Object Oriented Programming

Unit 2

C++ Access Specifiers

The **public** keyword is an access specifier. Access specifiers define how the members (attributes and methods) of a class can be accessed. In the example above, the members are public - which means that they can be accessed and modified from outside the code.
In C++, there are three access specifiers:

- Public - members are accessible from outside the class
- Private - members cannot be accessed (or viewed) from outside the class
- Protected - members cannot be accessed from outside the class, however, they can be accessed in inherited classes.

C++ Access Specifiers

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.

C++ Access Specifiers

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.

C++ Access Specifiers

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 sub-class(derived class) of that class.

Constructors in C++

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. There are 3 types of constructors:

- Default constructors
- Parametrized constructors
- Copy constructors

Default constructors

- Default constructor is the constructor which doesn't take any argument. It has no parameters.
- Even if we do not define any constructor explicitly, the compiler will automatically provide a default constructor implicitly.

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.

Copy Constructor

- Copy Constructor is a type of constructor which is used to create a copy of an already existing object of a class type
- A copy constructor is a member function which initializes an object using another object of the same class.

Destructors in C++

- Destructor is a member function which destructs or deletes an object.
- Destructors have same name as the class preceded by a tilde (~). Destructors don't take any argument and don't return anything

When is destructor called?

A destructor function is called automatically when the object goes out of scope:

- the function ends
- the program ends
- a block containing local variables ends
- a delete operator is called

Destructors in C++

Can there be more than one destructor in a class?

- No, there can only one destructor in a class with classname preceded by (~), no parameters and no return type.

When do we need to write a user-defined destructor?

- If we do not write our own destructor in class, compiler creates a default destructor for us.
- The default destructor works fine unless we have dynamically allocated memory or pointer in class.
- When a class contains a pointer to memory allocated in class, we should write a destructor to release memory before the class instance is destroyed.

C++ Overloading

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. 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.

C++ Operators Overloading

- 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.
- You can redefine or overload most of the built-in operators available in C++. Thus a programmer can use operators with user-defined types as well.

C++ Operators Overloading

Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list. Syntax: return-type class-name :: operator op(arg-list)

Where return type is the type of value returned by the specified operation and op is the operator being overloaded. The op is preceded by the keyword operator, operator op is the function name.

Operator that cannot be overloaded are as follows:

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

Why do we need Operator overloading in C++?

Operators in C++ like +, -, *, / can operate in datatypes like int, float, double etc as predefined operational meanings. But these operators can't operate in user-defined datatypes like objects without extension or adding some sort of code to alter their operational meaning. Such a way of extending the operational functionality of certain operators in C++ is called **operator overloading**.

This Pointer

- Every member function of the class is born with the pointer called "THIS"; which points to the object with which member function is associated.
- When a member function is invoked it comes into existence with the value of this set to the address of the object for which is it called.

This Pointer

- Every member function of the class is born with the pointer called "THIS"; which points to the object with which member function is associated.
- When a member function is invoked it comes into existence with the value of this set to the address of the object for which is it called.

Using this for Returning Values

- This pointer can be use to return the values from the member function.
- Since it points to the address of the object which is called the member function, we can return the object by value with the help of this pointer.

Using this for specifying memory address

- This pointer is created automatically inside the member function whenever the member function is invoked by the object.
- It hold the memory address of the object so it can be used to access the memory address of the object.

Using this for accessing the data member

- This pointer can also be used to access the data members inside the member function.
- It can be done with the help of arrow operator $(->)$.
- Arrow operator is the combination of hyphen $(-)$ and greater than operator $(>)$.

Static Data Member and Member Function

Static Data Member A data member of a class can be qualified as static. The properties of a static member variable are similar to that of C-programmings static variable. A static data member has certain special characteristics. They are:-

- It is initialized to zero when the first object of its class is created. No other initialization is permitted.
- 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 visible only within the class, but its lifetime is the entire program.

Static Data Member and Member Function

A static variable is normally used to maintain value common to the entire class. For e.g, to hold the count of objects created. Note that the type and scope of each static member variable must be declared outside the class definition. This is necessary because the static data members are stored separately rather than as a part of an object.

Declaration

static data-type member-name;

Defining the static data member

It should be defined outside of the class following this syntax:

data-type class-name :: member-name =value;

If you are calling a static data member within a member function, member function should be declared as static (i.e. a static member function can access the static data members)

Static Data Member and Member Function

Static Member Function

like a static member variable, we can also have static member functions. A member function that is declared static has the following properties

- A static function can have access to only other static members (function or variable) declared in the same class.
- A static member function can be called using the class name (instead of its object) as follows-
  Class-name::Function-name();