C++ Programming

Simply Easy Learning

by Bhimashankar T

Introduction

- C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.
- C++ is regarded as a middle-level language, as it comprises a combination of both high-level and low-level language features.
- C++ was developed by Bjarne Stroustrup starting in 1979 at Bell Labs in Murray Hill, New Jersey, as an enhancement to the C language and originally named C with Classes but later it was renamed C++ in 1983.
- C++ is a superset of C, and that virtually any legal C program is a legal C++ program.
- Note: A programming language is said to use static typing when type checking is performed during compile-time as opposed to run-time.

Comparison between C and C++

C Programming	C++ Programming
Top-Down approach	Bottom-up approach
Function - driven	Object - driven
Cannot use functions inside structures	can use functions inside structures
Data is not secured	Data is hidden (secured)
Procedural oriented language	Procedural and Object oriented language
Cannot do overloading with C	Supports both operator and function overloading
main() need not return any value	main() should return a value

Object-Oriented Programming

Object-oriented programming is based on the following

- Classes
- Objects

C++ fully supports object-oriented programming, including the four pillars of object-oriented development:

- Encapsulation
- Data hiding / abstraction
- Inheritance
- Polymorphism

C++ Basic Syntax

C++ program can be defined as a collection of objects that communicate via invoking each other's methods.

Let us now briefly look into what do 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.
- **Instant Variables** - Each object has its unique set of instant variables. An object's state is created by the values assigned to these instant 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++ Program Structure

Let us look various parts of the above program:

- The C++ language defines several headers, which contain information that is either necessary or useful to your program. For this program, the header <iostream> is needed.

- The line using namespace std; tells the compiler to use the std namespace. Namespaces are a relatively recent addition to C++.

- The next line // main() is where program execution begins. is a single-line comment available in C++. Single-line comments begin with // and stop at the end of the line.

- The line int main() is the main function where program execution begins.

- The next line cout << "Hello World"; causes the message "Hello World" to be displayed on the screen.

- The next line return 0; terminates main( )function and causes it to return the value 0 to the calling process.

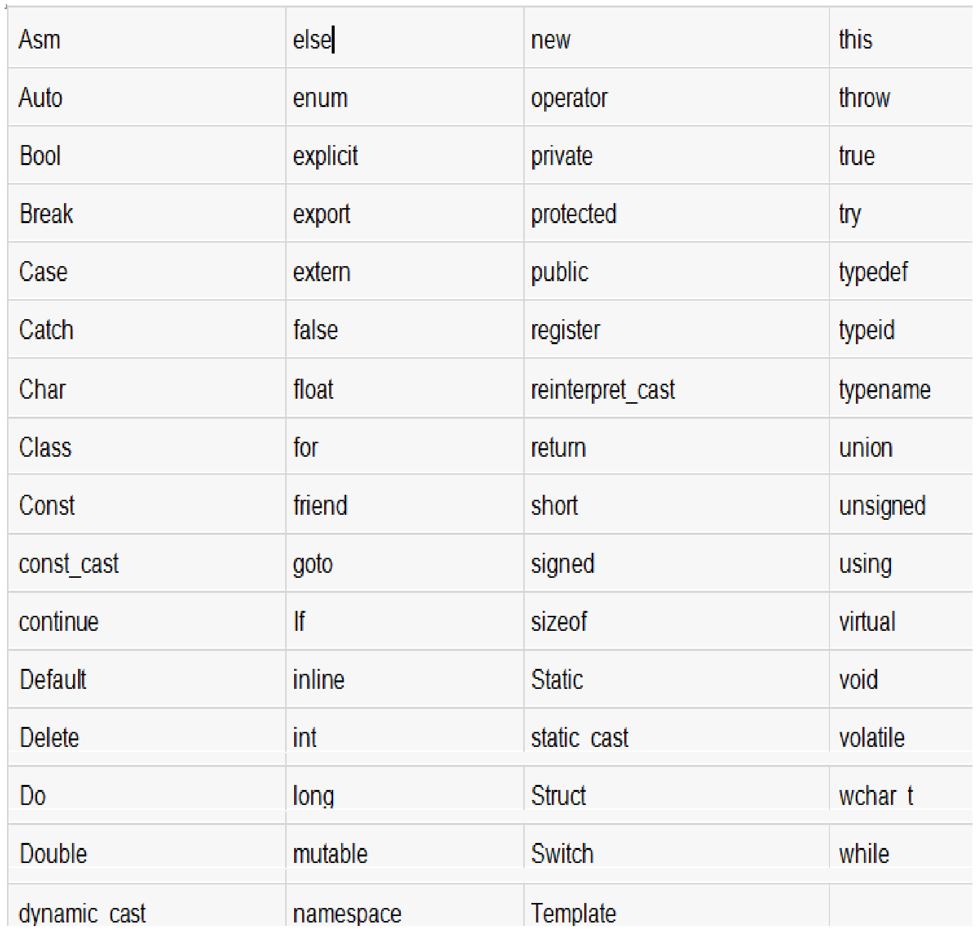
C++ Identifiers:

A C++ identifier is a name used to identify a variable, function, class, module, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9).

C++ does not allow punctuation characters such as @, $, and % within identifiers. C++ is a case-sensitive programming language. Thus, Manpower and manpower are two different identifiers in C++.

C++ Keywords:

The following list shows the reserved words in C++. These reserved words may not be used as constant or variable or any other identifier names.



Asm	else	new	this
Auto	enum	operator	throw
Bool	explicit	private	true
Break	export	protected	try
Case	extern	public	typedef
Catch	false	register	typeid
Char	float	reinterpret_cast	typename
Class	for	return	union
Const	friend	short	unsigned
const_cast	goto	signed	using
continue	If	sizeof	virtual
Default	inline	Static	void
Delete	int	static cast	volatile
Do	long	Struct	wchar t
Double	mutable	Switch	while
dynamic cast	namespace	Template	

Data Types

Primitive Built-in Types:

C++ offer the programmer a rich assortment of built-in as well as user defined data types. Following table lists down seven basic C++ data types

Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void
Wide character	wchar_t

Modifier Types:

C++ allows the char, int, and double data types to have modifiers preceding them. A modifier is used to alter the meaning of the base type so that it more precisely fits the needs of various situations.

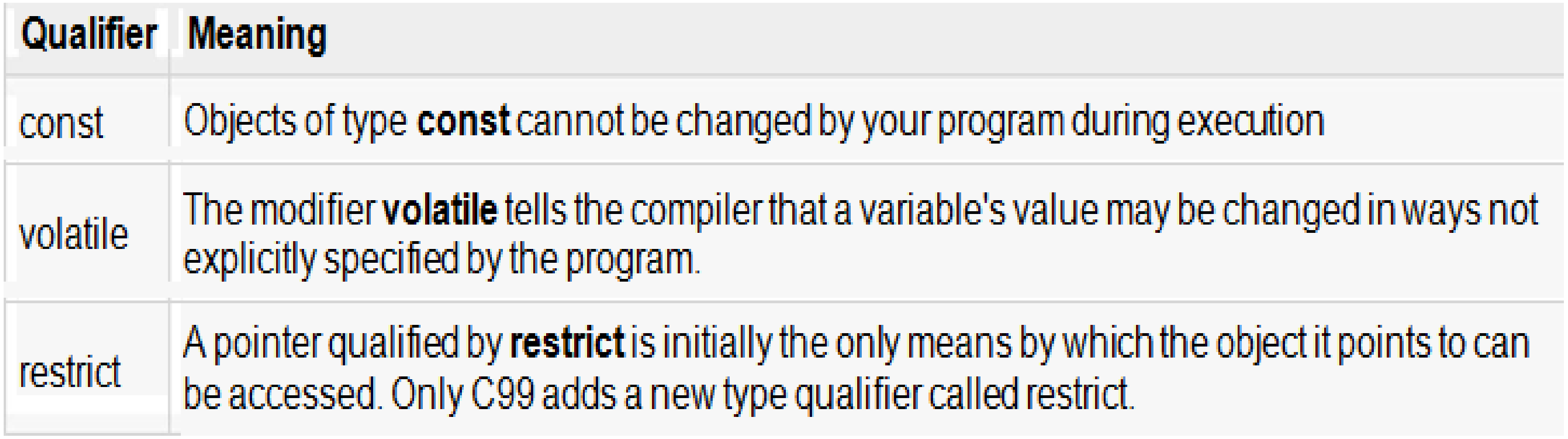
The data type modifiers are listed here:

- signed
- unsigned
- long
- short

Data Types

Type Qualifiers in C++:

The type qualifiers provide additional information about the variables they precede.



Qualifier	Meaning
const	Objects of type **const** cannot be changed by your program during execution
volatile	The modifier **volatile** tells the compiler that a variable's value may be changed in ways not explicitly specified by the program.
restrict	A pointer qualified by **restrict** is initially the only means by which the object it points to can be accessed. Only C99 adds a new type qualifier called restrict.

Storage Classes

A storage class defines the scope (visibility) and life-time of variables and/or functions within a C++ Program. These specifiers precede the type that they modify. There are following storage classes, which can be used in a C++ Program

- **auto :**

  The **auto** storage class is the default storage class for all local variables.

  Example: auto int month;

- **register :**

  The **register** storage class is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can't have the unary '&' operator applied to it (as it does not have a memory location).

  The **register** should only be used for variables that require quick access such as counters. It should also be noted that defining 'register' does not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register depending on hardware and implementation restrictions

  Example: register int miles;

Storage Classes

- **static :**

  The **static** storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls.

  The **static** modifier may also be applied to global variables. When this is done, it causes that variable's scope to be restricted to the file in which it is declared.

  Example: static int count =10; /* Global variable */

- **extern :**

  The **extern** storage class is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern' the variable cannot be initialized as all it does is point the variable name at a storage location that has been previously defined.

  When you have multiple files and you define a global variable or function, which will be used in other files also, then *extern* will be used in another file to give reference of defined variable or function. Just for understanding, *extern* is used to declare a global variable or function in another file.

  Example: extern int count;

Storage Classes

- **mutable** :

  The **mutable** specifier applies only to class objects, which are discussed later in this tutorial. It allows a member of an object to override constness. That is, a mutable member can be modified by a const member function

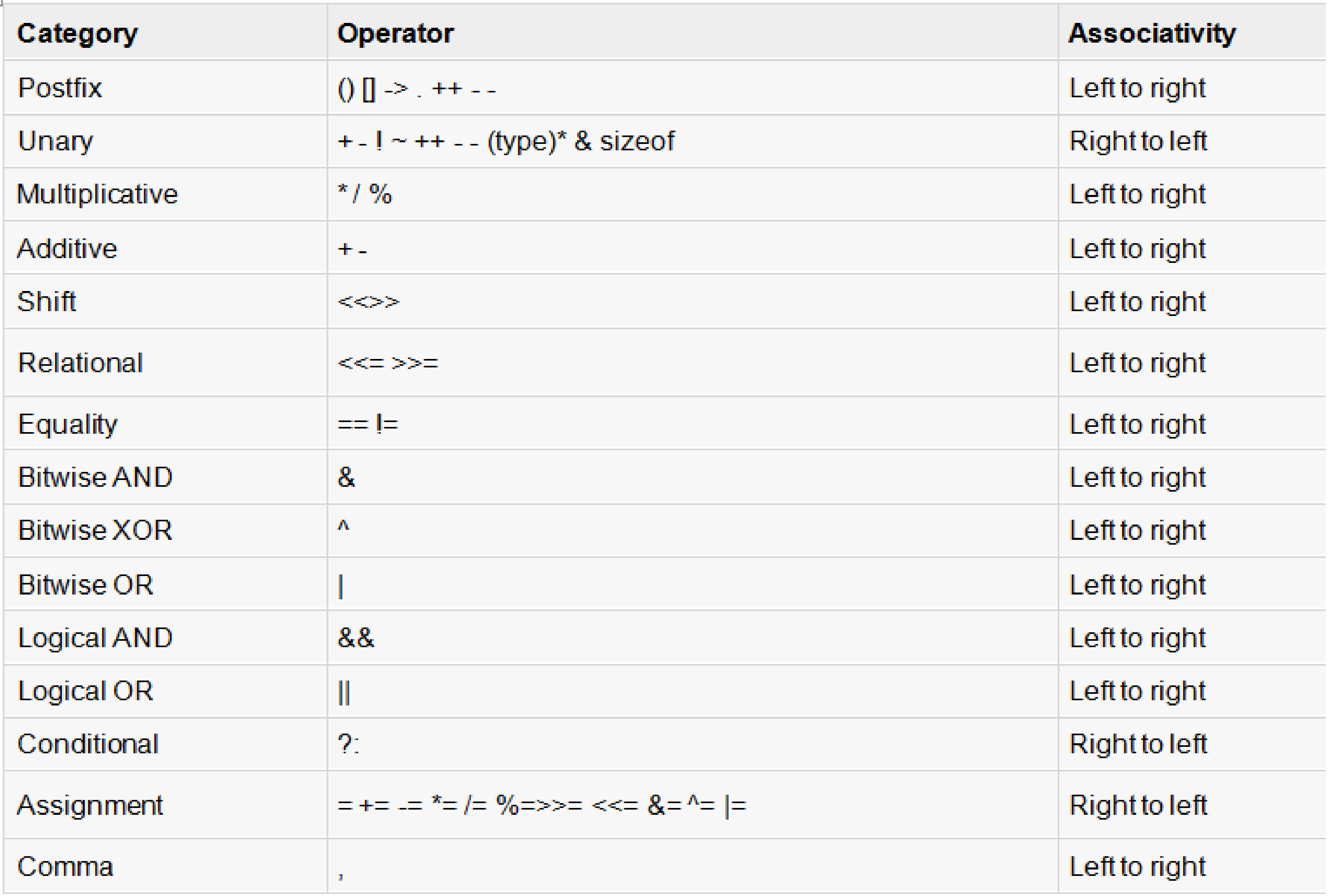
Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

C++ is rich in built-in operators and provides the following types of operators:

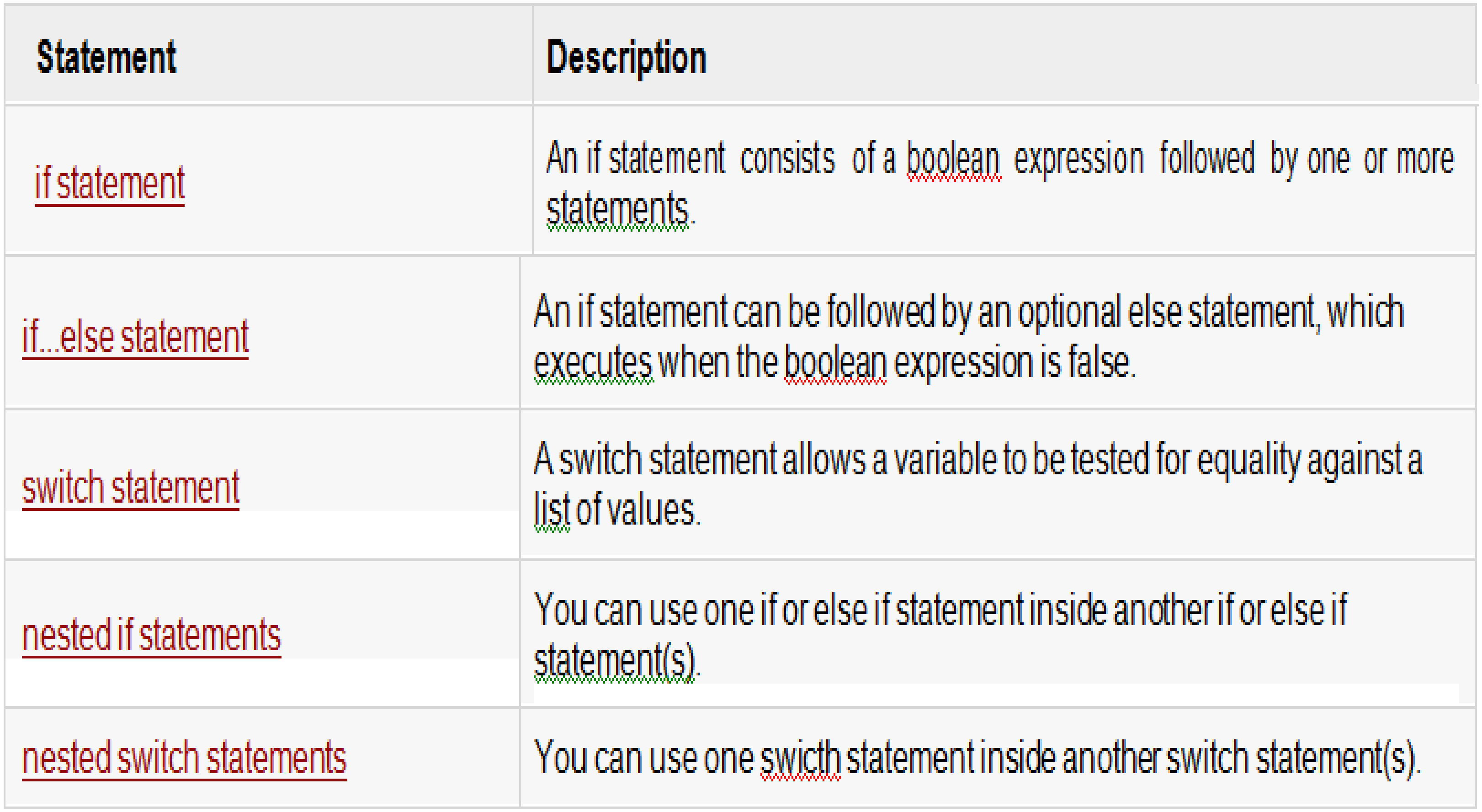
- Arithmetic Operators : **+** , **-**, *****, **/**, **%**, **++**, **--**
- Relational Operators : **==, !=, >, <, >=, <=**
- Logical Operators : **&&, ||, !**
- Bitwise Operators : **&, |, ^, ~, <<, >>,**
- Assignment Operators : **=**
- Misc Operators : **sizeof(), ?, comma(,), dot(.), arrow(->), reference or address of (&), pointer(*)**

Operators Precedence in C++



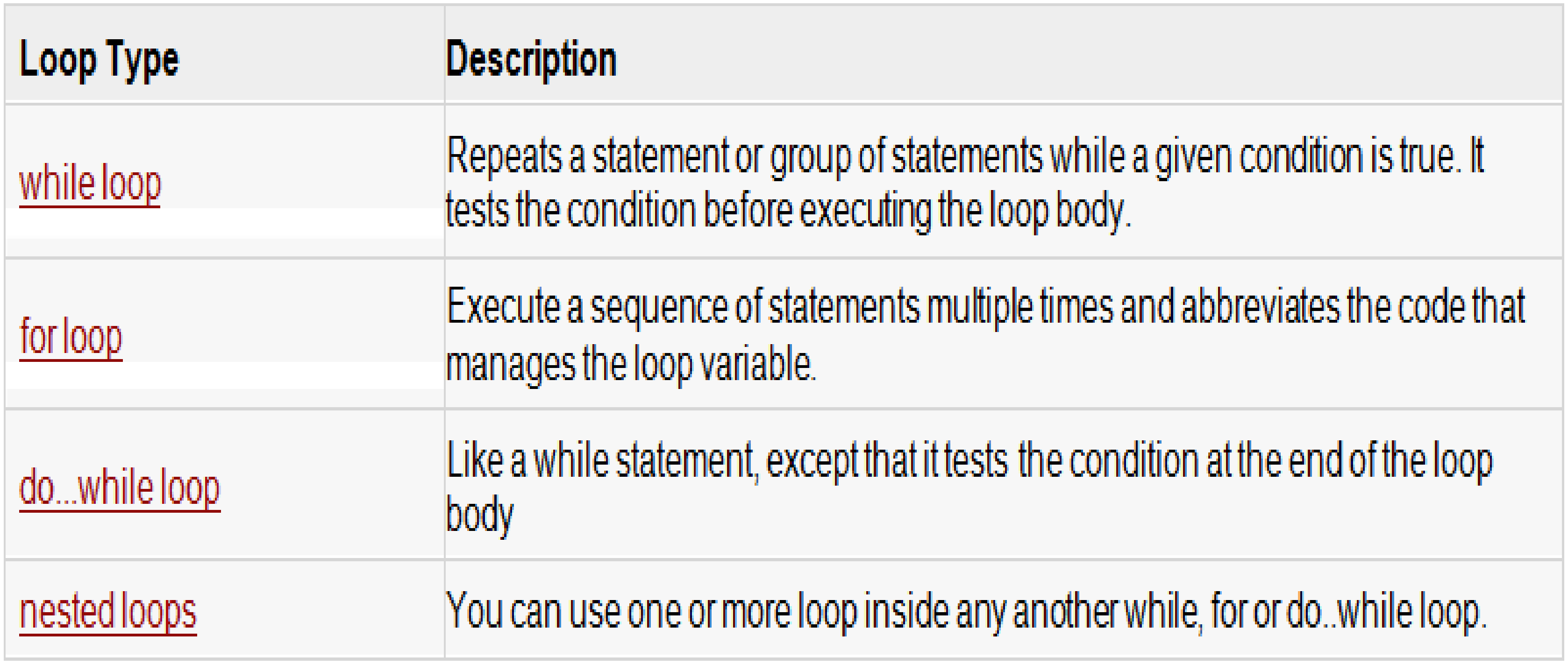
Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<<>>	Left to right
Relational	<<= >>=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Decision Making



Statement	Description
if statement	An if statement consists of a boolean expression followed by one or more statements.
if...else statement	An if statement can be followed by an optional else statement, which executes when the boolean expression is false.
switch statement	A switch statement allows a variable to be tested for equality against a list of values.
nested if statements	You can use one if or else if statement inside another if or else if statement(s).
nested switch statements	You can use one swicth statement inside another switch statement(s).

C++ Loop Types



Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
do...while loop	Like a while statement, except that it tests the condition at the end of the loop body
nested loops	You can use one or more loop inside any another while, for or do..while loop.

Functions

A function is a group of statements that together perform a task. Every C++ program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.

You can divide up your code into separate functions.

Defining a Function:

The general form of a C++ function definition is as follows:

```
return_type function_name( parameter list )
 {
body of the function

}
```

- **Return Type**: A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.

- **Function Name**: This is the actual name of the function. The function name and the parameter list together constitute the function signature.

- **Parameters**: A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

- **Function Body**: The function body contains a collection of statements that define what the function does.

Functions

Example:

```
// function returning the max between two
numbers
int max(int num1, int num2)
{
       // local variable declaration
       int result;
       if (num1 > num2)
               result = num1;
       else
               result = num2;
 return result;
}
```

Calling a Function:

```
#include <iostream>
using namespace std;
// function declaration
 int max(int num1, int num2);

int main ()
{
  local variable declaration:
  int a = 100;

  int b = 200; int ret;
  calling a function to get max value.
  ret = max(a, b);
  cout << "Max value is : " << ret << endl;
  return 0;
}
```

Functions

Call by value :

```
void swap(int x, int y)
{
    int  temp;
    temp = x; /* save the value of x */
    x = y; /* put y into x */
    y = temp; /* put x into y */
 }

#include <iostream>
using namespace std;
 // function declaration
void swap(int x, int y);
int main ()
{
   // local variable declaration:
   int a = 100;
   int b = 200;
   cout << "Before swap, value of a :" << a << endl;
   cout << "Before swap, value of b :" << b << endl;

   // calling a function to swap the values.
   swap(a, b);
   cout << "After swap, value of a :" << a << endl;
   cout << "After swap, value of b :" << b << endl;
   return 0;
}
```

Call by pointer :

```
void swap(int *x, int *y)
 {
    int temp;
    temp = *x; /* save the value at address x */
     *x = *y; /* put y into x */
     *y = temp; /* put x into y */
 }

#include <iostream>
using namespace std;
// function declaration
void swap(int *x, int *y);
int main ()
{
    // local variable declaration:
    int a = 100;
    int b = 200;
    cout << "Before swap, value of a :" << a << endl;
    cout << "Before swap, value of b :" << b << endl;
    //calling a function to swap the values.
    swap(&a, &b);
    cout << "After swap, value of a :" << a << endl;
    cout << "After swap, value of b :" << b << endl;
    return 0;
}
```

Functions

Call by reference:

```
void swap(int &x, int &y)
{
    int  temp;
    temp = x; /* save the value of x */
    x = y; /* put y into x */
    y = temp; /* put x into y */
 }

#include <iostream>
using namespace std;
 // function declaration
void swap(int &x, int &y);
int main ()
{
   // local variable declaration:
   int a = 100;
   int b = 200;
   cout << "Before swap, value of a :" << a << endl;
   cout << "Before swap, value of b :" << b << endl;

   // calling a function to swap the values.
   swap(a, b);
   cout << "After swap, value of a :" << a << endl;
   cout << "After swap, value of b :" << b << endl;
   return 0;
}
```

Default Values for Parameters :

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

int sum(int a, int b=20)
{
    int result;
    result = a + b;
    return (result);
 }

int main ()
 {
     // local variable declaration:
     int a = 100;
     int b = 200;
     int result;
     // calling a function to add the values.
     result = sum(a, b);
     cout << "Total value is :" << result << endl;
     // calling a function again as follows.
     result = sum(a);
     cout << "Total value is :" << result << endl;

     return 0;
 }
```