using namespace std;

int main()
{
        int a = 6, b = 4;

        // Binary AND operator
        cout << "a & b is " << (a & b) << endl;

        // Binary OR operator
        cout << "a | b is " << (a | b) << endl;

        // Binary XOR operator
        cout << "a ^ b is " << (a ^ b) << endl;

        // Left Shift operator
        cout << "a<<1 is " << (a << 1) << endl;
```

```
    // Right Shift operator
    cout << "a>>1 is " << (a >> 1) << endl;

    // One's (two's) Complement operator
    cout << "~(a) is " << ~(a) << endl;

    return 0;
}
```

Explanation:

Let's go through each operation:

1. **Binary AND operator (`a & b`):**
   - The **AND** operator compares each bit of `a` and `b` and returns `1` if both bits are `1`; otherwise, it returns `0`.
   - In binary:

$$a = 6 \rightarrow 0110 \quad (in\ 4\ bits)$$

$$b = 4 \rightarrow 0100 \quad (in\ 4\ bits)$$

     Applying the AND operation:

$$0110\ \&\ 0100 = 0100 \ \rightarrow\ 4$$

   - **Output**: `a & b is 4`
2. **Binary OR operator (`a | b`):**
   - The **OR** operator compares each bit of `a` and `b` and returns `1` if at least one of the bits is `1`.
   - Applying the OR operation:

$$0110\ |\ 0100 = 0110 \ \rightarrow\ 6$$

   - **Output**: `a | b is 6`

3. **Binary XOR operator (`a ^ b`):**
   - The **XOR** operator compares each bit of `a` and `b` and returns `1` if the bits are different; otherwise, it returns `0`.
   - Applying the XOR operation:
   $$0110 \ \wedge \ 0100 = 0010 \ \rightarrow \ 2$$
   - **Output**: `a ^ b is 2`
4. **Left Shift operator (`a << 1`):**
   - The **left shift** operator shifts all bits of `a` one position to the left, and `0` fills the vacated bit on the right.
   - Shifting `0110` one position to the left:
   $$0110 << 1 = 1100 \ \rightarrow \ 12$$
   - **Output**: `a<<1 is 12`
5. **Right Shift operator (`a >> 1`):**
   - The **right shift** operator shifts all bits of `a` one position to the right, discarding the rightmost bit and filling the leftmost bit with `0`.
   - Shifting `0110` one position to the right:
   $$0110 >> 1 = 0011 \ \rightarrow \ 3$$
   - **Output**: `a>>1 is 3`
6. **One's Complement operator (`~a`):**
   - The **one's complement** operator inverts all bits of `a`.
   - For a 4-bit representation:
   $$a = 6 \rightarrow 0000 \ 0110$$
   Applying the one's complement operator:
   $$0000 \ 0110 = 1111 \ 1001 \quad (in \ 8 \ bits)$$
   - This result is the two's complement form of `-7`.
   - **Output**: `~(a) is -7`

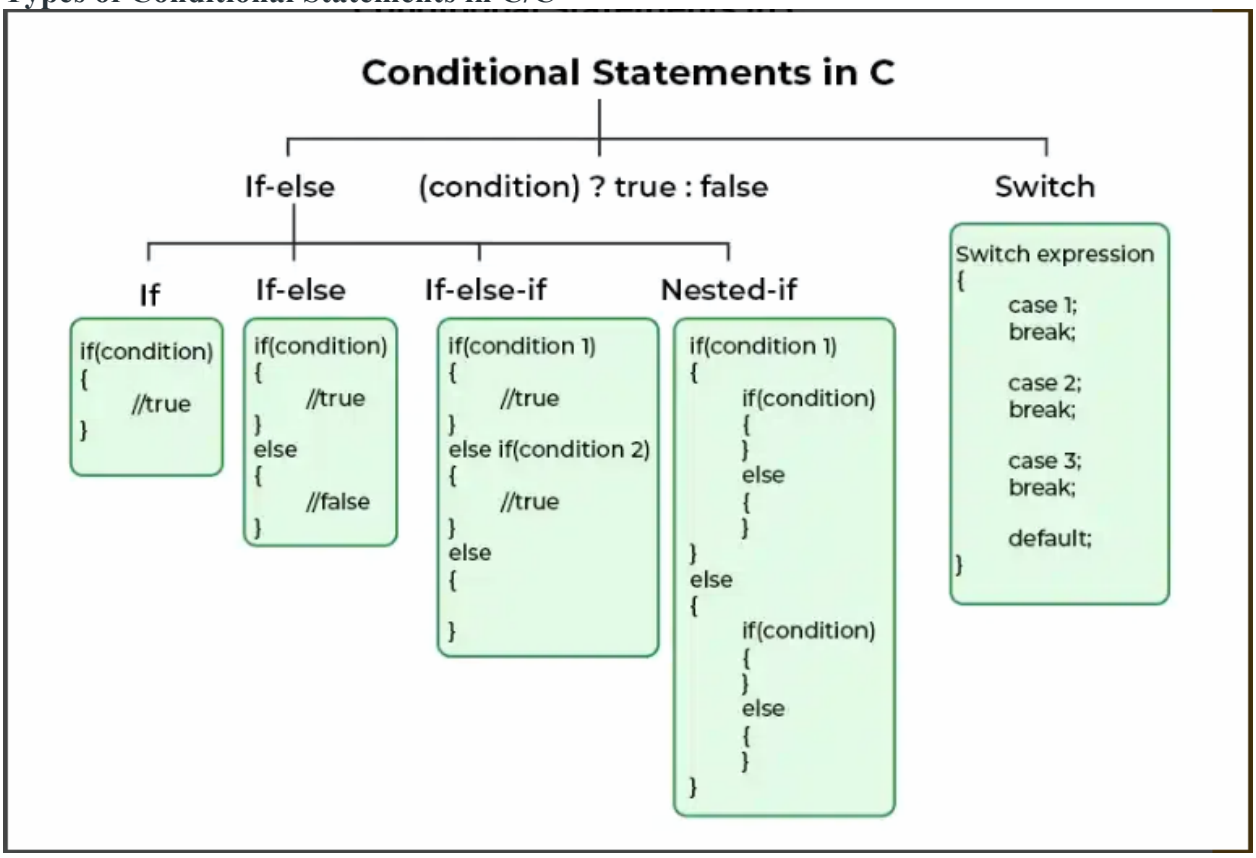
Control Statements

Decision Making in C / C++ (if , if..else, Nested if, if-else-if)

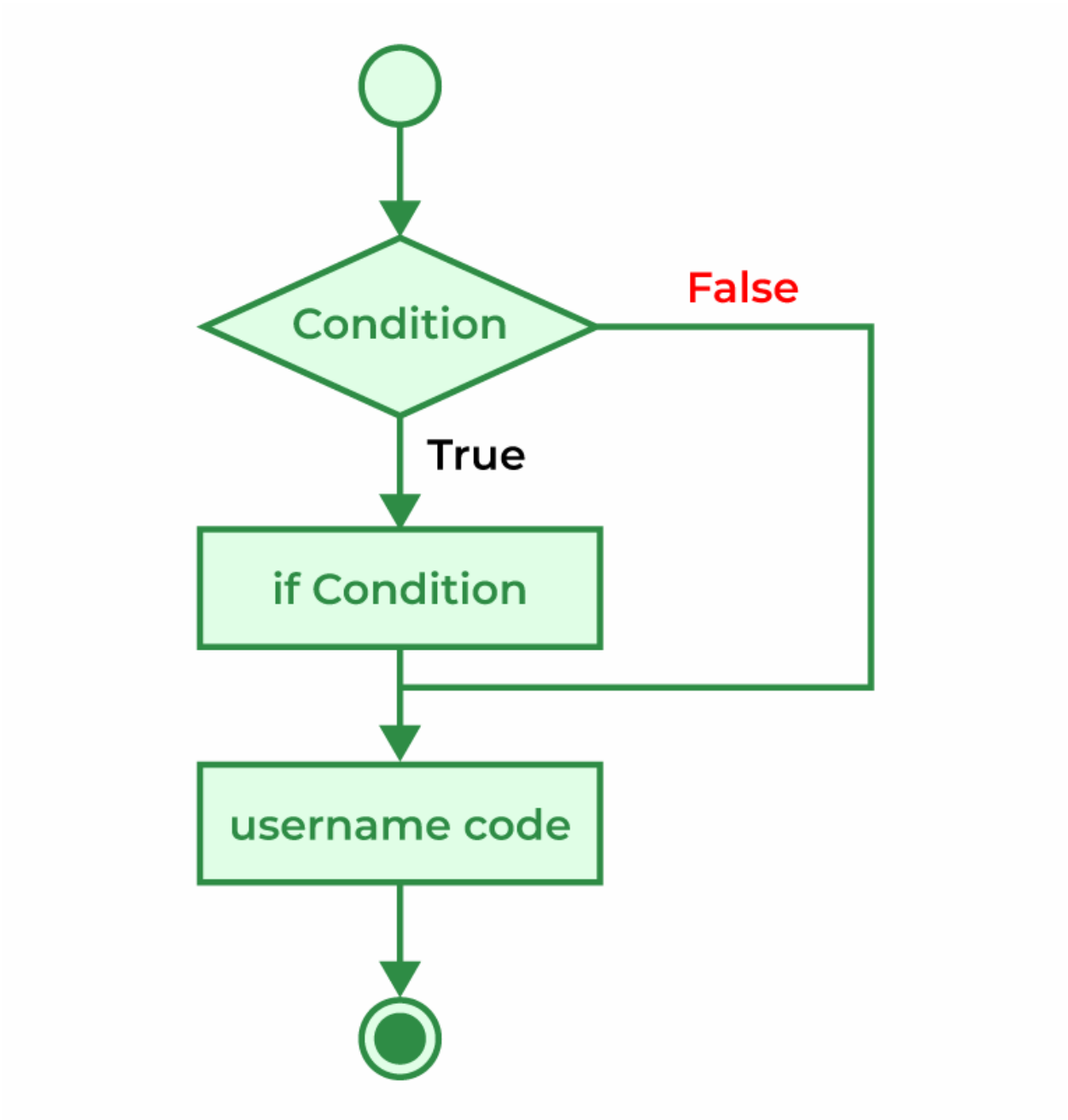
Need of Conditional Statements

There come situations in real life when we need to make some decisions and based on these decisions, we decide what should we do next. Similar situations arise in programming also where we need to make some decisions and based on these decisions we will execute the next block of code. For example, in C if x occurs then execute y else execute z. There can also be multiple conditions like in C if x occurs then execute p, else if condition y occurs execute q, else execute r. This condition of C else-if is one of the many ways of importing multiple conditions.

Types of Conditional Statements in C/C++



Flowchart of if Statement

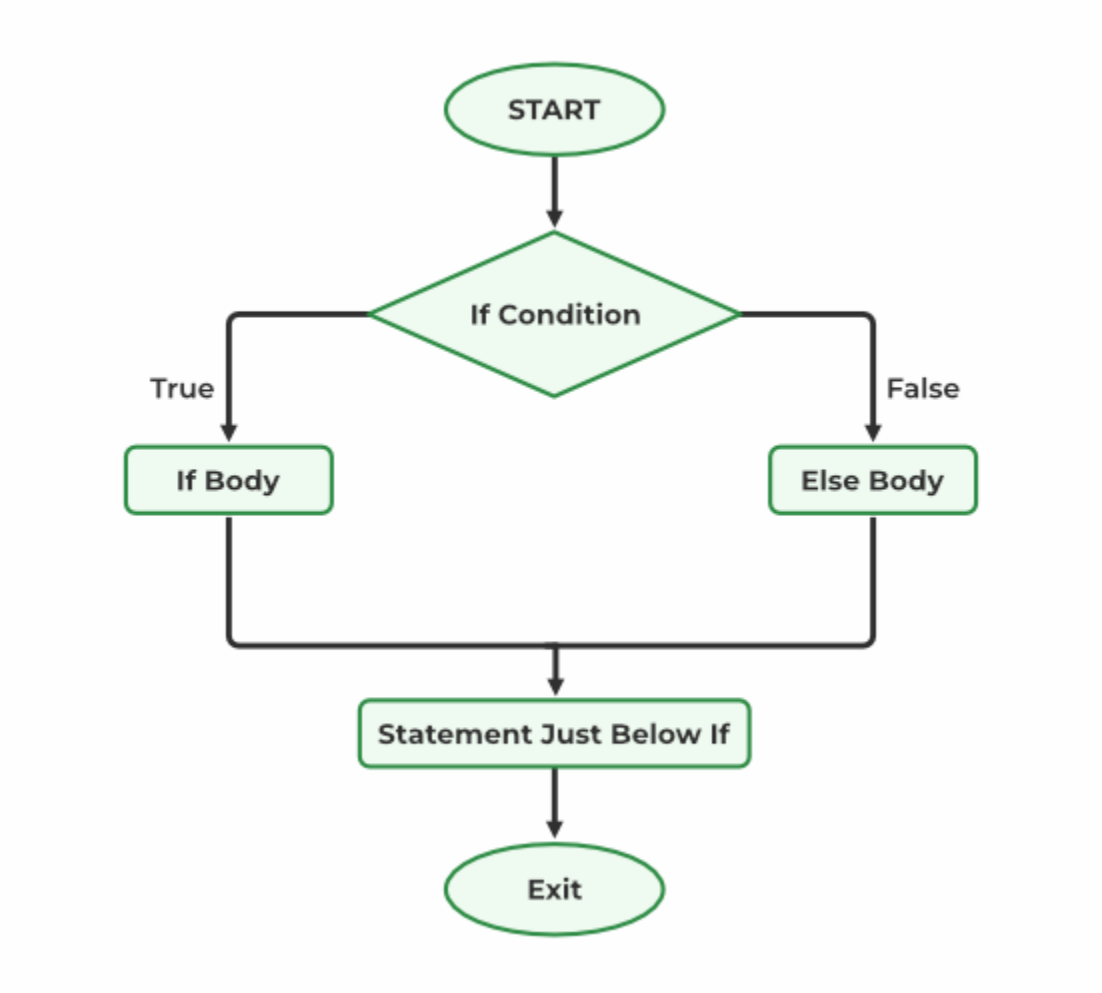


Example of if in C/C++

```
// C++ program to illustrate If statement
#include <iostream>
using namespace std;

int main()
```

```
{
        int i = 10;

        if (i > 15) {
                cout << "10 is greater than 15";
        }

        cout << "I am Not in if";
}
```

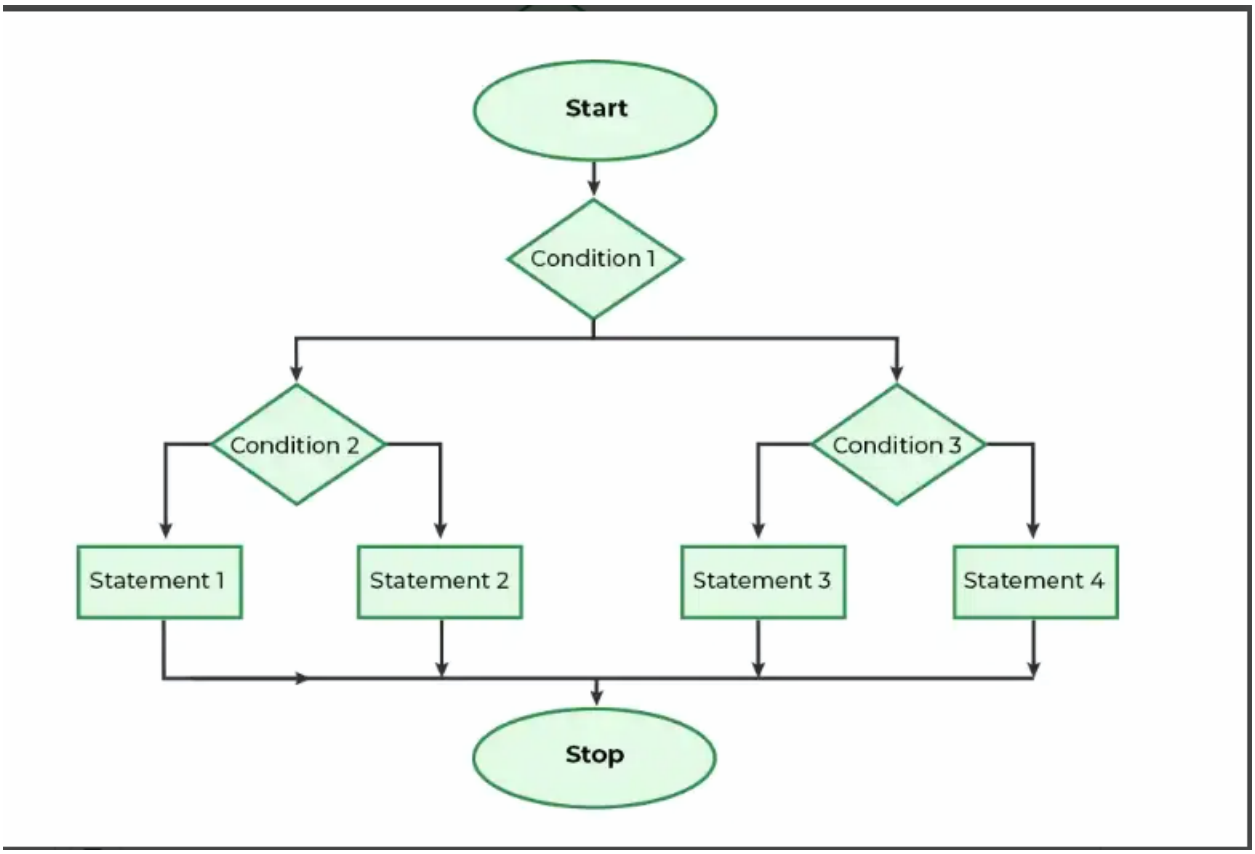
Flowchart of if-else Statement




```
// C++ program to illustrate if-else statement
#include <iostream>
using namespace std;

int main()
```

```
{
        int i = 20;

        if (i < 15)
                cout << "i is smaller than 15";
        else
                cout << "i is greater than 15";

        return 0;
}
```

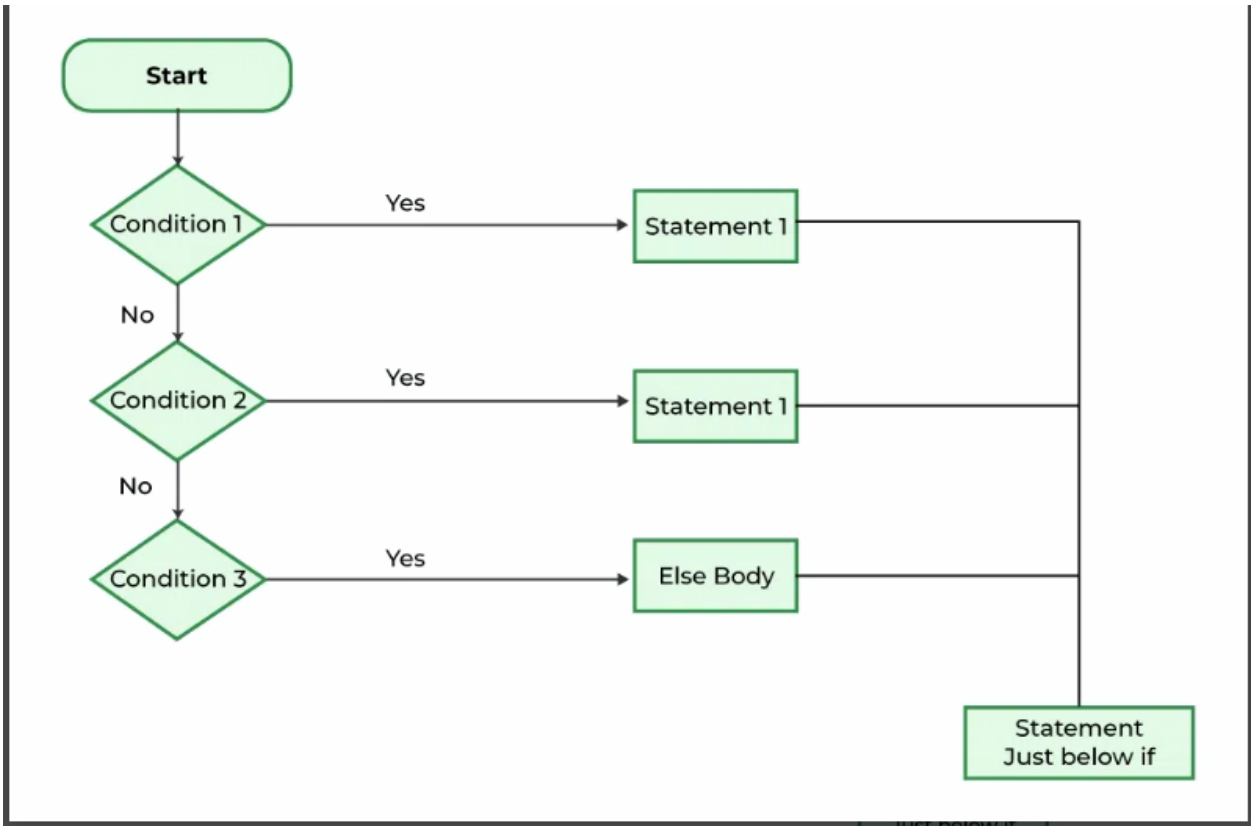
3. Nested if-else in C/C++




```
// C++ program to illustrate nested-if statement
#include <iostream>
using namespace std;

int main()
{
        int i = 10;
```

```
        if (i == 10) {
                // First if statement
                if (i < 15)
                        cout << "i is smaller than 15\n";

                // Nested - if statement
                // Will only be executed if
                // statement above is true
                if (i < 12)
                        cout << "i is smaller than 12 too\n";
                else
                        cout << "i is greater than 15";
        }

        return 0;
}
```

4. if-else-if Ladder in C/C++

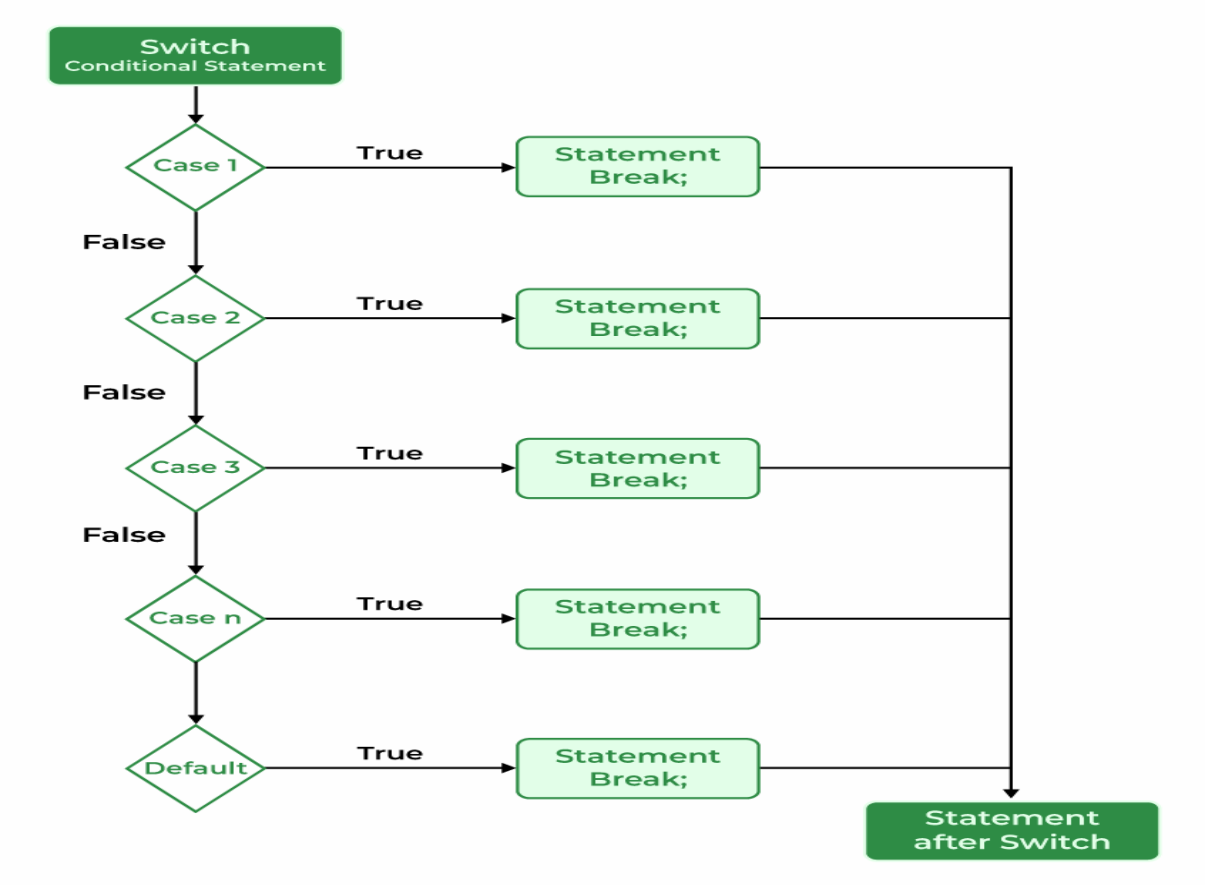



```
// C++ program to illustrate if-else-if ladder
#include <iostream>
using namespace std;
```

```
int main()
{
        int i = 20;

        if (i == 10)
                cout << "i is 10";
        else if (i == 15)
                cout << "i is 15";
        else if (i == 20)
                cout << "i is 20";
        else
                cout << "i is not present";
}
```

5. switch Statement in C/C++

The switch case statement is an alternative to the if else if ladder that can be used to execute the conditional code based on the value of the variable specified in the switch statement. The switch block consists of cases to be executed based on the value of the switch variable.



```
// C++ program to demonstrate syntax of switch
#include <iostream>
using namespace std;

// Driver Code
int main()
{
	// switch variable
	char x = 'A';

	// switch statements
	switch (x) {
	case 'A':
		cout << "Choise is A";
		break;
	case 'B':
		cout << "Choise is B";
		break;
	case 'C':
		cout << "Choise is C";
		break;
	default:
		cout << "Choice other than A, B and C";
		break;
	}
	return 0;
}
```

Jump statements in C++

Jump statements are used to manipulate the flow of the program if some conditions are met. It is used to terminate or continue the loop inside a program or to stop the execution of a function.

Types of Jump Statements in C++

In C++, there is four jump statement

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

```
// C++ program to demonstrate the
// continue statement
#include <iostream>
using namespace std;
```

```
// Driver code
int main()
{
	for (int i = 1; i < 10; i++) {

		if (i == 5)
			continue;
		cout << i << " ";
	}
	return 0;
}

// C++ program to demonstrate the
// break statement
#include <iostream>
using namespace std;

// Driver Code
int main()
{
	for (int i = 1; i < 10; i++) {

		// Breaking Condition
		if (i == 5)
			break;
		cout << i << " ";
	}
	return 0;
}

// C++ program to demonstrate the
// return statement
#include <iostream>
using namespace std;

// Driver code
int main()
{
	cout << "Begin ";

	for (int i = 0; i < 10; i++) {
```

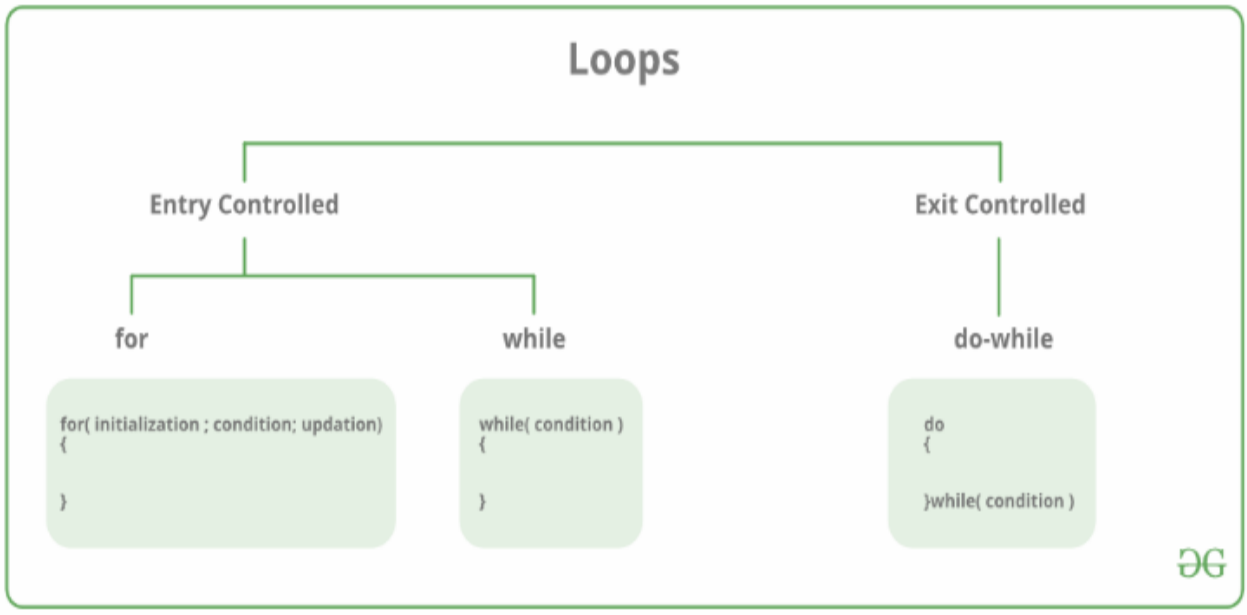
```
                // Termination condition
                if (i == 5)
                        return 0;
                cout << i << " ";
        }
        cout << "end";

        return 0;
}

// C++ program to demonstrate the return
// statement in void return type function
#include <iostream>
using namespace std;

// Function to find the greater element
// among x and y
void findGreater(int x, int y)
{
        if (x > y) {
                cout << x << " "
                        << "is greater"
                        << "\n";
                return;
        }
        else {
                cout << y << " "
                        << "is greater"
                        << "\n";
                return;
        }
}

// Driver Code
int main()
{
        // Function Call
        findGreater(10, 20);

        return 0;
}
```

```
// C++ program to demonstrate the
// goto statement
#include <iostream>
using namespace std;

// Driver Code
int main()
{
        int n = 4;

        if (n % 2 == 0)
                goto label1;
        else
                goto label2;

label1:
        cout << "Even" << endl;
        return 0;

label2:
        cout << "Odd" << endl;
}
```

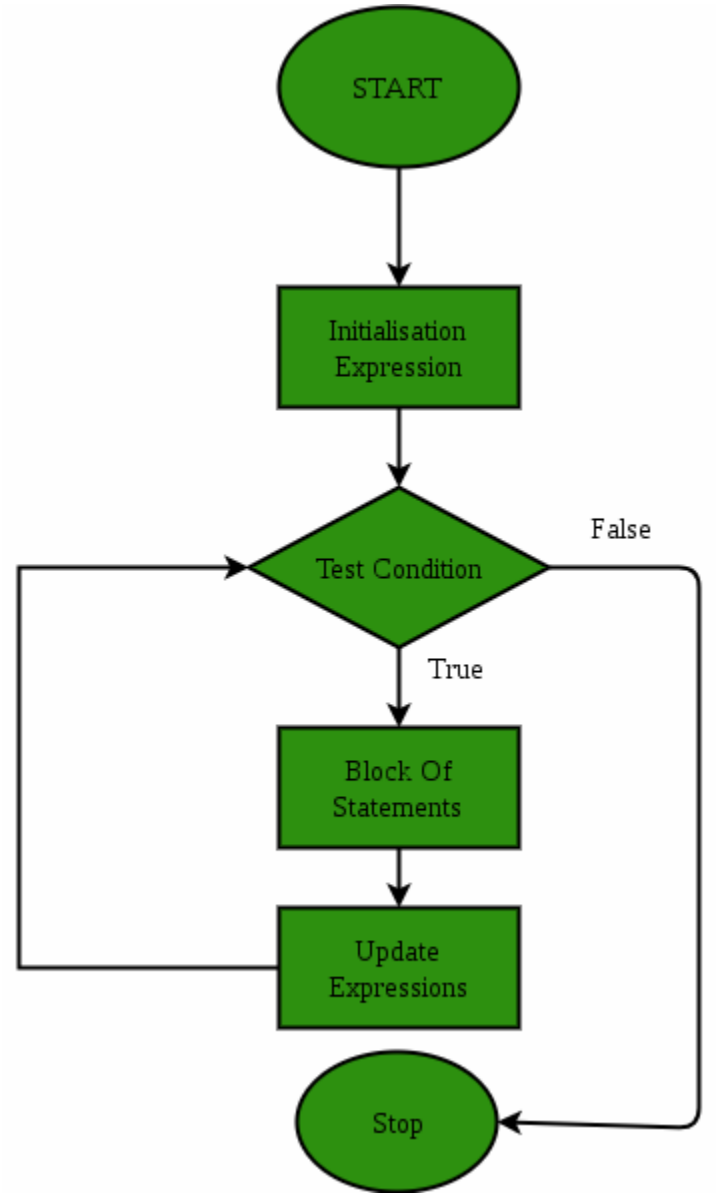
C++ Loops

There are mainly two types of loops:

1. **Entry Controlled loops**: In this type of loop, the test condition is tested before entering the loop body. **For Loop** and **While Loop** is entry-controlled loops.
1. **Exit Controlled Loops**: In this type of loop the test condition is tested or evaluated at the end of the loop body. Therefore, the loop body will execute at least once, irrespective of whether the test condition is true or false. the do-while **loop** is exit controlled loop.

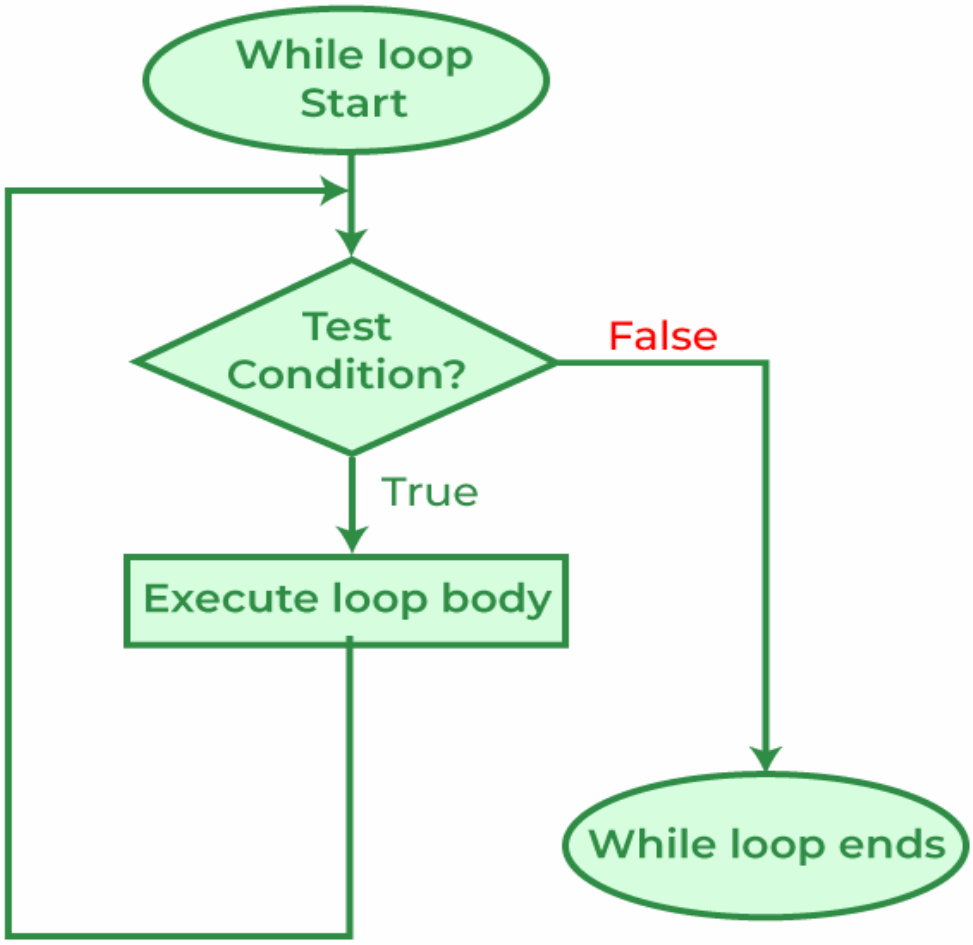


Flow Diagram of for loop:




```
// C++ program to Demonstrate for loop
#include <iostream>
using namespace std;

int main()
{
	for (int i = 1; i <= 5; i++) {
		cout << "Hello World\n";
	}

	return 0;
}
```

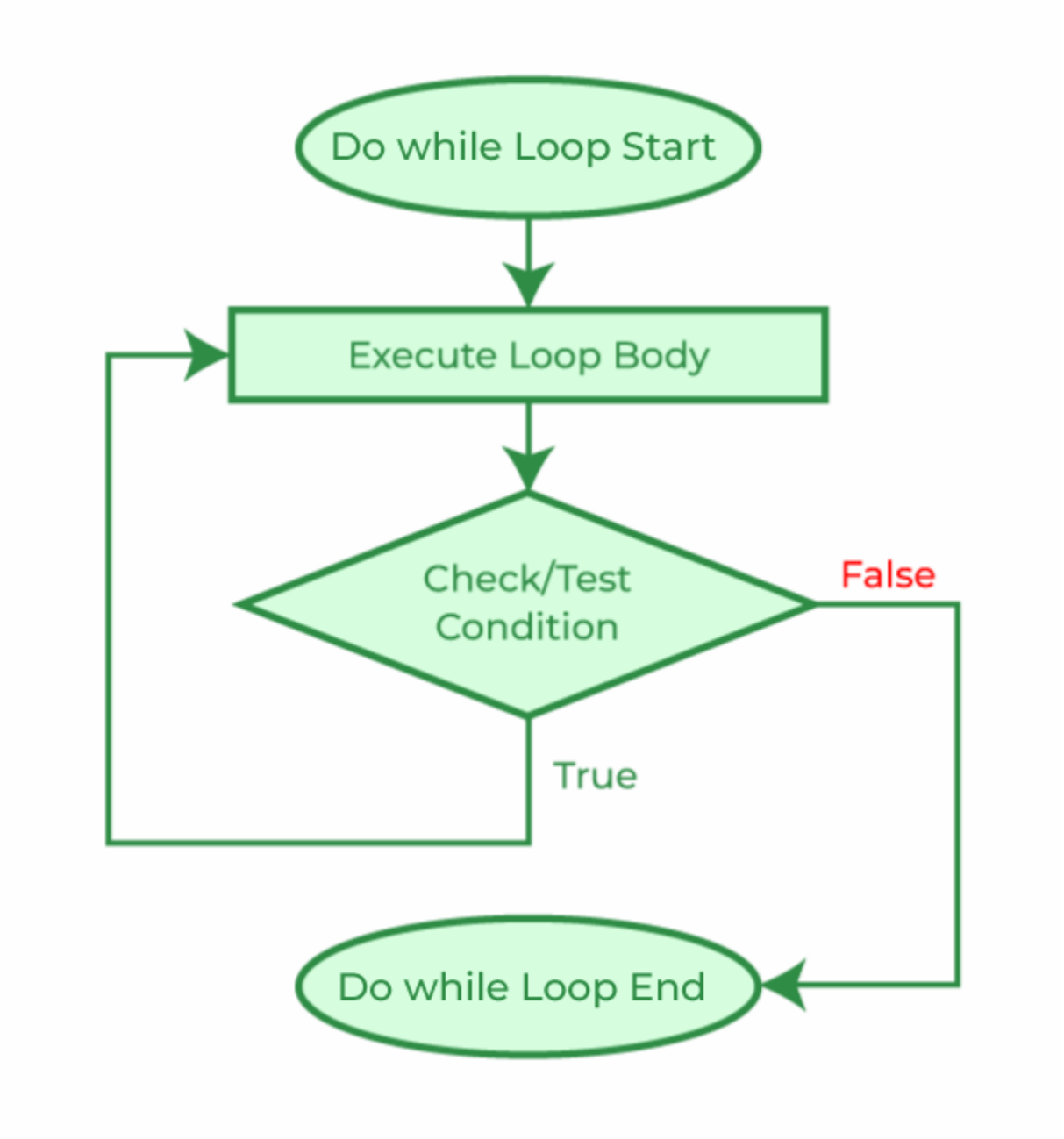
While Loop-




```
// C++ program to Demonstrate while loop
#include <iostream>
using namespace std;

int main()
{
        // initialization expression
        int i = 1;

        // test expression
        while (i < 6) {
                cout << "Hello World\n";

                // update expression
```

```
            i++;
        }

        return 0;
}
```

Do-while loop




```
// C++ program to Demonstrate do-while loop
```

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

int main()
{
	int i = 2; // Initialization expression

	do {
		// loop body
		cout << "Hello World\n";

		// update expression
		i++;

	} while (i < 1); // test expression

	return 0;
}
```

Using While loop:

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

int main()
{

	while (1)
		cout << "This loop will run forever.\n";
	return 0;
}
```

Using the Do-While loop:

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

int main()
{

	do {
		cout << "This loop will run forever.\n";
	} while (1);

	return 0;
}
```

decrementing loops.

Sometimes we need to decrement a variable with a looping condition.

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

int main() {

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

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

int main() {
	//first way is to decrement in the condition itself
	int i=5;
	while(i--){
	cout<<i<<" ";
	}
	cout<<endl;
	//second way is to decrement inside the loop till i is 0
	i=5;
	while(i){
	cout<<i<<" ";
	i--;
	}
	return 0;
}
```

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

int main() {
	int i=5;
do{
	cout<<i<<" ";
}while(i--);
}
```