Shree Ganeshay Nmah

Object Oriented Programming

- Page 28 (imp)

OOPs?

- Object-oriented programming – As the name suggests uses objects in programming. 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.
- There are some basic concepts of OOPs
    - Class
    - Objects
    - Encapsulation
    - Abstraction
    - Polymorphism
    - Inheritance
    - Dynamic Binding
    - Message Passing

Class

- Class 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/object of that class.
- Class is like structures.
- For e.g. class of cars where its object are diff-2 brand and

  Data member are speed-limit,milege, model, price and

  Member function maybe stearing, clutching, accelerating

  Diff-2 object/brand have diff-1 data-member value.

Friend Class/function

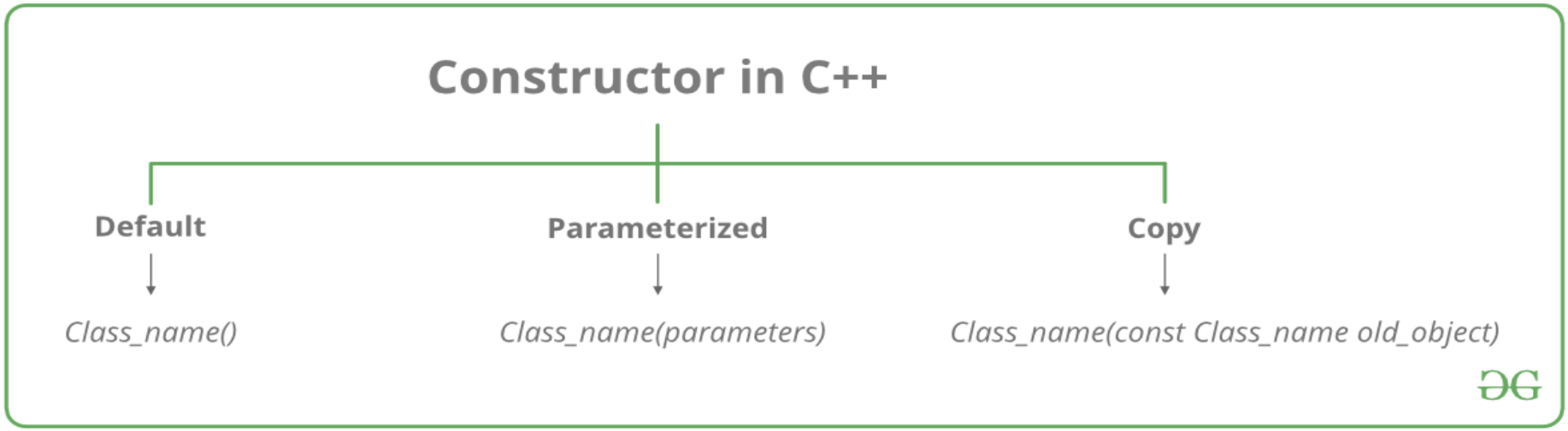
- A friend class can access private and protected members of other classes in which it is declared as a friend. It is sometimes useful to allow a particular class to access private and protected members of other classes.
- For example, a LinkedList class may be allowed to access private members of Node.

- **Friend Function**
- They are the non-member functions that can access and manipulate the private and protected members of the class for they are declared as friends.
- A friend function can be:
  - A global function
  - A member function of another class

- **Advantages of Friend Functions**
  - A friend function is able to access members without the need of inheriting the class.
  - The friend function acts as a bridge between two classes by accessing their private data.
  - It can be used to increase the versatility of overloaded operators.
  - It can be declared either in the public or private or protected part of the class.
- **Disadvantages of Friend Functions**
  - Friend functions have access to private members of a class from outside the class which violates the law of data hiding.
  - Friend functions cannot do any run-time polymorphism in their members.

- 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.
- Constructors :-
  - Constructors are special class members which are called by the compiler every time an object of that class is instantiated/created. 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

The prototype constructor class_name (list-of-parameters);



Destructors

- Destructor is an instance member function that is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed.
- A destructor is also a special member function like a constructor. Destructor destroys the class objects created by the constructor.
- Destructor has the same name as their class name preceded by a tilde (~) symbol.
- It is not possible to define more than one destructor.
- The destructor is only one way to destroy the object created by the constructor. Hence destructor can-not be overloaded.
- Destructor neither requires any argument nor returns any value.
- It is automatically called when an object goes out of scope.
- Destructor release memory space occupied by the objects created by the constructor.
- In destructor, objects are destroyed in the reverse of an object creation.

Private Destructor

- Whenever we want to control the destruction of objects of a class, we make the destructor private.
- For dynamically created objects, it may happen that you pass a pointer to the object to a function and the function deletes the object. If the object is referred after the function call, the reference will become dangling.

Shallow vs Deep Copy

	Shallow Copy	Deep copy
1.	When we create a copy of object by copying data of all member variables as it is, then it is called shallow copy	When we create an object by copying data of another object along with the values of memory resources that reside outside the object, then it is called a deep copy
2.	A shallow copy of an object copies all of the member field values.	Deep copy is performed by implementing our own copy constructor.
3.	In shallow copy, the two objects are not independent	It copies all fields, and makes copies of dynamically allocated memory pointed to by the fields
4.	It also creates a copy of the dynamically allocated objects Means both object point to same memory	If we do not create the deep copy in a rightful way then the copy will point to the original, with disastrous consequences.

- Static data members are class members that are declared using **static** keywords.
  - Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.
  - It is initialized before any object of this class is created, even before the main starts.
  - It is visible only within the class, but its lifetime is the entire program.
- **Syntax:**
  - static data_type data_member_name;

- A static member function is independent of any object of the class.
- A static member function can be called even if no objects of the class exist.
- A static member function can also be accessed using the class name through the scope resolution operator.
- A static member function can access static data members and static member functions inside or outside of the class.
- Static member functions have a scope inside the class and cannot access the current object pointer.
- **The reason we need Static member function:**
- Static members are frequently used to store information that is shared by all objects in a class.
- For instance, you may keep track of the quantity of newly generated objects of a specific class type using a static data member as a counter. This static data member can be increased each time an object is generated to keep track of the overall number of objects.

This pointer

- This cann’t be reassigned.
- this -> mem_data
- This->mem_fun

Local Class

- A local class name can only be used locally i.e., inside the function and not outside it.
- The methods of a local class must be defined inside it only.
- A local class can have static functions but, not static data members.
- Like a class in global scope now the local class has its global scope inside the block only

Class	Structure
1. Members of a class are private by default.	1. Members of a structure are public by default.
2. An instance of a class is called an ‘object’.	2. An instance of structure is called the ‘structure variable’.
3. Member classes/structures of a class are private by default but not all programming languages have this default behavior eg Java etc.	3. Member classes/structures of a structure are public by default.
4. It is declared using the class keyword.	4. It is declared using the struct keyword.
5. It is normally used for data abstraction and further inheritance.	5. It is normally used for the grouping of data
6. NULL values are possible in Class.	6. NULL values are not possible.

Encapsulation

- Encapsulation in C++ is defined as the wrapping up of data and information in a single unit.
- **Two Important property of Encapsulation**
  - **Data Protection:** Encapsulation protects the internal state of an object by keeping its data members private. Access to and modification of these data members is restricted to the class's public methods.
  - **Information Hiding:** Encapsulation hides the internal implementation details of a class from external code. Only the public interface of the class is accessible.

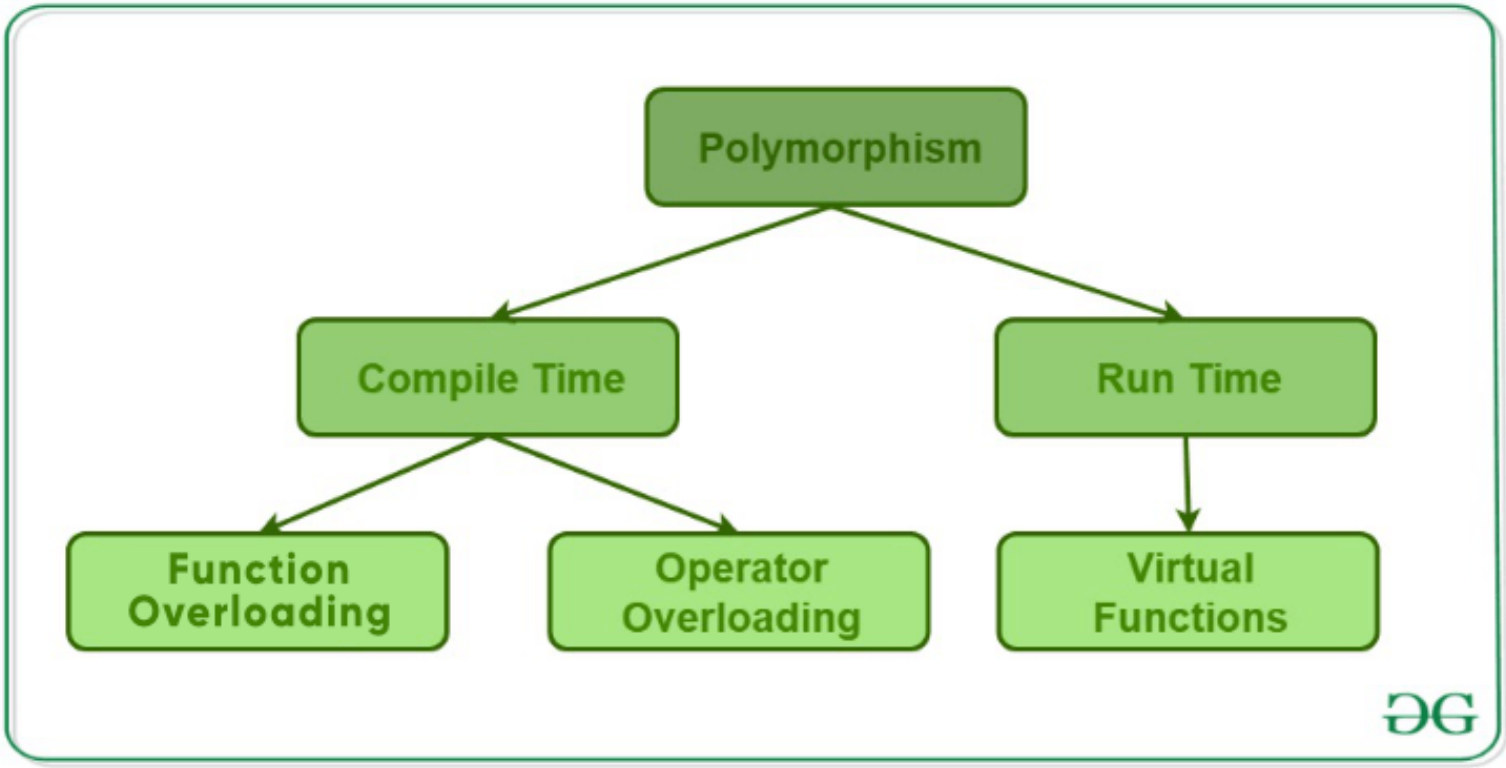
- The function which we are making inside the class must use only member variables, only then it is called *encapsulation*.
- If we don't make a function inside the class which is using the member variable of the class then we don't call it encapsulation.

Abstraction

- Data abstraction means providing only essential information about the data to the outside world, hiding the background details or implementation.
- **Types of Abstraction:**
  - **Data abstraction –** This type only shows the required information about the data and hides the unnecessary data.
  - **Control Abstraction –** This type only shows the required information about the implementation and hides unnecessary information.

Polymorphism

- Types of Polymorphism
  - Compile-time Polymorphism
  - Runtime 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.
- A Function cannot have same Signature in C++
- Signature is fun_name & the **number** and the **type of arguments.**
- Function Prototype :- return_type + signature

Check b/w the function calls and actual Definition

- Rule 1 : Exact Match (jo function call kiya h vo exact match ho kissi se)
    - Actual argument type = Formal argument type
- Rule 2: Type Promotion (only 2)
    - Char into int only and float into double
- Rule 3: Type Conversion (all possible implicit conversion)

Note :- Any one of the folowing Rule will applicable at a time

Function that cann’t Overload

- 1). If Differs only in return type
- 2). If two function have same name and parameters but one of them having static keyword then not
- 3). Parameter declarations that differ only in a pointer * versus an array [] are equivalent.
- 4). Parameter declarations that differ only in a function pointer * versus an functionn are equivalent.
    - void fun(int ()); and void fun(int (*)()) // not overload fun -ambigious error
- 5).Parameters that differ only in the presence or absence of const and/or volatile are equivalent. // not overload
    - But if pointer or refrence as parametrs then const work // overload

Operator Overloading

- Operator performs various operation

Syntax :-

  - Return_Type classname :: operator op(Argument list) { Function Body }
  - // This can be done by declaring the function

- Rules :-
  - In the case of a **non-static member function**, the binary operator should have only one argument and the unary should not have an argument.
  - In the case of a **friend function**, the binary operator should have only two arguments and the unary should have only one argument.
  - All the class member objects should be public if operator overloading is implemented.
  - Operators that cannot be overloaded are **.* :: ?:**
  - Operators that cannot be overloaded when declaring that function as friend function are **= () [] ->**.
  - The operator function must be either a non-static (member function) or a friend function.

Operator can Be Overload

+	–	*	?	%	?	&	|	~
!	=	<	>	+=	-=	*=	?=	%=
?=	&=	|=	<<	>>	<<=	>>=	==	!=
<=	>=	&&	||	++	—	,	->*	->
( )	[ ]	new	delete	new[]	delete []			

Operator Cann’t Overload

- 1) Scope Resolution Operator (::)
- **2)** Ternary or Conditional Operator (?:)
- **3)** Member Access or Dot operator (.)
- **4)** Pointer-to-member Operator (.*)
- **5)** Object size Operator (sizeof)
- **6)** Object type Operator(typeid)
- **7)** static_cast (casting operator)
- **8)** const_cast (casting operator)
- **9)** reinterpret_cast (casting operator)
- **10)** dynamic_cast (casting operator)

Runtime Polymorphism

- This type of polymorphism is achieved by **Function Overriding**.
- Late binding and dynamic polymorphism are other names for runtime polymorphism.
- The function call is resolved at runtime in runtime polymorphism.
- In contrast, with compile time polymorphism, the compiler determines which function call to bind to the object after deducing it at runtime. (in pointer this reflected)
- Two types :-
  - Function Overriding
  - Virtual Function

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.
- If we run function as child_class object then at runtime at first function is searched in child class if there is no function then it goes to parent class for searching
- Child to parent // valid
- Parent to child class // invalid search

Note :- There is an Error with pointers so Virtual function come into play

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.
- **Some Key Points About Virtual Functions:**
  - Virtual functions are Dynamic in nature cannot static.
  - 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.
  - A virtual function can be a friend function of another class.
  - Virtual functions should be accessed using a pointer or reference of base class type to achieve runtime polymorphism.
  - The prototype of virtual functions should be the same in the base as well as the derived class.
  - They are always defined in the base class and overridden in a derived class. It is not mandatory for the derived class to override (or re-define the virtual function), in that case, the base class version of the function is used.
  - A class may have a virtual destructor but it cannot have a virtual constructor.

Logic of Vir. Func

- In multilevel inheritence if virtual function define in base class
- Then all inherit class need not to use virtual A->B->C
- Virtual Fucntions binded at Runtime
  - 1). Late binding (Runtime) is done in accordance with the content of the pointer (i.e. location pointed to by pointer) and Early binding (Compile-time) is done according to the type of pointer

- **Note:** Never call a virtual function from a CONSTRUCTOR or DESTRUCTOR

Working of Virtual Functions (concept of VTABLE and VPTR)

- As discussed here, if a class contains a virtual function then the compiler itself does two things.
- If an object of that class is created then a **virtual pointer (VPTR)** is inserted as a data member of the class to point to the VTABLE of that class. For each new object created, a new virtual pointer is inserted as a data member of that class.
- Irrespective of whether the object is created or not, the class contains as a member **a static array of function pointers called VTABLE**. Cells of this table store the address of each virtual function contained in that class.

Virtual Desructors

- As a guideline, any time you have a virtual function in a class, you should immediately add a virtual destructor (even if it does nothing). This way, you ensure against any surprises later.

Inheritance

- The capability of a class to derive properties and characteristics from another class is called **Inheritance**.
- The derived class inherits all the properties of the base class, without changing the properties of base class and may add new features to its own.
  - **Sub Class:** The class that inherits properties from another class is called Subclass or Derived Class.
  - **Super Class:** The class whose properties are inherited by a subclass is called Base Class or Superclass.
- Syntax :-
  - class derived_class_name : access-specifier base_class_name {

    // body

    }

- **Modes of Inheritance:**
  - **Public Mode**: In this mode the public member of the base class will become public in the derived class and protected members of the base class will become protected in the derived class.
  - **Protected Mode**: Both public members and protected members of the base class will become protected in the derived class.
  - **Private Mode**: If we derive a subclass from a Private base class. Then both public members and protected members of the base class will become Private in the derived class.

Note : Sub-class cannot access the private member of parent class

Types Of Inheritance

- Single inheritance :-
  - Only one class is super class of derived class
- Multilevel inheritance
  - Like nested Intheritence base_clasee ---->sub_class -----> sub_sub_class
- Multiple inheritance
  - An sub_class has multiple parent/super class
- Hierarchical inheritance
  - Like combination of Multiple + Multilevel
- Hybrid inheritance
  - Hybrid of any of two types of Inheritence

- In general, it is not allowed to call the grandparent’s constructor directly, it has to be called through parent class. It is allowed only when ‘virtual’ keyword is used.