Operator overloading

17) When should a class overload the assignment operator and define the copy constructur?

→ The answer is same as Copy Constructor. If a class doesn't contain pointers, then there is no need to write assignment operator and copy constructor.

→ The compiler creates a default copy constructor and assignment operators for every class.

→ The compiler created copy constructor and assignment operator may not be sufficient when we have pointers or any run time allocation of resource like file handle, a network connection.

→ The **assignment operator** (operator=) is used to copy values from one object to another *already exiting operator*.

➢ **Assignment vs Copy constructor :**

→ The purpose of the copy constructor and the assignment operator are almost another *already exiting operator*.

→ However, the copy constructor initializes new objects, whereas the assignment operator replaces the contents of existing objects

.

→ The difference between the copy constructor and the assignment operator causes a lot of confusion for new programmers, but it's really not all that difficult.

- If a new object has to be created before the copying can occur, the copy constructor is used (note: this includes passing or returning objects by value).
- If a new object does not have to be created before the copying can occur, the assignment operator is used.

➢ **Overloading the assignment operator**

→ Overloading the assignment operator (operator=) is fairly straightforward, with one specific caveat that we'll get to. The assignment operator must be overloaded as a function member.

For example :

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

// A class without user defined assignment operator
class Test
{
int *ptr;
public:
Test (int i = 0)      { ptr = new int(i); }
void setValue (int i) { *ptr = i; }
void print()          { cout << *ptr << endl; }
};

int main()
{
Test t1(5);
Test t2;
t2 = t1;
t1.setValue(10);
t2.print();
return 0;
}
```

Output of above program is **"10".**

- **We can handle the above problem in two ways.**

1) Do not allow assignment of one object to other object. We can create our own dummy assignment operator and make it private.
2) Write your own assignment operator that does deep copy

- **Same is true for Copy Constructor**.

example of overloading assignment operator :

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

class Test
{
int *ptr;
public:
Test (int i = 0)      { ptr = new int(i); }
void setValue (int i) { *ptr = i; }
void print()          { cout << *ptr << endl; }
Test & operator = (const Test &t);
};

Test & Test::operator = (const Test &t)
{
// Check for self assignment
if(this != &t)
*ptr = *(t.ptr);

return *this;
}

int main()
{
Test t1(5);
Test t2;
t2 = t1;
t1.setValue(10);
t2.print();
return 0;
}
```

Output: 5

Operator overloading

17) When should a class overload the assignment operator and define the copy constructur?

An assignment operator is called when an object is already initialised and then again assigned a new value from another existing object.

EX:

```
#include<iostream>
Using namespace std;
Class simple
{
Private:
        Char* data;
Public:
Simple() { data=Null;}
Simple(char*S);
{
        Strcpy(data,S);
}
Char getdata()
{
        Return data;
}
Void setdata(char*S)
{
        if(data!=NULL)
Strcpy(data,S);
}
};
int main()
{
        Simple A("Mickey Mouse");
        Simple B;
        B=A;
        Simple(A);
        cout<<A.getdata()<<endl;
        cout<<B.getdata()<<endl;
        cout<<C.getdata()<<endl;
        A.setdata ("Donald Duck");
        cout<<A.getdata()<<endl;
        cout<<B.getdata()<<endl;
        cout<<C.getdata()<<endl;
        return 0;
        }
```

The copy assignment operator often just called the "assignment operator" is a special case of assignment operator where the source(right hand side)and destination(left hand side) are of the same class type.

Copy constructor:

Copy constructor is used to create a copy of an already existing object of a calss type.It is usually of the from X(X&) where X is the class name.

Syntax:

Classname(classname &object name)

Example:

```
#include<iostream>
```

```
        Using namespace std;
        Class Sample;
        {
        Private:
        Int x,y;
        Public:
        Sample(int x , int y)
        {
        X=x;
        Y=y;
        }
        Sample(Sample &sam)
        {
        X=sam.x;
        Y=sam.y;
        }

        void display()
        {
        Cout<<x<<"”<<y<<endl;
        }
        };
        main()
        {
        Samplea1(10,15);
        Sample a2=01;
        cout<<”Normal constructor:”;
        a1.display();
        cout<<”Copy constructor:”;
        a2.display();
}
```

18)What is overloading in OOP? What is function overloading and operator overloading in C++? Give suitable example.

Some Questions Related to C++

- Overloading refers to the ability to use a single identifier to define multiple methods of the class that differ in their input and output parameters.
- Overloading methods are generally used when they conceptually execute the same task but with a slightly different set of parameters.

Function overloading:

C++ allows the specification of more than one function of the same name in the same scope.These are called overloaded functions.

Overloaded functions enable you to supply different semantics for a function depending on the types and number of the arguments.

Operator overloading:

C++ provides a special function to change the current functionality of same operators within its class which is often called as operator overloading.

Operator overloading is the method by which we can change the function of the same specific operators to do saome different task.

Syntax:

```
Return type calssname::operator op(argument)
{
Function body
}
```

Operator function must be either non static(member function) or friend function.

Operator overloading can be done by using three approaches ,they are

1) Overloading unary operator
2) Overloading binary operator
3) Overloading binary operator using friend function

Example of function overloading

```
#include<iostream>
using namespace std;
class Data;
{
void print(int i);
{
{
cout<<"Printing int:"<<i<<endl;
}
void print(double f)
{
cout<<”Printing float:”<<f<<endl;
}
void print(char* c)
{
cout<<”Printing character:”<<c<<endl;
}
};
main()
{
Data d;
d.print(5);
d.print(5.636);
d.print(“Priyal kathrani”);
}
```

Example of operator overloading

```
#include<iostream>
using namespace std;
class Complex
{
Private:
        int real,imag;
Public:
        complex(int r=0,int i=0)
{
real=r;
imag=i;
}
complex operator+(complex &obj)
{
        complex res;
res.real=real+obj.real;
res.imag=real+obj.imag;
return res;
}
void print()
{
cout<<real<<:+i<<imag<<endl;
}
};
main()
{
complex c1(10,5);
complex c2(2.4);
complex c3=c1+c2;
c3.print();
}
```

19)What is conversion function? How is it created. Explain with example.

→ **Conversion functions** (C++ only) You can define a member **function** of a class, called a **conversion function**, that converts from the type of its class to another specified type.

→ A conversion function that belongs to a class X specifies a conversion from the class type X to the type specified by the conversion type.

→ The following code fragment shows a conversion function called operator int():

Ex :

```
class Y {
  int b;
public:
  operator int();
};
Y::operator int() {
  return b;
}
void f(Y obj) {
  int i = int(obj);
  int j = (int)obj;
  int k = i + obj;
}
```

→ All three statements in function f(Y) use the conversion function Y::operator int().

→ Classes, enumerations, typedef names, function types, or array types cannot be declared or defined in the *conversion_type*.

→ You cannot use a conversion function to convert an object of type A to type A, to a base class of A, or to void.

→ Conversion functions have no arguments, and the return type is implicitly the conversion type. Conversion functions can be inherited. You can have virtual conversion functions but not static ones.

Inheritance

20)Explain the difference between overriding and overloading member function of a base class in a derived class.

Ans:

Overloading	**Overriding**
Provides functionality to reduce method name for different arguments.	Provides functionality to override a behaviour which the class have inherited from parent class.
Occurs within a single class may also occur in child class.	Occurs in two classes that have child-parent or is a relationship.
Must have different argument list.	Must have the same argument list.
May have different return types.	Must have the same or variant return type.
May have different access specifier.	Must not have a more restrictive access modifier but may have less restrictive access specifier.
May throw different exceptions.	Must not throw new or broader checked eception but may throw no error vheckrd exceptions or any unchecked exception.
Function Overriding occurs when one class is inherited from another class.	Function Overloading can occur without inheritance

21)State how in private, public and protected inheritance ,the member of the base class are inherited by a derived class.

➢ **Public inheritance :**

→ Public inheritance is by far the most commonly used type of inheritance. In fact, very rarely will you see or use the other types of inheritance, so your primary focus should be on understanding this section.

→ Fortunately, public inheritance is also the easiest to understand. When you inherit a base class publicly, inherited public members stay public, and inherited protected members stay protected.

→ Inherited private members, which were inaccessible because they were private in the base class, stay inaccessible.

➢ **Private inheritance :**

→ private inheritance, all members from the base class are inherited as private. This means private members stay private, and protected and public members become private.

→ Note that this does not affect the way that the derived class accesses members inherited from its parent! It only affects the code trying to access those members through the derived class.

➢ **Protected inheritance:**

→ Protected inheritance is the last method of inheritance. It is almost never used, except in very particular cases.

→ With protected inheritance, the public and protected members become protected, and private members stay inaccessible.

Ex :

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

class A
{
public:
   int x;
protected:
   int y;
private:
   int z;
};

class B : public A
{
   // x is public
   // y is protected
   // z is not accessible from B
};

class C : protected A
{
   // x is protected
   // y is protected
```

```
    // z is not accessible from C
};

class D : private A    // 'private' is default for classes
{
    // x is private
    // y is private
    // z is not accessible from D
};
```

22)What is containership? How does it differ from inheritance?

Containership also referred to as composition allows a class to contain an object of a different class as a member data.

Ex: class A could contain an object of class b as a member.Here all the public methods defined in b can be executed within the class A. class A becomes the container while class B becomes the container class.This can also be enquired as class A is composed of class B.

Inheritance	**Containership**
Inheritance is the ability for a class to inherit properties and behaviour from a parent class by extending.	Containership is the ability of a class to contain object of different classes as a member data if a class is extended.
It inherits all the public and protected properties/behaviour and those behaviour may be overridden by the sub class.	Class is contained in another, the container does not get the ability change or add behaviour to contained.
Inheritance represents an "is-a" relationship.	Containership represent a "has-a" relationship

23. what is the ambiguity that arises in multiple inheritance? How it can be overcome. Explain with example.

- The order in which the superclass or parents are given define which property and/or method will be accessible by the conflict causing name; the others will remain hidden.
- The subclass must resolve the conflict by providing a property and/or method with the name and by defining how to use the ones from its parents .
- C++ user resolve the ambiguity with the ancestral class and a scope resolution operator.
- The first solution is not very convenient because it introduces implicit consequences, depending on the order in which classes inherit from each other.

- For the second case, subclasses must explicitly redefine properties which are involved in a naming conflict.

Example:

```
#include<iostream>
using namespacec std;
class LivingThing
{
protected:
void breathe()
{
cout<<"I am breathing as a living thing";<<endl;
}
};
class Animal : protected LiningThing
{
protected:
void breath()
{
cout<<I'm breathing as an animal."<<endl;
}
};
class Reptile  : protected LiningThing
{
protected:
void Grawl()
{
cout<<I'm breathing as an reptile."<<endl;
}
};
class Snake : protected Animal,protected Reptile
{
protected:
void breathe()
{
cout<<I'm breathing as an snake."<<endl;
}
void crawl()
{
cout<<I'm crawling as an snake."<<endl;
}
};
main()
{
Snake s;
s.breathe()
s.crawl();
}
```

Output:

```
I'm breathing as a snake.
I'm crawling as a snake.
```

Pointers and Virtual Functions

24. Suppose that you have the declaration int *numptr;. what is the difference between the expressions: *numptr and &numptr?

Difference between *numptr and &numptr :

***numptr**	**&numptr**
we are writing *numptr then it is called as integer pointer variable	we are writing &numptr then it is called as reference variable
pointer can point to a perticular memory address and pointer variable has own memory address	reference variable dosn't have it's own memory address
It is that kind of variable which can store the address of another variable.	it is just alias name of variable to which it is pointing.
In pointer variable we have two memory spaces 1)memory of address 2)memory of pointer variable.	where for reference variable we have only one memory address for varibale and refrence varibale.

25. What are the different type of polymorphism? What is the difference between compile time binding and runtime binding? Write down synonyms of the same functionality.

Polymorphism is a feature of OOPC that allows the object to have differently in different condition.

Two types of polymorphisms:

1) Compile time polymorphism: - This is also known as static or early binding.
2) Runtime polymorphism: - This is also known as dynamic or late binding.

Compile time binding	**Runtime binding**
The static binding happens at the compile time and early binding.	The dynamic binding happens at the run time binding and late.
The function definition and the function call are linked during the compile time.	The function calls are not resolved until runtime so they are bound.
It happens when all information needed to call a function is available at the compile time.	It happens when all information for a function call cannot be determined at compile time.
It can be achieved using the normal function calls, function overloading and operator overloading.	It can be achieved using the virtual functions.
It is flexible since a single function can handle different type of objects at runtime.	This significantly reduce the size of codebase and also makes the source code mode readable.
Static binding result in faster.	Dynamic binding result in slower.

26. Write down a short note on Abstract class. Is it legal to have an abstract class with all member functions pure virtual?

→ Abstract Class is a class which contains atleast one Pure Virtual function in it.
→ Abstract classes are used to provide an Interface for its sub classes.
→ Classes inheriting an Abstract Class must provide definition to the pure virtual function, otherwise they will also become abstract class.

➢ **Characteristics of Abstract Class :**

→ Abstract class cannot be instantiated, but pointers and refrences of Abstract class type can be created.

→ Abstract class can have normal functions and variables along with a pure virtual function.

→ Abstract classes are mainly used for Upcasting, so that its derived classes can use its interface.

→ Classes inheriting an Abstract Class must implement all pure virtual functions, or else they will become Abstract too.

Ex :

```
//Abstract base class
class Base
{
   public:
   virtual void show() = 0;    // Pure Virtual Function
};

class Derived:public Base
{
   public:
   void show()
   {
      cout << "Implementation of Virtual Function in Derived class\n";
   }
};

int main()
{
   Base obj;   //Compile Time Error
   Base *b;
   Derived d;
   b = &d;
   b->show();
}
```

27. What are the difference between reference and pointer?

Reference	Pointers
A reference & is like an alias to an existing variable	A pointer is a variable that holds a memory address.
Reference unlike pointer have to be initialized at the point of definition.	Pointer can be initialized at any time.
A reference can refer to only one object during its lifetime i.e. a reference cannot be rebound.	A pointer can point to many different objects during its lifetime.
Array of reference cannot be created as each reference in the array cannot be initialized at the time of creation.	Array of pointer can be created.
No operator is needed to de reference a reference.	One hasto use an explicit operator to de reference a pointer.
A valid reference must refer to an object.	Pointer need not refer to any object.
Reference cannot be NULL.	A pointer even a constant pointer can have a NULL value.

28. What is a virtual function? Write rules for virtual function. Explain with example.

→ A virtual function a member function which is declared within base class and is re-defined (Overriden) by 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 Run-time.

➤ **Rules for Virtual Functions :**

❶ They Must be declared in public section of class.

❷ Virtual functions cannot be static and also cannot be a friend function of another class.

❸ Virtual functions should be accessed using pointer or reference of base class type to achieve run time polymorphism.

❹ The prototype of virtual functions should be same in base as well as derived class.

❺ They are always defined in base class and overridden in derived class. It is not mandatory for derived class to override (or re-define the virtual function), in case base class version of function is used.

❻ A class may have virtual destructor but it cannot have a virtual constructor.

❼ Compile-time(early binding) VS run-time(late binding) behavior of Virtual Functions

Program :

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

class base
{
public:
   virtual void print ()
   { cout<< "print base class" <<endl; }
```

```
    void show ()
    { cout<< "show base class" <<endl; }
};

class derived:public base
{
public:
    void print ()
    { cout<< "print derived class" <<endl; }

    void show ()
    { cout<< "show derived class" <<endl; }
};

int main()
{
    base *bptr;
    derived d;
    bptr = &d;

    //virtual function, binded at runtime
    bptr->print();

    // Non-virtual function, binded at compile time
    bptr->show();
}
```

Output:

```
print derived class
show base class
```

29. Explain C++ stream classes. Describe briefly the feature of I/O system supported by c++ with reference to c++ stream class.

The c++ I/O system contains a heirchy of classes that are used to define various streams to deal with both the console and disk file.

These classes are called stream classes.
These classes are declared in iostream class.

Input stream: - The source stream that providesdata to the program.
Output stream: - The destination stream that receive output from the program.

The data in the input stream can come from keyboard or any other storage device.

The data in the output stream can go to the screen any other storage device.

The c++ I/O Ios provides the asic support for formatted and unformatted I/O operators.

istream: provides the facilities for formatted and unformatted input.
ostream: provides the facilities for formatted output.

30. What do you mean by manipulators? How do we create user defined manipulator? Discuss with example.

Manipulators are special functions that can be used for getting formatted output.

Before using the manipulators the file iomanip should be included in the program.

Manipulators are classified in two parts:

1) Manipulator operators
2) Manipulator functions

Designing our own Manipulators:
We can design our own Manipulator for certain special purpose.

Syntax:

```
ostream &manip_name(ostream &output)
{
        return output;
}
```

Example:

```
ostream &unit(ostream &output)
{
        output<<"inches";
        return output;
}
cout<<36<<unit;
```

it will produce “36 inches”.

31. Explain different file operations. How many ways are there to open file.

→ Ofstream : Stream class to write on files
→ ifstream : Stream class to read from files
→ fstream : Stream class to both read and write from files.

➢ **<u>Open a file :</u>**

→ In order to open a file with a stream object we use its member function open:

Syntax : fobj.open(“abc.txt”);

ios::in	Open for input operations.
ios::out	Open for output operations.
ios::binary	Open in binary mode.
ios::ate	Set the initial position at the end of the file. If this flag is not set, the initial position is the beginning of the file.
ios::app	All output operations are performed at the end of the file, appending the content to the current content of the file.
ios::trunc	If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.

➢ **Close the file :**

→ it is used for close the file

Syntax : fobj.close();

➢ **Checking state flags :**

bad() :

Returns true if a reading or writing operation fails. For example, in the case that we try to write to a file that is not open for writing or if the device where we try to write has no space left.

fail() :

Returns true in the same cases as bad(), but also in the case that a format error happens, like when an alphabetical character is extracted when we are trying to read an integer number.

eof() :

Returns true if a file open for reading has reached the end.

good() :

It is the most generic state flag: it returns false in the same cases in which calling any of the previous functions would return true. Note that good and bad are not exact opposites (good checks more state flags at once).

➢ **seekg() and seekp() :**

→ These functions allow to change the location of the *get* and *put positions*. Both functions are overloaded with two different prototypes.

syntax :

seekg (offset, direction);
seekp (offset, direction);

→ Using this prototype, the get or put position is set to an offset value relative to some specific point determined by the parameter direction. offset is of type streamoff. And direction is of type seekdir, which is an *enumerated type* that determines the point from where offset is counted from, and that can take any of the following values:

ios::beg	offset counted from the beginning of the stream
ios::cur	offset counted from the current position
ios::end	offset counted from the end of the stream

➢ **Ways of open the file :**

→ 2 ways of open the file

(a) Using constructor

(b) Using open function

(a) using constructor :

Ex :

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

int main () {
  ofstream myfile;
  myfile.open ("example.txt");
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
```

(b) using open function :

Ex :

```
#include <iostream>
#include <fstream>
using namespace std;
int main ()
{
   ofstream fin("myfile", ios::out) ;
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
}
```

32. Briefly explain error handling functions of file.

a) eof()
   Returns true (non-zero) value of end of file is encountered while reading otherwise return false (zero).

b) fail()
   Return true when an input or output operation has failed.

c) bad()
   Returns true if an invalid operation is attempted or any unrecoverable error has occurred.

   However, if it is false it may be possible to recover from any other error reported and continue operation.

d) good()
   Return true if no error has occurred.