OOPs is the principle of design and development of programs using modular approach

OOP combines both the data and the functions that data into a single unit called object. It follows bottom up design technique and is piecing together the smaller systems to give rise to more complex systems.

Major component of the OOP s is the class.

Encapsulation: The process of binding data members (variables , properties) and member functions into a single unit. A class is the best example.

Abstraction: Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

Inheritance: a mechanism of reusing and extending existing classes without modifying them, thus producing hierarchical relationships between them.

Polymorphism: It means ability to take more than one form.

Everything is c++ is associated with classes and objects

Attributes and methods are basically variables and functions to the class. These are often referred to as “class members”

```
Class MyClass{
        Public:
        Int myNum;
        String myString;
};
```

```
Class MyClass{
Public:
        Void myMethod(){
                Cout<<”Hello World!”;
        }
};
```

Pb3

```
Class MyClass{
Public:
        Void myMethod();
};
Void MyClass::myMethod(){
        Cout<<"Hello World!";
}
```

Pb4

```
Class Student{
Public:
String name;
Int rollNo;
Static int age;

Void display(){
//method body
}
};
```

Access specifier

Access specifiers define how the member (attributed and methods) of a class can be accessed:

Public – members are accessible from outside the class

Private – members cannot be accessed from outside the class

Protected – members cannot be accessed from outside the class, however, they can be accessed in inherited classes.

By default, all members of a class are private.

Pb5

```
Class MyClass{
        Public:
        Int x;
        Private:
        Int y;
};

Int main(){
MyClass myObj;
myObj.x = 5;
myObj.y = 5;
return 0;
}
```

Pb6

```
Class Box{
Public:
Static int objectCount;
};
Int Box::objectCount=0;
```

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

```
Class Box{
        Public:
                Static int objectCount;
                Static int getCount(){
                        Return length;
};
```

```
Box.getCount();
```

Pb7

```
Class Example{
Public:
        Int a;
        Void add(Example E)
        {
                a=a+E.a;
        }
};
```

Static Function Members

Pb8

```
Class Box{
Public:
Static int objectCount;
Static int getCount(){
        Return objectCount;
}
Private:
        Double length;
};
```

```
Box::getCount();
```

Friend Function and classes

A friend class can access private and protected members of other class in which it is declared as friend

```
Class A{
Private:
Int a;

Public:
A(){
a=0;
}
Friend class B;
};

Class B{
Private:
Int b;
Public:
Void showA(A& x){
Cout<<x.a;
}
};

Pb9
Class Data{
Public:
Int a;
Void print(){
Cout<<”a is”<<a;
}
};
```

```
Data d, *dp;
Dp=&d;
Int Data::*ptr=&Data::a;
d.*ptr=10;
d.print();

dp->*ptr=20;
dp->print();

pb10
class Data{
public:
int f(float){
return 1;
}
};

Int(Data::*fp1)(float)=&Data::f;
```