Assignment 2

Page No.
Date

① State the OOP's need and any 2 Characteristic of C++. Mention advantages of OOP approch over the func^n.

→

As the name suggests, Object-Oriented Programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming where Class and Objects are used.

i) Using OOPs methodology, one can enhance the code reusability and save development time.
ii) Using functionalities like data abstraction and hiding, OOPs ensure the security of code.
iii) OOP concepts like inheritance help to eliminate the need for code redundancy.

Characteristics of C++

i) It is Object-Oriented Programming
ii) Machine independent.
iii) Multi-threading.
iv) High-level language.

Advantages of OOP:

i) OOP language allows to break the program into the bit-sized problems that can be solved easily
ii) The new technology promises greater programmer productivity, better quality of software and lesser maintainance cost.
iii) OOP systems can be easily upgraded from small to large systems.
iv) It is very easy to partition the work in a project based on objects.

② What is constructor ? Explain all types of constructor.

→ Constructors in C++ is a special method that is invoked automatically at the time of object creation. The constructor has same name as the class or structure.
It constructs the values i.e provides data for the object which is why it is known as constructors.

Syntax (constructor within class) :

```
class_Name (list-of-parameters){
        // Constructor defination.
}
```

Syntax (outside the class) :

```
class_Name :: class_Name (list of parameters){
        // defination.
}
```

Types of Constructors :

1. Default Constructors :

Default constructor is the constructor which doesn't take any argument. It has no parameters

```
class example {
  public:
    int x, y;
    example (){
        x = 15;
        y = 17;
    }
};
int main (){
    example one;
    cout << "a: " << one.a << endl << "b: " << one.b << endl;
}
```

2. Parameterized Constructors :

It is possible to pass arguments to constructors Typically, these arguments help initialize an object when it is created.

```
class example {
  private:
    int x, y;
  public:
    example (int a, int b) {
        x = a;
        y = b;
    }
    int getX() { return x; }
    int getY() { return y; }
};
```

```
int main(){
    example para(11, 33);
    cout << " para.x : " << para.getX() << endl;
    cout << " para.y : " << para.getY() << endl;
};
```

3. Copy Constructor :

A copy constructor is a member function that intializes an object using another object of the same class.

```
class example {
    int id;
  Public :
    void assign (int a){
        id = a;
    }
    void display (){
        cout << " ID is → " << id;
    }
};
int main (){
    example copy1;
    copy1.assign (111);
    copy1.display ();
    example copy2 (copy1);
    copy2.display ();
    return 0;
}
```

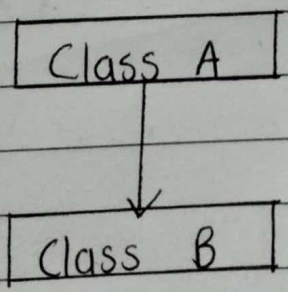
③ Explain different types of inheritance.

→

The capability of a class to derive properties and characteristics from another class is called inheritance. The existing class is known as the 'base class'. The new class created is called 'derived class'.

Types of inheritance:
1) Single inheritance
2) Multilevel inheritance
3) Multiple inheritance
4) Hierarchical inheritance
5) Hybrid inheritance.

1) Single Inheritance: In single inheritance, a class is allowed to inherit from only one class.



For example:

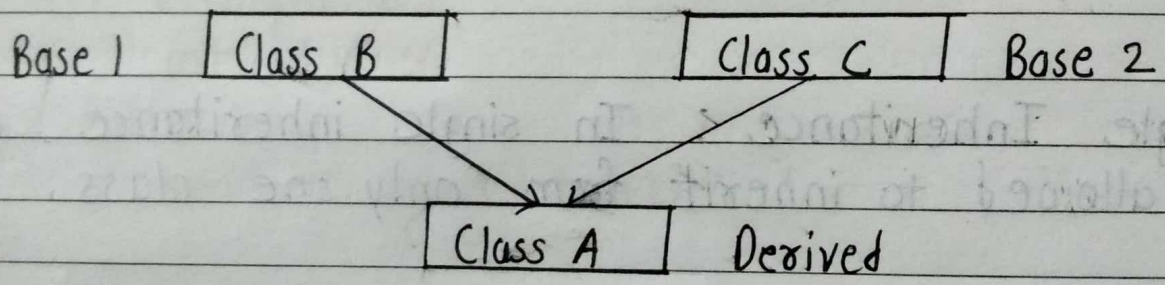
```
class Vehicle {
    public:
    Vehicle(){
        cout << "This is a vehicle \n";
    }
};
```

```
class Car : public Vehicle { };

int main (){
        Car obj;
        return 0;
}
```

Output : This is a vehicle.

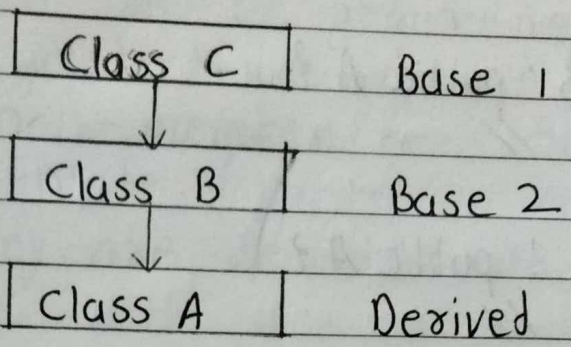
2) Multiple Inheritance : Multiple inheritance is a feature of C++ where a class can inherit from more than one class.



Syntax:

```
class B {
    // statements
};
class C {
    // statements
};
class A : public B, public C {
    // statements
};
```

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



Syntax :

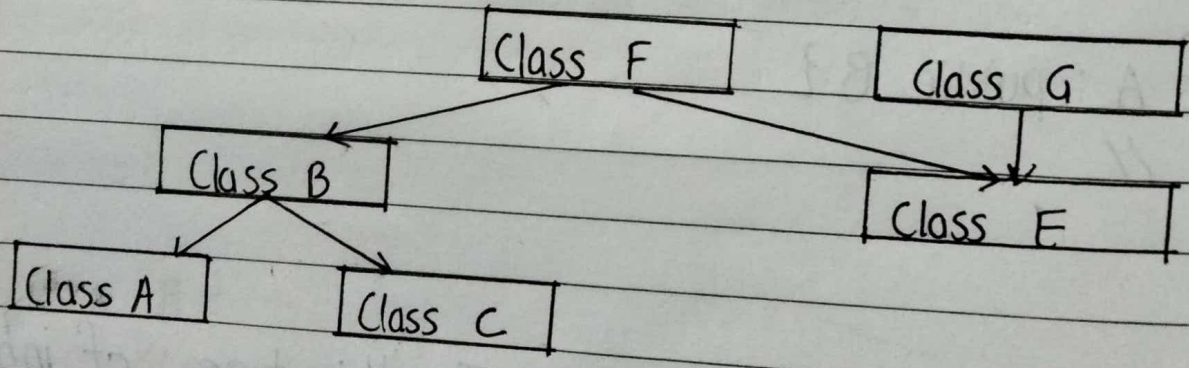
```
class C {
    //
};
class B : public C {
    //
};
class A : public B {
    //
};
```

4) Hierarchical Inheritance : In this type of inheritance, more than one subclass is inherited from a single base class.

Syntax :

```
class A {
    //
};
class B : public A {
    //
};
class C : public A {
    //
};
class D : public A {
    //
};
```

5) Hybrid Inheritance : Hybrid inheritance is implemented by combining more than one type of inheritance. For ex, combining Hierarchical and Multiple inheritance



④ What is friend function? State its uses.

→

A friend function can be granted special access to private and protected members of a class in c++. They are non-member func^n that can access and modify the private & protected members of the class for they are declared as friends.

Syntax:

```
class class_Name {
    friend data_type func_name (arguments);
};
```

In the above syntax, the friend function is preceded by the keyword friend. The friend function can be defined anywhere in the program.

- A friend function is declared when we want to access the non-public data members of a particular class.

⑤ What are virtual function ? Explain with example. How it is different from simple func^n.

→

A virtual function is also known as virtual methods. It is a member function that is declared within a base class and is re-defined by 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 derived class's version of method.

① Virtual functions ensure that the correct function is called for an object, regardless of the type of reference used for the function call.
② Virtual func^n are declared with a virtual keyword in a base class.
③ Virtual functions cannot be static
④ The prototype of virtual functions should be the same in the base as well as the derived class.
⑤ Virtual functions can be a friend function of another class.

```
class base{
  public :
      virtual void print(){
          cout<<" Print Base"<<endl;
      }
      void show(){
          cout<<" Show Base\n";
      }
};
```

```
class derived : public base {
    public :
        void print (){
            cout << " Print Derived \n";
        }
        void show (){
            cout << "Show Derived \n";
        }
};
int main (){
    base * bptr;
    derived d;
    bptr = &d;

    bptr -> print ();
    bptr -> show ();
    return 0;
}
```

Output :
Print Derived
Show Base

Limitations of Virtual Functions :

1) Slower
2) Difficult to Debug :

i) Virtual function get executed at run-time and normal function get executed at compile time.
ii) A virtual function cannot static whereas normal function can be static.

Q.6) Compare Compile & runtime polymorphism. What is polymorphism? how achieved at ① Compile time ② Runtime.

→

Polymorphism is one of the most important OOPs concepts. Its is a concept by which we can perform single task in multiple ways. There are two types of polymorphism one is compile-time polymorphism and another is run-time polymorphism

Compile-time polymorphism:
1) Function overloading.
2) Operator overloading.

Run time polymorphism.
1) Function overriding
2) Virtual function.

	Compile-time polymorphism	Runtime polymorphism.
i)	Compile-time polymorphism means binding is occuring at compile time	i) Runtime polymorphism where at run time we come to know which method is going to invoke.

ii)	It can be achieved through static binding	ii) It can be achieved through dynamic binding.
iii)	Inheritance is not involved	iii) Inheritance is involved.
iv)	Function overloading is an example of compile time polymorphism	iv) Function overriding is an example of runtime polymorphism.

Example of Compile-time

```
Class Example {
public:
    void func (int x){
        cout << " First function " <<endl;
    }
    void func (int x, int y){
        cout << "Second function" <<endl;
    }
};

int main(){
    Example obj;
    obj.func (7);
    obj.func (8,9);
    return 0;
}
```

Output:
First function
Second function.

Example of runtime.

```
class Animal {
public:
    string color = "Black";
};

class Dog : public Animal {
public: string color = "Red";
};

int main (void) {
    Animal d = Dog();
    cout << d.color;
}
```

Output : Black.

---

(7) Difference between inheritance and polymorphism.

→

	Inheritance	Polymorphism
i)	Inheritance is one in which a new class is created that inherits the features from the already existing.	i) Polymorphism is that which can be defined in multiple forms.
ii)	It is basically applied to classes.	ii) It is basically applied to functions or methods.
iii)	It supports the concept of reusability and reduce code length in oop.	iii) Polymorphism allows the objec to decide which form of the function to implement at compile time and run-time.

iv) Inheritance can be single, hybrid, multiple, hierarchical and multilevel inheritance	iv) It can be compiled-time polymorphism (overload) as well as run-time polymorphism (overriding).
v) It is used in pattern designing.	v) It is also used in pattern designing.
vi) Example: The class bike can be inherit from the class of two-wheel vehicles, which is turn could be a sub-class of vehicles	vi) Example: The class bike can have method name set_color(), which changes the bike's color based on the name of color you have entered.

(8) What is difference between compile-time & run time binding?

Compile time binding	Run time binding.
i) Compile time binding also called as static or early binding	i) Runtime binding also called as Dynamic or late binding.
ii) It happens at compile time	ii) It happens at run time.
iii) It can be achieved during normal func^n calls, function - operator overloading	iii) It is achieved with the use of virtual functions.
iv) It provides less flexibility compared to dynamic binding	iv) It provides more flexibility as different types of objects at runtime can be handled by a single function call making the source code more readable.

v) When all the information needed to call a function is available at the compile time, static binding occurs.	v) When the compiler cannot determine all the information to resolve the function call, dynamic binding occurs.
vi) Ex. Function overloading & operator overloading	vi) Ex. Function overriding.