EXERCISE

Q1. Select the best answer for the following MCQs.

i. A constructor is called whenever _____

a. An object is destroyed
b. An object is created
c. A class is declared
d. A class is used

ii. Destructor is used for _____

a. Initializing the values of data members in an object
b. Initializing arrays
c. Freeing memory allocated to the object of the class when it was created
d. Creating an object

iii. The name of destructor is always preceded by the symbol _____

a. +
b. %
c. .
d. ~

iv. Constructors are usually used for _____

a. Constructing programs
b. Running classes
c. Initializing objects
d. All of the above

v. **Inheritance is used to _____**

a. Increase the size of a program

b. Make the program simpler

c. Provide the facility of code reusability

d. Provide the facility of data hiding

Answers

i.	An object is creates	ii.	Freeing memory allocated to the object of the class when it was created
iii.	~	iv.	Initializing object
v.	Provide the facility of code reusability		

EXTENSIVE QUESTIONS

Q2. Write long answers of the following questions

1. Explain the terms object and class with examples.

Answer

1. Object

Objects are instances of class which holds the data variables declared in class and the member functions work on these class objects. In other words, a variable of type class is called object

Instance of Class

In C++ when we declare a variable of type class we call it instantiating, the class and the variable itself is called an instance or object of that class.

Syntax

The general syntax for creating an object of a class is given below

Class_name Object_name ;

Example

For the class CRectangle the object can be created as follows

CRectangle R1: // R1 is the object of class CRectangle

CRectangle R1, R2, R3:

Here R1, R2, R3 are the objects of the same class CRectangle that share the same date member and member functions. The definition of a class does not occupy any memory. It only defines what the class looks like. In order to use a class, a variable of that class type must be declared. When an object is created then memory is set aside for all the data members and member functions of that class.

Program

Consider the following program to implement the class CRectangle

```
// Object creation in Class

#include <iostream.h>

#include <conio.h>

Class CRectangle

{

Private:

Int X, Y:
```

```
Public:
Void set_values (int a, int b)
{
X = a;
Y = b;
}
Int area ()
{
Cout <<"area of the rectangle=" <<(X*Y);
}
};      // End of class
Int main ()
{
CRetangle R1;
R1 set_values (44, 22);          // call to set_values function
R1 area ();                      // call to area function
Getch();
Return 0;
}
```

In this program the member function set-values and area are accessed by

R1.set_values (44, 22) and R.area()

Output of the Program

Area of the rectangle = 968

2. **Class**

Class is a user defined data type, which holds its own data members and member functions. These can be accessed and used by creating objects

Structure of Class

A class consists of both attributes of real world objects and functions. Functions perform operations on these attributes. The attributes are called data members and the operations are called member functions.

Formal of Class

The general format is

```
Class class_name
{
Access_specifier_1          // body of the class
Member 1:
Access specifier_2:
Member 2:
}
```

Here class_name is a valid identifier for the class which is followed by the body of the class that is enclosed in pair of braces

Body of class

The body of the class consists of

i. Data members
ii. Member functions

Program

Consider the following example to understand the concept of class

```
// class declaration example
Class CRectangle
{
Private:
Int x, y:                    // declaration of data members
Public:
Void set_values (int a, int b):     // declaration of member functions
Int area ():                 // declaration of member functions
Int area ():
};
```

Explanation

The above statements declare a class named CRectangle. The class consists four members

i. Two members of type int (members x and member y) with private access
ii. Two member functions with public access set_values () and area ()

In this example only declarations for the member functions has given and not their definition

iii. How objects are created to access members of a class?

Answer

Class has two members

i. Data members
ii. Member function

i. **Data Member**

The attributes of a class are called data members. Mostly data members are declared under the private access specifier to make them private to the class in which they are used

Data members can also be used under public access specifier

Explanation

The following lines of code are taken from the program discussed above to explain data members.

For each object of a class a separate copy of the data members is created in memory

```
Class CRectangle
{
Private:
Int X;
Int y;
Public:
};
```

3. **Member Function**

   The functions declared or defined inside the body of the class are called member functions. These member functions are usually written under the public access specifier to operate on the data members of the class. Member functions can also be written in the private area of the class but the practice is to write them in the public area

Program

```
// example of a class with its answers

#include <iostream>
```

```
#include <conio.h>
Class data
{
Public:
Int Month;
Int Day;
Int Year;
Void SetDate (int nDay, int nMonth, int nYear)
{
Day= nDay;
Month = nMonth;
Year = nYear;
}
Void showDate ()
{
Cout <<"Day of birth:"<<Day<<end1;
Cout<<"Month of birth:"<<Month<<End1;
Cout<<"Year of birth:">>Year;
};
Int main ()
{                                   // Object creation
D1. setDate (21. 07. 2010);         // call to Set Date function
D1. ShowDate ();
```

```
Getch ():

Return 0:

}
```

Output of the Program

```
Day of birth : 21

Month of birth : 07

Year of birth : 2010
```

ii. **What are access specifiers? Explain private and public specifier with examples.**

Answer

Access Specifier

Data members and member functions are used within the access specifiers

Private Access Specifier

Private access specifier tells the compiler that the members defined in a class by preceding this specific are access only within the class and its friend function private members are not accessible outside the class

Program 1

Consider the following sample program to demonstrate private access specifier

```
// private access specifier example 1

#include <iostream.h>

#include <conio.h>

Class TestPrivate

{
```

```
Private:
Int x;
Void Test ()
{
X=10;
Cout<<"X is private member and accessed inside the class"<<end1;
Cout<<"value of X<<v<<end1;
};
Int main()
{
Test private P1;        // P1 is an object
P1. Test ()             // valid
// P1.X = 10            // invalid
Getch ();
Return 0;
}
```

Public Access Specifier

Public members are accessible from anywhere where the object is visible i.e. within the class in the derived classes and in the main function

Program

Consider the following program to demonstrate the visibility of public members of a class

```
// public access specifier example 1
```

```
#include <iostream>
#include <conio.h>
Class PublicTest
{
Public:
Int X= 10;                    // Public data member
Void showPublic ()                  // public member function
{
Cout<<"X="<<X;
}
}
Int main ()
{
Public Test PT1;              // PT1 is an object to class Public Test
PT1. X;                       // access private data members
PT1. showPublic ();           // access public member function
Getch ();
Return 0;
}
```

Explanation

In the above program the class Public test is defined that comprise of one data members X and one member function showPublic () in its public part. The data member 'X' is accessed within the member function showPublic and outside the class i.e. in main

() function without generating any error message. Similarly, the member function ShowPublic is also accessed in main ()

Program

Here is an example of a class that uses all access specifiers

```
// private and public access specifier program

#include <iostream.h>

#include <conio.h>

Class Access

{

Int A:                  // private by default

Void getA()             // private by default

{

Cout <<"I am private member accessible only thorough public member function"

<<End1;

}

Public:

Int B;                  // public

Void GetB()

{

Get A'

Cout << "I am B in public"<<end1;

}

};
```

```
Int main ()
{
Access cAccess:          // cAccess is an object of the class Access
// cAccess. A = 2;       // WRONG because A is private data member
// cAccess. Get A ();    // Wrong because Get A () is private member function
cAccess. B= 10;          // okay because B is public data members
cAccess. Get B ();       // okay because Get b () is public member function
getch ();
return 0;
}
```

Output of the Program

```
I am private member accessible only through public
Member function
I am B in public
```

iii. What is data hiding? Explain

Answer

Data hiding

Data hiding is one of the C++ features in which the members of a class we are protected against illegal access for outside the class. In C++ we have the facility of hiding data using different levels in classes

i. Private
ii. Protected
iii. Public

It is also called Explanation

Explanation

Private data members and member functions can only be accessed by the members of the same class defining them. They cannot be accessed from outside the class. Similarly, only the class defining them and its derived classes can access protected members of a class. By using friendly function the private and protected members (data members and member function) of a class can also be accessed.

Living of Hiding Members of a Class

The following shows the level of hiding members of a class

Access	Public	Protected	Private
Access of members (data functions) in the same class	Yes	Yes	Yes
Access of members (data functions) in the derived class	Yes	Yes	No
Access of members (data functions) from outside the class i.e. from main ()	Yes	No	no

iv. Explain constructor and destructor class members functions with examples

Answer

Constructor

A constructor is a special type of member function that initializes an object automatically when it is created compiler identifies a given member function is a constructor by its name and the return type.

Rules for Naming Constructors

Unlike functions, constructors have specific rule/features for how they must be named

i. Constructors have the same name as that of class

ii. Constructors have no return type

iii. Constructor is always public

Program

Consider the following simple program to explain the concept of constructor in classes

```
// Constructor Example

#include <iostream.h>

#include <conio.h>

Class ConstTest

{

Public:

constTest () // Constructor

{

Cout <<"I am Constructor";

Int main ()

{                    // CT is an object to the class ConstTest

Getch ();

Return 0;

}
```

Output of the Program

I am constructor

Explanation

In the above examples, a class **ConstTest** is defined which having the constructor. **ConstTest ()** in its public part. In **main()** program, there is no explicit call to this constructor and it is automatically called when the object CT of this class is created.

Destructors

Destructors are special member functions that are executed when an object is destroyed. Destructor fulfils the opposite functionality of constructor and thus de-allocates the memory already allocated to an object during its creation. They are called automatically when objects are destroyed. They are denoted by – destructor

Features of Destructor

Destructors have the following specific features

i. Destructor has the same name as that of the class preceded by a tilde symbol
ii. Destructor cannot take arguments
iii. Destructor has no return type

Note: the second feature given above, implies that only one destructor may exist per class, as there is no way to overload destructors since they cannot be differentiated from each other based on arguments

Program- explain the concept of destructor

Consider the following program to explain the concept of destructor

```
// constructors and destructors example

#include <iostream.h>

#include <conio.h>

Class A

{

Public

A ()              // constructor function

Cout <<"I am constructor"<<end1;

}
```

```
A ()            // destructor function
{
Cout <<"I am destructor":
}
};
Int main ()
{
A a1,
Getch ():
Return 0:
}
```

Output of the Program

```
I am constructor
I am destructor
```

Explanation

In this example, A () is a destructor. When the program runs and the object a1 is created. Constructor A() is called displaying the message "I am constructor" but when the control goes out of the program and the object is destroyed by A() a message. "I am destructor" Is displayed to verify that the destructor is executed.

v. **Explain inheritance and polymorphism with examples**

Answer

Inheritance

A key feature of C++ classes in which new classes are created from existing classes is called inheritance. Inheritance uses the concept of parent and child class.

Parent Class/ Base Class

A parent class is the class from which others classes are derived. It is also called base class.

Child class/ Sub Class

A sub class is a class which inherits features from the base class. A sub class is also called child class or derived class.

Examples of Inheritance

A few examples of inheritance from daily life are given below

i. Consider Polygon as base class which has a series of child classes that describe polygons i.e. Square and Pentagon. The have certain common properties such as both can be described by means of only two sides and angles.
ii. We can also represent patient class into Indoor and Outdoor patients in which both have some common features like name, father, name age, address and disease but indoor patient have Ward No and Bed No as its unique features and the outdoor patient have Next data of visit as its unique features. It can be represented as follows

Syntax of using Inheritance in Classes

```
// syntax of using inheritance in classes
Class A                    // base class
{
}
Class B: public A      // class B is derived from class A
```

{

}

Program

A complete C++ program for the above sketch is given hereunder

```
// inheritance of classes
#include <iostream.h>
#include <conio.h.>
Class A      // base class
{
Public:
Void showa ()
{
Cout <<"I am in base class A"<<end1;
}
}
Class B. public A    // class B is derived from class A
{
Public
Void show b()
{
Public
Void show b ()
{
```

```
Cout <<"I am in derived class B" << end1:
}
}
Int main ()
{
B b1:  // object of the derived class
B1. Show a ():       // object of B inheriting function of base class b1
Getch ():
Return 0:
}
```

Output of the Program

```
I am in base class A
I am in derived class B
```

Explanation

In this example, we have two classes A and B. class B is publically derived from class A and has therefore right to access the member function of class A. in main (), we have created only object b1 for class B and have called the member functions show a() of base class A and show b () of class B that have produced the output as shown above.

Polymorphism

Polymorphism is the ability to use an operator or function in multiple ways. Polymorphism gives different meanings or functionality to the operators or functions. Poly refers to many signifies that there are many uses of these operators and functions. It is the use of a single function or operator In many ways.

Method to perform Polymorphism

In C++ one of the following concepts can achieve polymorphism

i. Function name overloading
ii. Operator overloading
iii. Virtual functions

i. **Function overloading**

Function overloading is the concept of using same function name for related but slightly different purposes in a single program

Examples

Consider the following simple examples to understand the concept of polymorphism with respect to operator overloading

```
1          + 10;
```

The above statement refers to integer addition with the help of + operator. He same + operator can also be used for addition of two floating point numbers or two strings (concatenation) as shown hereunder

```
7.15 + 3.78
```

Polymorphism is a powerful feature of every object Oriented Programming (OOP) language especially of C++.

2. **Operator Overloading**

   A single operator + behaves differently in different contexts such as integer and float addition and strings concatenation. This concept is known as operator overloading.

3. **Virtual function**

   A virtual function or virtual method is a function or method whose behavior can be overridden within an inheriting class by a function with the same signature. This concept is a very important part of the polymorphism portion of object oriented programming (OOP)

Lab Activities

- Write a C++ program implementing a class with the name Circle having two functions with the names, GetRadius and CalArea. The functions should get the value for the radius from the user and then calculate the area of the circle and display the results.

Program

```
#include <iostream>

Using namespace std:

// circle class

Class circle

{

        Public:

        Circle ()

{

   }

        Public:

        Int radius;

        Double area;

        // get method to get radius

        Void GetRadius ()

{

        Cout <<"Enter radius: ";
```

```
        Cin>> radius;
    }
    // calculate area
    Void Cal Area ()
    {
        Area = 3.14 * radius * radius;
        Cout << "Area of Circle: << area << end1;
    }
}
// main class
Int main ()
{
    Circle c;
    c.Get radius ();
    c.CalArea ();
    }
```

- **Write a C++ program to implementing inheritance between Employee (base class) and Manager (derived class)**

Program

```
#include <iostream>
#include <bits/stds++. H>
Using namespace std;
```

```
// base class
Class Employee
{
        Public:
        Int id_e:
// Sub class inheriting from Base Class (Parent)
Class Manager : public Employee
{
        Public
        Int id_m:
};
// main function
Int main ()
{
        Manager obj1:
        / / an object of class child has all data members
        // and member functions of class parent
        Obj1. Id m = 5:
        Cout<<"Child id is "<< obj1. Id m << end1;
        Cout <<"Parent id is "<< obj1. Id_e << end1;
        Return 0;
}
```

- Write a C++ program implementing a class with the name, Time. This class should have a constructor to initialize the time, hours, minutes and seconds, initially to zero. The class should have another function name ToSecond to convert and display the time pass by object into seconds.

Program

```
#include <iostream>
Using namespace std;
Class time
{
        Public:
                Int h, m, s totalSec = 0;
        Public:
                Time ()
        {
                H=0;
                M=0;
                S=0;
                Cout << "Object is created: "<<End1;
}
        Time ()
{
        Cout <<"Object is being deleted" << end1;
}
Void toSecond ()
```

```
Cout<<"Enter time in hour, min or sec";

Cin >> h >> m >> s;

H= h*3600;

m= m * 60;

total sec= h + m +s;

        cout <<"The total seconds are "<<total Sec<<end1;

        }

}

Int main ()

{

        Time t;

        t. to second ();

}
```