Contents

[**CHAPTER 1: INTODUCTION TO C++** 3](#_Toc203125132)

[What is C++? 3](#_Toc203125133)

[Advantages of learning C++: 3](#_Toc203125134)

[Features of C++ 4](#_Toc203125135)

[Applications of C++ Programming 4](#_Toc203125136)

[C++ Program Structure 5](#_Toc203125137)

[Comments in C++ 5](#_Toc203125138)

[**Types of C++ Comments** 5](#_Toc203125139)

[**CHAPTER TWO: TOKENS IN C++** 6](#_Toc203125140)

[Tokens in C++ 6](#_Toc203125141)

[Keywords 6](#_Toc203125142)

[Identifiers 6](#_Toc203125143)

[C++ Constants/Literals 7](#_Toc203125144)

[Defining Constants 7](#_Toc203125145)

[C++ Data Types 7](#_Toc203125146)

[**1.** **Primitive Built-in Types** 8](#_Toc203125147)

[**2.** **Derived Data Types** 8](#_Toc203125148)

[**3.** **User-Defined Data Types** 8](#_Toc203125149)

[typedef Declarations 9](#_Toc203125150)

[C++ Variables and Types 9](#_Toc203125151)

[**Lvalues and Rvalues** 9](#_Toc203125152)

[**Variable Scope in C++** 9](#_Toc203125153)

[C++ Modifier Types 10](#_Toc203125154)

[Type Qualifiers in C++ 10](#_Toc203125155)

[Storage Classes in C++ 10](#_Toc203125156)

[**CHAPTER TWO: C++ DECISION MAKING STATEMENTS** 12](#_Toc203125157)

[C++ if...else statement 12](#_Toc203125158)

[if...else if...else Statement 12](#_Toc203125159)

[**C++ switch statement** 13](#_Toc203125160)

[Loop Control Statements 13](#_Toc203125161)

[C++ while loop 13](#_Toc203125162)

[C++ for loop 14](#_Toc203125163)

[do...while loop 16](#_Toc203125164)

[**CHAPTER 3 : ESSENCE OF OBJECTS AND CLASSES** 17](#_Toc203125165)

[C++ Object Oriented 17](#_Toc203125166)

[C++ Class Definitions 18](#_Toc203125167)

[Define C++ Objects 18](#_Toc203125168)

[Accessing the Data Members 19](#_Toc203125169)

[Class Member Functions 20](#_Toc203125170)

[Defining Class Member Functions 20](#_Toc203125171)

[Calling (Accessing) Member Functions 20](#_Toc203125172)

[Class Access Modifiers 21](#_Toc203125173)

[Public Access Modifier 21](#_Toc203125174)

[Private Access Modifier 21](#_Toc203125175)

[Protected Access Modifier 21](#_Toc203125176)

[CHAPTER 4: C++ Inheritance 22](#_Toc203125177)

[Base and Derived Classes 22](#_Toc203125178)

[CHAPTER 5: 23](#_Toc203125179)

[C++ Overloading (Operator and Function) 23](#_Toc203125180)

[Function Overloading in C++ 23](#_Toc203125181)

[Example of Function Overloading 23](#_Toc203125182)

[Function Overriding in C++ 24](#_Toc203125183)

[Polymorphism in C++ 24](#_Toc203125184)

[Virtual Function 24](#_Toc203125185)

[Pure Virtual Functions 25](#_Toc203125186)

[C++ Class Constructor and Destructor 25](#_Toc203125187)

[The Class Constructor 25](#_Toc203125188)

[Types of Constructors in C++ 26](#_Toc203125189)

[The Class Destructor 27](#_Toc203125190)

[C++ Files and Streams 28](#_Toc203125191)

[Opening a File 28](#_Toc203125192)

[Closing a File 29](#_Toc203125193)

# **CHAPTER 1: INTODUCTION TO C++**

## What is C++?

**C++** is a middle-level programming language developed by Bjarne Stroustrup starting in 1979 at Bell Labs. **C++** runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. C++ is an extension of the C programming language with object-oriented programming concepts. Or, we can say, "C++ is a superset of C programming with additional implementation of object-oriented concepts".

## Advantages of learning C++:

* C++ is very close to hardware, so you get a chance to work at a low level, which gives you a lot of control in terms of memory management, better performance, and finally, robust software development.
* **C++ programming** gives you a clear understanding of object-oriented programming. You will understand low level implementation of polymorphism when you implement virtual tables and virtual table pointers, or dynamic type identification.
* C++ is one of the evergreen programming languages and is loved by millions of software developers. If you are a great C++ programmer, then you will never sit without work, and more importantly, you will get highly paid for your work.
* C++ is the most widely used programming language in application and system programming. So you can choose your area of interest in software development.
* C++ really teaches you the difference between compiler, linker, and loader, different data types, storage classes, variable types, their scopes, etc.

## Features of C++

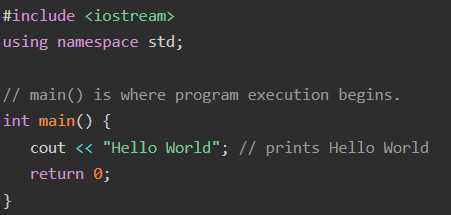
* **C language compatibility**: C++ provides backward compatibility with C; it supports all features of C language.
* **Object-oriented programming**: C++ supports the concepts of OOPs such as objects & classes, encapsulation, data binding, inheritance, and polymorphism.
* **Compiled language:**C++ is a compiler language where the complete code is converted into machine language, which makes it a faster programming language.
* **Standard template library:**C++ provides many data structures and algorithms-related library collections, such as template libraries for contains, iterators, algorithms, etc.
* **Dynamic memory management**: C++ provides two operators **new** and **delete** that help to allocate and deallocate memory blocks dynamically.
* **Exception handling**: C++ provides try, catch, and throw blocks for exceptional handling, which were not available in C programming.

## Applications of C++ Programming

* **Application Software Development** - C++ programming has been used in developing almost all the major Operating Systems like Windows, Mac OSX and Linux. Apart from the operating systems, the core part of many browsers, like Mozilla Firefox and Chrome have been written using C++. C++ also has been used in developing the most popular database system called MySQL.
* **Programming Languages Development** - C++ has been used extensively in developing new programming languages like C#, Java, JavaScript, Perl, UNIX's C Shell, PHP, Python, and Verilog, etc.
* **Computation Programming** - C++ is the best friend of scientists because of its fast speed and computational efficiencies.
* **Games Development** - C++ is extremely fast, which allows programmers to do procedural programming for CPU-intensive functions and provides greater control over hardware, because of which it has been widely used in the development of gaming engines.
* **Embedded System**- C++ is being heavily used in developing medical and engineering applications like software for MRI machines, high-end CAD/CAM systems, etc.

## C++ Program Structure

* **Header file inclusion section**: This is the section where we include all required header files whose functions we are going to use in the program.
* **Namespace section**: This is the section where we use the namespace.
* **The main() section**: In this section, we write our main code. The main() function is an entry point of any C++ programming code from where the program's execution starts.



* The C++ language defines several headers, which contain information that is either necessary or useful to your program. For this program, the header [**<iostream>**](https://www.tutorialspoint.com/cpp_standard_library/iostream.htm) 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.

## Comments in C++

Program comments are explanatory statements that you can include in the C++ code. These comments help anyone reading the source code. All programming languages allow for some form of comments.

# **Types of C++ Comments**

**1. C++ Single-line Comments**

A single-line comment starts with **//**, extending to the end of the line. These comments can last only till the end of the line, and the next line leads to a new comment.

**2. C++ Multi-line Comments**

Multi-line comments start with **/\*** and end with **\*/**. Any text in between these symbols is treated as a comment only.

**Purpose/Advantages of Comments**

* To represent a short and concise step in the program for users to understand better.
* To explain a step in a detailed way that is not expressed explicitly in the code.
* To leave different hints for users to grab in the code itself.
* To leave comments for fun or recreation.
* To temporarily disable part of the code for debugging purposes.
* To add metadata to the code for future purposes.
* To create documentation for the code, for example, in Github pages.

# **CHAPTER TWO: TOKENS IN C++**

## Tokens in C++

C++ tokens are the smallest individual units of a program, that the compiler recognizes and processes. Together, they form the syntax of C++ code, the same as combining words to form a sentence.

Here are the following C++ tokens given below

* Keywords
* Identifiers
* Constants
* Variables
* Operators
* Punctuation

## Keywords

Keywords are reserved words which have fixed meaning, and its meaning cannot be changed. The meaning and working of these keywords are already known to the compiler. C++ has more numbers of keyword than C, and those extra ones have special working capabilities.

List of some commonly used keywords

* **Control flow:** if, else, switch, case, break, continue, return, goto
* **Data types:** int, char, double, float, bool, void, long, short
* **Storage classes:** static, extern, mutable, register, volatile
* **Object-oriented programming:** class, struct, public, private, protected, virtual, this, friend
* **Other:** namespace, using, typedef, const, sizeof, typeid, template, new, delete

## Identifiers

Identifiers are names given to different entries such as variables, structures, and functions, classes, objects, arrays, etc. Also, identifier names should have to be unique because these entities are used in the execution of the program.

Identifier naming conventions

* Only alphabetic characters starting with a letter (A-Z, a-z), digits (0-9), and underscores (\_) are permitted.
* The first letter must be an **alphabet** or **underscore (\_)** not a number.
* Identifiers are case sensitive.
* Reserved keywords can not be used as an identifier's name.

## C++ Constants/Literals

Constants refer to fixed values that the program may not alter and they are called **literals**.

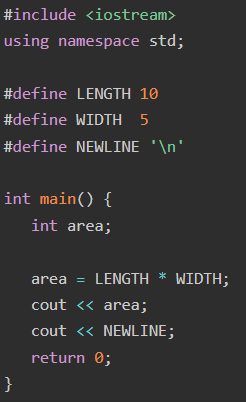
Constants can be of any of the basic data types and can be divided into:

* Integer Literals
* Floating-Point Literals
* Characters Literals
* Strings Literals
* Boolean Literals

## Defining Constants

There are two simple ways in C++ to define constants −

* Using **#define** preprocessor.
* Using **const** keyword.



## C++ Data Types

### **Primitive Built-in Types**

|  |  |
| --- | --- |
| **Type** | **Keyword** |
| Boolean | bool |
| Character | char |
| Integer | int |
| Floating point | float |
| Double floating point | double |
| Valueless | void |

### **Derived Data Types**

Data types which are obtained from pre-defined data types in C++ are known as Derived Data Types. These can be classified into four categories, namely –

Array- An array is a series of elements of same data type.

Function

Pointer - A pointer is a reference to an element defined previously. The value of the pointer returns the address location of the element which is associated with it.

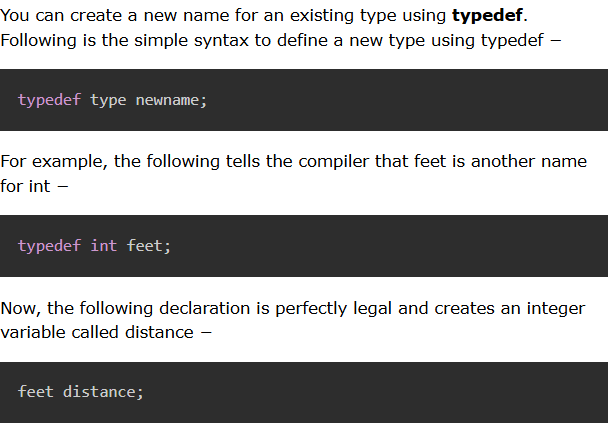
Reference - A reference variable is used to create a copy of a variable with the same reference.

### **User-Defined Data Types**

Data types which are defined by the user intuitively without using any pre-defined data types are known as User-Defined Data Types. These data types can be further categorized into five types, namely –

* **Class** - A class is a defined in Object Oriented Programming as a custom data type which is used to construct an object.
* **Structure (struct)-** In structure data type, the user can introduce multiple primitive data types inside the struct body.
* **Union**- Union is similar to a structure. In this, the memory location of all variables is same, and all variables share the same reference. Hence, a change in one value leads to all other values getting changed.
* **Enumeration (Enum)-** Enumeration or simply enum is a user-defined data type that is used to give name to integer constants in a program.

## typedef Declarations



## C++ Variables and Types

A variable provides us with named storage that our programs can manipulate.

**Rules for variable naming conventions in C++ −**

* Keywords cannot be used as variable names.
* The variable name cannot contain spaces.
* Hyphen (-) cannot be used within the variable names.
* Variable names must not start with special characters and numbers. It should be either an uppercase or lowercase character or an underscore (\_).

### **Lvalues and Rvalues**

There are two kinds of expressions in C++ −

* **lvalue** − Expressions that refer to a memory location is called "lvalue" expression. An lvalue may appear as either the left-hand or right-hand side of an assignment.
* **rvalue** − The term rvalue refers to a data value that is stored at some address in memory. An rvalue is an expression that cannot have a value assigned to it which means an rvalue may appear on the right- but not left-hand side of an assignment.

Variables are lvalues and so may appear on the left-hand side of an assignment. Numeric literals are rvalues and so may not be assigned and can not appear on the left-hand side.

### **Variable Scope in C++**

**Local Variables**

Variables that are declared inside a function or block are local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own.

**Global Variables**

Global variables are defined outside of all the functions, usually on top of the program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration.

## C++ Modifier Types

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

## Type Qualifiers in C++

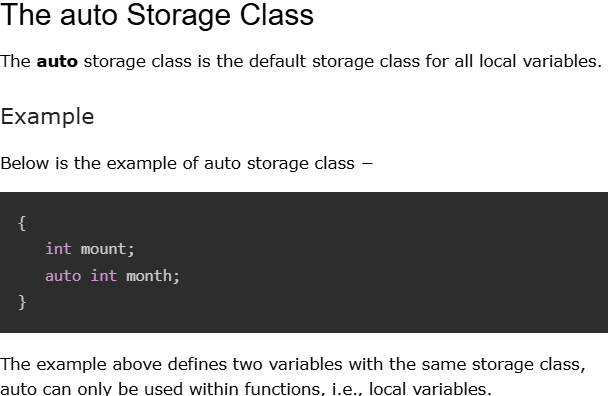
The type qualifiers provide additional information about the variables they precede.

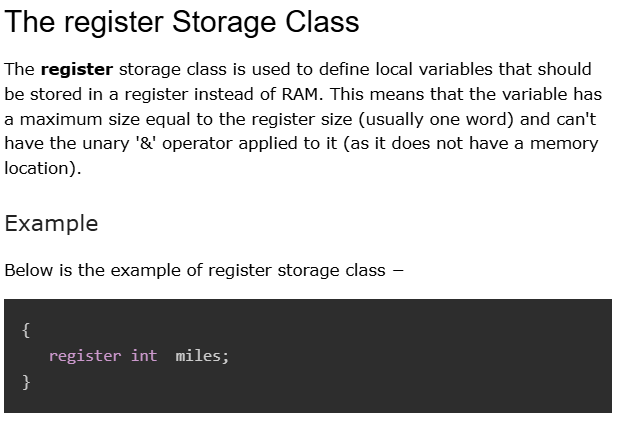
1. **Const** - Objects of type const cannot be changed by your program during execution.
2. **Volatile -** The modifier volatile tells the compiler that a variable's value may be changed in ways not explicitly specified by the program.
3. **Restrict** - A pointer qualified by restrict is initially the only means by which the object it points to can be accessed.

## Storage Classes in C++

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
* register
* static
* extern
* mutable





**The static Storage Class**

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 extern Storage Class**

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.

# **CHAPTER TWO: C++ DECISION MAKING STATEMENTS**

## C++ if...else statement

An **if** statement can be followed by an optional **else** statement, which executes when the boolean expression is false.

Syntax

The syntax of an if...else statement in C++ is −

if(boolean\_expression)

{

// statement(s) will execute if the boolean expression is true

}

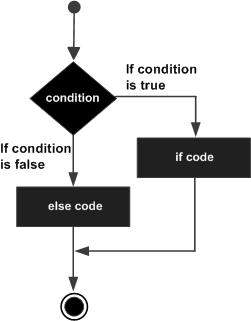
else

{

// statement(s) will execute if the boolean expression is false

}

If the boolean expression evaluates to **true**, then the **if block** of code will be executed, otherwise **else block** of code will be executed.

Flow Diagram

## if...else if...else Statement

An **if** statement can be followed by an optional **else if...else** statement, which is very usefull to test various conditions using single if...else if statement.

if(boolean\_expression 1)

{

// Executes when the boolean expression 1 is true

}

else if( boolean\_expression 2)

{

// Executes when the boolean expression 2 is true

}

else if( boolean\_expression 3)

{

// Executes when the boolean expression 3 is true

}

else

{

// executes when the none of the above condition is true.

}

# **C++ switch statement**

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

The syntax for a **switch** statement in C++ is as follows –

switch(expression)

{

case constant-expression : statement(s);

break; //optional

case constant-expression : statement(s);

break; //optional

// you can have any number of case statements.

default : //Optional statement(s);

}

## Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

## C++ while loop

A **while** loop statement repeatedly executes a target statement as long as a given condition is true.

while(condition)

{

statement(s);

}

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.



## C++ for loop

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

for ( init; condition; increment )

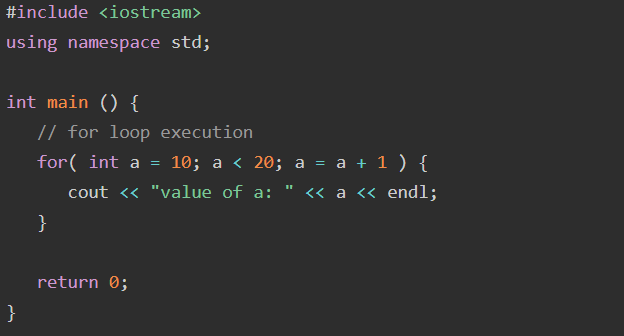
{

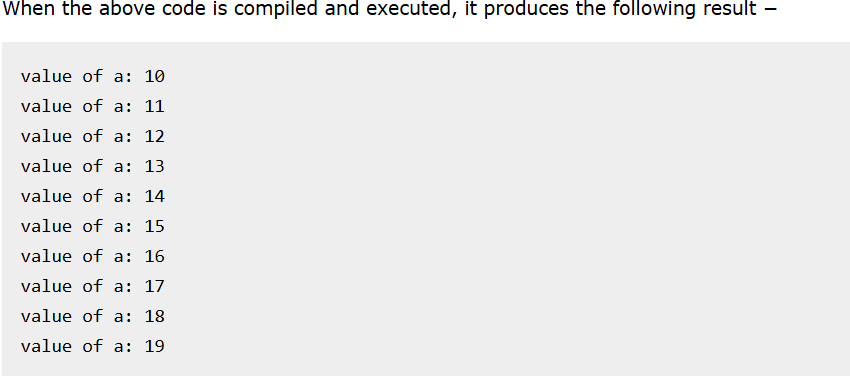
statement(s);

}

* The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
* Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
* After the body of the for loop executes, the flow of control jumps back up to the **increment** statement. This statement can be left blank, as long as a semicolon appears after the condition.
* The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the for loop terminates.







## do...while loop

Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop checks its condition at the bottom of the loop.

A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

do

{

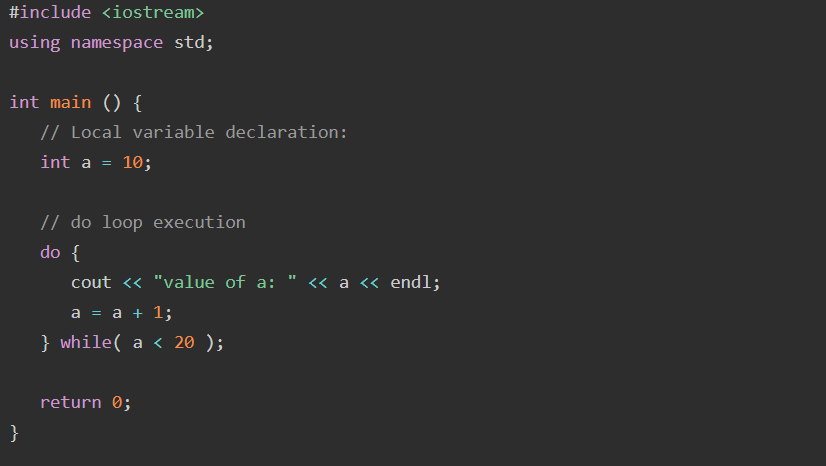
statement(s);

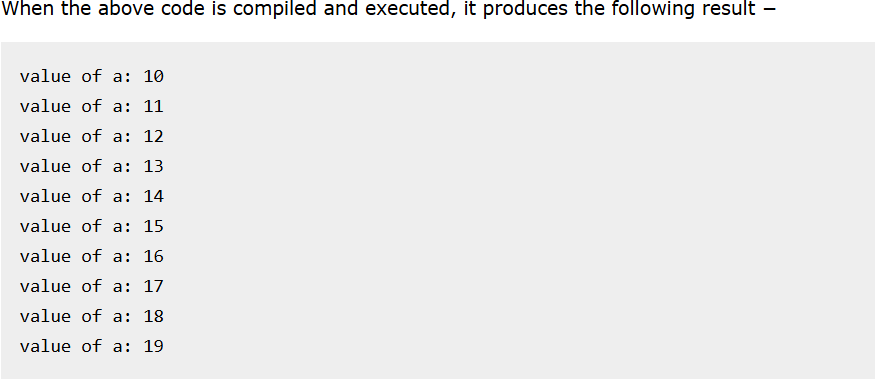
} while

( condition );

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again. This process repeats until the given condition becomes false.





# **CHAPTER 3 : ESSENCE OF OBJECTS AND CLASSES**

## C++ Object Oriented

**Object**

This is the basic unit of object-oriented programming. That is both data and function that operate on data are bundled as a unit called as object.

**Class**

When you define a class, you define a blueprint for an object. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

**Abstraction**

Data abstraction refers to, providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.

For example, a database system hides certain details of how data is stored and created and maintained. Similar way, C++ classes provide different methods to the outside world without giving internal detail about those methods and data.

**Encapsulation**

Encapsulation is placing the data and the functions that work on that data in the same place. While working with procedural languages, it is not always clear which functions work on which variables but object-oriented programming provides you framework to place the data and the relevant functions together in the same object.

**Inheritance**

One of the most useful aspects of object-oriented programming is code reusability. As the name suggests Inheritance is the process of forming a new class from an existing class that is from the existing class called as base class, new class is formed called as derived class.

This is a very important concept of object-oriented programming since this feature helps to reduce the code size.

**Polymorphism**

The ability to use an operator or function in different ways in other words giving different meaning or functions to the operators or functions is called polymorphism. Poly refers to many. That is a single function or an operator functioning in many ways different upon the usage is called polymorphism.

## C++ Class Definitions

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

A class definition starts with the keyword **class** followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. For example, we defined the Box data type using the keyword **class** as follows −

class Box

{

public: double length;

// Length of a box double breadth;

// Breadth of a box double height;

// Height of a box

};

The keyword **public** determines the access attributes of the members of the class that follows it. A public member can be accessed from outside the class anywhere within the scope of the class object. You can also specify the members of a class as **private** or **protected** which we will discuss in a sub-section.

## Define C++ Objects

A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types. Following statements declare two objects of class Box –

Box Box1; // Declare Box1 of type Box

Box Box2; // Declare Box2 of type Box

Both of the objects Box1 and Box2 will have their own copy of data members.

## Accessing the Data Members

The public data members of objects of a class can be accessed using the direct member access operator (.). Let us try the following example to make the things clear −

#include <iostream>

using namespace std;

class Box {

public:

double length; // Length of a box

double breadth; // Breadth of a box

double height; // Height of a box

};

int main() {

Box Box1; // Declare Box1 of type Box

double volume = 0.0; // Store the volume of a box here

// box 1 specification

Box1.height = 5.0;

Box1.length = 6.0;

Box1.breadth = 7.0;

// volume of box 1

volume = Box1.height \* Box1.length \* Box1.breadth;

cout << "Volume of Box1 : " << volume <<endl;

return 0;

}

When the above code is compiled and executed, it produces the following result −

Volume of Box1 : 210

It is important to note that private and protected members can not be accessed directly using direct member access operator (.). We will learn how private and protected members can be accessed.

## Class Member Functions

A member function of a class is a function that has its definition or its prototype within the class definition like any other variable. It operates on any object of the class of which it is a member, and has access to all the members of a class for that object.

Let us take previously defined class to access the members of the class using a member function instead of directly accessing them −

class Box {

public:

double length; // Length of a box

double breadth; // Breadth of a box

double height; // Height of a box

double getVolume(void);// Returns box volume

};

## Defining Class Member Functions

Member functions can be defined within the class definition or separately using **scope resolution operator, :** −. Defining a member function within the class definition declares the function **inline**, even if you do not use the inline specifier. So either you can define **Volume()** function as below −

**Defining Member Function inside the Class**

class Box {

public:

double length; // Length of a box

double breadth; // Breadth of a box

double height; // Height of a box

double getVolume(void) {

return length \* breadth \* height;

}

};

**Defining Member Function outside of the Class**

If you like, you can define the same function outside the class using the **scope resolution operator** (::) as follows –

double Box::getVolume(void) {

return length \* breadth \* height;

}

*Here, only important point is that you would have to use class name just before :: operator.*

## Calling (Accessing) Member Functions

A member function will be called using a dot operator (**.**) on a object where it will manipulate data related to that object only as follows −

Box myBox; // Create an object

myBox.getVolume(); // Call member function for the object

## Class Access Modifiers

C++ access modifiers are used for data hiding implementation. Data hiding is one of the important features of object-oriented programming, which allows the functions of a program to access directly the internal representation of a class type.The access restriction to the class members is specified by the labeled **public, private,** and **protected** sections within the class body. The keywords public, private, and protected are called access specifiers.

A class can have multiple public, protected, or private labeled sections. Each section remains in effect until either another section label or the closing right brace of the class body is seen. The default access for members and classes is private.

class Base {

public:

// public members go here

protected:

// protected members go here

private:

// private members go here

};

## Public Access Modifier

The public access modifier defines public data members and member functions that are accessible from anywhere outside the class but within a program. You can set and get the value of public variables without any member function.

## Private Access Modifier

The **private** access modifier defines private data members and member functions that cannot be accessed, or even viewed from outside the class. Only the class and friend functions can access private members.

By default all the members of a class would be private, for example in the following class **width** is a private member, which means until you label a member, it will be assumed a private member.

## Protected Access Modifier

The **protected** access modifier defines protected data members and member functions that are very similar to a private member, but it provides one additional benefit that they can be accessed in child classes, which are called derived classes.

# CHAPTER 4: C++ Inheritance

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base class**, and the new class is referred to as the **derived class**.

The idea of inheritance implements the is a relationship. For example, mammal IS-A animal, dog IS-A mammal hence dog IS-A animal as well and so on.

## Base and Derived Classes

A class can be derived from more than one classes, which means it can inherit data and functions from multiple base classes. To define a derived class, we use a class derivation list to specify the base class(es). A class derivation list names one or more base classes and has the form −

class derived-class: access-specifier base-class

Where access-specifier is one of **public, protected,** or **private**, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.

**EXAMPLE:**

#include <iostream>

using namespace std;

// Base class

class Shape {

public:

int width;

int height;

};

// Derived class

class Rectangle: public Shape {

public:

int getArea() {

return (width \* height);

}

};

int main(void) {

Rectangle Rect;

Rect.setWidth(5);

Rect.setHeight(7);

// Print the area of the object.

cout << "Total area: " << Rect.getArea() << endl;

return 0;

}

# CHAPTER 5:

# C++ Overloading (Operator and Function)

## Function Overloading in C++

Function overloading in C++ allows you to define multiple functions with the same name but different parameters. Function overloading is used to achieve polymorphism which is an important concept of object-oriented programming systems.Function overloading in C++ allows you to define multiple functions with the same name but different parameters. Function overloading is used to achieve polymorphism which is an important concept of object-oriented programming systems.

Example of Function Overloading

In the following example, we are defining three different functions with the same name but different parameters. This example demonstrates the implementation of function overloading –

#include<iostream>

using namespace std;

// Adding two integers (Function definition 1)

int addition(int a, int b) {

return a + b;

}

// Adding three integers (Function definition 2)

int addition(int a, int b, int c) {

return a + b + c;

}

// Adding two floating-point numbers (Function definition 3)

float addition(float a, float b) {

return a + b;

}

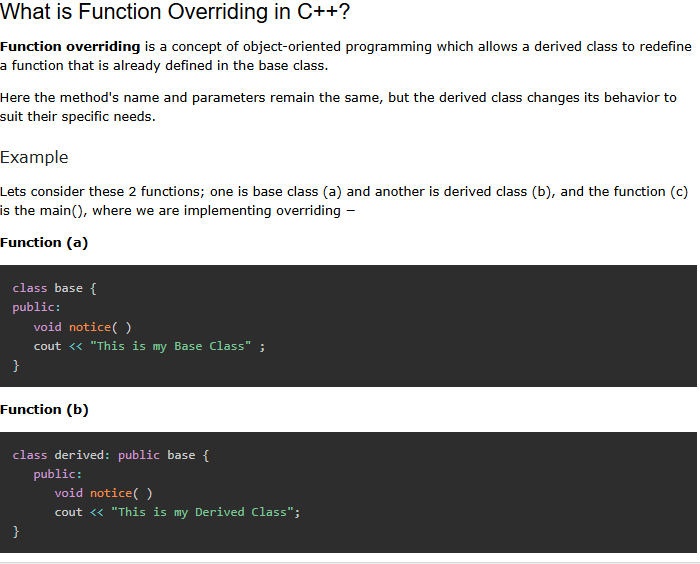
Output

30.8

60

30

# Function Overriding in C++



# Polymorphism in C++

The word polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

## Virtual Function

A virtual function is a function in a base class that is declared using the keyword virtual. Defining in a base class a virtual function, with another version in a derived class, signals to the compiler that we don't want static linkage for this function.

## Pure Virtual Functions

It is possible that you want to include a virtual function in a base class so that it may be redefined in a derived class to suit the objects of that class, but that there is no meaningful definition you could give for the function in the base class.

We can change the virtual function area() in the base class to the following –

Class Shape {

protected:

int width, height;

public:

Shape(int a = 0, int b = 0) {

width = a;

height = b;

}

// pure virtual function

virtual int area() = 0;

};

The = 0 tells the compiler that the function has no body and above virtual function will be called pure virtual function.

# C++ Class Constructor and Destructor

## The Class Constructor

A class constructor is a special member function of a class that is executed whenever we create new objects of that class.

A constructor will have exact same name as the class and it does not have any return type at all, not even void. Constructors can be very useful for setting initial values for certain member variables.

Example

Following example explains the concept of constructor –

*#include <iostream>*

*using namespace std;*

*class Line {*

*public:*

*void setLength( double len );*

*double getLength( void );*

*Line(); // This is the constructor*

*private:*

*double length;*

*};*

*// Member functions definitions including constructor*

*Line::Line(void) {*

*cout << "Object is being created" << endl;*

*}*

*void Line::setLength( double len ) {*

*length = len;*

*}*

*double Line::getLength( void ) {*

*return length;*

*}*

*// Main function for the program*

*int main() {*

*Line line;*

*// set line length*

*line.setLength(6.0);*

*cout << "Length of line : " << line.getLength() <<endl;*

*return 0;*

*}*

When the above code is compiled and executed, it produces the following result −

Object is being created

Length of line : 6

## Types of Constructors in C++

**1. Default Constructor**

A constructor that takes no arguments.

Automatically provided by the compiler if no constructor is defined.

class MyClass {

public:

MyClass() {

cout << "Default constructor called!" << endl;

}

};

2. Parameterized Constructor

A constructor that takes parameters to initialize objects with specific values.

class MyClass {

public:

int x;

MyClass(int a) {

x = a;

}

};

3. Copy Constructor

Initializes an object using another object of the same class.

Syntax: ClassName (const ClassName &obj)

**class MyClass {**

**public:**

**int x;**

**MyClass(int a) { x = a; }**

**MyClass(const MyClass &obj) {**

**x = obj.x;**

**}**

**};**

## The Class Destructor

A destructor is a special member function of a class that is executed whenever an object of it's class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class.

A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc.

Example

Following example explains the concept of destructor –

#include <iostream>

using namespace std;

class Line {

public:

void setLength( double len );

double getLength( void );

Line(); // This is the constructor declaration

~Line(); // This is the destructor: declaration

private:

double length;

};

// Member functions definitions including constructor

Line::Line(void) {

cout << "Object is being created" << endl;

}

Line::~Line(void) {

cout << "Object is being deleted" << endl;

}

void Line::setLength( double len ) {

length = len;

}

double Line::getLength( void ) {

return length;

}

// Main function for the program

int main() {

Line line;

// set line length

line.setLength(6.0);

cout << "Length of line : " << line.getLength() <<endl;

return 0;

}

When the above code is compiled and executed, it produces the following result −

Object is being created

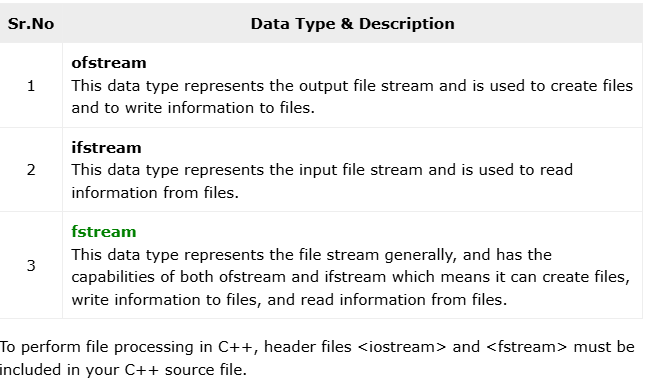
Length of line : 6

Object is being deleted

# C++ Files and Streams

So far, we have been using the iostream standard library, which provides cin and cout methods for reading from standard input and writing to standard output respectively.

This notes will teach you how to read and write from a file. This requires another standard C++ library called fstream, which defines three new data types –



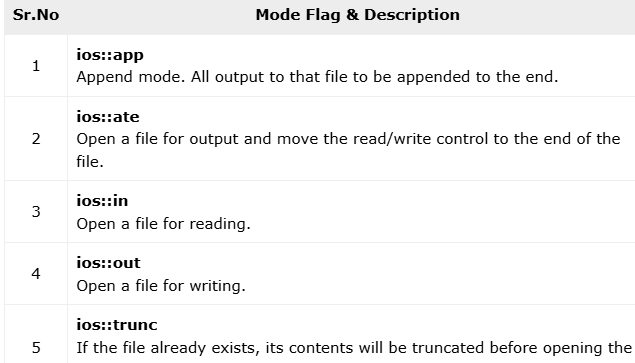
## Opening a File

A file must be opened before you can read from it or write to it. Either ofstream or fstream object may be used to open a file for writing. And ifstream object is used to open a file for reading purpose only.

Following is the standard syntax for open() function, which is a member of fstream, ifstream, and ofstream objects.

void open(const char \*filename, ios::openmode mode);

Here, the first argument specifies the name and location of the file to be opened and the second argument of the open() member function defines the mode in which the file should be opened.



## Closing a File

When a C++ program terminates it automatically flushes all the streams, release all the allocated memory and close all the opened files. But it is always a good practice that a programmer should close all the opened files before program termination.

Following is the standard syntax for close() function, which is a member of fstream, ifstream, and ofstream objects.

void close();