**SEM/YEAR : III / II** **DEPT : ECE**

**SUB CODE : EC2202** **SUB NAME : DS & OOPS**

## UNIT I- PRINCIPLES OF OBJECT ORIENTED PROGRAMMING

## QUESTION BANK WITH ANSWER

## PART A

**1. What is the output of the following program, if it is correct? Otherwise indicate the mistake: [May 2006]**

```
int  l=10;
Void main []
{int l=20;
{int l=30;
cout<<l<<::l;
}}
```

The program is in correct due to syntax error. Within the main function, there is no need of another opening braces in the int l=20; and also closing braces.

**2. Difference between Class and structure? [April -2010, Dec-2012]**

- Class is the ADT where as structure is udt.
- Class needs access specifier such as private, public & private where as structure members can be accessed by public by default & don't need any accessfiers.
- Class is oops where structure is borrowed from traditional structured [pop] concept.

**3. What is abstract Class? [Nov-2009]**

An abstract class is a class that is designed to be specifically used as a base class. An abstract class contains at least one pure virtual function. You declare a pure virtual function by using a pure specifier [= 0] in the declaration of a virtual member function in the class declaration

**4. List out the advantages of new operator over malloc[] [Dec-2012]**

- It automatically computes the size of the data object.
- It automatically returns the correct pointer type
- It is possible to initialize the objects while creating_ the memory space.
- It can be overloaded.

**5. What are the basic concepts of OOS? [ April -2011]**

- Objects.
- Classes.
- Data abstraction and Encapsulation.
- Inheritance.
- Polymorphism.
- Dynamic binding.
- Message passing

**6. What is the difference between local variable and data member? [Nov-2011]**

- A data member belongs to an object of a class whereas local variable belongs to its current scope.
- A local variable is declared within the body of a function and can be used only from the point at which it is declared to the immediately following closing brace.
- A data member is declared in a class definition, but not in the body of any of the class member functions.
- Data members are accessible to all member function of the class.

**7. What is the function parameter? Difference between parameter and Argument. [Nov-2011]**

A **function parameter** is a variable declared in the prototype or declaration of a function:

```
void foo[int x]; // prototype -- x is a parameter
void foo[int x] // declaration -- x is a parameter
{
}
```

An **argument** is the value that is passed to the function in place of a parameter

**8. What is data hiding? [April -2011,Nov-2010]**

The insulation of data from direct access by the program is called as data hiding or information binding.

The data is not accessible to the outside world and only those functions, which are wrapped in the class, can access it.

**9. What are the advantages of Default Arguments? [Nov-2010]**

The function assigns a default value to the parameter which does not have a matching argument in the function call. They are useful in situations where some arguments always have the same value.

e.g., float amt [float P, float n, float r = 0.15];

**10. List out the basic concepts of Object Oriented Programming. [Nov-2009]**

- Objects
- Classes
- Data Abstraction and Encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding
- Message Passing

**11. What are abstract classes? [Nov 2009, Apr 2013]**

Classes containing at least one pure virtual function become abstract classes. Classes inheriting abstract classes must redefine the pure virtual functions; otherwise the derived classes also will become abstract. Abstract classes cannot be instantiated.

**12. Define abstraction and Encapsulation [Apr 2011]**

**Data Abstraction**

Abstraction refers to the act of representing the essential features without including the background details or explanations.

**Data Encapsulation**

The wrapping up of data and functions into a single unit is known as data encapsulation.

**13. What is the Need for Static Members [April 2011]**

Class members can be declared using the storage class specifier static in the class member list. Only one copy of the static member is shared by all objects of a class in a program. When you declare an object of a class having a static member, the static member is not part of the class object.

**14. Define Polymorphism. [Apr 2013]**

Polymorphism is another important oops concept. Polymorphism means the ability to take more than one form. For example, an operation may exhibit **different behavior in different instances**. Behavior depends upon the types of data used in the operation.

**15. What do you mean by pure virtual functions? [Dec2008]**

A pure virtual member function is a member function that the base class forces derived classes to provide. Any class containing any pure virtual function cannot be used to create object of its own type.

**16. Write a C++ program to check the given integer is prime or composite number. [Apr-2010]**

```
#include<conio.h>
#include<stdio.h>
int main[]
 {
  int num,d,ctr;
  clrscr[];
  printf["\n Enter a number="];
  scanf["%d",&num];
  d=1;
  ctr=0;
  while[d<=num]
   {
    if[num%d==0]
      ctr++;
```

```
 d=d+1;
}
if[ctr==2]
 printf["\n %d is a prime number",num];
else
 printf["\n %d is a composite number",num];
getch[];
return[0];
}
```

**17. What is function Prototype? [DEC 2011]**

A function prototype or function interface in C, Perl, PHP or C++ is a declaration of a function that omits the function body but does specify the function's return type, name and argument types. While a function definition specifies what a function does, a function prototype can be thought of as specifying its interface.

**18. List out four Storage Classes in C++ [Nov 2008]**

Storage classes are used to specify the lifetime and scope of variables. How storage is allocated for variables and how variable is treated by complier depends on these storage classes.

These are basically divided into 5 different types :

1. Global variables
2. Local variables
3. Register variables
4. Static variables
5. Extern variables

**19. What is an identifier?**

Identifiers are names for various programming elements in c++ program. such as variables, arrays, function, structures, union, labels ect., An identifier can be Composed only of uppercase, lower case letter, underscore and digits, but should start only with an alphabet or an underscore.

**20. What is a keyword?**

Keywords are word whose meanings have been already defined in the c compiler. They are also called as reserved words.

(ex) main(), if, else, else, if, scanf, printf, switch, for, goto, while ect.,

**21. List out the benefits of oops.**

• Can create new programs faster because we can reuse code
• Easier to create new data types
• Easier memory management
• Programs should be less bug-prone, as it uses a stricter syntax and type checking.
• `Data hiding', the usage of data by one program part while other program parts cannot access the data Will whiten your teeth

**22. List out the application of oops.**

• Client server computing
• Simulation such as flight simulations.
• Object-oriented database applications.
• Artificial intelligence and expert system
• Computer aided design and manufacturing systems.

**23. Define data hiding.**

The purpose of the exception handling mechanism is to provide a means to detect and report an "exceptional circumstance" so that appropriate action can be taken.

**24. What is the use of scope resolution operator?**

In C, the global version of the variable cannot be accessed from within the inner block. C++ resolves this problem by introducing a new operator :: called the scope resolution operator. It is used to uncover a hidden variable.

Syntax:

:: variable name

**25. When will you make a function inline?**

When the function definition is small, we can make that function an inline function and we can mainly go for inline function to eliminate the cost of calls to small functions.

**26. What is overloading?**

Overloading refers to the use of the same thing for different purposes.

There are 2 types of overloading:

• Function overloading
• Operator overloading

**27. What is the difference between normal function and a recursive function?**

A recursive function is a function, which call it whereas a normal function does not.

Recursive function can't be directly invoked by main function

**28. What are objects? How are they created?**

Objects are basic run-time entities in an object-oriented programming system. The class variables are known as objects. Objects are created by using the syntax:

classname obj1,obj2,…,objn;

(or) during definition of the class:

```
class classname
{
-------
}obj1,obj2,…,objn;
```

**29. List some of the special properties of constructor function.**

• They should be declared in the public section.
• They are invoked automatically when the objects are created.
• They do not have return types, not even void and cannot return values.
• Constructors cannot be virtual.

Like other C++ functions, they can have default arguments

**30. Describe the importance of destructor.**

A destructor destroys the objects that have been created by a constructor upon exit from the program or block to release memory space for future use. It is a member function whose name

is the same as the class name but is preceded by a tilde.
Syntax:
~classname(){ }

**31. What do you mean by friend functions?**

C++ allows some common functions to be made friendly with any number of classes, thereby allowing the function to have access to the private data of thse classes. Such a function need not be a member of any of these classes. Such common functions are called friend functions.

**32. What are member functions?**

Functions that are declared within the class definition are referred as member function.

**33. Define dynamic binding.**

Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run-time.

**34.Write any four properties of constructor.(DEC 2010)**

- Constructors should be declared in the public section.
- They are invoked automatically when the objects are created.
- They do not have return types
- They cannot be inherited

**35.List any four Operators that cannot be overloaded.(DEC 2010) (DEC 2009) (DEC 2011)**

- Class member access operator (. , .*)
- Scope resolution operator (::)
- Size operator ( sizeof )
- Conditional operator (?:)

**36.What is a Destructor? (DEC 2012)**

A destructor is used to destroy the objects that have been created by a constructor. It is a special member function whose name is same as the class and is preceded by a tilde '~' symbol. When an object goes out from object creation, automatically destructor will be executed.

**Example:**

```
class File {

public:

~File(); //destructor declaration

};

File::~File()
```

{

close(); // destructor definition

}

**37.What is the Need for initialization of object using Constructor? (DEC 2012)**

If we fails to create a constructor for a class, then the compiler will create a constructor by default in the name of class name without having any arguments at the time of compilation and provides the initial values to its data members. So we have to initialize the objects using constructor

**38.What is a Copy Constructor (DEC 2009)**

A copy constructor is used to declare and initialize an object from another object. It takes a reference to an object of the same class as an argument

Eg: integer i2(i1);

would define the object i2 at the same time initialize it to the values of i1. Another form of this statement is

Eg: integer i2=i1;

The process of initializing through a copy constructor is known as copy initialization.

**39.Give an example for a Copy Constructor (JUNE 2013)**

```
#include<iostream>

#include<conio.h>

using namespace std;

class Example       {

   // Variable Declaration

   int a,b;

   public:
```

```
    //Constructor with Argument
    Example(int x,int y)          {
    // Assign Values In Constructor
    a=x;
    b=y;
    cout<<"\nIm Constructor";
    }
    void Display()    {
    cout<<"\nValues :"<<a<<"\t"<<b;
    }
};
int main()              {
        Example Object(10,20);
        //Copy Constructor
        Example Object2=Object;
        // Constructor invoked.
        Object.Display();
        Object2.Display();
        // Wait For Output Screen
        getch();
        return 0;
```

}

**40.What is the Need for Destructors? (June 2013)**

- Destructor is used to destroy an object.
- By destroying the object memory occupied by the object is released.

**41.Explain the functions of Default Constructor(MAY 2011)**

- The main function of the constructor is, if the programmer fails to create a constructor for a class, then the compiler will create a constructor by default in the name of class name without having any arguments at the time of compilation and provides the initial values to its data members.
- Since it is created by the compiler by default, the no argument constructor is called as default constructor.

**42.What is the need for Overloading an operator(MAY 2011)**

- To define a new relation task to an operator, we must specify what it means in relation to the class to which the operator is applied.
- This is done with the help of a special function called operator function.
- It allows the developer to program using notation closer to the target domain and allow user types to look like types built into the language.

Or

- The ability to tell the compiler how to perform a certain operation when its corresponding operator is used on one or more variables.

**43.What is the function of get and put function (MAY 2010)**

- Cin.get(ch) reads a character from cin and stores what is read in ch.
- Cout.put(ch) reads a character and writes to cout

## PART B

**1. Explain the basic concepts of object oriented programming in detail with example. [Apr 2010, Nov 2010, April 2011]**

**Objects:**

Object is the basic unit of object-oriented programming. Objects are identified by its unique name. An object represents a particular instance of a class. There can be more than one instance of a class. Each instance of a class can hold its own relevant data.
An Object is a collection of data members and associated member functions also known as methods.

**Classes:**

Classes are data types based on which objects are created. Objects with similar properties and methods are grouped together to form a Class. Thus a Class represents a set of individual objects. Characteristics of an object are represented in a class as Properties. The actions that can be performed by objects become functions of the class and are referred to as Methods. For example consider we have a Class of Cars under which Santro Xing, Alto and WaganR represents individual Objects. In this context each Car Object will have its own, Model, Year of Manufacture, Color, Top Speed, Engine Power etc., which form Properties of the Car class and the associated actions i.e., object functions like Start, Move, and Stop form the Methods of Car Class.No memory is allocated when a class is created. Memory is allocated only when an object is created, i.e., when an instance of a class is created.

**Inheritance:**

Inheritance is the process of forming a new class from an existing class or base class. The base class is also known as parent class or super class. The new class that is formed is called derived class. Derived class is also known as a child class or sub class. Inheritance helps in reducing the overall code size of the program, which is an important concept in object-oriented programming.

**Data Abstraction:**

Data Abstraction increases the power of programming language by creating user defined data types. Data Abstraction also represents the needed information in the program without presenting the details.

**Data Encapsulation:**

Data Encapsulation combines data and functions into a single unit called Class. When using Data Encapsulation, data is not accessed directly; it is only accessible through the

functions present inside the class. Data Encapsulation enables the important concept of data hiding possible.

**Polymorphism:**

Polymorphism allows routines to use variables of different types at different times. An operator or function can be given different meanings or functions. Polymorphism refers to a single function or multi-functioning operator performing in different ways.

**Dynamic Binding** :

Binding refers to the linking of procedure call to the code to be executed in response to the call. Dynamic Binding or Late Binding means that the code associated with a given procedure call is known only at the run-time.

**Message Passing** :

Objects communicate between each other by sending and receiving information known as messages. A message to an object is a request for execution of a procedure. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

**2. What are the features of Object Oriented Programming? – [Apr 2010, Nov 2010, April 2011]**

- Emphasis is on data rather than procedure.
- Programs are divided into objects.
- Data structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are tied together.
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through functions.
- New data and functions can easily be added whenever necessary.
- Follows bottom-up approach.

**3. Explain the idea of Classes, data abstraction and Data Encapsulation.[Nov-2009]**

**Classes**

A Class is a collection of objects of similar type.

**Syntax**

Class class-name;

**Example**

Class mango;

Where, mango is the name of the class.

**Data Abstraction and Encapsulation**

- The wrapping up of the data and functions into a single unit is known as encapsulation. The insulation of data from direct access by the program is called data hiding.

- Abstraction refers to the act or representing essential features without including the background details. The attributes are sometimes called as data members and functions that operating on that data is called member function.
- A class using the data abstraction concept is known as AbstractData Type [ADT].

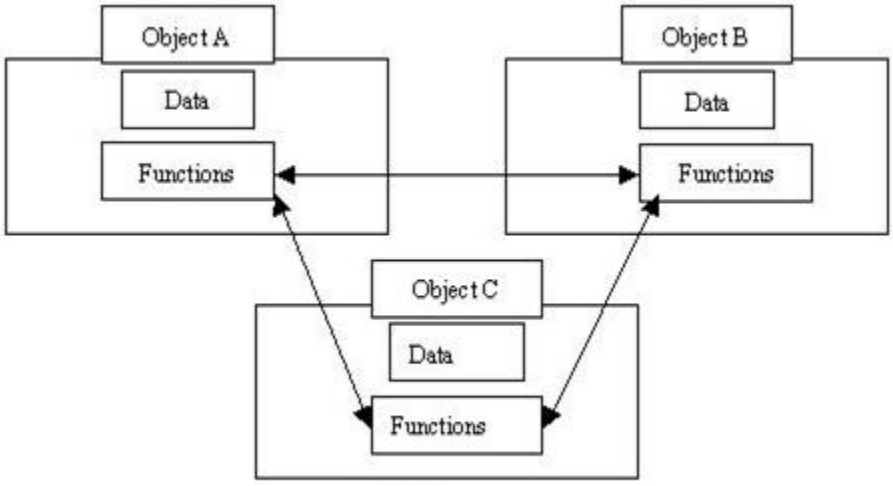
**4. Explain the characteristic features of Object Oriented programming. Give example to support your explanation?[May 2006]**

**Emphasis is on data rather than procedure.**

- The concept of a data class makes it possible to define subclasses of data objects that share some or all of the main class characteristics. Called inheritance, this property of OOP forces a more thorough data analysis, reduces development time, and ensures more accurate coding.
- Since a class defines only the data it needs to be concerned with, when an instance of that class [an object] is run, the code will not be able to accidentally access other program data. This characteristic of data hiding provides greater system security and avoids unintended data corruption.
- The definition of a class is reuseable not only by the program for which it is initially created but also by other object-oriented programs [and, for this reason, can be more easily distributed for use in networks].
- The concept of data classes allows a programmer to create any new data type that is not already defined in the language itself.

**Programs are divided into objects.**

OOP treats data as a critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the functions that operate on it and protects it from accidental modification from outside functions. OOP allows us to decompose a problem into a number of entities called objects and then builds data and functions around these entities. The organization of data and functions in object-oriented programs is shown here.



- Data structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are tied together.
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through functions.
- New data and functions can easily be added whenever necessary.
- Follows bottom-up approach.

Object-oriented programming is the most recent concept among programming paradigms is means different things to different people. It is therefore important to have a working definite object-oriented programming before we proceed further. Our definition of object-oriented programming is as follows “Object-oriented programming is an approach that provides a way of molding programs by creating partitioned memory area for both data, and functions that can be used as templates for creating copies of such modules on demand.”That is, an object is considered to be a partitioned area of computer memory that stores data and set of operations that can access that data. Since the memory partitions are independent, the objects are used in a variety of different programs without modifications.

**5. Write a C++ program to Multiply two matrices and print the result [Apr-2010]**

```
#include<stdio.h>

#include<conio.h>

void main[]

{

int Matrix A[9][9] , MatrixB[9][9] , Matrixsproduct [9][9] ;

int n , i , j , k;                /* 'i' used for rows and 'j' used for columns */

int Row1 , Row2 , Column1 , Column2;

clrscr[];

printf[" Enter the order of Matrix A\n"];

scanf["%d * %d " , &Row1 , &Column1];

printf[" Enter the order of Matrix B\n"];

scanf["%d * %d " , &Row2 , &Column2];

if[Column1 == Row2]

{

printf[" Enter the elements of Matrix A\n"];

for[i=0 ; i<Row1 ; i++]
```

```
{
for[j=0 ; j<Column1 ; j++]
{
scanf["%d" , &Matrix A[i][j] ];
}
}
printf[" Enter the elements of Matrix B\n"];
for[i=0 ; i<Row2 ; i++]
{
for[j=0 ; j<Column2 ; j++]
{
scanf["%d" , &Matrix B[i][j] ];
}
}
```

**/* Product of two Matrices */**

```
for[i=0 ; i<Row1 ; i++]
{
for[j=0 ; j<Column2 ; j++]
{
Matrixproduct[i][j] = 0 ;
for[k=0 ; k<Row2 ; k++]
{
Matrixproduct[i][j] = Matrixproduct[i][j] + [ Matrix A[i][k] * Matrix B[k][j] ];
}
}
}
```

```
printf[" Product Matrix\n"];
for[i=0 ; i< Row1 ; i++]
{
for[j=0 ;j< Column2;j++]
{
printf["%d" , Matrixproduct[i][j] ];
}
printf["\n"];
}
}                                                    /* End of if */
else
{
printf[" Invalid order so Multiplication not possible\n"];
}                                                    /* End of main[] */
```

**Output**

```
Enter the order of matrix A
2*2
Enter the order of matrix B
2*2
Enter the elements of matrix A
1
2
3
4
Enter the elements of matrix A
5
```

6

7

8

Product matrix

19 22

43 50

**6. Write a C++ program that inputs two numbers and outputs the largest number using class. [Nov-2009]**

```
class name
{
private:
int a;
int b;
public:
void show[]
{
cout<<"enter the a and b value";
cin<<a<<b;
if[a>b]
cout<<"a is largest";
else
cout<<"b is largest";
}
};
void main[]
{
name n;
n.show[];
}
```

**7. Write a C++ program using inline function [April-2010]**

Inline function is expanded in line when it is invoked. The compiler replaces the function call with the corresponding function code.

**Syntax**

1. datatype function[parameters] { statements; }

2. inline datatype class::function [parameters] { statements;}

The first syntax example declares an inline function by default. The syntax must occur within a class definition. The second syntax example declares an inline function explicitly. Such definitions do not have to fall within the class definition.

**Sample**

```
/* First example: Implicit inline statement */

int num;

// global num

class cat

{

public:

char* func[void] { return num; }

char* num;

}

/* Second example: Explicit inline statement */

inline char*

cat::func[void] {

return num; }
```

Program to define function cube [ ] as inline for calculating cube:

```
void main [ ]

{

clrscr [ ];

int cube [int];

int j, k, v = 5;

j = cube [3];

k = cube [v];

cout << “\n Cube of J = ”<<j;

cout << “\n Cube of K = ”<<k;
```

```
}
inline int cube
[int h]
{
return [h*h*h];
}
```

**Output**

Cube of J=27

Cube of K=125

**8. Explain briefly about function overloading with a suitable example and also its principles. [April-2011, Dec-2012]**

Same function name for a number of times for different purposes. Defining multiple functions with same is function overloading.

**Example**

```
int sqr [int];
float sqr [float];
int main [ ]
{
int a = 15;
float b = 2.5;
cout <<"Square =" << sqr [a] << "\n";
cout <<"Square =" << sqr [b] << "\n";
return 0;
}
int sqr [int s]
{
Return [s * s];
```

```
}

float sqr [float j]

{

Return [j * j];

}
```

Output:

Square = 225

Square = 6.25

Principles:

- If two functions have similar type of data type and arguments function can't be overloaded.
- Passing constant values directly instead of variable also results in ambiguity.
- To overcome this declares the prototype declaration of all overloaded function before main [ ] function and also pass variable as an argument instead of values.

**9. Explain Friend Function and it's with suitable example and also its characteristics.[Nov 2010, Dec 2012]**

- A function that is declared with a keyword friend is known as friend function. A function can be declared as a friend in any number of classes.
- Friend function must be preceded with a keyword friend.
- The function defined elsewhere in the program like normal C++ function. Function definition doesn't use either keyword friend or scope resolution operator.
- Friend function although not a member function, it has full rights to access the private members of a class

**Declaration**

```
Class class-name

{

Public:

.....

Friend void xyz [void];

};
```

**Example**

```
#include <iostream.h>
```

```
class sample
{
int a, b;
public:
void setvalue[ ]
{
a=25; b=40;
}
friend float mean [sample s];
};
float mean [sample s]
{
return float [s.a + s.b] / 2.0;
}
int main [ ]
{
sample x;
x.setvalue [ ];
cout << “Mean value = “ << mean [x] << “\n”;
return 0;
}
```

Output:

Mean Value = 32.5

**Characteristics**

It’s not in the scope of the class to which it has been declared as friend.

- Since it is not in the scope of the class it can’t be called using object of that class.

- It can be invoked like a normal function without help of any object.
- It can be declared either in public or private part of the class.
- It has object as argument.
- Often used in operator overloading.

**10. Explain different types of constructor with suitable examples.(DEC 2010,DEC 2012,DEC 2009)**

A constructor is a special member function whose task is to initialize the objects of its class. It is special because its name is same as class name. The constructor is invoked whenever an object of its associated class is created. It is called constructor because it constructs the values of data members of the class

**Syntax:**

```
integer Class
{
……
public:
integer( );//constructor
……
}
```

**Types of constructors:**

1. Parameterized constructors
2. Default constructors
3. Copy constructors

**Refer Question 15 for copy and question 18 for parameterized constructor**

**11. Write a C++ program that takes two values of time (hr, min ,sec) and output their sum using constructor and Operator Overloading. (DEC 2009)**

```
#include<iostream.h>
#include<conio.h>
```

```
#include<iomanip.h>
class time
{
int h,m,s;
public:
void read();
void print();
time operator+(time t2);
};
void time::read()
{
cout<<"\nEnter hour,minutes and seconds\n";
cin>>h>>m>>s;
}
void time::print()
{
cout<<"\nTime is-> "<<setfill('0')<<setw(2)<<h;
cout<<":"<<setfill('0')<<setw(2)<<m;
cout<<":"<<setfill('0')<<setw(2)<<s<<endl;
}
time time::operator+(time t2)
{
time t;
t.h=h+t2.h;
t.m=m+t2.m;
t.s=s+t2.s;
return t;
}
void main()
{
clrscr();
time t1,t2,t3;
t1.read();
t1.print();
```

```
t2.read();
t2.print();
t3=t1+t2;
cout<<"\nTime1+ Time2:\n";
t3.print();
getch();
}
```

**12. Write a C++ program to find the area of the Square,Rectangle,Circle using Function Overloading .(DEC 2010)**

```
#include<iostream.h>
#include<conio.h>
  int area(int);
  int area(int,int);
  float area(float,double=3.14);
  float area(float,float,float=0.5);
   void main()
{
   int s,l,b;
   float r,bas,h;
    clrscr();
    cout<<"\n\nAREA OF SQUARE CIRCLE TRIANGLE AND RECTANGLE\n\n";
    cout<<"Enter Square side :";
    cin>>s;
    cout<<"Enter Rectangle length and breadth:";
```

```
    cin>>l>>b;
    cout<<"\nEnter Radius ";
    cin>>r;
    cout<<"\n Enter base and height";
    cin>>bas>>h;
int sqr=area(s);
   cout<<"\nThe Area of Square : "<<sqr;
   int rect=area(l,b);
  cout<<"\nThe Area of Rectangle : "<<rect;
  float circle=area(r);
  cout<<"\nThe Area of Circle : "<<circle;
  float tri=area(bas,h);
  cout<<"\nThe Area of Triangle : "<<tri;
  getch();
}
  int area(int a)
 {
  return(a*a);
  }
 int area(int x,int y)
 {
   return(x*y);
  }
```

```
 float area(float b,double c)
{
  return(c*b*b);
 }
float area(float a,float b,float c)
{
return(a*b*c);
}
```

**OUTPUT**

**Area of Square Circle Triangle And Rectangle**

**Enter Square side: 5**

**Enter Rectangle length and breadth: 10 5**

**Enter Radius:5**

**Enter base and height:5 4**

**The Area of Square : 25**

**The Area of Rectangle : 50**

**The Area of Circle : 78.5**

**The Area of Triangle :10**

**13. Explain with programs the various types of type conversions. (JUNE 2013)**

It is the process of converting one type into another. In other words converting an expression of a given type into another is called type casting. There are two ways of achieving the type conversion namely:

- **Automatic Conversion** otherwise called as **Implicit Conversion**
- **Type casting** otherwise called as **Explicit Conversion**

This is not done by any conversions or operators. In other words value gets automatically converted to the specific type in which it is assigned.

**example:**

#include <iostream.h> void main() {

Short x=6000; int y; y=x; }

In the above example the data type short namely variable x is converted to int and is assigned to the integer variable y.

So as above it is possible to convert short to int, int to float and so on.

**Type casting otherwise called as Explicit Conversion**

Explicit conversion can be done using type cast operator and the general syntax for doing this is,

datatype (expression);

- Here in the above datatype is the type which the programmer wants the expression to gets changed as
- In C++ the type casting can be done in either of the two ways mentioned below namely:

  C-style casting C++-style casting

  The C-style casting takes the synatx as

  (type) expression

  This can also be used in C++.
- Apart from the above the other form of type casting that can be used specifically in C++ programming language namely C++-style casting is as below namely:

  type (expression)

- This approach was adopted since it provided more clarity to the C++ programmers rather than the C-style casting. Say for instance the as per C-style casting

  (type) firstVariable * secondVariable
- is not clear but when a programmer uses the C++ style casting it is much more clearer as below

type (firstVariable) * secondVariable

**Let us see the concept of type casting in C++ with a small example:**

```
#include <iostream.h>
void main()
{
int a;
float b,c;
 cout<< “Enter the value of a:”;
 cin>>a;
cout<<  “Enter the value of b:”;
cin>>b;
c=float(a)+b;
cout<<Enter The value of c is:”<<c;
}
```

**The output of the above program is**

Enter the value of a: 10 Enter the value of b: 12.5 The value of c is: 22.5

- In the above program a is declared as integer and b and c are declared as float. In the type conversion statement namely

  c = float(a)+b;

- The variable a of type integer is converted into float type and so the value 10 is converted as 10.0 and then is added with the float variable b with value 12.5 giving a resultant float variable c with value as 22.5

**14. Write a C++ program to explain how the member function can be accessed using operators (MAY 2011)**

```
#include <iostream>

using namespace std;

class Box
{
   public:
      double length;         // Length of a box
      double breadth;        // Breadth of a box
      double height;         // Height of a box

      // Member functions declaration
```

```
      double getVolume(void);
      void setLength( double len );
      void setBreadth( double bre );
      void setHeight( double hei );
};

// Member functions definitions
double Box::getVolume(void)
{
    return length * breadth * height;
}

void Box::setLength( double len )
{
    length = len;
}

void Box::setBreadth( double bre )
{
    breadth = bre;
}

void Box::setHeight( double hei )
{
    height = hei;
}

// Main function for the program
int main( )
{
   Box Box1;               // Declare Box1 of type Box
   Box Box2;               // Declare Box2 of type Box
   double volume = 0.0;     // Store the volume of a box here

   // box 1 specification
   Box1.setLength(6.0);
   Box1.setBreadth(7.0);
   Box1.setHeight(5.0);

   // box 2 specification
   Box2.setLength(12.0);
   Box2.setBreadth(13.0);
   Box2.setHeight(10.0);

   // volume of box 1
   volume = Box1.getVolume();
   cout << "Volume of Box1 : " << volume <<endl;

   // volume of box 2
   volume = Box2.getVolume();
```

```
   cout << "Volume of Box2 : " << volume <<endl;
   return 0;
}
```

**Output**

```
Volume of Box1 : 210
Volume of Box2 : 1560
```

**15. Explain Copy Constructor with an example (MAY 2011) (MAY 2010) (DEC 2011)**

Copy constructor is

- a constructor function with the same name as the class
- used to make deep copy of objects.

There are 3 important places where a copy constructor is called.

1. When an object is created from another object of the same type .
2. When an object is passed by value as a parameter to a function .
3. When an object is returned from a function.

If a copy constructor is not defined in a class the compiler itself defines one. This will ensure a shallow copy. If the class does not have pointer variables with dynamically allocated memory then one need not worry about defining a copy constructor. It can be left to the compiler's discretion.

**Example Program**

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
using namespace std;
class test
{
public:
        int a;
        test()
        {
                a=0;
                cout<<a;
        }
```

```
        test(int pa)
        {
                a = pa;
                cout<<a;
        }
        test(test &t) //copy constructor
        {
                a=t.a;
                cout<<"\n t.a"<< t.a;
        }
};
void main()
{
        test t(5);
        cout<<"\n"<<t.a;
        test t1(t);
        cout<<"\n"<<t1.a;
        getchar();
}
```

**16. Explain Friend function with an example (MAY 2010)**

- when a data is declared as private inside a class, then it is not accessible from outside the class.
- A function that is not a member or an external class will not be able to access the private data.
- A programmer may have a situation where he or she would need to access private data from non-member functions and external classes. For handling such cases, the concept of Friend functions is a useful tool.
- A friend function is used for accessing the non-public members of a class. A class can allow non-member functions and other classes to access its own private data, by making them friends.

- Thus, a friend function is an ordinary function or a member of another class. The friend function is written as any other normal function, except the function declaration of these functions is preceded with the keyword friend.
- The friend function must have the class to which it is declared as friend passed to it in argument the keyword friend is placed only in the function declaration of the friend function and not in the function definition.
- It is possible to declare a function as friend in any number of classes. When a class is declared as a friend, the friend class has access to the private data of the class that made this a friend.
- A friend function, even though it is not a member function, would have the rights to access the private members of the class.
- It is possible to declare the friend function as either private or public. The function can be invoked without the use of an object.
- The friend function has its argument as objects, seen in example below.

```
#include<iostream.h>
#include<conio.h>
class  base
{
    int val1,val2;
   public:
    void get()
    {
       cout<<"Enter two values:";
       cin>>val1>>val2;
    }
    friend float mean(base ob);
};
```

```
float mean(base ob)
{
   return float(ob.val1+ob.val2)/2;
}
void main()
{
    clrscr();
    base obj;
    obj.get();
    cout<<"\n Mean value is : "<<mean(obj);
    getch();
}
```

**Output:**

Enter two values: 10, 20

Mean Value is: 15

**17. Explain the Syntax for Operator Overloading. How many arguments are needed in the definition of an Overloaded binary operator? Give an example. (MAY 2010)**

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
class complex
{
```

```
private:
float real;
float imag;
public:
complex() { real=imag=0.0; }
complex(int r,int i) //conversion constructor
{
real = r;
imag = i; }
complex(double r, double i)//conversion constructor
{
real = r;
imag = i;}
friend istream& operator>>(istream &, complex &);
friend ostream& operator<<(ostream &, complex &);
complex operator+(complex);
complex operator-(complex);
complex operator*(complex);
complex operator/(complex);

friend double condou(complex t); //complex–>double
};double condou(complex t)
{
return t.real+t.imag;
}istream& operator >>(istream &in, complex &c)
{
cout<<"\nReal Part:";
in>>c.real;
cout<<"Imag Part:";
in>>c.imag;
return in;
}ostream& operator<<(ostream &out, complex &c)
{
if (c.imag<0)
```

```
out<<c.real<<c.imag<<"i";
else
out<<c.real<<"+"<<c.imag<<"i";
return out;
}complex complex::operator+(complex c)
{
complex temp;
temp.real = real+c.real;
temp.imag = imag+c.imag;
return tem p;
}complex complex::operator-(complex c)
{
complex temp;
temp.real = real-c.real;
temp.imag = imag-c.imag;
return temp;
}complex complex::operator*(complex c)
{
complex temp;
temp.real = real*c.real-imag*c.imag;
temp.imag = real*c.imag+imag*c.real;
return temp;
}complex complex::operator/(complex c)
{
complex temp;
float qt;
qt = c.real*c.real+c.imag*c.imag;
temp.real = (real*c.real+imag*c.imag)/qt;
temp.imag = (imag*c.real-real*c.imag)/qt;
return temp;
}void main()
{
complex c1, c2, c3,c4(4,9),c5(3.23004,4.666304444);
double t;
```

```
clrscr();
t=condou(c5);
cout<<"\nEnter complex number 1: ";
cin>>c1;
cout<<"\nEnter complex number 2: ";
cin>>c2;
cout<<"\nEnter complex numbers are:";
cout<<"\nComplex 1: "<<c1;
cout<<"\nComplex 2: "<<c2;
c3=c1+c2;
cout<<"\nResult of addition is:"<<c3;
c3=c1-c2;
cout<<"\nResult of subtraction is:"<<c3;
c3=c1*c2;
cout<<"\nResult of multiplication is:"<<c3;
c3=c1/c2;
cout<<"\nResult of division is:"<<c3;
cout<<"\nInteger–>complex:"<<c4;
cout<<"\nDouble–>complex:"<<c5;
cout<<"\nConverted to double"<<t; getch();
}
```

**OUTPUT**:

Enter complex number 1:

Real Part:12

Imag Part:24

Enter complex number 2:

Real Part:89

Imag Part:67

Enter complex numbers are:

Complex 1: 12+24i

Complex 2: 89+67i

Result of addition is:101+91i

Result of subtraction is:-77-43i

Result of multiplication is:-540+2940i

Result of division is:0.215633+0.107333i

Integer->complex:4+9i

Double->complex:3.23004+4.6663i

Converted to double7.89634

**18. Explain Parameterized constructor and Explicit constructor with suitable example(JUNE 2011)**

**Parameterized constructor**

C++ permits us to achieve this objects bt passing argument to the constructor function when the object are created . The constructor that can take arguments are called parametrized constructors.

**Example**

```
#include<iostream>

#include<conio.h>

using namespace std;

class Example       {

   // Variable Declaration

   int a,b;

   public:
```

```
    //Constructor
    Example(int x,int y)          {
    // Assign Values In Constructor
    a=x;
    b=y;
    cout<<"Im Constructor\n";
    }
    void Display()    {
    cout<<"Values :"<<a<<"\t"<<b;
    }
};
int main()              {
        Example Object(10,20);
        // Constructor invoked.
        Object.Display();

        // Wait For Output Screen
        getch();
        return 0;
}
```

**Output**

Im Constructor

Values :10 20

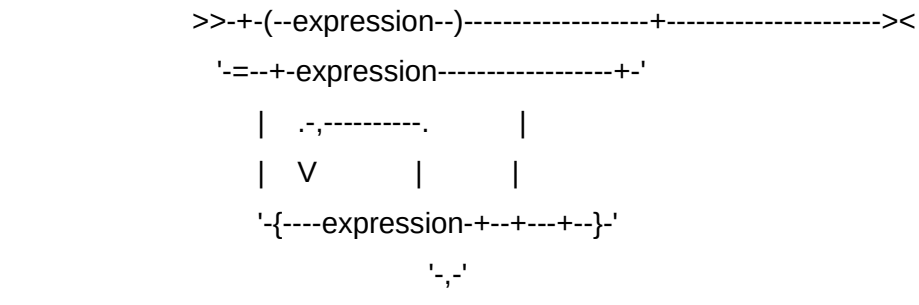
**Explicit constructor**

- A class object with a constructor must be explicitly initialized or have a default constructor. Except for aggregate initialization, explicit initialization using a constructor is the only way to initialize nonstatic constant and reference class members.
- A class object that has no constructors, no virtual functions, no private or protected members, and no base classes is called an aggregate. Examples of aggregates are C-style structures and unions.

You explicitly initialize a class object when you create that object. There are two ways to initialize a class object:

- Using a parenthesized expression list. The compiler calls the constructor of the class using this list as the constructor's argument list.
- Using a single initialization value and the = operator. Because this type of expression is an initialization, not an assignment, the assignment operator function, if one exists, is not called.
- The type of the single argument must match the type of the first argument to the constructor. If the constructor has remaining arguments, these arguments must have default values.

Initializer syntax




The following example shows the declaration and use of several constructors that explicitly initialize class objects:

```
// This example illustrates explicit initialization
// by constructor.
#include <iostream>
using namespace std;
class complx {
  double re, im;
public:

  // default constructor
  complx() : re(0), im(0) { }
  // copy constructor
  complx(const complx& c) { re = c.re; im = c.im; }
  // constructor with default trailing argument
  complx( double r, double i = 0.0) { re = r; im = i; }
  void display() {
    cout << "re = "<< re << " im = " << im << endl;
  }
};
int main() {
  // initialize with complx(double, double)
  complx one(1);
  // initialize with a copy of one
  // using complx::complx(const complx&)
  complx two = one;
  // construct complx(3,4)
  // directly into three
  complx three = complx(3,4);
  // initialize with default constructor
  complx four;
  // complx(double, double) and construct
  // directly into five
  complx five = 5;
  one.display();
  two.display();
```

```
  three.display();
  four.display();
  five.display();
}
```

**The above example produces the following output:**

```
re = 1 im = 0
re = 1 im = 0
re = 3 im = 4
re = 0 im = 0
re = 5 im = 0
```

**19. Give access rules for accessing static data members. (4) (AUC MAY 2008)**

Classes can contain static member data and member functions. When a data member is declared as static, only one copy of the data is maintained for all objects of the class. (For more information, see Static Member Functions.)

Static data members are not part of objects of a given class type; they are separate objects. As a result, the declaration of a static data member is not considered a definition. The data member is declared in class scope, but definition is performed at file scope. These static members have external linkage. The following example illustrates this:

```
// static_data_members.cpp
class BufferedOutput
{
public:
   // Return number of bytes written by any object of this class.
   short BytesWritten()
   {
      return bytecount;
   }
```

```
  // Reset the counter.
  static void ResetCount()
  {
    bytecount = 0;
  }

  // Static member declaration.
  static long bytecount;
};

// Define bytecount in file scope.
long BufferedOutput::bytecount;

int main()
{
}
```

In the preceding code, the member bytecount is declared in class BufferedOutput, but it must be defined outside the class declaration.

Static data members can be referred to without referring to an object of class type. The number of bytes written using BufferedOutput objects can be obtained as follows:

```
long nBytes = BufferedOutput::bytecount;
```

For the static member to exist, it is not necessary that any objects of the class type exist.

Static members can also be accessed using the member-selection (. and –>) operators.

**20. Write a program to overload unary operator ++ for incrementing distance. Assume that the distance class has feet and inches as data members. (12) (AUC MAY 2008)**

```
#include <iostream>

using namespace std;

class Distance {

  int feet,inch;

public:

  Distance() { feet= 0;inch=0 }

  Distance(int i, int j) {feet = i; inch = j; }


 Distance operator++();


  void show() ;

} ;




Distance Distance::operator++()

{

  feet++;

  inch++;

  return *this;
```

```
}
void Distance::show()
{
  cout << feet << ", ";
  cout << inch << ", ";
}

int main()
{
  Distance a(1, 2);

  cout << "Original value of a: ";
  a.show();

  ++a;
  cout << "Value after ++a: ";
  a.show();

  return 0;
}
```

**21. What is operator overloading? How many arguments are required in the definition of an overload binary operator. (6) (AUC MAY 2008)**

C++ has the ability to provide the operators with a special meaning for a data type. This mechanism of giving such special meanings to an operator is known as Operator overloading. It provides a flexible option for the creation of new definitions for C++ operators.

**Defining operator overloading**

To define an additional task to an operator, we must specify what it means in relation to the class to which the operator is applied. This is done with the help of a special function, called operator function, which describes the task.

```
return type classname :: operator op(arglist)
{
      Function body          // task defined
}
```

where *return type* is the type of value returned by the specified operation and *op* is the operator being overloaded. The *op* is preceded by the keyword **operator**. **operator** *op* is the function name.

Operator functions must be either member functions or friend functions. A basic difference is that a friend function will have only one argument for unary operators and two for binary operators, while a member function has no arguments for unary operators and one for binary operators.

This is because the object used to invoke the member function is passed implicitly, which is not so in case of friend functions. Operator functions are declared in the class using prototype as follows:

```
vector operator+(vector);                    // vector addition
vector operator-();                          // unary minus
friend vector operator+(vector,vector);      // vector addition
friend vector operator-(vector);             // unary minus
vector operator-(vector &a);                 // subtraction
int operator==(vector);                      // comparison
friend int operator==(vector,vector)         // comparison
```

**vector** is a data type of **class** and may represent both magnitude and direction (as in physics and engineering) or a series of points called elements (as in mathematics)

The process of overloading involves the following steps:

- Create a class that defines the data type that is to be used in the overloading operation.
- Declare the operator function operator op() in the public part of the class. It may be either a member function or a friend function.
- Define the operator function to implement the required operations.

**22. Explain the various types of constructors. (4)**

**Default Constructor:-** Default Constructor is also called as Empty Constructor which has no arguments and It is Automatically called when we creates the object of class but Remember name of Constructor is same as name of class and Constructor never declared with the help of Return Type. Means we can't Declare a Constructor with the help of void Return Type. , if we never Pass or declare any Arguments then this called as the Copy Constructors.

**Parameterized Constructor: -** This is another type Constructor which has some Arguments and same name as class name but it uses some Arguments So For this We have to create object of Class by passing some Arguments at the time of creating object with the name of class. When we pass some Arguments to the Constructor then this will automatically pass the Arguments to the Constructor and the values will retrieve by the Respective Data Members of the Class.

**Copy Constructor:-** The copy constructor is a constructor which creates an object by initializing it with an object of the same class, which has been created previously. The copy constructor is used to: Initialize one object from another of the same type.

- Copy an object to pass it as an argument to a function.
- Copy an object to return it from a function.

If a copy constructor is not defined in a class, the compiler itself defines one.If the class has pointer variables and has some dynamic memory allocations, then it is a must to have a copy constructor. The most common form of copy constructor is shown here:

```
Class name (const class name &obj)
 {
   // body of constructor
}
```

**23. Write a C++ program that takes the (x,y) coordinates of two points and outputs the distance between them using constructors. (12) (AUC DEC 2009)**

```
#include <iostream>
#include <math.h>
using namespace std;
```

```
struct Point {
    double x;
    double y;
    Point() { //constructor 1
        x = 0;
        y = 0;
    }
    Point(double _x, double _y) { //constructor 2
        x = _x;
        y = _y;
     }
     };

    int main() {
    double x1, x2, y1, y2;
    cout << "Enter coordinate of two points " << endl << endl;
    cout << "Point 1" << endl;
    cout << " x: ";
    cin >> x1;
    cout << " y: ";
    cin >> y1;
    cout << "Point 2" << endl;
    cout << " x: ";
    cin >> x2;
    cout << " y: ";
    cin >> y2;
    Point p1(x1, y1), p2(x2, y2); //use constructor
    double dx = p2.x - p1.x;
    double dy = p2.y - p1.y;
    double distance = sqrt(pow(dx, 2) + pow(dy, 2));
    cout << endl << "Distance between two points is " << distance << endl;
    return 0;
}
```