# What is a structuredmethodology

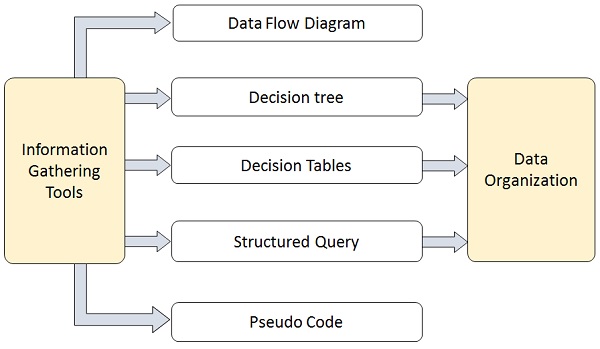
A structuredmethodologyis a development method that allows the analyst to understand the system and its activities in a logical way.

It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by the user.

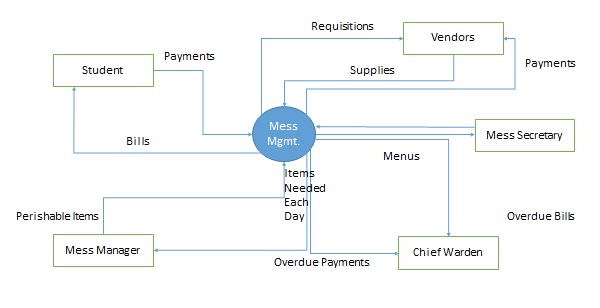
It has the following attributes −

* It is a graphic which specifies the presentation of the application.
* It divides the processes so that it gives a clear picture of system flow.
* It is logical rather than physical i.e., the elements of the system do not depend on vendor or hardware.
* It is an approach that works from high-level overviews to lower-level details.

# Tools fora structured methodology



# Final Output in a structured methodology

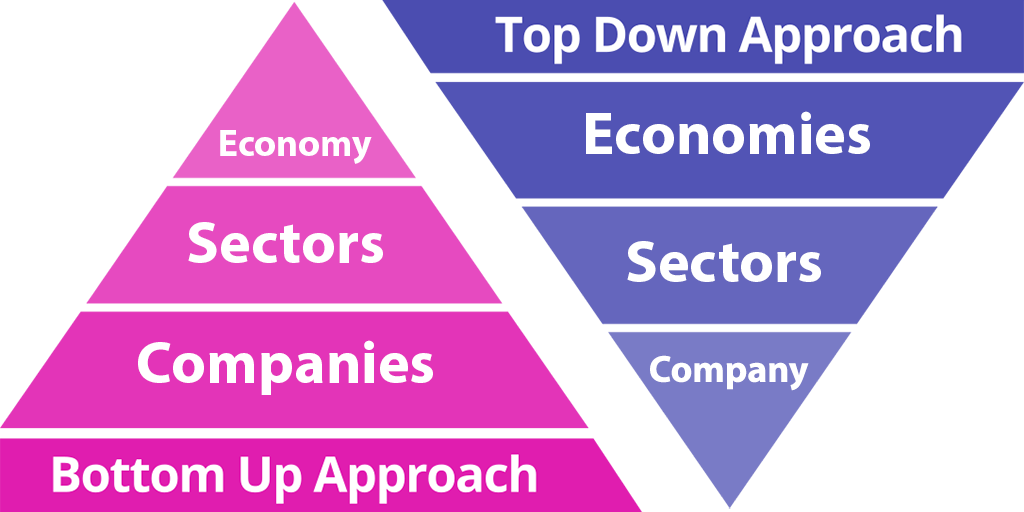


# What is a Top**-**Downapproach

A top-down approach (also known as stepwise design and in some cases used as a synonym of decomposition) is essentially the breaking down of a system to gain insight into its computational sub-systems in a reverse engineering fashion. In a top-down approach, an overview of the system is formulated, specifying but not detailing any first-level subsystems. Each subsystem is then refined in yet greater detail, sometimes in many additional subsystem levels, until the entire specification is reduced to base elements. A top-down model is often specified with the assistance of "black boxes", these make it easier to manipulate. However, black boxes may fail to elucidate elementary mechanisms or be detailed enough to realistically validate the model. Top-down approach starts with the big picture. It breaks down from there into smaller segments.

# What is a Bottom-Upapproach

A **bottom-up** approach is the piecing together of systems to give rise to more complex systems, thus making the original systems sub-systems of the emergent system. Bottom-up processing is a type of information processing based on incoming data from the environment to form a perception. From a Cognitive Psychology perspective, information enters the eyes in one direction (sensory input, or the "bottom"), and is then turned into an image by the brain that can be interpreted and recognized as a perception (output that is "built up" from processing to final cognition). In a bottom-up approach, the individual base elements of the system are first specified in great detail. These elements are then linked together to form larger subsystems, which then, in turn, are linked, sometimes in many levels, until a complete top-level system is formed. This strategy often resembles a "seed" model, whereby the beginnings are small but eventually grow in complexity and completeness. However, "organic strategies" may result in a tangle of elements and subsystems, developed in isolation and subject to local optimization as opposed to meeting a global purpose.



# Object-Orientation – A Paradigm Shift

Object orientated approach is a way of modeling software architecture so that it uses (depending on the language) classes, objects, interfaces, abstractions, access rules, and patterns to describe a software solution.  It allows for inheritance, code re-use, DRY (don't repeat yourself), clear separation of data from implementation and a clear way of organizing and creating a maintainable software solution for a team of developers.  Keep in mind that OOP means different things in different languages.  I will describe class-based object orientation as this is my preferred method of working with this method of programming.  (See: C#, Java, PHP) Objects in these languages come from classes through instantiation. (saying - give me a new Object from Class) - this is the way of transforming the object description into a usable object

# Object-Oriented Programming

Object-oriented programming (OOP) is a programming paradigm based upon objects (having both data and methods) that aims to incorporate the advantages of modularity and reusability. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

The important features of object-oriented programming are −

* The bottom-up approach in program design
* Programs organized around objects, grouped in classes
* Focus on data with methods to operate upon object’s data
* The interaction between objects through functions
* Reusability of design through the creation of new classes by adding features to existing classes

Some examples of object-oriented programming languages are C++, Java, Smalltalk, Delphi, C#, Perl, Python, Ruby, and PHP.

Grady Booch has defined object-oriented programming as *“a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships”*.

# What is C++

“C++ is a statically-typed, free-form, (usually) compiled, multi-paradigm, intermediate-level general-purpose middle-level programming language.”

In simple terms, C++ is a sophisticated, efficient and a general-purpose programming language based on C. It was developed by Bjarne Stroustrup in 1979.

Many of today’s operating systems, system drivers, browsers and games use C++ as their core language. This makes C++ one of the most popular languages today.

Since it is an enhanced/extended version of C programming language, C and C++ are often denoted together as C/C++

While Bjarne Stroustrupwas working in AT&T Bell Labs in 1979, he faced difficulties in analyzing UNIX kernel for distributed systems. The current languages were either too slow or too low level. So, he set forward to create a new language.

For building this language, he chose C. Why C? Because it is a general-purpose language and is very efficient as well as fast in its operations.

He used his knowledge of object-oriented model from SIMULA and began working on class extensions to C. His aim was to create a language with far higher level of abstraction while retaining the efficiency of C.

This new programming language was named C with Classes, but was later renamed to C++ (++ refers to the increment operator in C).

#### C++98

When C++ was first released in 1985, there were no official standards released. It was only until 1998 that C++ was first standardized which was known as C++98.

#### C++03

In 2003, a new version of C++ standard was published. C++03 wasn’t really a new standard altogether but a bug fix release identified with C++98 “**to ensure greater consistency and portability**”.

#### C++11 (C++0x)

The next major standard for C++ was released in 2011 and it was named C++11. Since, C++ committee was sure this update would be released within 2009, they unofficially named it C++0x. Later, when they didn’t, Stroustrup joked that C++0x went hexadecimal - C++0xB (C++11). Nice save.

#### C++14 (C++1y)

C++14 is the current iteration of C++ released in 2014. Like C++03, it included mainly bug fixes and simple improvements to C++11.

#### C++17 (C++1z)

The supposedly next iteration to C++ which is planned to be rolled out in 2017. It is expected to have many new features. Most of the features planned for this version are already completed.

### Features of C++

Being a general-purpose language, C++ is undoubtedly feature-rich. Going through all the features will take you some time but, as a beginner, below are the most important features you should know.

1. **C++ is fast**  
     
   Since, C++ is an extended version of C, the C part of it is very low level.  
     
   This offers a huge boost in speed that high level languages like Python, Java don’t give you.
2. **C++  is statically typed**  
     
   C++ is a statically typed programming language.  
     
   In simple terms, C++ doesn’t allow the compiler to make assumptions about the type of data e.g. 10 is different from “10” and you have to let C++ know which one you are talking about.  
     
   This helps the compiler catch errors and bugs before execution of the program.
3. **C++ is a multi-paradigm programming language**  
     
   C++ supports at least 7 different styles of programming and gives developers the freedom to choose one at their will.  
     
   Unlike Java and Python, you don’t need to use objects to solve every task (if it’s not necessary).  
     
   You can choose the programming style that fits your use case.
4. **Object oriented programming with C++**  
     
   Object oriented programming helps you solve a complex problem intuitively.  
     
   With its use in C++, you are able to divide these complex problems into smaller sets by creating objects.
5. **Power of standard library (Standard template library - STL)**  
     
   The power of C++ extends with the use of standard libraries contained in it.  
     
   These libraries contain efficient algorithms that you use extensively while coding.  
     
   This saves ample amount of programming effort, which otherwise would have been wasted **reinventing the wheel**.

.

First Programing

#include <iostream>

using namespace std;

int main()

{

cout<<"Hello World!";

return 0;

}

What is #include <iostream>?

This statement includes the header file into the application so that you are able to use the operations included in them. Also, you can create your own header files and include them in your program using the #include.

What is iostream?

iostream is what you call the header file. It is a standard C++ input/output library file.

It comes packaged with the compiler/IDE and contain mechanisms to get the information from the user and print same or added information to a file, screen or any other media.

What is #include?

The #include iostream file, into the program. This ensures that now you’re able to use the operations, iostream operations (like: taking input from user, displaying output on the screen), in the program.

What is using namespace std;”?

The statement is intuitive in itself, you are “using” the “namespace” “std” in your file.

We use the namespace std to make it easier to reference operations included in that namespace.

If we hadn’t used the namespace, we’d have written std::cout instead of cout. This tells the compiler that every cout is actually std::cout.

What’s a namespace?

It’s a region where your code resides. It limits or expands the scope of your code to one or more files.

Why do you use namespace?

Like two persons can have the same name, variables and functions in C++ can have same names as well. The use of namespace is to avoid the confusion of which variables/functions you are referencing to.

What is std?

std is a standard namespace used in C++.

Semicolon ”;”

The semicolon is a terminal. It terminates a statement. When missed or incorrectly used, it will cause a lot of issues.

int main () { }

As the name suggests, it is the main function of the program. The code inside { } is called the body and is executed first when you run your C++ program.

It is one code that is mandatory in a C++ program. If you just have this line of code alone, your program will be valid.

cout<< “Hello World!”;

This statement prints “Hello World!” onto the output screen.

The cout is an object of standard output stream. What this means is, it outputs/prints the data after <<, i.e. Hello World! into a stream (in this case, the output screen).

What is a stream?

Stream is basically a sequence of objects, usually bytes. It can describe files, input/output terminal, sockets, etc.

What is <<?

<< is the insertion operator used to write formatted data into the stream.

What is return 0;?

This statement returns 0 ‘zero’.

This is called a return statement. It isn’t mandatory to return anything from the main () function but is rather a convention. If not return, the compiler returns a status automatically.

Why zero in return statement?

It denotes Exit status of the application that basically the tells system “The program worked fine.”

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

C++ Basic Input/Output?

C++ I/O operation is using the stream concept. Stream is the sequence of bytes or flow of data. It makes the performance fast.

If bytes flow from main memory to device like printer, display screen, or a network connection, etc, this is called as **output operation.**

If bytes flow from device like printer, display screen, or a network connection, etc to main memory, this is called as **input operation.**

# I/O Library Header Files

|  |  |
| --- | --- |
| <iostream> | It is used to define the **cout, cin and cerr** objects, which correspond to standard output stream, standard input stream and standard error stream, respectively. |
| <iomanip> | It is used to declare services useful for performing formatted I/O, such as **setprecision and setw.** |
| <fstream> | It is used to declare services for user-controlled file processing. |

Cout:The **cout** is a predefined object of **ostream** class. It is connected with the standard output device, which is usually a display screen. The cout is used in conjunction with stream insertion operator (<<) to display the output on a console

# Cin:The **cin** is a predefined object of **istream** class. It is connected with the standard input device, which is usually a keyboard. The cin is used in conjunction with stream extraction operator (>>) to read the input from a console.

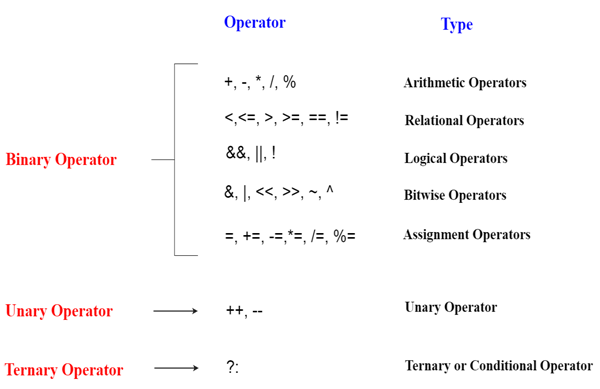
|  |  |
| --- | --- |
| **Types** | **Data Types** |
| Basic Data Type | int, char, float, double, etc |
| Derived Data Type | array, pointer, etc |
| Enumeration Data Type | enum |
| User Defined Data Type | structure |

# C++ Operators

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise etc.

There are following types of operators to perform different types of operations in C language.

* Arithmetic Operators
* Relational Operators
* Logical Operators
* Bitwise Operators
* Assignment Operator
* Unary operator
* Ternary or Conditional Operator
* Misc Operator



# Precedence of Operators in C++

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

# 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().

# 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.

### Object

Any entity that has state and behaviour is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

### Class

**Collection of objects** is called class. It is a logical entity.

### Inheritance

**When one object acquires all the properties and behaviours of parent object** i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

### Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In C++, we use Function overloading and Function overriding to achieve polymorphism.

### Abstraction

**Hiding internal details and showing functionality** is known as abstraction. For example: phone call, we don't know the internal processing.

In C++, we use abstract class and interface to achieve abstraction.

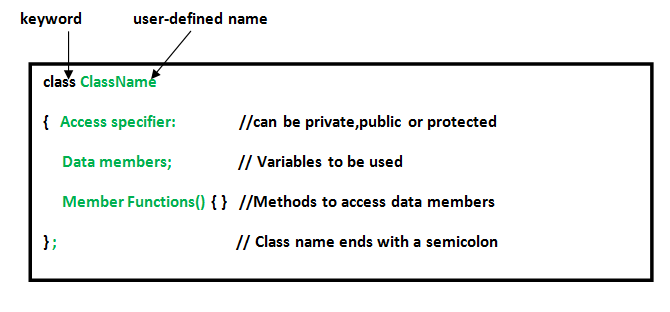
### Encapsulation

**Binding (or wrapping) code and data together into a single unit is known as encapsulation.** For example: capsule, it is wrapped with different medicines.

**Class:** The building block of C++ that leads to Object Oriented programming is a **Class**. It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.  
For Example: Consider the Class of **Cars**. There may be many cars with different names and brand but all of them will share some common properties like all of them will have *4 wheels*, *Speed Limit*, *Mileage range* etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

* A Class is a user defined data-type which has data members and member functions.
* Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.
* In the above example of class *Car*, the data member will be *speed limit*, *mileage* etc and member functions can be *apply brakes*, *increase speed* etc.

An **Object** is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.



**Member Functions in Classes**

There are 2 ways to define a member function:

* Inside class definition
* Outside class definition

To define a member function outside the class definition we have to use the scope resolution :: operator along with class name and function name.

# Access Modifiers in C++

Access modifiers are used to implement an important feature of Object Oriented Programming known as **Data Hiding**.

Access modifiers or Access Specifiers in a class are used to set the accessibility of the class members. That is, it sets some restrictions on the class members not to get directly accessed by the outside functions.

There are 3 types of access modifiers available in C++:

1. **Public**
2. **Private**
3. **Protected**

**Public**: All the class members declared under public will be available to everyone. The data members and member functions declared public can be accessed by other classes too. The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

**Private**: The class members declared as **private** can be accessed only by the functions inside the class. They are not allowed to be accessed directly by any object or function outside the class. Only the member functions or the friend functions are allowed to access the private data members of a class.

**Protected**: Protected access modifier is similar to that of private access modifiers, the difference is that the class member declared as Protected are inaccessible outside the class but they can be accessed by any subclass(derived class) of that class.

**What is constructor?**  
A constructor is a member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object(instance of class) create. It is special member function of the class.

**How constructors are different from a normal member function?**

A constructor is different from normal functions in following ways:

* Constructor has same name as the class itself
* Constructors don’t have return type
* A constructor is automatically called when an object is created.
* If we do not specify a constructor, C++ compiler generates a default constructor for us (expects no parameters and has an empty body)

**Types of Constructors**

1. **Default** **Constructors:** Default constructor is the constructor which doesn’t take any argument. It has no parameters.
2. **Parameterized Constructors:** It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor’s body, use the parameters to initialize the object.
3. **Copy constructors:** A copy constructor is a member function which initializes an object using another object of the same class.

## When Copy Constructor is called

Copy Constructor is called in the following scenarios:

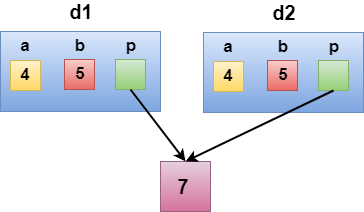
* When we initialize the object with another existing object of the same class type. For example, Student s1 = s2, where Student is the class.
* When the object of the same class type is passed by value as an argument.
* When the function returns the object of the same class type by value.

## Two types of copies are produced by the constructor:

* Shallow copy
* Deep copy

## Shallow Copy

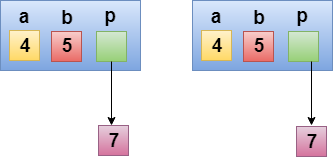
* The default copy constructor can only produce the shallow copy.
* A Shallow copy is defined as the process of creating the copy of an object by copying data of all the member variables as it is.



In the above case, a programmer has not defined any constructor, therefore, the statement **Demo d2 = d1;** calls the default constructor defined by the compiler. The default constructor creates the exact copy or shallow copy of the existing object. Thus, the pointer p of both the objects point to the same memory location. Therefore, when the memory of a field is freed, the memory of another field is also automatically freed as both the fields point to the same memory location. This problem is solved by the **user-defined constructor** that creates the **Deep copy**.

## Deep copy

Deep copy dynamically allocates the memory for the copy and then copies the actual value, both the source and copy have distinct memory locations. In this way, both the source and copy are distinct and will not share the same memory location. Deep copy requires us to write the user-defined constructor.



In the above case, a programmer has defined its own constructor, therefore the statement **Demo d2 = d1;** calls the copy constructor defined by the user. It creates the exact copy of the value types data and the object pointed by the pointer p. Deep copy does not create the copy of a reference type variable.

Destructor is another special member function that is called by the compiler when the scope of the object ends.

# C++ this Pointer

In C++ programming, **this** is a keyword that refers to the current instance of the class. There can be 3 main usage of this keyword in C++.

* It can be used **to pass current object as a parameter to another method.**
* It can be used **to refer current class instance variable.**
* It can be used **to declare indexers.**

# C++ static

In C++, static is a keyword or modifier that belongs to the type not instance. So instance is not required to access the static members. In C++, static can be field, method, constructor, class, properties, operator and event.

## Advantage of C++ static keyword

**Memory efficient:** Now we don't need to create instance for accessing the static members, so it saves memory. Moreover, it belongs to the type, so it will not get memory each time when instance is created.

## C++ Static Field

A field which is declared as static is called static field. Unlike instance field which gets memory each time whenever you create object, there is only one copy of static field created in the memory. It is shared to all the objects.

It is used to refer the common property of all objects such as rateOfInterest in case of Account, companyName in case of Employee etc.

# C++ Enumeration

Enum in C++ is a data type that contains fixed set of constants.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY) , directions (NORTH, SOUTH, EAST and WEST) etc. The C++ enum constants are static and final implicitly.

C++ Enums can be thought of as classes that have fixed set of constants.

## Points to remember for C++ Enum

* enum improves type safety
* enum can be easily used in switch
* enum can be traversed
* enum can have fields, constructors and methods
* enum may implement many interfaces but cannot extend any class because it internally extends Enum class

# C++ Friend function

If a function is defined as a friend function in C++, then the protected and private data of a class can be accessed using the function.

By using the keyword friend compiler knows the given function is a friend function.

For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword friend.

**Characteristics of a Friend function:**

* The function is not in the scope of the class to which it has been declared as a friend.
* It cannot be called using the object as it is not in the scope of that class.
* It can be invoked like a normal function without using the object.
* It cannot access the member names directly and has to use an object name and dot membership operator with the member name.
* It can be declared either in the private or the public part.

# C++ Inheritance

In C++, inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. In such way, you can reuse, extend or modify the attributes and behaviors which are defined in other class.

In C++, the class which inherits the members of another class is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.

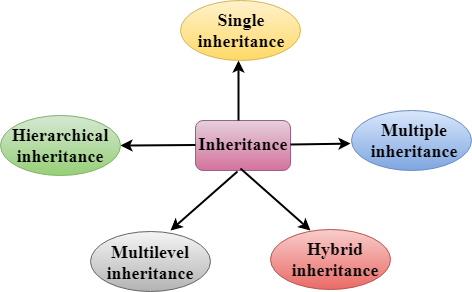
## Advantage of C++ Inheritance

**Code reusability:** Now you can reuse the members of your parent class. So, there is no need to define the member again. So less code is required in the class.

## Types Of Inheritance

**C++ supports five types of inheritance:**

* Single inheritance
* Multiple inheritance
* Hierarchical inheritance
* Multilevel inheritance
* Hybrid inheritance



## Derived Classes

A Derived class is defined as the class derived from the base class.

The Syntax of Derived class:

1. class derived\_class\_name :: visibility-mode base\_class\_name
2. {
3. // body of the derived class.
4. }

**Where,**

**derived\_class\_name:** It is the name of the derived class.

**visibility mode:** The visibility mode specifies whether the features of the base class are publicly inherited or privately inherited. It can be public or private.

**base\_class\_name:** It is the name of the base class.

* When the base class is privately inherited by the derived class, public members of the base class becomes the private members of the derived class. Therefore, the public members of the base class are not accessible by the objects of the derived class only by the member functions of the derived class.
* When the base class is publicly inherited by the derived class, public members of the base class also become the public members of the derived class. Therefore, the public members of the base class are accessible by the objects of the derived class as well as by the member functions of the base class.

### Note:

* In C++, the default mode of visibility is private.
* The private members of the base class are never inherited.

## C++ Single Inheritance

**Single inheritance** is defined as the inheritance in which a derived class is inherited from the only one base class.

C++ Inheritance

Where 'A' is the base class, and 'B' is the derived class.

## C++ Single Level Inheritance Example: Inheriting Fields

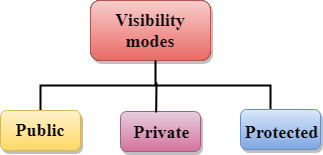
When one class inherits another class, it is known as single level inheritance. Let's see the example of single level inheritance which inherits the fields only.

## How to make a Private Member Inheritable

The private member is not inheritable. If we modify the visibility mode by making it public, but this takes away the advantage of data hiding.

C++ introduces a third visibility modifier, i.e., **protected**. The member which is declared as protected will be accessible to all the member functions within the class as well as the class immediately derived from it.

**Visibility modes can be classified into three categories:**



* **Public**: When the member is declared as public, it is accessible to all the functions of the program.
* **Private**: When the member is declared as private, it is accessible within the class only.
* **Protected**: When the member is declared as protected, it is accessible within its own class as well as the class immediately derived from it.

## Visibility of Inherited Members

|  |  |  |  |
| --- | --- | --- | --- |
| **Base class visibility** | **Derived class visibility** | | |
| **Public** | **Private** | **Protected** |
| Private | Not Inherited | Not Inherited | Not Inherited |
| Protected | Protected | Private | Protected |
| Public | Public | Private | Protected |

## C++ Multilevel Inheritance

**Multilevel inheritance** is a process of deriving a class from another derived class.

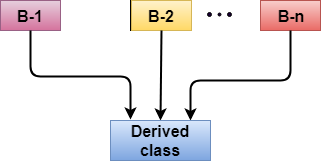
C++ Inheritance

## C++ Multi Level Inheritance Example

When one class inherits another class which is further inherited by another class, it is known as multi level inheritance in C++. Inheritance is transitive so the last derived class acquires all the members of all its base classes.

## C++ Multiple Inheritance

**Multiple inheritance** is the process of deriving a new class that inherits the attributes from two or more classes.

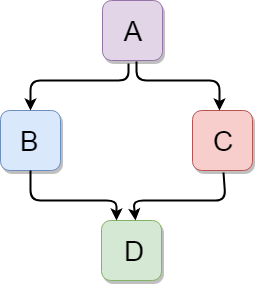


## Ambiquity Resolution in Inheritance

Ambiguity can be occurred in using the multiple inheritance when a function with the same name occurs in more than one base class.

## C++ Hybrid Inheritance

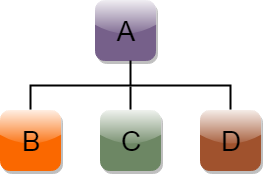
Hybrid inheritance is a combination of more than one type of inheritance.



Let's see a simple example:

## C++ Hierarchical Inheritance

Hierarchical inheritance is defined as the process of deriving more than one class from a base class.



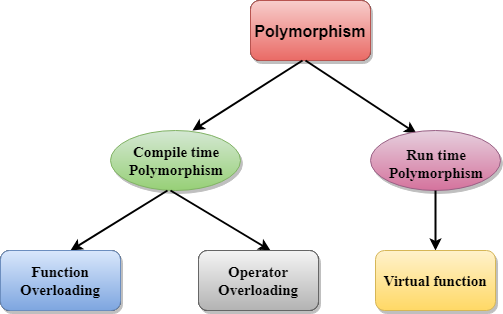
# C++ Aggregation (HAS-A Relationship)

In C++, aggregation is a process in which one class defines another class as any entity reference. It is another way to reuse the class. It is a form of association that represents HAS-A relationship.

# C++ Polymorphism

The term "Polymorphism" is the combination of "poly" + "morphs" which means many forms. It is a greek word. In object-oriented programming, we use 3 main concepts: inheritance, encapsulation, and polymorphism.

**There are two types of polymorphism in C++:**



* **Compile time polymorphism**: The overloaded functions are invoked by matching the type and number of arguments. This information is available at the compile time and, therefore, compiler selects the appropriate function at the compile time. It is achieved by function overloading and operator overloading which is also known as static binding or early binding. Now,

**Run time polymorphism**: Run time polymorphism is achieved when the object's method is invoked at the run time instead of compile time. It is achieved by method overriding which is also known as dynamic binding or late binding.

# C++ Overloading (Function and Operator)

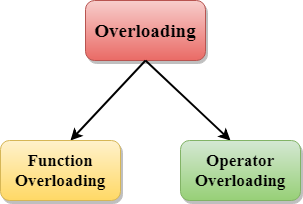
If we create two or more members having the same name but different in number or type of parameter, it is known as C++ overloading. In C++, we can overload:

* methods,
* constructors, and
* indexed properties

It is because these members have parameters only.

## Types of overloading in C++ are:

* Function overloading
* Operator overloading



## C++ Function Overloading

Function Overloading is defined as the process of having two or more function with the same name, but different in parameters is known as function overloading in C++. In function overloading, the function is redefined by using either different types of arguments or a different number of arguments. It is only through these differences compiler can differentiate between the functions.

The **advantage** of Function overloading is that it increases the readability of the program because you don't need to use different names for the same action.

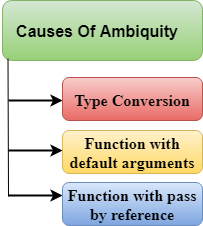
## Function Overloading and Ambiguity

When the compiler is unable to decide which function is to be invoked among the overloaded function, this situation is known as **function overloading**.

When the compiler shows the ambiguity error, the compiler does not run the program.

**Causes of Function Overloading:**

* Type Conversion.
* Function with default arguments.
* Function with pass by reference.



## C++ Operators Overloading

Operator overloading is a compile-time polymorphism in which the operator is overloaded to provide the special meaning to the user-defined data type. Operator overloading is used to overload or redefines most of the operators available in C++. It is used to perform the operation on the user-defined data type. For example, C++ provides the ability to add the variables of the user-defined data type that is applied to the built-in data types.

The advantage of Operators overloading is to perform different operations on the same operand.

**Operator that cannot be overloaded are as follows:**

* Scope operator (::)
* Sizeof
* member selector(.)
* member pointer selector(\*)
* ternary operator(?:)

## Rules for Operator Overloading

* Existing operators can only be overloaded, but the new operators cannot be overloaded.
* The overloaded operator contains atleast one operand of the user-defined data type.
* We cannot use friend function to overload certain operators. However, the member function can be used to overload those operators.
* When unary operators are overloaded through a member function take no explicit arguments, but, if they are overloaded by a friend function, takes one argument.
* When binary operators are overloaded through a member function takes one explicit argument, and if they are overloaded through a friend function takes two explicit arguments.

# C++ Function Overriding

If derived class defines same function as defined in its base class, it is known as function overriding in C++. It is used to achieve runtime polymorphism. It enables you to provide specific implementation of the function which is already provided by its base class.

# C++ virtual function

* A C++ virtual function is a member function in the base class that you redefine in a derived class. It is declared using the virtual keyword.
* It is used to tell the compiler to perform dynamic linkage or late binding on the function.
* There is a necessity to use the single pointer to refer to all the objects of the different classes. So, we create the pointer to the base class that refers to all the derived objects. But, when base class pointer contains the address of the derived class object, always executes the base class function. This issue can only be resolved by using the 'virtual' function.
* A 'virtual' is a keyword preceding the normal declaration of a function.
* When the function is made virtual, C++ determines which function is to be invoked at the runtime based on the type of the object pointed by the base class pointer.

## Late binding or Dynamic linkage

In late binding function call is resolved during runtime. Therefore compiler determines the type of object at runtime, and then binds the function call.

**Rules of Virtual Function**

* Virtual functions must be members of some class.
* Virtual functions cannot be static members.
* They are accessed through object pointers.
* They can be a friend of another class.
* A virtual function must be defined in the base class, even though it is not used.
* The prototypes of a virtual function of the base class and all the derived classes must be identical. If the two functions with the same name but different prototypes, C++ will consider them as the overloaded functions.
* We cannot have a virtual constructor, but we can have a virtual destructor
* Consider the situation when we don't use the virtual keyword.

## Pure Virtual Function

* A virtual function is not used for performing any task. It only serves as a placeholder.
* When the function has no definition, such function is known as "**do-nothing**" function.
* The "**do-nothing**" function is known as a **pure virtual function**. A pure virtual function is a function declared in the base class that has no definition relative to the base class.
* A class containing the pure virtual function cannot be used to declare the objects of its own, such classes are known as abstract base classes.
* The main objective of the base class is to provide the traits to the derived classes and to create the base pointer used for achieving the runtime polymorphism.

# Interfaces in C++ (Abstract Classes)

Abstract classes are the way to achieve abstraction in C++. Abstraction in C++ is the process to hide the internal details and showing functionality only. Abstraction can be achieved by two ways:

1. **Abstract class**
2. **Interface**

Abstract class and interface both can have abstract methods which are necessary for abstraction.

## C++ Abstract class

In C++ class is made abstract by declaring at least one of its functions as <>strong>pure virtual function. A pure virtual function is specified by placing "= 0" in its declaration. Its implementation must be provided by derived classes.

Let's see an example of abstract class in C++ which has one abstract method draw(). Its implementation is provided by derived classes: Rectangle and Circle. Both classes have different implementation.

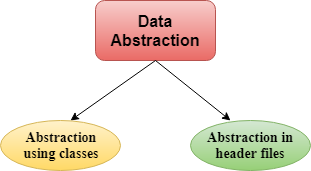
# Data Abstraction in C++

* Data Abstraction is a process of providing only the essential details to the outside world and hiding the internal details, i.e., representing only the essential details in the program.
* Data Abstraction is a programming technique that depends on the seperation of the interface and implementation details of the program.
* Let's take a real life example of AC, which can be turned ON or OFF, change the temperature, change the mode, and other external components such as fan, swing. But, we don't know the internal details of the AC, i.e., how it works internally. Thus, we can say that AC seperates the implementation details from the external interface.
* C++ provides a great level of abstraction. For example, pow() function is used to calculate the power of a number without knowing the algorithm the function follows.

In C++ program if we implement class with private and public members then it is an example of data abstraction.

**Data Abstraction can be achieved in two ways:**

* Abstraction using classes
* Abstraction in header files.



**Abstraction using classes:** An abstraction can be achieved using classes. A class is used to group all the data members and member functions into a single unit by using the access specifiers. A class has the responsibility to determine which data member is to be visible outside and which is not.

**Abstraction in header files:** An another type of abstraction is header file. For example, pow() function available is used to calculate the power of a number without actually knowing which algorithm function uses to calculate the power. Thus, we can say that header files hides all the implementation details from the user.

**Access Specifiers Implement Abstraction:**

* **Public specifier:** When the members are declared as public, members can be accessed anywhere from the program.
* **Private specifier:** When the members are declared as private, members can only be accessed only by the member functions of the class.

## Advantages Of Abstraction:

* Implementation details of the class are protected from the inadvertent user level errors.
* A programmer does not need to write the low level code.
* Data Abstraction avoids the code duplication, i.e., programmer does not have to undergo the same tasks every time to perform the similar operation.
* The main aim of the data abstraction is to reuse the code and the proper partitioning of the code across the classes.
* Internal implementation can be changed without affecting the user level code.

# C++ Namespaces

Namespaces in C++ are used to organize too many classes so that it can be easy to handle the application.

For accessing the class of a namespace, we need to use namespacename::classname. We can use **using** keyword so that we don't have to use complete name all the time.

In C++, global namespace is the root namespace. The global::std will always refer to the namespace "std" of C++ Framework.