بسم الله الرحمن الرحيم

مقام معظم رهبری:

علم، پایه‌ی پیشرفت همه‌جانبه‌ی یک کشور است.

۱۳۹۰/۱۱/۱۴

Qt Training in C++

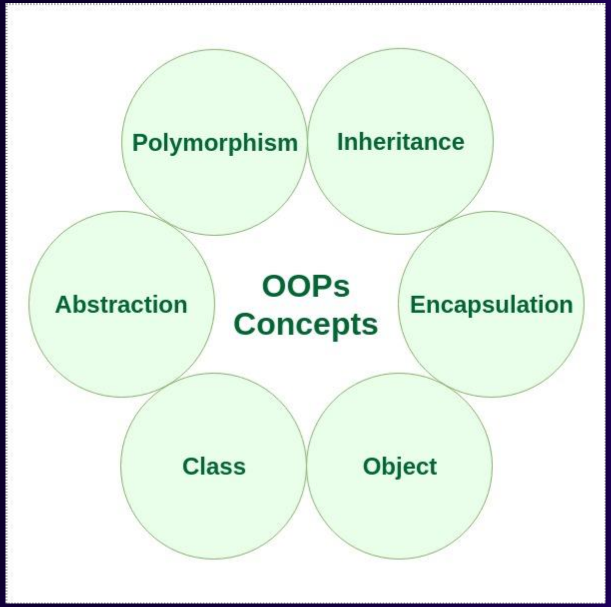
Lecturer: Ali Panahi

Spring 1402

Object Oriented Programming in C++

Object Oriented

- Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc. in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.
- Basic concepts:
  - Class
  - Objects
  - Encapsulation
  - Abstraction
  - Polymorphism
  - Inheritance
  - Dynamic Binding
  - Message Passing



Object Oriented Programming in C++

Class

- A Class is a user-defined data type that has data members and member functions.
- Data members are the data variables and member functions are the functions used to manipulate these variables together these data members and member functions define the properties and behavior of the objects in a Class.
  - For Example: Consider the Class of Cars. There may be many cars with different names and brands but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range, etc. So here, the Car is the class, and wheels, speed limits, and mileage are their properties.

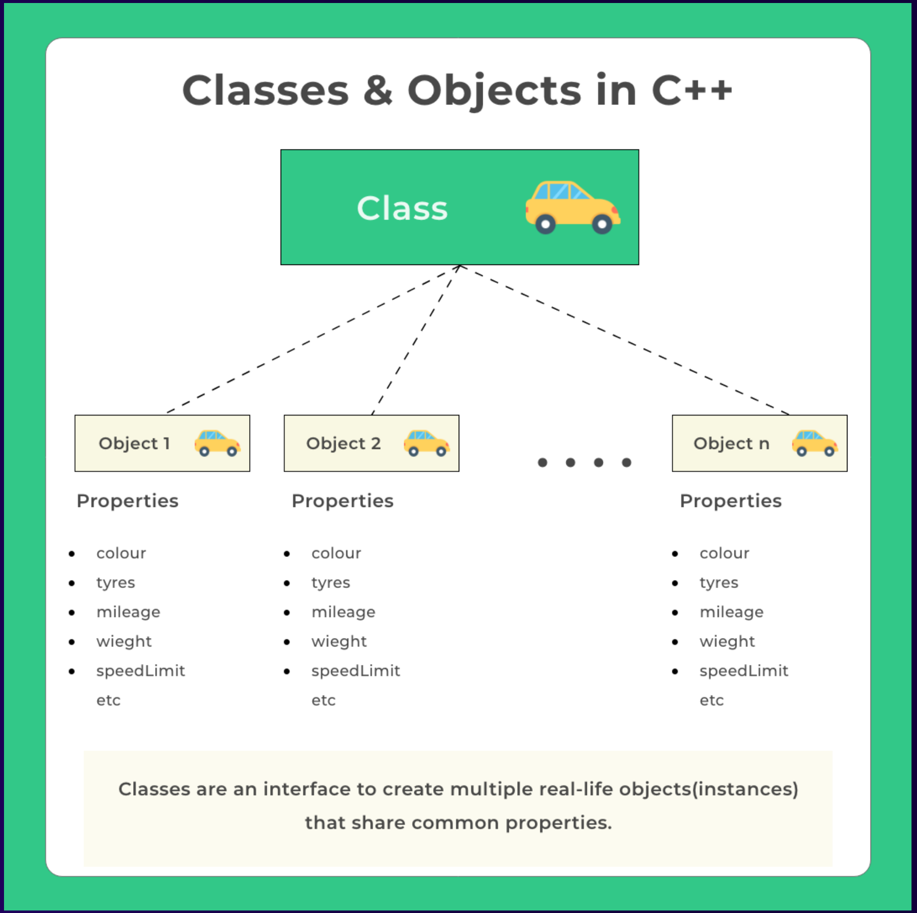
Object Oriented Programming in C++

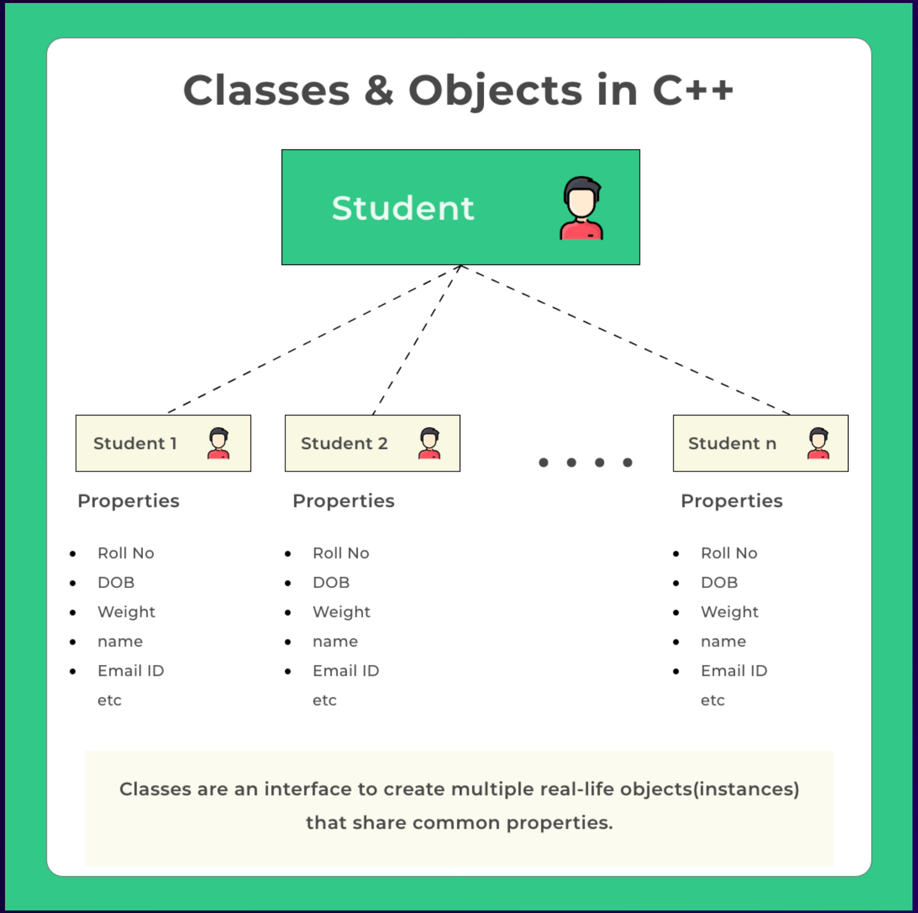
Object

- An Object is an identifiable entity with some characteristics and behavior. 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.
- When a program is executed the objects interact by sending messages to one another. Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other's data or code, it is sufficient to know the type of message accepted and the type of response returned by the objects.

Object Oriented Programming in C++

Class vs Object

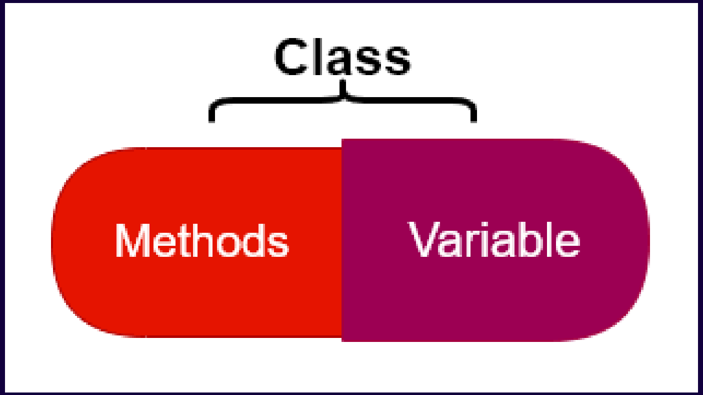




Object Oriented Programming in C++

Encapsulation

- Encapsulation is an Object Oriented Programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse. Data encapsulation led to the important OOP concept of data hiding.
- Data encapsulation is a mechanism of bundling the data, and the functions that use them and data abstraction is a mechanism of exposing only the interfaces and hiding the implementation details from the user.



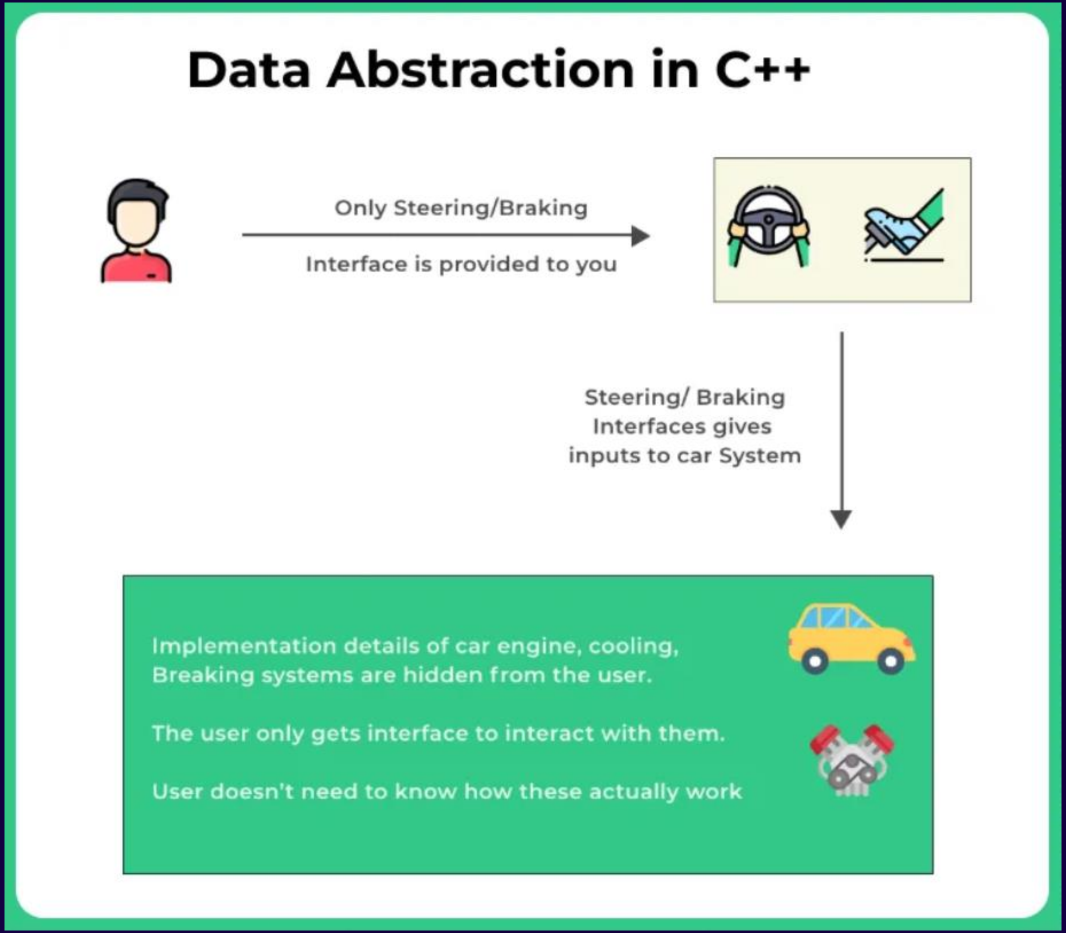
Object Oriented Programming in C++

Abstraction

- Data abstraction is one of the most essential and important features of object-oriented programming in C++. Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.
  - Abstraction using Classes: We can implement Abstraction in C++ using classes. The class helps us to group data members and member functions using available access specifiers. A Class can decide which data member will be visible to the outside world and which is not.
  - Abstraction in Header files: One more type of abstraction in C++ can be header files. For example, consider the pow() method present in math.h header file. Whenever we need to calculate the power of a number, we simply call the function pow() present in the math.h header file and pass the numbers as arguments without knowing the underlying algorithm according to which the function is actually calculating the power of numbers.

Object Oriented Programming in C++

Abstraction


Data Abstraction in C++
Only Steering/Braking
Interface is provided to you
Steering/ Braking
Interfaces gives
inputs to car System
Implementation details of car engine, cooling,
Breaking systems are hidden from the user.
The user only gets interface to interact with them.
User doesn't need to know how these actually work

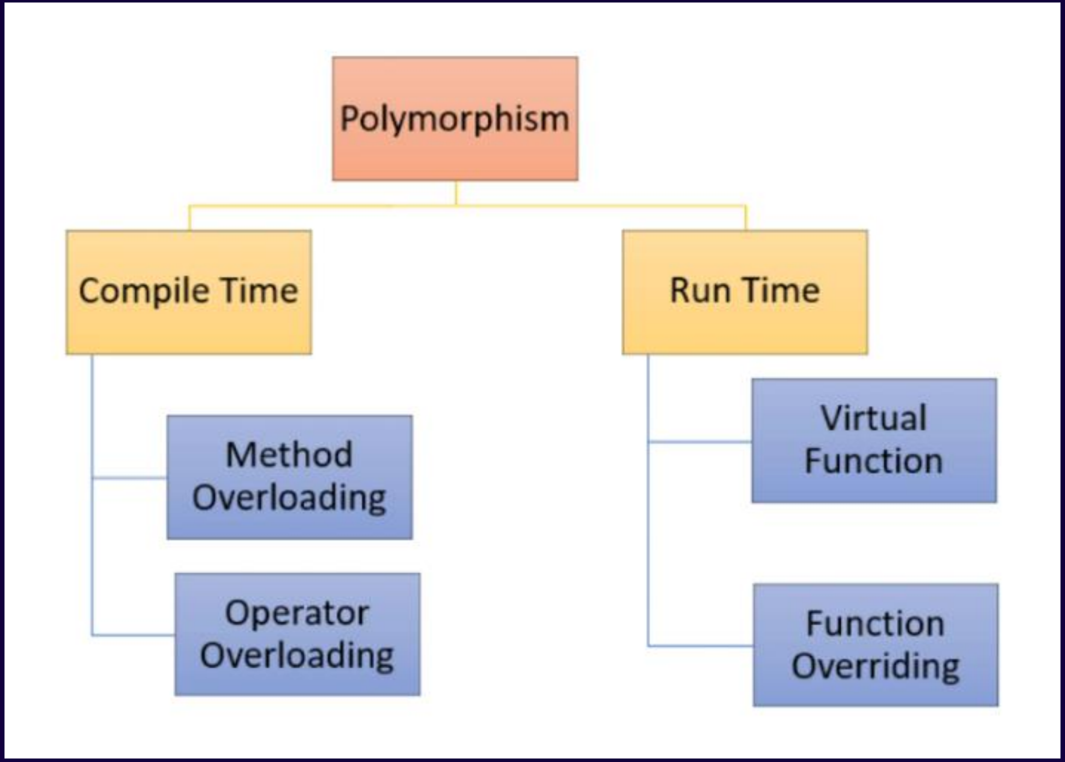
Object Oriented Programming in C++

Polymorphism

- Polymorphism in C++ means, the same entity (function or object) behaves differently in different scenarios.
  1. Compile Time Polymorphism
     - Function Overloading
     - Operator Overloading
  2. Runtime Polymorphism
     - Function overriding
     - Virtual Function

Object Oriented Programming in C++

Polymorphism



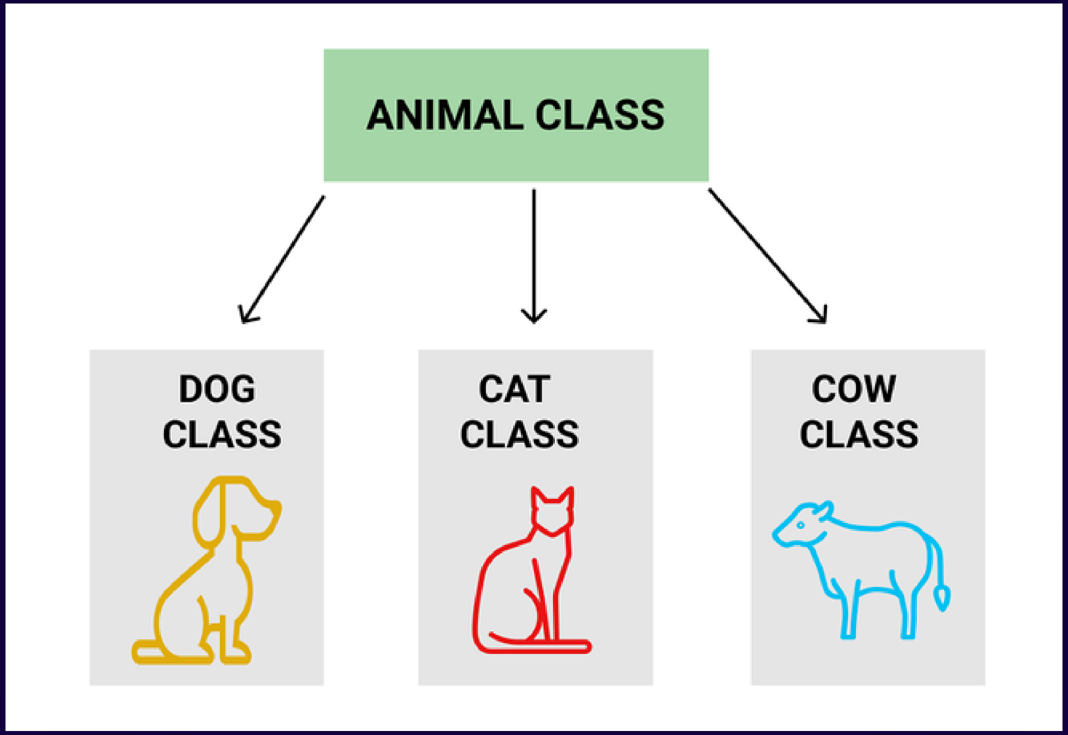
Object Oriented Programming in C++

Inheritance

- The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important features of Object-Oriented Programming.
  - Sub Class: The class that inherits properties from another class is called Sub class or Derived Class.
  - Super Class: The class whose properties are inherited by a sub-class is called Base Class or Superclass.
  - Reusability: Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Object Oriented Programming in C++

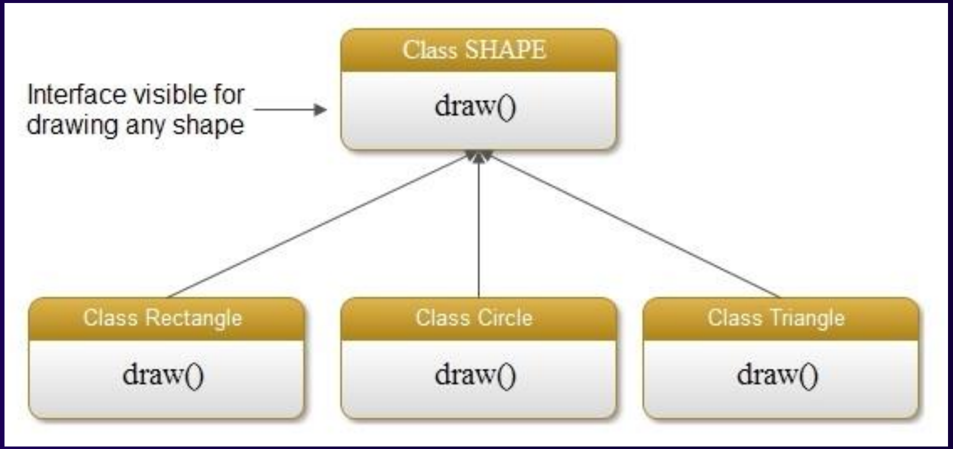
Inheritance

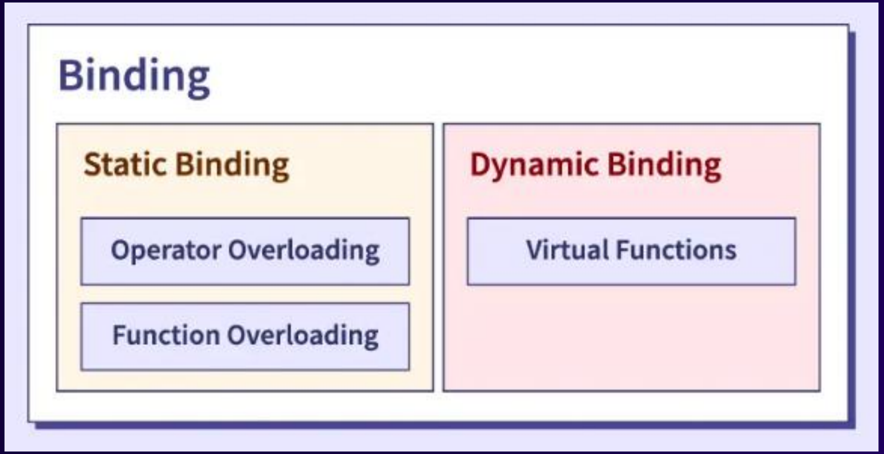


Object Oriented Programming in C++

Dynamic Binding

- In dynamic binding, the code to be executed in response to the function call is decided at runtime. C++ has virtual functions to support this. Because dynamic binding is flexible, it avoids the drawbacks of static binding, which connected the function call and definition at build time.

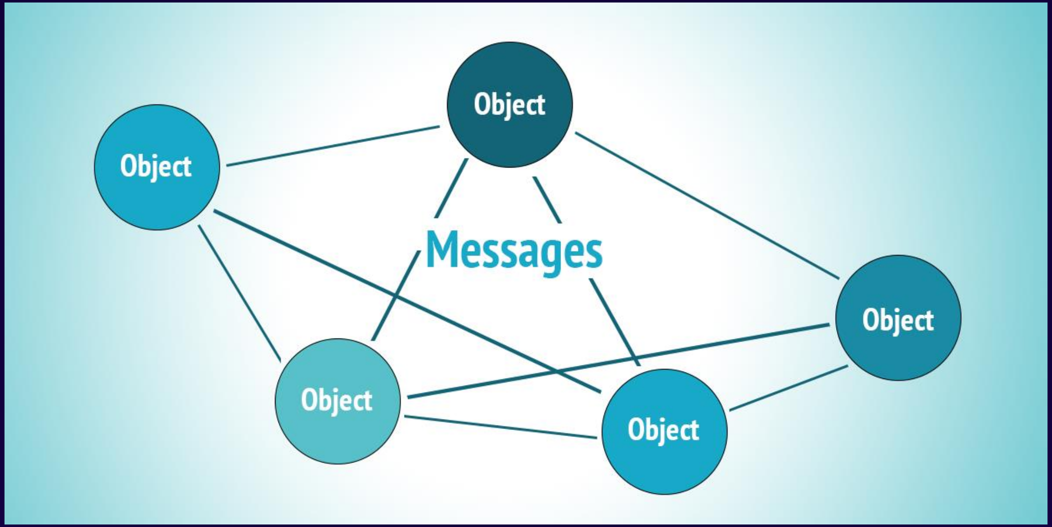




Object Oriented Programming in C++

Message Passing

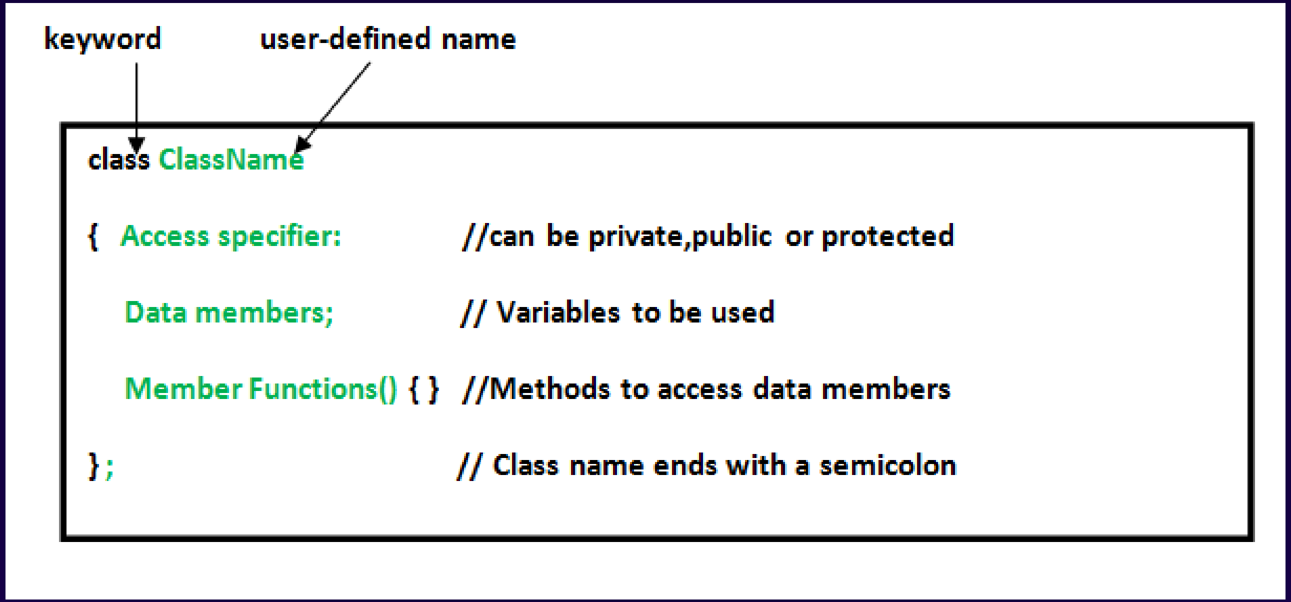
- Objects communicate with one another by sending and receiving information. A message for an object is a request for the execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function, and the information to be sent.



Object Oriented Programming in C++

Classes and Objects

- Defining Class and Declaring Objects
  - A class is defined in C++ using the keyword class followed by the name of the class. The body of the class is defined inside the curly brackets and terminated by a semicolon at the end. Default modifier is private.



```
class ClassName
{  Access specifier:          //can be private,public or protected
   Data members;              // Variables to be used
   Member Functions() { }     //Methods to access data members
};                            // Class name ends with a semicolon
```

Object Oriented Programming in C++

Qt

Classes and Objects

- Declaring Objects
  - When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

```
ClassName ObjectName;
```

- Accessing data members and member functions
  - The data members and member functions of the class can be accessed using the dot(‘.’) operator with the object. For example, if the name of the object is obj and you want to access the member function with the name printName() then you will have to write obj.printName().
  - The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member. This access control is given by Access modifiers in C++. There are three access modifiers: public, private, and protected.

Object Oriented Programming in C++

Classes and Objects

- 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 the class name and function name.
- Constructors
  - Constructors are special class members which are called by the compiler every time an object of that class is instantiated. Constructors have the same name as the class and may be defined inside or outside the class definition. There are 3 types of constructors:
    - ✓ Default Constructors
    - ✓ Parameterized Constructors
    - ✓ Copy Constructors

    Note: A Copy Constructor creates a new object, which is an exact copy of the existing object. The compiler provides a default Copy Constructor to all the classes.

```
class-name (class-name &){}
```

Object Oriented Programming in C++

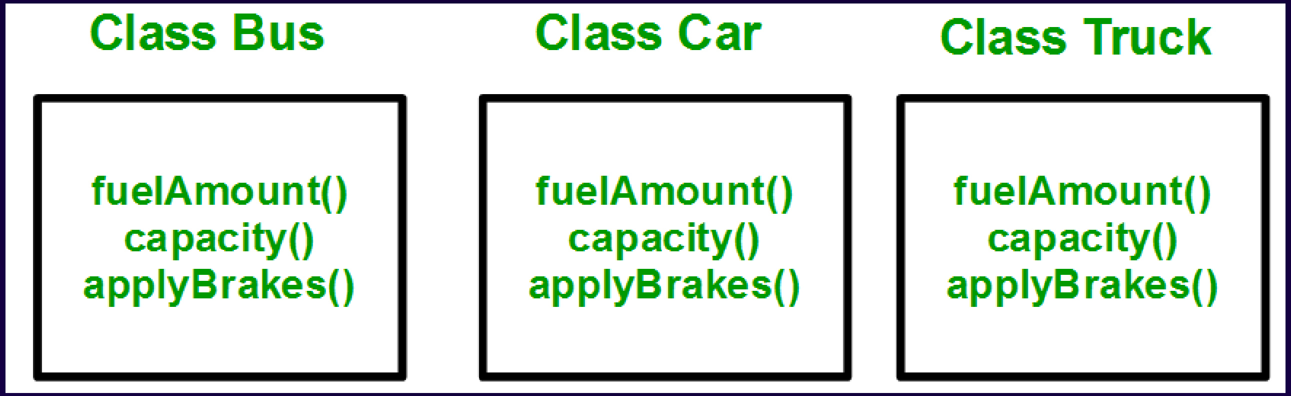
Classes and Objects

- Destructors
  - Destructor is another special member function that is called by the compiler when the scope of the object ends.
- Interesting Fact (Rare Known Concept)
  - Many people might say that it's a basic syntax and we should give a semicolon at the end of the class as its rule defines in cpp. But the main reason why semi-colons are there at the end of the class is compiler checks if the user is trying to create an instance of the class at the end of it.
  - Yes just like structure and union, we can also create the instance of a class at the end just before the semicolon. As a result, once execution reaches at that line, it creates a class and allocates memory to your instance.

Object Oriented Programming in C++

Inheritance in C++

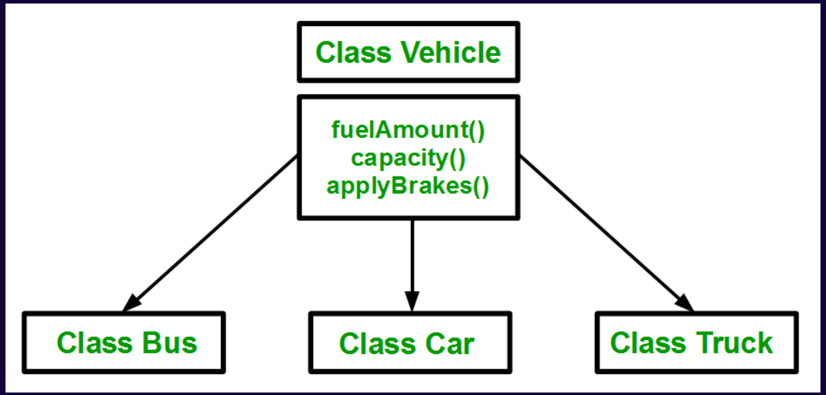
- Why and when to use inheritance?
  - Consider a group of vehicles. You need to create classes for Bus, Car, and Truck. The methods fuelAmount(), capacity(), applyBrakes() will be the same for all three classes. If we create these classes avoiding inheritance then we have to write all of these functions in each of the three classes as shown below figure:



Object Oriented Programming in C++

Inheritance in C++

- Why and when to use inheritance?
  - You can clearly see that the above process results in duplication of the same code 3 times. This increases the chances of error and data redundancy. To avoid this type of situation, inheritance is used.



  - Using inheritance, we have to write the functions only one time instead of three times as we have inherited the rest of the three classes from the base class (Vehicle).

Object Oriented Programming in C++

Inheritance in C++

- Implementing inheritance in C++
  - access-specifier either of private, public or protected. If neither is specified, PRIVATE is taken as default.

```
class  <derived_class_name> : <access-specifier> <base_class_name>
{
        //body
}
```

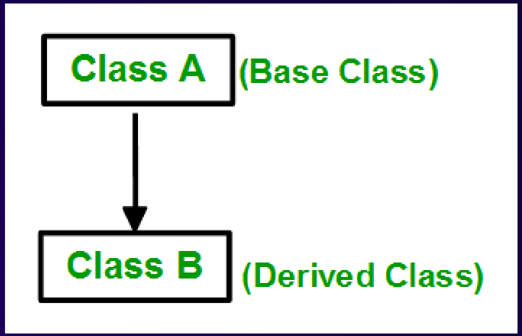
  - Note: When a base class is privately inherited by the derived class, public members of the base class becomes the private members of the derived class and therefore, the public members of the base class can only be accessed by the member functions of the derived class.
- Modes of Inheritance

Base class member access specifier	Type of Inheritence		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

Object Oriented Programming in C++

Inheritance in C++

- Types Of Inheritance:
  1. Single inheritance
  2. Multilevel inheritance
  3. Multiple inheritance
  4. Hierarchical inheritance
  5. Hybrid inheritance
  6. Multipath inheritance
- Single Inheritance: In single inheritance, a class is allowed to inherit from only one class. i.e. one subclass is inherited by one base class only.



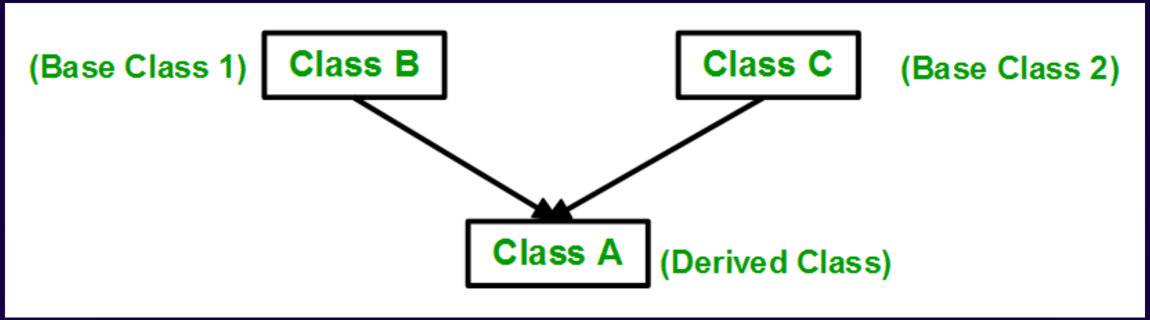
```
class subclass_name : access_mode base_class
{
  // body of subclass
};
```

Object Oriented Programming in C++

Inheritance in C++

- Multiple Inheritance: Multiple Inheritance is a feature of C++ where a class can inherit from more than one class. i.e one subclass is inherited from more than one base class.

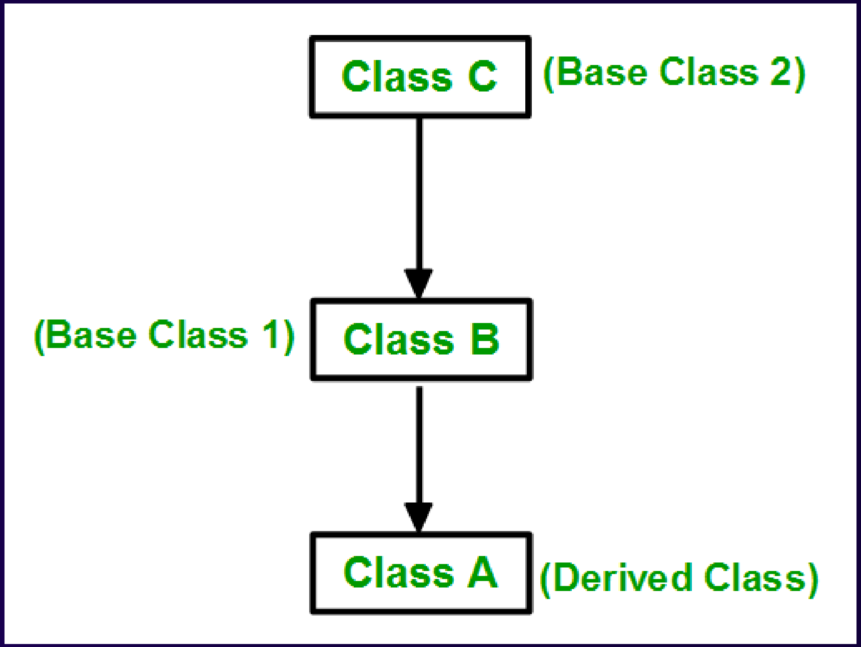
```
class subclass_name : access_mode base_class1, access_mode base_class2, ....
{
  // body of subclass
};
```



Object Oriented Programming in C++

Inheritance in C++

- Multilevel Inheritance: In this type of inheritance, a derived class is created from another derived class.

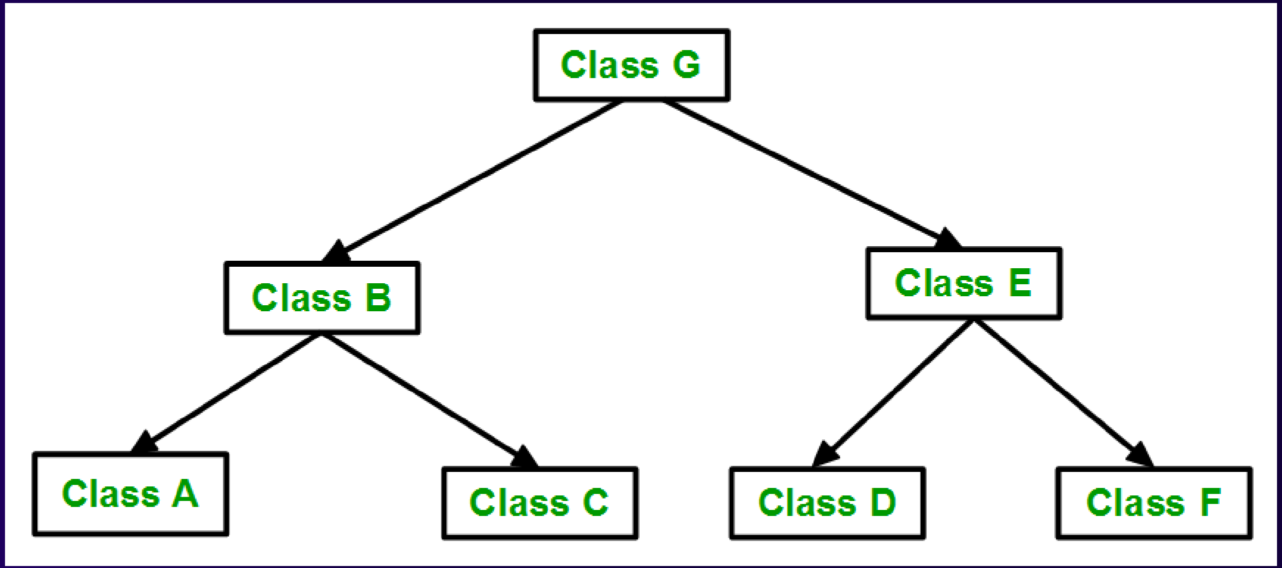


```
class C
{
... .. ...
};
class B:public C
{
... .. ...
};
class A: public B
{
... ... ...
};
```

Object Oriented Programming in C++

Inheritance in C++

- Hierarchical Inheritance: In this type of inheritance, more than one subclass is inherited from a single base class. i.e. more than one derived class is created from a single base class.

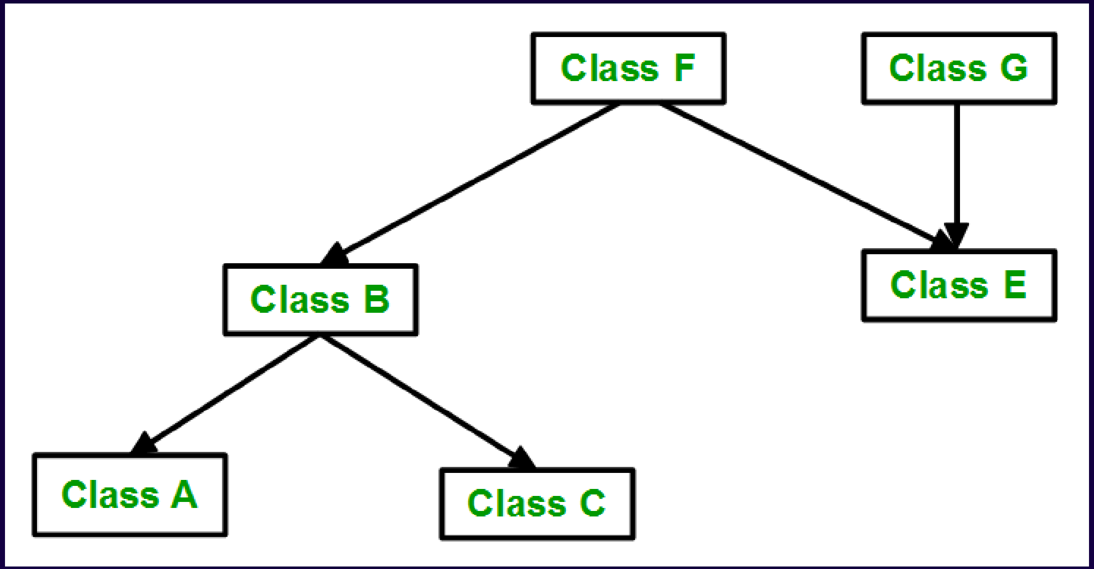


```
class A
{
    // body of the class A.
}
class B : public A
{
    // body of class B.
}
class C : public A
{
    // body of class C.
}
class D : public A
{
    // body of class D.
}
```

Object Oriented Programming in C++

Inheritance in C++

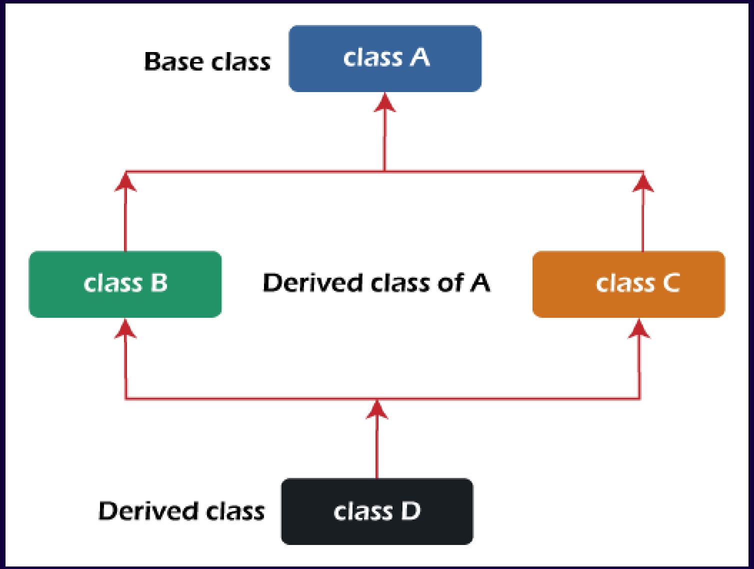
- Hybrid (Virtual) Inheritance: Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance. Below image shows the combination of hierarchical and multiple inheritances:



Object Oriented Programming in C++

Inheritance in C++

- Multipath inheritance (A special case of hybrid inheritance): A derived class with two base classes and these two base classes have one common base class is called multipath inheritance. Ambiguity can arise in this type of inheritance.



Object Oriented Programming in C++

Inheritance in C++

- Multipath inheritance (A special case of hybrid inheritance)
  - There are 2 Ways to Avoid this Ambiguity:
    1. Avoiding ambiguity using the scope resolution operator: In this case there are multiple copies of the parent class
    2. Avoiding ambiguity using the virtual base class: In this case there is a copy of the parent class.

```
class A
{
    member;
}
class B : public A {}
class C : public A {}
class D : public B, public C {}

D d;
d.B::member = x;
d.C::member = x;
```

```
class A
{
    member;
}
class B : virtual public A {}
class C : virtual public A {}
class D : public B, public C {}

D d;
d.member = x;
```

Object Oriented Programming in C++

Compile-Time Polymorphism

- Function Overloading
  - When there are multiple functions with the same name but different parameters, then the functions are said to be overloaded, hence this is known as Function Overloading. Functions can be overloaded by changing the number of arguments or/and changing the type of arguments.
- Operator Overloading
  - C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading. For example, we can make use of the addition operator (+) for string class to concatenate two strings. We know that the task of this operator is to add two operands. So a single operator '+', when placed between integer operands, adds them and when placed between string operands, concatenates them.

Object Oriented Programming in C++

Runtime Polymorphism

- Function Overriding
  - Function Overriding occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.
  - The function in the parent class must be defined as virtual.
- Runtime Polymorphism with Data Members
  - Runtime Polymorphism can be achieved by data members in C++.
- Virtual Function
  - A virtual function is a member function that is declared in the base class using the keyword virtual and is re-defined (Overridden) in the derived class.
    - Virtual functions are Dynamic in nature.
    - They are defined by inserting the keyword “virtual” inside a base class and are always declared with a base class and overridden in a child class
    - A virtual function is called during Runtime

ClassName() { //Constructor's Body }

Object Oriented Programming in C++

Qt

Constructor

- A constructor is a member function of a class that has the same name as the class name. It helps to initialize the object of a class. It can either accept the arguments or not.

```
ClassName()
{
    //Constructor's Body
}
```

Destructor

- Like a constructor, Destructor is also a member function of a class that has the same name as the class name preceded by a tilde(~) operator. It helps to deallocate the memory of an object. It is called while the object of the class is freed or deleted.

```
~ClassName()
{
      //Destructor's Body
}
```

Object Oriented Programming in C++

Virtual Destructor

- Deleting a derived class object using a pointer of base class type that has a non-virtual destructor results in undefined behavior. To correct this situation, the base class should be defined with a virtual destructor.

Pure Virtual Destructor

- The inclusion of a pure virtual destructor in a C++ program has no negative consequences. Pure virtual destructors must have a function body because their destructors are called before those of base classes; if one is absent, object destruction will fail since there won't be anything to call when the object is destroyed. Making a pure virtual destructor with its definition allows us to create an abstract class easily.

Note: Only Destructors can be Virtual. Constructors cannot be declared as virtual.

Object Oriented Programming in C++

Pure Virtual Functions and Abstract Classes in C++

- Sometimes implementation of all function cannot be provided in a base class because we don't know the implementation. Such a class is called abstract class. For example, let Shape be a base class. We cannot provide implementation of function draw() in Shape, but we know every derived class must have implementation of draw(). Similarly an Animal class doesn't have implementation of move() (assuming that all animals move), but all animals must know how to move. We cannot create objects of abstract classes.
- A pure virtual function (or abstract function) in C++ is a virtual function for which we can have implementation, But we must override that function in the derived class, otherwise the derived class will also become abstract class.
  - Some Interesting Facts:
    1. A class is abstract if it has at least one pure virtual function.
    2. We can have pointers and references of abstract class type.
    3. If we do not override the pure virtual function in derived class, then derived class also becomes abstract class.
    4. An abstract class can have constructors.
    5. An abstract class in C++ can also be defined using struct keyword.

Object Oriented Programming in C++

static_cast

- static_cast is used for ordinary typecasting. It is responsible for the implicit type of coercion and is also called explicitly. We should use it in cases like converting the int to float, int to char, etc.

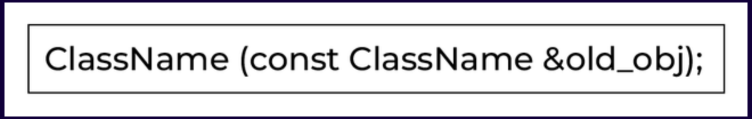
dynamic_cast

- In C++, we can treat the derived class’s reference or pointer as the base class’s pointer. This method is known as upcasting in C++. But its opposite process is known as downcasting, which is not allowed in C++. So, the dynamic_cast in C++ promotes safe downcasting. We can only perform this in polymorphic classes, which must have at least one virtual function.
- dynamic_cast is useful when you don't know what the dynamic type of the object is. It returns a null pointer if the object referred to doesn't contain the type casted to as a base class (when you cast to a reference, a bad_cast exception is thrown in that case).

Object Oriented Programming in C++

Copy Constructor

- A copy constructor is a member function that initializes an object using another object of the same class. In simple terms, a constructor which creates an object by initializing it with an object of the same class, which has been created previously is known as a copy constructor.



```
ClassName (const ClassName &old_obj);
```

- Characteristics of Copy Constructor
  1. The copy constructor is used to initialize the members of a newly created object by copying the members of an already existing object.
  2. Copy constructor takes a reference to an object of the same class as an argument.
  3. The process of initializing members of an object through a copy constructor is known as copy initialization.
  4. It is also called member-wise initialization because the copy constructor initializes one object with the existing object, both belonging to the same class on a member-by-member copy basis.
  5. The copy constructor can be defined explicitly by the programmer. If the programmer does not define the copy constructor, the compiler does it for us.

Object Oriented Programming in C++

Copy Constructor

- Types of Copy Constructors
  1. Default Copy Constructor
  2. User Defined Copy Constructor
- When is the copy constructor called? In C++, a Copy Constructor may be called in the following cases:
  1. When an object of the class is returned by value.
  2. When an object of the class is passed (to a function) by value as an argument.
  3. When an object is constructed based on another object of the same class.
  4. When the compiler generates a temporary object.

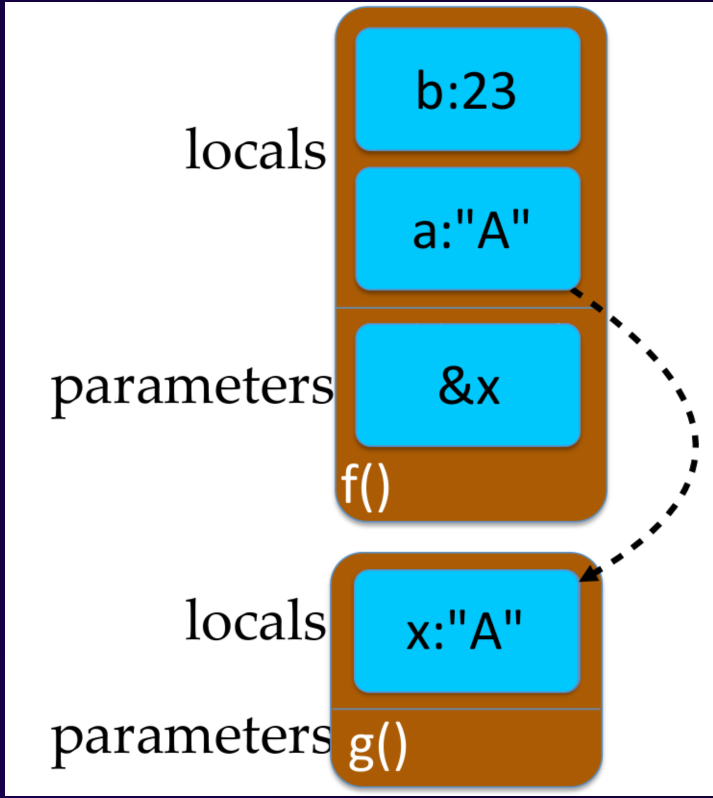
  Note: It is, however, not guaranteed that a copy constructor will be called in all these cases, because the C++ Standard allows the compiler to optimize the copy away in certain cases, one example is the return value optimization (sometimes referred to as RVO).
- Copy Elision

  In copy elision, the compiler prevents the making of extra copies which results in saving space and better the program complexity(both time and space); Hence making the code more optimized.

Object Oriented Programming in C++

Copy Constructor

- RVO



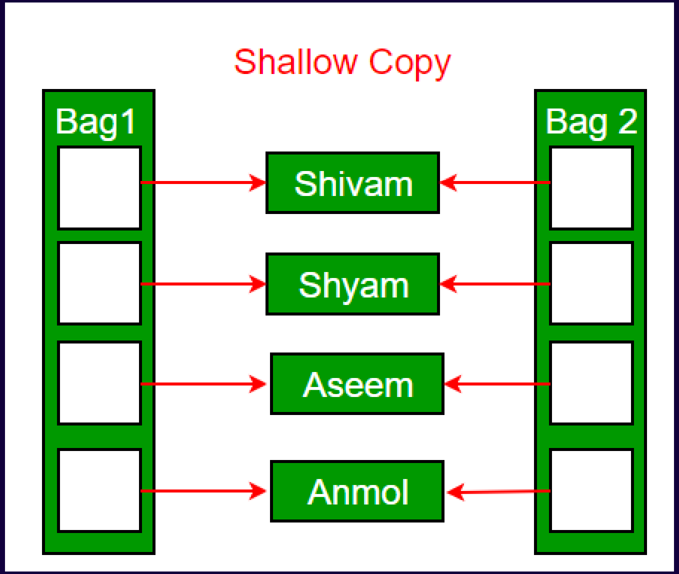
```
std::string f()
{
    std::string a{"A"};
    int b{23};
    ...
    return a;
}

void g()
{
    std::string x{f()};
}
```

Object Oriented Programming in C++

Copy Constructor

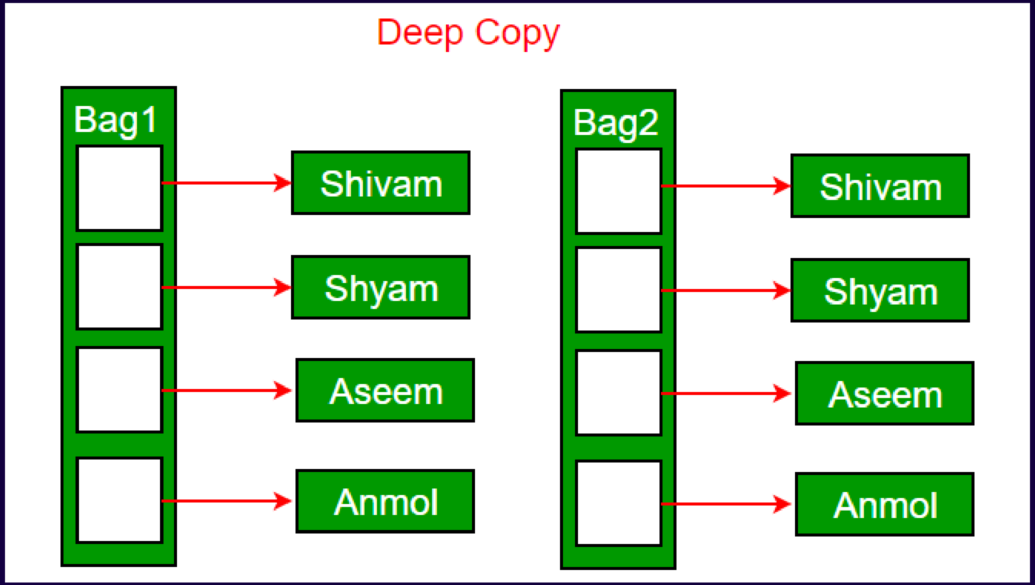
- When is a user-defined copy constructor needed?
  - If we don’t define our own copy constructor, the C++ compiler creates a default copy constructor for each class which does a member-wise copy between objects. The compiler-created copy constructor works fine in general. We need to define our own copy constructor only if an object has pointers or any runtime allocation of the resource like a file handle, a network connection, etc.
  - The default constructor does only shallow copy.



Object Oriented Programming in C++

Copy Constructor

- When is a user-defined copy constructor needed?
  - Deep copy is possible only with a user-defined copy constructor. In a user-defined copy constructor, we make sure that pointers (or references) of copied objects point to new memory locations.



Copy Constructor

- Copy constructor vs Assignment Operator
  - The main difference between Copy Constructor and Assignment Operator is that the Copy constructor makes a new memory storage every time it is called while the assignment operator does not make new memory storage.
  - A copy constructor is called when a new object is created from an existing object, as a copy of the existing object. The assignment operator is called when an already initialized object is assigned a new value from another existing object. In the bellow example (1) calls the copy constructor and (2) calls the assignment operator. See this for more details.

```
MyClass t1, t2;
MyClass t3 = t1;  // ----> (1)
t2 = t1;          // -----> (2)
```

Thank You

By Ali Panahi