Spring- 22

Ans. To Question No.1(a)

Operator Overloading:

Operator overloading is a compile-time polymorphism in which the operator is overloaded to provide the special meaning to the user-defined data type. Operator overloading is used to overload or redefines most of the operators available in C++.

Rules for Operator Overloading:

- Existing operators can only be overloaded, but the new operators cannot be overloaded.
- The overloaded operator contains at least one operand of the user-defined data type.
- We cannot use friend function to overload certain operators. However, the member function can be used to overload those operators.

Ans. To Question No.1(b)

```
#include<iostream>
using namespace std;
class Float
{
   float mem;
public:
   Float(): mem(5) {}
   Float(float x): mem(x) {}
   Float operator + (Float a)
   {
      Float temp;
      temp.mem = mem + a.mem;
      return temp;
   }
   void show()
   {
      cout<<mem<<endl;
   }
};
int main()
{
   Float a = 10.6,b = 5.3,c;
   c = a + b;
   cout<<"a + b = ";
   c.show();
   return 0;
}
```

Ans. To Question No.1(b or)

```
# include <iostream >
using namespace std ;
class coord
{
   int x,y;
public :
   coord()
   {
      x=0;
      y=0;
   };
   coord (int i,int j)
   {
      x=i;
      y=j;
   }
   void get_xy(int &i,int &j)
   {
      i=x;
      j=y;
   }
   friend coord operator && (coord ob1,coord ob2);
};
coord operator &&(coord ob1,coord ob2)
{
   coord temp;
   temp.x = ob1.x && ob2.x;
   temp.y = ob1.y && ob2.y;
   return temp;
}
int main ()
{
   coord o1 (10, 10), o2 (1, 3),o3;
   int x,y;
   o3=o1&&o2;
   o3.get_xy(x,y);
   if(x==y)
      cout << "o1 && o2 is true \n";
   else
      cout << "o1 && o2 is false \n";
   return 0;
}
```

Ans. To Question No.1(c)

A friend function cannot be used to overload the assignment operator (=). Because-
Assignment (=) operator is a special operator that will be provided by the constructor to the class when programmer has not provided(overloaded) as member of the class. (like copy constructor). When programmer is overloading = operator using friend function, two = operation will exist:
1) compiler is providing = operator
2) programmer is providing (overloading) = operator by friend function.
Then ambiguity will be created and compiler will give an error.

Ans. To Question No.1(d)

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 +.

It is an idea of giving special meaning to an existing operator in C++ without changing its original meaning.
Example:

```
int a;
float b,sum;
sum=a+b;
```

Here, variables “a” and “b” are of types “int” and “float”, which are built-in data types. Hence the addition operator ‘+’ can easily add the contents of “a” and “b”. This is because the addition operator “+” is predefined to add variables of built-in data type only.

Operator Overloading does not mean changing the function and behavior of that particular operator forever or for the built-in data types. It only facilitates to perform operations on user defined data types similar to the ones we do on integral types.

Ans. To Question No.2(a)

No. A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.

Accessibility	Private	Public	Protected
From own class	Yes	Yes	Yes
From derived class	No	Yes	Yes
Outside derived class	No	Yes	No

Ans. To Question No.2(b)

```
#include <iostream>
using namespace std ;
class A
{
public:
   void f1()
   {
      cout<<"Class A"<<endl;
   }
};
class B
{
public:
   void f2()
   {
      cout<<"Class B"<<endl;
   }
};
class C
{
public:
   void f3()
   {
      cout<<"Class C"<<endl;
   }
};
class D:public C,public B,public A
{
public:
   void f4()
   {
      cout<<"Class D"<<endl;
   }
```

```
};
int main ()
{
   D d;
   d.f1();
   d.f2();
   d.f3();
   d.f4();
   return 0;
}
```

Ans. To Question No.2(c)

To call the parameterized constructor of base class inside the parameterized constructor of sub class, we have to mention it explicitly.
Whenever the derived class's default constructor is called, the base class's default constructor is called automatically.
The parameterized constructor of base class cannot be called in default constructor of sub class, it should be called in the parameterized constructor of sub class.

```
#include <iostream>
using namespace std;
class Parent
{
   int x;

public:
   Parent(int i)
   {
      x = i;
      cout << "Inside base class's parameterized "
         "constructor"
         << endl;
   }
};
class Child : public Parent
{
public:
   Child(int x): Parent(x)
   {
      cout << "Inside derived class's parameterized "
         "constructor"
         << endl;
   }
};
```

```
int main()
{
   Child obj1(10);
   return 0;
}
```

Ans. To Question No.2(d)

claas A and claas B has same type of member function with no parameter so its ambiguous to compiler when we calling it from derived class.
Correct Code:

```
#include <iostream>
using namespace std;
class A
{
public:
  void cheers()
  {
     cout<<"Class A:Hip-Hip hooray";
  }
};
class B
{
public:
   void cheers()
   {
      cout<<"Class B:Hip-Hip hooray";
   }
};
class C: public A,public B
{
};
int main()
{
   C obc;
   obc.A::cheers();
   return 0;
}
```

Output: Class A:Hip-Hip hooray

Ans. To Question No.3(a)

Virtual Function:

A virtual function is a member function which is declared within a base class and is re-defined (overridden) by a derived class. When virtual functions are used, a program that appears to be calling a function of one class may in reality be calling a function of a different class.

```
#include<iostream>
using namespace std;
class base
{
public:
   virtual void print()
   {
      cout << "print base class\n";
   }
   void show()
   {
      cout << "show base class\n";
   }
};
class derived : public base
{
public:
   void print()
   {
      cout << "print derived class\n";
   }
   void show()
   {
      cout << "show derived class\n";
   }
};
int main()
{
   base *bptr;
   derived d;
   bptr = &d;
   bptr->print();
   bptr->show();
   return 0;
}
```

Explanation:

Runtime polymorphism is achieved only through a pointer (or reference) of base class type. Also, a base class pointer can point to the objects of base class as well as to the objects of derived class. In above code, base class pointer 'bptr' contains the address of object 'd' of derived class.

Late binding (Runtime) is done in accordance with the content of pointer (i.e. location pointed to by pointer) and Early binding (Compile time) is done according to the type of pointer, since print() function is declared with virtual keyword so it will be bound at runtime (output is print derived class as pointer is pointing to object of derived class) and show() is non-virtual so it will be bound during compile time (output is show base class as pointer is of base type).

Ans. To Question No.3(b)

Early binding:

Early binding essentially refers to those events that can be known at compile time. Specifically, it refers to those function calls that can be resolved during compilation.

Late binding:

Late binding refers to events that must occur at run time. A late bound function call is one in which the address of the function to be called is not known until the program runs. In C++, a virtual function is a late bound object.

Pros and Cons of Early binding:

The main advantage of early binding (and the reason that it is so widely used) is that it is very efficient. Calls to functions bound at compile time are the fastest types of function calls. The main disadvantage is lack of flexibility.

Pros and Cons of Late binding:

The main advantage of late binding is flexibility at run time. Its primary disadvantage is that there is more overhead associated with a function call. This generally makes such calls slower than those that occur with early binding. Because of the potential efficiency trade-offs, you must decide when it is appropriate to use early binding and when to use late binding.

Ans. To Question No.3(b or)

If a virtual function is specified by placing "= 0" in its declaration then the function called pure virtual function.

```
#include<iostream>
using namespace std;
class Base
{
public:
   virtual void show() = 0;
};
class Derv1 : public Base
{
public:
   void show()
   {
      cout << "Derv1\n";
   }
```

```
};

class Derv2 : public Base
{
public:
   void show()
   {
      cout << "Derv2\n";
   }
};
int main()
{
   Base* arr[2];
   Derv1 dv1;
   Derv2 dv2;
   arr[0] = &dv1;
   arr[1] = &dv2;
   arr[0]->show();
   arr[1]->show();
   return 0;
}
```

Ans. To Question No.3(c)

```
#include<iostream>
using namespace std;
class Base //base class
{
public:
   virtual void show()
   {
      cout << "Base\n";
   }
};
class Derv1 : public Base
{
public:
   void show()
   {
      cout << "Derv1\n";
```

```
    }
};
class Derv2 : public Base
{
public:
    void show()
    {
        cout << "Derv2\n";
    }
};
int main()
{
    Derv1 dv1;
    Derv2 dv2;
    Base* ptr;
    ptr = &dv1;
    ptr->show();
    ptr = &dv2;
    ptr->show();
    return 0;
}
```

Ans. To Question No.4(a)

When we write a simple c++ program, the first thing that we see when a program doesn't work correctly are bugs. Most common bugs are logical errors and syntactic errors. We can overcome these bugs by debugging but sometimes we come across some problems other than logical or syntax errors. These are known as Exceptions.

Advantages of exception handling:

(a) Exception handling can control run tune errors that occur in the program.
(b) It can avoid abnormal termination of the program and also shows the behavior of program to users.
(c) It can provide a facility to handle exceptions, throws message regarding exception and completes the execution of program by catching the exception
(d) It can separate the error handling code and normal code by using try-catch block.
(e) It can produce the normal execution flow for a program.
(f) It can implement a clean way to propagate error. that is. when an invoking method cannot manage a particular situation, then it throws an exception and asks the invoking method to deal with such situation.

Ans. To Question No.4(b)

```
#include<iostream>
using namespace std;
template<class X> int Min(X a,X b)
{
   if(a<b)
      return a;
   else
      return b;
}
int main()
{
   int a,b;
   cin>>a>>b;
   cout<<Min(a,b)<<endl;
   return 0;
}
```

Ans. To Question No.4(c)

Generic Function:

A generic function defines a general set of operations that will be applied to various types of data. A generic function has the type of data that it will operate upon passed to it as a parameter. A generic function is created using the keyword **template**. The normal meaning of the word template accurately reflects the keyword's use in C++. It is used to create a template (or framework) that describes what a function will do, leaving it to the compiler to fill in the details as needed.

Generic Class:

To defining generic functions, we can also define generic classes. When we do this, we create a class that defines all algorithms used by that class, but the actual type of the data being manipulated will be specified as a parameter when objects of that class are created. Generic classes are useful when a class contains generalizable logic. For example, the same algorithm that maintains a queue of integers will also work for a queue of characters. Also, the same mechanism that maintains a linked list of mailing addresses will also maintain a LinkedList of auto part information. By using a generic class, we can create a class that will maintain a queue, a linked list, and so on for any type of data. The compiler will automatically generate the correct type of object based upon the type we specify when the object is created.

Ans. To Question No.5(a)

Stream:

A C++ stream is a flow of data into or out of a program, such as the data written to cout or read from cin.

In general, a Stream will be an input stream or, an output stream. InputStream − This is used to read data from a source. OutputStream − This is used to write data to a destination.

Stream classes hierarchy :

```
#include <iostream>
#include <cstdlib>
int main()
{
   using namespace std;
   cout << "Enter your age: ";
   int nAge;
   cin >> nAge;
   if (nAge <= 0)
   {
      cerr << "Oops, you entered an invalid age!" << endl;
      exit(1);
   }
   cout << "You are " << nAge << " years old" << endl;
   return 0;
}
```

Ans. To Question No.5(b)

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

int main()
{
   cout.setf(ios::showpos);
   cout.setf(ios::scientific);
   cout << 123 << " " << 123.23 << endl;

   cout.precision(2);
   cout.width(10);
   cout << 123 << " ";
   cout.width(10);
   cout << 123.23 << endl;
```

```
   cout.fill('#');
   cout.width(10);
   cout << 123 << " ";
   cout.width(10);
   cout << 123.23;

   return 0;
}
```

Ans. To Question No.5(c)

```
#include <iostream>
#include <fstream>

using namespace std;

int main(){

  char Name[200];
  char semester[50];
  char code[20];
  char title[100];

  fstream file;
  file.open ("WhoAreYou.txt", ios::out | ios::in );

  cout << "Write text to be written on file." << endl;
  cout<<"Write your name: ";
  cin.getline(Name, sizeof(Name));
  file << Name << endl;
  cout<<"Write Semester: ";
  cin.getline(semester, sizeof(semester));
  file << semester << endl;
  cout<<"Write Course Code: ";
  cin.getline(code, sizeof(code));
  file << code << endl;
  cout<<"Write Course Title: ";
  cin.getline(title, sizeof(title));
  file << title << endl;

  file >> Name;
  file >> semester;
  file >> code;
```

```
 file >> title;

 cout<<"*Output From File*"<<endl;
 cout << "Name: "<<Name << endl;
 cout << "Semester: "<<semester << endl;
 cout << "Course Code: "<<code << endl;
 cout <<"Course Title: "<< title << endl;
 file.close();
 return 0;
}
```