Assignment 4
Object-Oriented Programming Features

1.1 What are the basic elements of Object-Oriented Programming?

SOL. Basic elements of Object-Oriented Programming are :

- Abstraction
- Encapsulation
- Polymorphism
- Inheritance

1.2 "Abstraction and encapsulation are complementary to each other." Justify this statement.

SOL. Encapsulation and abstraction are complementary to each other. Abstraction complements encapsulation by exposing to the user of an object a clear and well-defined interface to that object. So without abstraction, there would be no meaning of encapsulation.

Similarly, abstraction would not be useful without encapsulation. If the object and function would not be together, functions would be operating from outside. So data would not be safe from the outside world. It would defeat the actual purpose of abstraction.

1.3 What is Inheritance? What are its advantages?

SOL. Inheritance is the capability of a class to derive properties and characteristics from another class. Inheritance is one of the most important feature of Object Oriented programming. The main advantages of inheritance are code reusability and readability. When child class inherits the properties and functionality of parent class, we need not to write to same code again and again in child class. This makes it easier to reuse the code.

1.4 Write short notes on the following:
i) Data abstraction
ii) Encapsulation
iii) Polymorphism

SOL. i)Data Abstraction: Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation. We can implement Abstraction in c++ using classes.
Class helps us to group data members and member functions using available access specifiers

ii)Encapsulation: Encapsulation is defined as wrapping up of data and information under a single unit. In Object Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulates them. Encapsulation also leads to data abstraction or hiding, It can be implemented using Class and access specifiers.

iii)Polymorphism: Polymorphism is a powerful feature of the object-oriented programming language C++. In Polymorphism "Poly" means "many" and "morph" means "form". Thus, polymorphism is the ability to take many different forms at the same time. Polymorphism enables one common interface for many implementations; thus, object can act differently under different circumstances.

1.5 Differentiate between run-time and compile-time polymorphism.

SOL. Runtime polymorphism: This type of polymorphism is achieved by Function overriding. Function overriding occurs when a derived class has a definition class has a definition for one of the member functions of the base class. That base function is said to be overridden.

Compile time polymorphism: This type of polymorphism is achieved by function overloading or operator overloading.

- Function overloading: When there are multiple functions with same name but different parameters then these functions are said to be overloaded.
- Operator Overloading: C++ also provide option to overload operators. Like we can use ’+’ to add two numbers as well as concatenate two strings.

1.6 Define classes and objects.

SOL. Class: A class 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.

Object: An object is an instance of a class. When a class is defined, no memory is allocated but when it is initialized(when object is created) memory is allocated.

1.7 What is the basic difference between structures and classes in C++?

SOL. A class is a user-defined blueprint or prototype from which objects are created. Basically, a class combines the fields and methods(member functions which defines actions) into a single unit. A structure is a collection of variables of different data types under a single unit.

1.8 Design a class student with name, age, course, address, and college name. Also define member functions to input values for a student, to display details of a student and to modify details of a student.

SOL.

```
class student{
string name;
int age;
string course;
string address;
string college_name;
public:
void inputv()
{
   cout<<"enter name : ";
   getline(cin, name);
   cout<<"\nenter age : ";
```

```
    cin.ignore();
    cin>>age;
    cin.ignore();
    cout<<"\nenter course : ";
    getline(cin, course);
    cout<<"\n enter address : ";
    getline(cin, address);
    cout<<"\nenter college_name : ";
    getline(cin, college_name);
}
void outputv()
{
    cout<<"Name : "<<name<<endl;
    cout<<"Age : "<<age<<endl;
    cout<<"Course : "<<course<<endl;
    cout<<"Address : "<<address<<endl;
    cout<<"College Name : "<<college_name<<endl;
}
void modifyv()
{
    cout<<"enter name : ";
    getline(cin, name);
    cout<<"\nenter age : ";
    cin.ignore();
    cin>>age;
    cin.ignore();
    cout<<"\nenter course : ";
    getline(cin, course);
    cout<<"\n enter address : ";
    getline(cin, address);
    cout<<"\nenter college_name : ";
    getline(cin, college_name);
}
};
```

1.9 Discuss various methods for defining member functions of a class with the help of suitable examples.

SOL. Member function can be defined inside the class definition or separately using scope resolution operator.
For example:

Defining member function inside the class

```
class rectangle{
   public:
      double length;
      double breadth;

double area()
{ return length*breadth; }
};
```

Defining member function outside the class

```
class rectangle{
   public:
      double length;
      double breadth;
};
double rectangle::area()
{ return length*breadth; }
```

1.10 For Exercise 1.8 define the function main and create three objects of class Student.

SOL.

```
#include <iostream>
using namespace std;
int main()
```

```
{
    student obj1,obj2,obj3;
    obj1.inputv();
    obj2.inputv();
    obj3.inputv();
    return 0;
}
```

1.11 Describe the public and private access specifier in a class with the help of suitable examples.

SOL. Public access specifier: All the members declared under the public specifier will be available to everyone. They can be accessed from anywhere in the program using direct member access operator(.) with the object of that class.

Example:

```
#include <iostream>
using namespace std;
class circle
{
   public:
      double radius;
      double area()
      {
         return 3.14*radius*radius;
      }

};
int main()
{
      circle obj;
      obj.radius = 5.5;
      cout<<"area "<<obj.area();
}
```

Output: area 94.985

Private access specifier: The class member declared as private can be accessed only by the member function of the class. They are not allowed to be accessed directly by any object or function outside the class.

For example:

```
#include <iostream>
using namespace std;
class circle{
      private:
//private data members
double radius;
public:
     double area()
          {
//member function can access private data members
               return 3.14*radius*radius;
          }

};
int main()
{
      circle obj;
      obj.radius = 1.5; //it will cause error
      cout<<"area "<<obj.area();
      return 0;
}
```

Output:
Error: 'double circle::radius' is private
double radius;

1.12 State whether the following statements are true or false?
i. Default access specifier of a class is public.
SOL. False, default specifier of a class is private.

ii. Public member function of a class can access all type of class data whether public or private.
SOL. True, public member function can access all types of class data.

iii. A private member function of a class cannot be directly accessed outside the Class.
SOL. True

1.13 Find the error(s) if any in the following code. Also correct the error(s) and give the output.

```
Class X
{
private:
int a,b;
void update(int a, int b);
public:
void read(){
cout<<"Enter the value of a:";
cin>>a;
cout<<"Enter the value of b:";

cin>>b;
}
void print(){
cout<<"a="<<a<<" "<<"b="<<b;
}
}
#include<iostream>
using namespace std;
int main()
{
```

X O;
O.a=10;
O.b=20;
O.print();
return 0;
}

SOL.

Error: 1) There should be a ';' after the definition of class
2) Object of class O cannot access the private data members directly.
It should call X::read()
Function for that purpose

```
Class X
{
private:
int a,b;
void update(int a, int b);
public:
void read(){

cout<<"Enter the value of a:";

cin>>a;

cout<<"Enter the value of b:";
cin>>b;
}
void print(){

cout<<"a="<<a<<" "<<"b="<<b;
}

}; //Error 1
#include<iostream>
using namespace std;
```

```
int main()
{
X O;
O.read(); //Error 2
O.print();
return 0;
}
```

Output:

```
Enter the value of a:10
Enter the value of b:20
a=10””b=20
```

1.14 Explain with the help of a suitable example, How objects are passed by-value and by-reference to a function?

SOL. **Pass by Value:** This creates a shallow local copy of the object in the function scope. Things you modify here won't be reflected in the object passed to it.

Let's assume 'X' is a class with some data members and member functions

Declaration

```
void funct(X x );
```

Calling

```
X x;
funct(x);
```

Pass by Reference: This passes a reference to the object to the function. Things you modify here will be reflected in the object passed to it. No copy of the object is created.

For example,

Declaration

```
void funct(X &x);
```

Calling

```
X x;
funct(x);
```

1.15 Is it possible to return objects from functions? Explain how?

SOL. Yes, it is possible to return objects from member function of class just like we would have done for any other data type.

For example:

```
//class definition
class X{
    public:
     int value;
     X add(X a, X b)
     {
          X c;
          c.value = a.value + b.value;
         //returning the object
         return c;
     }

};

//main function

int main()
{
X a,b,c;
a.value = 8;
b.value = 14;
```

```
c.value = 0;

cout<<"Initial value : \n";
cout<<a.value<<" "<<b.value<<" "<<c.value;
cout<<endl;

c = c.add(a,b);

cout<<"Final values : \n";
cout<<a.value<<" "<<b.value<<" "<<c.value;
cout<<endl;

return 0;
}
```

Output:
Initial values :
8 14 0

Final values :
8 14 22

1.16 Is it possible to create an array containing class objects? Explain with the help of suitable example.

SOL. Like array of any of other data types, an array of type class can also be created. Array of type class contains the objects of the class as its individual elements thus, it is also called array of object.

For example

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

class myclass{
int x;
public:
```

```
void setx(int i){x=i;}
int getx(){return x;}

};

void main()
{

myclass objs[4];
int i;

for(i=0; i<4;i++)
objs[i].setx(i);
for(i=0;i<4;i++)
cout<<"obj "<<i<<" : "<<obj[i].get(i)<<endl;
return 0;
}
```

Output:

```
obj 0 : 0
obj 1 : 1
obj 2 : 2
obj 3 : 3
```

1.17 State whether the following statements are true or false.

i. Objects can only be passed by reference.

SOL. False

ii. Objects can also be returned from functions.

SOL. True

iii. An array of objects cannot be created.

SOL. False

iv. Every object has a separate copy for its data as well as functions.

SOL. True

1.18 What is the purpose of using static data in a class? Explain with the help of suitable example.

SOL. When we declare a member of the class as static, it means no matter how many objects of the class are created, there is only one copy of the static member. All static data is initialized to zero when the first object is created, if no other initialization is present. Static data member are very uslful.

For example:

```
#include<iostream>
using namespace std;
class student{
private:
int roll_no;
public:
static int stcount;
//constructor definition
student(int r)
{
cout<<"contructor called\n";
roll_no=r;
stcount++;
}

};
int student::stcount = 0;
int main()
{
student st1(1);
student st2(2);
```

```
//total students

cout<<"total students : "<<student::stcount<<endl;
return 0;
}
```

OutPut:

```
constructor called
constructor called
total students : 2
```

1.19 Why static function cannot use non-static data? Justify your answer.

SOL. No, Static function of a class in C++ cannot access non-static variables, but, it can access static variable . However, non-static member function can access static and non-static variable both.

Static function is not associated with class object, means without object using class name only it can be called. whereas non-static variables are associated with objects. Every object has its own copy of non-static variable. Since, static function does not know about object, so, it is impossible for a static function to know on which class object or class instance it is being called.

1.20 Is it possible for static function to access non-static data, if it is invoked with a class object? Explain with the help of suitable example.

SOL. No, static function cannot access non-static data, even if it is invoked with a class object.

1.21 State whether the following statements are True or False.

i. Non-static function cannot access static data.

SOL. False

ii. Static data can be accessed by every object of the class.
SOL. True

iii. Static data and static function both can be accessed without making class objects.
SOL. True

1.22 Find the error(s) if any in the following code. Also correct the error(s) and give the output.

```
i) #include<iostream>
using namespace std;
class X
{
private:
int a;
static int b;
public:
void set(int i){a=I;}
static void display()

{
cout<<" The value of a:"<<a;
cout<<"The value of b:"<<b;
}

}
X:: b=0;
int main()

{
```

X O;
O.set(5);
O.display();
return 0;
}

SOL. **error 1:** Didn't put ';' after the definition of class
error 2: trying to access non static data member with static function

```
#include <iostream>
using namespace std;
class X
{
private:
int a;
static int b;
public:
void set(int i)
{
a=i;
}
void display() //error 2 correction
{
cout<<"The value of a:<<a";
cout<<"The value of b:"<<b;
}
}; //error 1 correction X::b=0;
int main()
{

X O;
O.set(5);
O.display();
return 0;
}
```

Output:

```
The value of a:5
The value of b:0
```

ii)

```
#include<iostream>
using namespace std;
class X
{
static const int i = 5;
};
const int X::i;
int main()
{
X O;
cout<<"i="<<X::i;
getchar();
return 0;
}
```

SOL. **error:** static member is declared in private scope so it cannot be accessed directly in main function

```
#include <iostream>
using namespace std;
class X
{
public: //error correction
static const int i =5;
};
const int X::I;
int main()
{
X O;
cout<<"i="<<X::i;
getchar();
return 0;
}
```

1.23 Define Constructor and Destructor.

SOL. Constructor - A constructor is a special function that has same name as that of class and is called automatically when an object of a class is created. It is used to initialize value to the data members of the class.

Destructor - A destructor is a member function that is invoked automatically when the object goes out of scope or is explicitly called. Its basic function is to de-allocate memory of the data members of the class.

1.24 What is default constructor? Explain.

SOL. Default Constructor: A default constructor is a constructor is a constructor that either has no parameters or if it has parameters, all the parameters have default values. If no user-defined constructor exists for a class A and one is needed, the compiler declares a default parameter-less constructor A::A().

1.25 Can you declare a constructor in private section? Justify your answer.

SOL. Yes, a constructor can be private. And you can call it with member functions (static or non) or friend functions.

For example:

```
class A{
    public:
        static: A create(int a)
                { return A(a); }

    private:
         int age;
        A (int a) :age(a) {}

        friend A createA(int a);

};
A createA(int a) { return A(a); }
```

here constructor is called indirectly with the friend function
creatA();

1.26 Can you overload constructors? Explain with the help of suitable example.

SOL. Constructor can be overloaded in a similar way as function overloading. Overloading construction have the same name as the class but the different number of arguments. Depending upon the number and the type of arguments passed, the corresponding constructors are called.

For example:

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

class person
{
private:
int age;
public:
person() { age = 20; } //without argument
person(int a) { age = a; } //with argument
int getage() { return age; }
};

int main()
{
person p1,p2(40);
cout<<"Age of person 1: "<<p1.getage()<<endl;
cout<<"Age of person 2: "<<p2.getage()<<endl;
return 0;
}
```

Output:

```
Age of person 1: 20
Age of person 2: 40
```

1.27 Why do we need destructors?

SOL. When an object of a class is created the constructor allocates memory for all the data members of the class, in order to deallocate the allocated memory we need destructor Functions.

1.28 Why destructors cannot be overloaded? Explain.

SOL. Overloading is possible when we have methods with same name but different signature. Since destructor is invoked automatically, and it does not have any parameter, there is only one destructor per class and thus destructor cannot be overloaded.

1.29 State whether the following statements are true or false.

i) It is necessary to define a constructor in every class.

SOL. False

ii) Destructor can be declared in private section.

SOL. True

iii) Destructors can also be overloaded.

SOL. False

iv) Destructors can also be declared as virtual.

SOL. True

v) Constructor can be a static member of the class.

SOL. False

vi) Constructors and destructors can be defined Inline.

SOL. True