What is C++?

C++ is a cross-platform language that can be used to create high-performance applications.

C++ was developed by Bjarne Stroustrup, as an extension to the C language.

C++ gives programmers a high level of control over system resources and memory.

The language was updated 3 major times in 2011, 2014, and 2017 to C++11, C++14, and C++17.

Why Use C++

C++ is one of the world's most popular programming languages.

C++ can be found in today's operating systems, Graphical User Interfaces, and embedded systems.

C++ is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs.

C++ is portable and can be used to develop applications that can be adapted to multiple platforms.

C++ is fun and easy to learn!

As C++ is close to C# and Java, it makes it easy for programmers to switch to C++ or vice versa

When we consider a C++ program, it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what a class, object, methods, and instant variables mean.

- **Object −** Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors - wagging, barking, eating. An object is an instance of a class.
- **Class −** A class can be defined as a template/blueprint that describes the behaviors/states that object of its type support.
- **Methods −** A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instance Variables −** Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.
- C++ Program Structure

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

```
// main() is where program execution begins.
int main() {
   cout << "Hello World"; // prints Hello World
   return 0;
}
```

C++ Variables

Variables are containers for storing data values.

In C++, there are different **types** of variables (defined with different keywords), for example:

- `int` - stores integers (whole numbers), without decimals, such as 123 or -123
- `double` - stores floating point numbers, with decimals, such as 19.99 or -19.99
- `char` - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- `string` - stores text, such as "Hello World". String values are surrounded by double quotes
- `bool` - stores values with two states: true or false

Declaring (Creating) Variables

To create a variable, specify the type and assign it a value:

Syntax

```
type variableName = value;
```

Example

Create a variable called **myNum** of type `int` and assign it the value **15**:

```
int myNum = 15;
cout << myNum;
```

Example

```
int myNum = 5;               // Integer (whole number without decimals)
double myFloatNum = 5.99;    // Floating point number (with decimals)
char myLetter = 'D';         // Character
string myText = "Hello";     // String (text)
bool myBoolean = true;       // Boolean (true or false)
```

C++ User Input

`cin` is a predefined variable that reads data from the keyboard with the extraction operator (`>>`).

```
int x;
cout << "Type a number: "; // Type a number and press enter
cin >> x; // Get user input from the keyboard
cout << "Your number is: " << x; // Display the input value
```

C++ is a general purpose, case-sensitive, free-form programming language that supports object-oriented, procedural and generic programming.

Object-Oriented Programming (OOPs)

C++ supports the object-oriented programming, the four major pillar of object-oriented programming (OOPs) used in C++ are:

1. Inheritance
2. Polymorphism
3. Encapsulation
4. Abstraction

C++ Standard Libraries

Standard C++ programming is divided into three important parts:

- The core library includes the data types, variables and literals, etc.
- The standard library includes the set of functions manipulating strings, files, etc.
- The Standard Template Library (STL) includes the set of methods manipulating a data structure.

C++ Functions

A function is a block of code that performs a specific task.To perform any task, we can create function. A function can be called many times. It provides modularity and code reusability.

Advantage of functions in C

There are many advantages of functions.

1) Code Reusability

By creating functions in C++, you can call it many times. So we don't need to write the same code again and again.

2) Code optimization

It makes the code optimized, we don't need to write much code.

Suppose, you have to check 3 numbers (531, 883 and 781) whether it is prime number or not. Without using function, you need to write the prime number logic 3 times. So, there is repetition of code.

But if you use functions, you need to write the logic only once and you can reuse it several times.

Types of Functions

There are two types of functions in C programming:

1. Library Functions: are the functions which are declared in the C++ header files such as ceil(x), cos(x), exp(x), etc.

2. User-defined functions: are the functions which are created by the C++ programmer, so that he/she can use it many times. It reduces complexity of a big program and optimizes the code.

C++ Function Declaration

The syntax to declare a function is:

```
returnType functionName (parameter1, parameter2,...) {
    // function body
}
```

Here's an example of a function declaration.

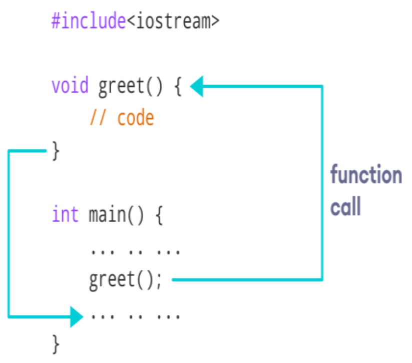
```
// function declaration
void greet() {
    cout << "Hello World";
}
```

Calling a Function

In the above program, we have declared a function named `greet()`. To use the `greet()` function, we need to call it.

Here's how we can call the above `greet()` function.

```
int main() {

    // calling a function
    greet();

}
```

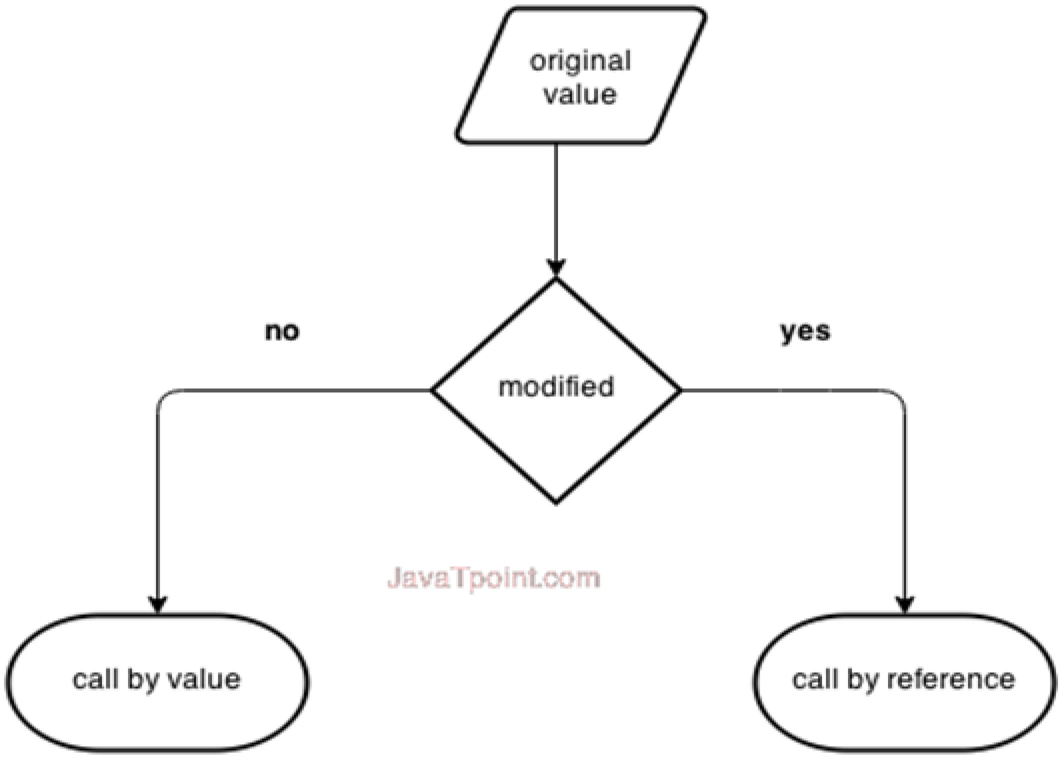


```
#include<iostream>

void greet() {
    // code
}

int main() {
    ... .. ...
    greet();
    ... .. ...
}
```

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

// declaring a function
void greet() {
    cout << "Hello there!";
}

int main() {

    // calling the function
    greet();

    return 0;
}
```

Call by value and call by reference in C++

There are two ways to pass value or data to function in C language: call by value and call by reference. Original value is not modified in call by value but it is modified in call by reference.



Call by value in C++

In call by value, **original value is not modified.**

In call by value, value being passed to the function is locally stored by the function parameter in stack memory location. If you change the value of function parameter, it is changed for the current function only. It will not change the value of variable inside the caller method such as main().

```
#include <iostream>
using namespace std;
void change(int data);
void main()
{
int data = 3;
change(data);
cout << "Value of the data is: " << data<< endl;
getch();
}
void change(int data)
{
data = 5;
}
```

Call by reference in C++

In call by reference, original value is modified because we pass reference (address).

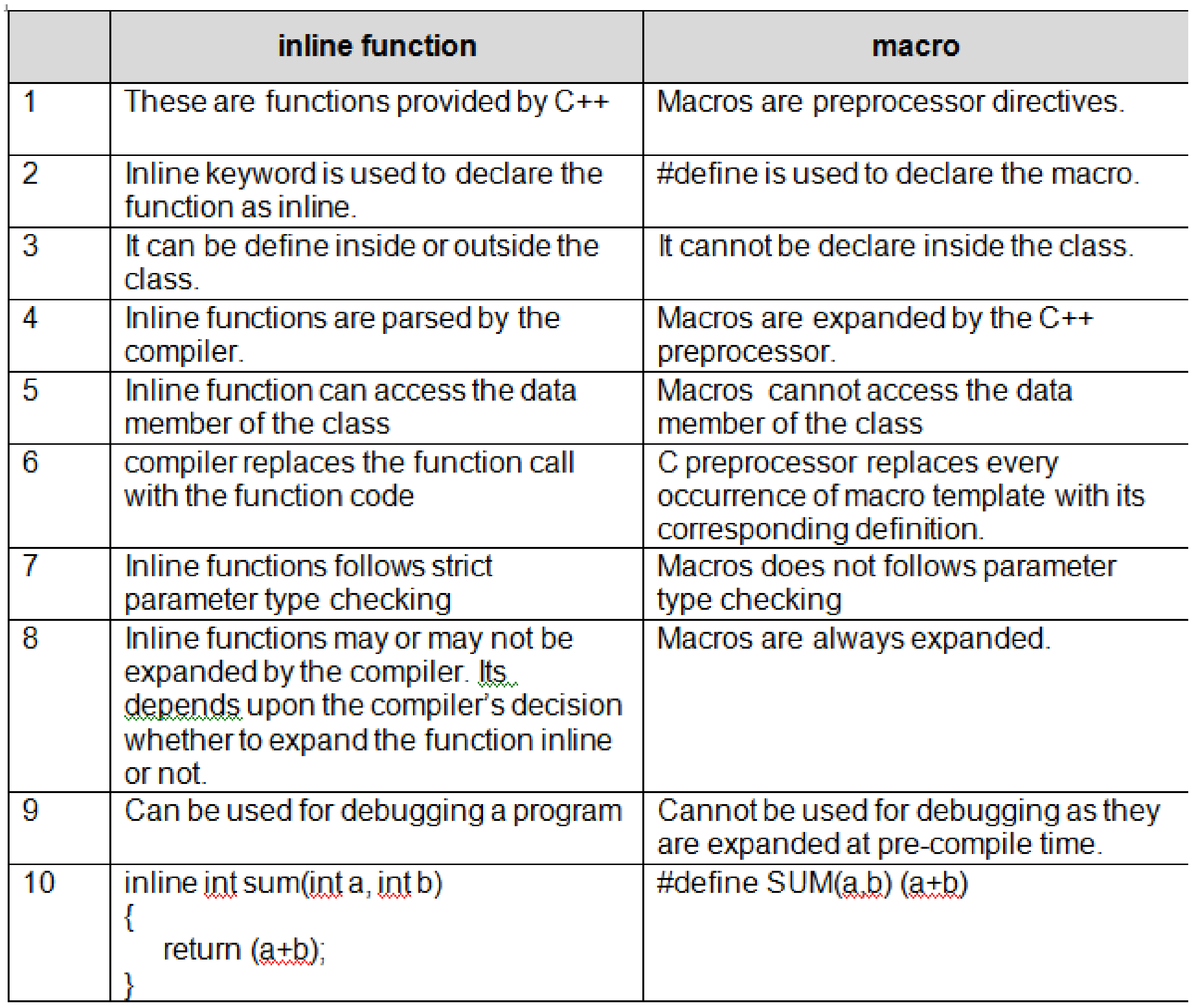
Here, address of the value is passed in the function, so actual and formal arguments share the same address space. Hence, value changed inside the function, is reflected inside as well as outside the function.

Note: To understand the call by reference, you must have the basic knowledge of pointers.

```
#include<iostream>
using namespace std;
void swap(int *x, int *y)
{
 int swap;
 swap=*x;
 *x=*y;
 *y=swap;
}
int main()
{
 int x=500, y=100;
 swap(&x, &y);  // passing value to function
 cout<<"Value of x is: "<<x<<endl;
 cout<<"Value of y is: "<<y<<endl;
 return 0;
}
```

No.	Call by value	Call by reference
1	A copy of value is passed to the function	An address of value is passed to the function
2	Changes made inside the function is not reflected on other functions	Changes made inside the function is reflected outside the function also
3	Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in same memory location

Inline vs Macro



	inline function	macro
1	These are functions provided by C++	Macros are preprocessor directives.
2	Inline keyword is used to declare the function as inline.	#define is used to declare the macro.
3	It can be define inside or outside the class.	It cannot be declare inside the class.
4	Inline functions are parsed by the compiler.	Macros are expanded by the C++ preprocessor.
5	Inline function can access the data member of the class	Macros cannot access the data member of the class
6	compiler replaces the function call with the function code	C preprocessor replaces every occurrence of macro template with its corresponding definition.
7	Inline functions follows strict parameter type checking	Macros does not follows parameter type checking
8	Inline functions may or may not be expanded by the compiler. Its depends upon the compiler's decision whether to expand the function inline or not.	Macros are always expanded.
9	Can be used for debugging a program	Cannot be used for debugging as they are expanded at pre-compile time.

10	inline int sum(int a, int b)
	{
	    return (a+b);
	}
		#define SUM(a,b) (a+b)

C++ Function Overloading

In C++, two functions can have the same name if the number and/or type of arguments passed is different.

These functions having the same name but different arguments are known as overloaded functions. For example:

```
// same name different arguments
int test() { }
int test(int a) { }
float test(double a) { }
int test(int a, double b) { }
```

Here, all 4 functions are overloaded functions.

Notice that the return types of all these 4 functions are not the same. Overloaded functions may or may not have different return types but they must have different arguments. For example,

```
// Error code
int test(int a) { }
double test(int b){ }
```

Here, both functions have the same name, the same type, and the same number of arguments. Hence, the compiler will throw an error.

```
#include <iostream>
 void print(int i) {
  cout << " Here is int " << i << endl;
}
void print(double  f) {
  cout << " Here is float " << f << endl;
}
void main() {
  print(10);
  print(10.8);
    getch();
}
```

Output:

```
Here is int 10

Here is float 10.8
```