C++ Theory

Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters. When a function name is overloaded with different jobs it is called Function Overloading. In Function Overloading "Function" name should be the same and the arguments should be different. Function overloading can be considered as an example of a polymorphism feature in C++.

The parameters should follow any one or more than one of the following conditions for Function overloading:

Parameters should have a different type

Parameters should have a different number

Parameters should have a different sequence of parameters.

Operator Overloading

In C++, we can make operators work for user-defined classes. This means 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 overload an operator '+' in a class like String so that we can concatenate two strings by just using +. Other example classes where arithmetic operators may be overloaded are Complex Numbers, Fractional Numbers, Big Integer, etc.

Operator overloading is a compile-time polymorphism. It is an idea of giving special meaning to an existing operator in C++ without changing its original meaning.

Operators that can be overloaded

We can overload

- Unary operators
- Binary operators
- Special operators ([ ], () etc)

But, among them, there are some operators that cannot be overloaded. They are

- Scope resolution operator : :
- Member selection operator
- Member selection through *

Pointer to member variable

- Conditional operator ? :
- Sizeof operator sizeof()

New and Delete Operator

The new and delete operators can also be overloaded like other operators in C++. New and Delete operators can be overloaded globally or they can be overloaded for specific classes.

C++ Theory

If these operators are overloaded using member function for a class, it means that these operators are overloaded only for that specific class.

If overloading is done outside a class (i.e. it is not a member function of a class), the overloaded 'new' and 'delete' will be called anytime you make use of these operators (within classes or outside classes). This is global overloading.

Syntax for overloading the new operator :

void* operator new(size_t size);

The overloaded new operator receives size of type size_t, which specifies the number of bytes of memory to be allocated. The return type of the overloaded new must be void*.The overloaded function returns a pointer to the beginning of the block of memory allocated.

Syntax for overloading the delete operator :

void operator delete(void*);

The function receives a parameter of type void* which has to be deleted. Function should not return anything.

NOTE: Both overloaded new and delete operator functions are static members by default. Therefore, they don't have access to this pointer .

Friend Function Like friend class, a friend function can be given a special grant to access private and protected members. A friend function can be:
a) A member of another class
b) A global function

- A friend function is a special function in C++ which in-spite of not being member function of a class has privilege to access private and protected data of a class.
- A friend function is a non member function or ordinary function of a class, which is declared as a friend using the keyword "friend" inside the class. By declaring a function as a friend, all the access permissions are given to the function.
- The keyword "friend" is placed only in the function declaration of the friend function and not in the function definition.
- When friend function is called neither name of object nor dot operator is used. However it may accept the object as argument whose value it want to access.
- Friend function can be declared in any section of the class i.e. public or private or protected.

Inline function:

C++ provides an inline functions to reduce the function call overhead. Inline function is a function that is expanded in line when it is called. When the inline function is called whole code of the inline function gets inserted or substituted at the point of inline function call. This substitution is performed by the C++ compiler at compile time. Inline function may increase efficiency if it is small.

C++ Theory

Inline functions provide following advantages:
1) Function call overhead doesn't occur.
2) It also saves the overhead of push/pop variables on the stack when function is called.
3) It also saves overhead of a return call from a function.
4) When you inline a function, you may enable compiler to perform context specific optimization on the body of function. Such optimizations are not possible for normal function calls. Other optimizations can be obtained by considering the flows of calling context and the called context.
5) Inline function may be useful (if it is small) for embedded systems because inline can yield less code than the function call preamble and return.

Static Member Functions

A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator ::.

A static member function can only access static data member, other static member functions and any other functions from outside the class.

Static member functions have a class scope and they do not have access to the this pointer of the class. You could use a static member function to determine whether some objects of the class have been created or not.

Default Arguments

A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the calling function doesn't provide a value for the argument. In case any value is passed, the default value is overridden.

Advantages of Default Arguments:

- Default arguments are useful when we want to increase the capabilities of an existing function as we can do it just by adding another default argument to the function.
- It helps in reducing the size of a program.
- It provides a simple and effective programming approach.
- Default arguments improve the consistency of a program.

A class is a building block in C++ that leads to Object-Oriented programming. It is a user-defined type that holds its own data members and member functions. These can be accessed by creating an instance of the type class.

Self-referential classes are a special type of classes created specifically for a Linked List and tree-based implementation in C++. To create a self-referential class, declare a data member as a pointer to an object of the same class.

Syntax:

C++ Theory

```
class Self
{
    private:
    int a;
    Self *srefer;
}
```

Key Points:

Many frequently used dynamic data structures like stacks, queues, linked lists, etc.. use self-referential members.

Classes can contain one or more members which are pointers to other objects of the same class.

this pointer holds an address of the next object in a data structure.

Abstract Classes

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 (For more info about where we provide implementation for such functions refer to this https://stackoverflow.com/questions/2089083/pure-virtual-function-with-implementation). A pure virtual function is declared by assigning 0 in declaration. A class is abstract if it has at least one pure virtual function.

A linked list is a sequence of data structures, which are connected together via links.

Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.

- Link − Each link of a linked list can store a data called an element.
- Next − Each link of a linked list contains a link to the next link called Next.
- LinkedList − A Linked List contains the connection link to the first link called First.

Constructors:

Constructor in C++ is a special method that is invoked automatically at the time of object creation. It is used to initialize the data members of new objects generally. The constructor in C++ has the same name as the class or structure. Constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object which is why it is known as constructors.

Constructor does not have a return value, hence they do not have a return type.

C++ Theory

Types of Constructors

1. Default Constructors: Default constructor is the constructor which doesn't take any argument. It has no parameters. It is also called a zero-argument constructor.

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.

- **Uses of Parameterized constructor:**
    1. It is used to initialize the various data elements of different objects with different values when they are created.
    2. It is used to overload constructors.
- **Can we have more than one constructor in a class?**
  Yes, It is called Constructor Overloading.

3. Copy Constructor:

A copy constructor is a member function that initializes an object using another object of the same class. A detailed article on Copy Constructor.

Whenever we define one or more non-default constructors(with parameters) for a class, a default constructor(without parameters) should also be explicitly defined as the compiler will not provide a default constructor in this case. However, it is not necessary but it's considered to be the best practice to always define a default constructor.

Copy constructor takes a reference to an object of the same class as an argument.

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.

Inheritance is a feature or a process in which, new classes are created from the existing classes. The new class created is called "derived class" or "child class" and the existing class is known as the "base class" or "parent class". The derived class now is said to be inherited from the base class.

When we say derived class inherits the base class, it means, 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. These new features in the derived class will not affect the base class. The derived class is the specialized class for the base class.

- 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

Types Of Inheritance:-

C++ Theory

Single inheritance

Multilevel inheritance

Multiple inheritance

Hierarchical inheritance

Hybrid inheritance

1. 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.

2. 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**.

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

4. 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.

5. Hybrid (Virtual) Inheritance: Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.

There are 2 Ways to Avoid this Ambiguity:

1) Avoiding ambiguity using the scope resolution operator: Using the scope resolution operator we can manually specify the path from which data member a will be accessed

Note: Still, there are two copies of ClassA in Class-D.
2) Avoiding ambiguity using the virtual base class:

Virtual Functions

A virtual function is a member function which is declared within a base class and is re-defined (overridden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

- Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.
- They are mainly used to achieve Runtime polymorphism
- Functions are declared with a **virtual** keyword in base class.
- The resolving of function call is done at runtime.

Rules for Virtual Functions

1. Virtual functions cannot be static.
2. A virtual function can be a friend function of another class.
3. Virtual functions should be accessed using pointer or reference of base class type to achieve runtime polymorphism.

4. The prototype of virtual functions should be the same in the base as well as derived class.
5. 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.
6. A class may have virtual destructor but it cannot have a virtual constructor.

Scope Operator:

The :: (scope resolution) operator is used to get hidden names due to variable scopes so that you can still use them. The scope resolution operator can be used as both unary and binary. You can use the unary scope operator if a namespace scope or global scope name is hidden by a particular declaration of an equivalent name during a block or class. For example, if you have a global variable of name my_var and a local variable of name my_var, to access global my_var, you'll need to use the scope resolution operator.

example

```
#include <iostream>
using namespace std;

int my_var = 0;
int main(void) {
  int my_var = 0;
  ::my_var = 1;  // set global my_var to 1
  my_var = 2;    // set local my_var to 2
  cout << ::my_var << ", " << my_var;
  return 0;
}
```

Output

This will give the output −

```
1, 2
```

The declaration of my_var declared in the main function hides the integer named my_var declared in global namespace scope. The statement ::my_var = 1 accesses the variable named my_var declared in global namespace scope.

You can also use the scope resolution operator to use class names or class member names. If a class member name is hidden, you can use it by prefixing it with its class name and the class scope operator. For example,

C++ Theory

Example

```
#include <iostream>
using namespace std;
class X {
  public:
  static int count;
};
int X::count = 10;  // define static data member

int main () {
  int X = 0;   // hides class type X
  cout << X::count << endl;   // use static member of class X
}
```

Output

This will give the output −

```
10
```

Class: A class in C++ is the building block that leads to Object-Oriented programming. 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 C++ 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.

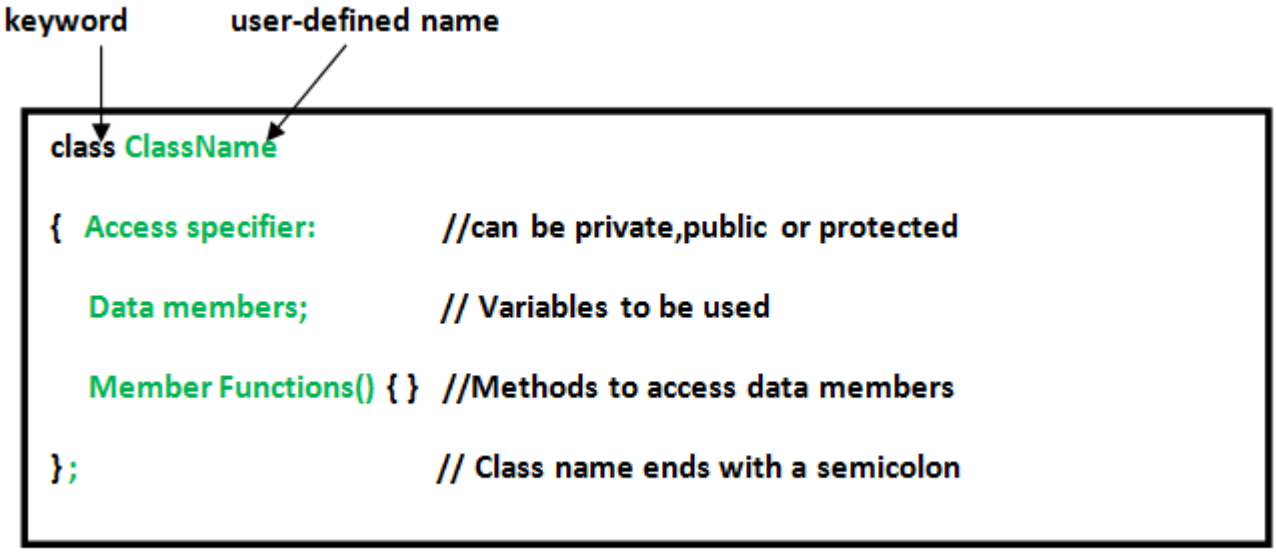
- A Class is a user defined data-type which has data members and member functions.
- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in 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.

Defining Class and Declaring Objects

C++ Theory

A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.



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.
Syntax:
ClassName ObjectName;
Accessing data members and member functions: The data members and member functions of class can be accessed using the dot('.') operator with the object. For example if the name of object is *obj* and you want to access the member function with the name *printName()* then you will have to write *obj.printName()* .

Accessing Data Members

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**.

```
// C++ program to demonstrate
// accessing of data members

#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    // Access specifier
    public:

    // Data Members
    string geekname;
```

```
    // Member Functions()
    void printname()
    {
      cout << "Geekname is: " << geekname;
    }
};

int main() {

    // Declare an object of class geeks
    Geeks obj1;

    // accessing data member
    obj1.geekname = "Abhi";

    // accessing member function
    obj1.printname();
    return 0;
}
```

Output:
Geekname is: Abhi

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.

```
// C++ program to demonstrate function
// declaration outside class

#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    public:
    string geekname;
    int id;

    // printname is not defined inside class definition
    void printname();

    // printid is defined inside class definition
    void printid()
    {
```

```
        cout << "Geek id is: " << id;
    }
};

// Definition of printname using scope resolution operator ::
void Geeks::printname()
{
    cout << "Geekname is: " << geekname;
}
int main() {

    Geeks obj1;
    obj1.geekname = "xyz";
    obj1.id=15;

    // call printname()
    obj1.printname();
    cout << endl;

    // call printid()
    obj1.printid();
    return 0;
}
```

Output:
Geekname is: xyz

Geek id is: 15

Note that all the member functions defined inside the class definition are by default **inline**, but you can also make any non-class function inline by using keyword inline with them. Inline functions are actual functions, which are copied everywhere during compilation, like pre-processor macro, so the overhead of function calling is reduced.

C++ Theory

Note: Declaring a friend function is a way to give private access to a non-member function.

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

```
// C++ program to demonstrate constructors

#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    public:
    int id;

    //Default Constructor
    Geeks()
    {
        cout << "Default Constructor called" << endl;
        id=-1;
    }

    //Parameterized Constructor
    Geeks(int x)
    {
        cout << "Parameterized Constructor called" << endl;
        id=x;
    }
};
int main() {

    // obj1 will call Default Constructor
    Geeks obj1;
    cout << "Geek id is: " <<obj1.id << endl;

    // obj2 will call Parameterized Constructor
    Geeks obj2(21);
    cout << "Geek id is: " <<obj2.id << endl;
    return 0;
}
```

Output:
Default Constructor called

Geek id is: -1

C++ Theory

Parameterized Constructor called

Geek id is: 21

A **Copy Constructor** creates a new object, which is exact copy of the existing object. The compiler provides a default Copy Constructor to all the classes.
Syntax:
class-name (class-name &){}

Destructors

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

```
// C++ program to explain destructors

#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    public:
    int id;

    //Definition for Destructor
    ~Geeks()
    {
        cout << "Destructor called for id: " << id <<endl;
    }
};

int main()
 {
    Geeks obj1;
    obj1.id=7;
    int i = 0;
    while ( i < 5 )
    {
        Geeks obj2;
        obj2.id=i;
        i++;
    } // Scope for obj2 ends here

    return 0;
  } // Scope for obj1 ends here
```

Output:
Destructor called for id: 0

Destructor called for id: 1

Destructor called for id: 2

Destructor called for id: 3

Destructor called for id: 4

Destructor called for id: 7