2068

Section A $(\mathbf{2 \times 10 = 20})$

1. What are the main features of the Object Oriented Programming? Explain with suitable practical examples.

Different features of object oriented programming are explained here below:

<u>Object:</u>

Objects are the entities in an object oriented system through which we perceive the world around us. We naturally see our environment as being composed of things which have recognizable identities & behavior. The entities are then represented as objects in the program. They may represent a person, a place, a bank account, or any item that the program must handle. For example Automobiles are objects as they have size, weight, color etc. as attributes (that is data) and starting, pressing the brake, turning the wheel, pressing accelerator pedal etc. as operation (that is functions).

<u>Class:</u>

Object consists of data and function tied together in a single unit. Functions are used to manipulate on the data. The entire construct of objects can be represented by a user defined data type in programming. The class is the user defined data type used to declare the objects. Actually objects are the variable of the user defined data type implemented as class. Once a class is defined, we can create any number of objects of its type. Each object that is created from the user defined type implemented as class is associated with the data type of that class. For example, manager, peon, secretary clerk are the objects of the class employee. Similarly, car, bus, jeep, truck are the objects of the class vehicle.

<u>Abstraction:</u>

Abstraction is representing essential features of an object without including the background details or explanation. It focuses the outside view of an object, separating its essential behavior from its implementation. We can manage complexity through abstraction. Let’s take an example of vehicle. It is constructed from thousands of parts. The abstraction allows the driver of the vehicle to drive without having detail knowledge of the complexity of the parts. The driver can drive the whole vehicle treating like a single object.

<u>Encapsulation:</u>

The mechanism of wrapping up of data and function into a single unit is called encapsulation. Because of encapsulation data and its manipulating function can be kept together. We can assume encapsulation as a protective wrapper that prevents the data being accessed by other code defined outside the wrapper. By making use of encapsulation we can easily achieve abstraction.

Inheritance:

Inheritance is the process by which objects of one class acquire the characteristics of object of another class. We can use additional features to an existing class without modifying it. This is possible by deriving a new class (derived class) from the existing one (base class).This process of deriving a new class from the existing base class is called inheritance. For examples, cars, trucks, buses, and motorcycles inherit all characteristics of vehicles.

Polymorphism:

Polymorphism means 'having many forms'. The polymorphism allows different objects to respond to the same operation in different ways, the response being specific to the type of object. The different ways of using same function or operator depending on what they are operating on is called polymorphism. Example of polymorphism in OOP is operator overloading, function overloading. Still another type of polymorphism exist which is achieved at run time also called dynamic binding.

2. Explain the role of constructor and destructor in Object Oriented Programming. Discuss user defined parameterized constructor with suitable example.

A constructor is a special member function that is executed automatically whenever an object is created. It is used for automatic initialization. Automatic initialization is the process of initializing object's data members when it is first created, without making a separate call to a member function. The name of the constructor is same as the class name. Types of constructors are: Default Constructor, Parameterized Constructor, and Copy Constructor. A constructor that does not take any parameter is called default constructor. A constructor that takes arguments is called a parameterized constructor. A copy constructor is called when an object is created by copying an existing object.

A destructor is a special member function that is executed automatically just before lifetime of an object is finished. A destructor has the same name as the constructor (which is the same as the class name) but is preceded by a tilde (~). Like constructors, destructors do not have a return value. They also take no arguments. Hence, we can use only one destructor in a class. The most common use of destructors is to deallocate memory that was allocated for the object by the constructor.

User defined parameterized constructor:

We can use parameterized constructors in two ways: by calling the constructor explicitly and by calling the constructor implicitly. Calling the constructor explicitly can also be termed as user defined parameterized constructor. The declaration

Rectangle R1 = Rectangle(5, 6, 7);

illustrates the user defined parameterized constructor.

3. Define a shape class (with necessary constructors and member functions) in Object Oriented Programming (abstract necessary attributes and their types). Write a complete code in C++ programming language.

- **Derive triangle and rectangle classes from shape class adding necessary attributes.**
- **Use these classes in main function and display the area of triangle and rectangle.**

```
#include<iostream.h>

#include<conio.h>

class shape

{       protected:

        float breadth, height, area;

        public:

        void getshapedata()

        {       cout<<"Enter breadth:"<<endl;

                cin>>breadth;

                cout<<"Enter height:"<<endl;

                cin>>height;

        }

};

class triangle: public shape

{       public:

        void calarea()

        {       area=(breadth * height)/2;

        }

        void display()

        {       cout<<"The area of triangle is"<<area<<endl;

        }

};

class rectangle: public shape

{       public:
        void calarea()
        {       area=breadth * height;
        }
        void display()
        {       cout<<"Area of rectangle is"<<area<<endl;
        }
};
int main()
{       triangle T;
        rectangle R;
        cout<<"Enter triangle data:"<<endl;
        T.getshapedata();
        cout<<"Enter rectangle data:"<<endl;
        R.getshapedata();
        T.calarea();
        R.calarea();
        T.display();
        R.display();
        return 1;
}
```

Section B $(\boldsymbol{8 \times 5 = 40})$

4. Why dynamic object is needed? Explain with suitable example.

The class declaration does not define any objects but only specifies what they will contain. Once a class has been declared, we can create variables (objects) of that type by using the class name (like any other built-in type variables). For example:
rectangle r;
creates a variable (object) r of type rectangle. We can create any number of objects from the same class. For example:
rectangle r1, r2, r3;
Objects can also be created when a class is defined by placing their names immediately after the closing brace. For example:

```
class rectangle {
………
………
………
}r1, r2, r3;
```

Dynamic object is needed due to following reasons:

- The Dynamic Object class enables us to define which operations can be performed on dynamic objects and how to perform those operations. For example, you can define what happens when you try to get or set an object property, call a method, or perform standard mathematical operations such as addition and multiplication.
- This class can be useful if you want to create a more convenient protocol for a library. For example, if users of your library have to use syntax like Scriptobj.SetProperty("Count", 1), you can provide the ability to use much simpler syntax, like scriptobj.Count = 1.

5. What is function overloading? Explain with suitable example.

Function overloading means having functions with the same name but with different signature. Signature includes method name and Parameters. These functions can be part of base class or derived class. For example:

```
#include <iostream>

int mul (int a, int b)

{         return (a*b);

}

float mul (float a, float b)

{         return (a*b);
}
int main ()
{       int x=5,y=2;
        float n=5.0,m=2.0;
        cout << mul (x,y);
        cout << "\n";
        cout << mul(n,m);
        cout << "\n";
        return 0;
}
```

6. Write a C++ program containing a possible exception. Use a try block to throw it and a catch block to handle it properly.

```
#include<iostream.h>
#include<conio.h>
void main()
{       clrscr();
        int a, b;
        cout<<"Enter values of a & b:\n";
        cin>>a>>b;
        try
        {       if(b == 0)
                throw b;
                else
                cout<<"Result = "<<(float)a/b;
        }
        catch(int i)
        {       cout<<"Divide by zero exception: b = "<<i;
        }
```

```
	cout<<"\nEND";
	getch();
}
```

7. Differentiate between base class and derived class with suitable examples.

Inheritance is the process of deriving new class from the existing class. Newly created class is called derived class (also called as child class or sub class) and existing old class is called as base class (also called as parent class or super class). If we derive a class, all the features of parent class are inherited into child class and also we can add extra new features to the child class. This inheritance provides us with the mechanism of adding new features to the class without directly modifying it. For example:

```
class person
{	char name[20];
	int age;
	Public:
	void getperson()
	{	cout<<"Enter name and age"<<endl;
		cin>>name>>age;
	}
	void displayperson()
	{	cout<<"Name:"<<name<<endl;
		cout<<"Age:"<<age<<endl;
	}
};
class employee: Public person		// employee is the derived class of person which is a base
class
{	Public:
	int eid, sal;
	void getemployee()
	{	cout<<"Enter ID and salary:"<<endl;
```

```
                cin>>eid>>sal;
        }
        void displayemployee()
        {       cout<<"EID:"<<eid<<endl;
                cout<<"Salary:"<<sal<<endl;
        }
};
void main()
{       clrscr();
        employee e;
        e.getperson();
        e.getemployee();
        cout<<"Employee details:"<<endl;
        e.displayperson();
        e.displayemployee();
        getch();
}
```

8. Differentiate between private, public and protected variables with suitable example.

When the base class is publicly inherited, all the public members of the base class become public members of the derived class. So, they can be accessed through the objects of the derived class. All the protected members become protected members of the derived class. When the base class is privately inherited, all the public and protected members of the base class become private members of the derived class. So, these cannot be accessed outside the class directly through the derived class object, but might be accessed through public functions in the derived class. Like general private members, these members can be used freely within the derived class. When a member is defined with protected access specifier, these members can be accessed from that class and also from the derived class of this base class. But cannot be accessed from any other function or class. i.e. protected members act as public for derived class and private for other classes.

Access specifier	Accessible from own class	Accessible from derived class	Accessible from objects outside the class
Public	Yes	Yes	Yes
Protected	Yes	Yes	No
Private	Yes	No	No

9. Differentiate between class from inheritance. Explain with suitable example.

Object-oriented programming (OOP) encapsulates data (attributes) and functions (behavior) into a single unit called classes. The data components of the class are called data members and the function components are called member functions. The data and functions of a class are intimately tied together. Class is a blueprint of real world objects. A programmer can create any number of objects of the same class. Classes have the property of information hiding. It allows data and functions to be hidden, if necessary, from external use.

Inheritance (or derivation) is the process of creating new classes, called derived classes, from existing classes, called base classes. The derived class inherits all the properties from the base class and can add its own properties as well. The inherited properties may be hidden (if private in the base class) or visible (if public or protected in the base class) in the derived class.

For example:

```
class A                         // class

{        members of A

};

class B : public A              // inherited class

{        members of B

};
```

10. Explain the role of polymorphism in Object Oriented Programming.

Polymorphism means state of having many forms. We know that polymorphism is implemented using the concept of overloaded functions and operators. In this case, polymorphism is called early binding or static binding or static linking. This is also called compile time polymorphism because the compiler knows the information needed to call the function at the compile time and, therefore; compiler is able to select the appropriate function for a particular call at the compile time itself. There is also another kind of polymorphism called run time polymorphism. In this type, the selection of appropriate function is done dynamically at run time. So, this is also called late binding or dynamic binding. C++ supports a mechanism known as virtual functions to achieve run time polymorphism. Run time polymorphism also requires the use of pointers to objects.

11. Explain about "this" pointer with suitable example.

The keyword "this" identifies a special type of pointer. Suppose that you create an object named x of class A, and class A has a non-static member function f(). If you call the function x.f(), the keyword "this" in the body of f() stores the address of x. You cannot declare the "this" pointer or make assignments to it. A static member function does not have a "this" pointer.

Example of "this" pointer:

```
#include <iostream>

using namespace std;

struct X

{       private:

        int a;

        public:

        void Set_a(int a)

        {       // The 'this' pointer is used to retrieve 'xobj.a'

                // hidden by the automatic variable 'a'

                this->a = a;

        }

        void Print_a()

        {       cout << "a = " << a << endl;

        }

};

int main()

{       X xobj;

        int a = 5;

        xobj.Set_a(a);
        xobj.Print_a();
}
```

12. Write a program to find the square root of given integer using inline function.

```
#include<iostream.h>
#include<conio.h>
inline void squareroot(int a)
{       int sq;
        sqroot=sqrt(a);
        cout<<"Square root="<<sqroot<<endl;
}
void main()
{       clrscr();
        int x;
        cout<<"Enter a number:"<<endl;
        cin>>x;
        squareroot(x);
        getch();
}
```

13. Write a program to convert inch into centimeter.

```
#include<iostream.h>
#include<conio.h>
void main()
{       float inch, cm;
        cout<<"Enter the inch:"<<endl;
        cin>>inch;
        cm=2.54*inch;
```

```
	cout<<"Equivalent centimeter is:"<<cm<<endl;
	getch();
}
```