Unit-I

INTRODUCTION TO OOPS

1) _____ are the basic run-time entities in an object oriented system.
A. **Object** B. Encapsulation C. Dynamic Binding D. Polymorphism

2) What is the wrapping up of data and functions into a single unit is known as?
A. Dynamic Binding B. Polymorphism C. Inheritance **D. Encapsulation**

3) What is a template representing a group of objects that share common properties and relationships called?
A. Object **B. Class** C. Inheritance D. Encapsulation

4) Which one of the below mentioned facilitate objects of one class to acquire theproperties of objects of another class
A. Polymorphism **B. Inheritance** C. Message Passing D.Encapsulation

5) The code associated with a given procedure call is not known until the time of callat run-time, this is due to?
A. Inheritance **B. Dynamic Binding** C.Polymorphism D.Data Abstraction

6) Which operator is used to define the member functions outside the class?
A. $ **B.: :** C. & D. *

7) Which of the following term is used for a function defined inside a class?
A. Member Variable **B. Member Function**
C. Class Function D. Static Function

8) Which function can be invoked like a normal function without the help of any object?
A) constant member function
B) private member function
C) static member function
D) friend function

9) Although not a member function, which of the below mentioned function has fullaccess rights to the private members of the class.
A) constant member function
B) private member function
C) static member function
D) friend function

10) In which method of passing arguments using an object as a function argument,a copy of the entire object is passed to the function
A) pass-by-value

B) pass-by-reference
C) pass-by-variable
D) pass-by-function

11) Can you identify an object-based programming language among these?
A.C++ B. Java **C. ADA** D. Smalltalk

12) Judge which of the below statements are True?
i. OOP software is not having set standards.
ii. To convert a real-world problem into an object oriented model is easy.
iii. The classes are overly generalized.
A. i and ii B. All of the above
C. i and iii D. i only

13) Identify which is the access specifier for the members of a class when not specifically mentioned or specified
A. Public **B. Private** C. Protected D. Static

14) Can you judge which kind of access means that members can be accessed by any function outside the class also.
A. **Public .**B. Private C. Protected D. Static

15) Predict the correct format for calling a memberfunction:
A. Object-name. function-name (actua1-arguments);
B. Object-name: function-name (actua1-arguments);
C. Object-name= function-name (actua1-arguments);
D. Object-name& function-name (actua1-arguments);

16) Select the valid statement to access, If getdata(int a, float b) is a member function ofthe class……
A .getdata (100, 25.5) B. **x.getdata(100, 25.5)**

C.x =getdata(100,25.5) D. getdata()

17) Can you suggest, how should the data member be declared in a class so as tomaintain values common to the entire class
A. Private B. public **C. static** D. protected

18) Which is most appropriate comment on following classdefinition?
```
class Student
{
int a;
public: float a;
};
```
a. Error: Same variable can't be used twice.
b. Error: Public must be declared first
c. Error: Data types are different for same variable.
d. It is correct

19) Identify which of the below mentioned syntax for class definitionis wrong
A) class student{ };
B) student class{ };
C) class student {public: student(int a){ } };
D) class student{ student(int a){} };

20) Identify which feature of OOP is used in below scenerario- "If a function canperform more than 1 type of tasks, where the function name remains same"
A) Encapsulation
B) Inheritance
C) Polymorphism
D) Abstraction

21) Can you Judge which of the following statements regarding inline functions iscorrect?
A. It speeds up execution B. It slows down execution
C. It increases the code size **D. Both A and C**

22) Can you predict ,what is correct about the static data member of a class?
A. A static member function can access all the data members of a class.
B. Static member remains common for all the objects of the same class.
C. Static data member is not initialized to 0 when first object of the class is created.
D. Static data member is can't be declared inside the class.

23) Identify which of the below can be used to call a static member function instead of objects.
A) variable name
B) function name
C) Class name
D) object name

24) Identify which of the following is the only technical difference betweenstructures and classes in C++?
A. Members are by default protected in structures but private in classes.
B. Members are by default private in structures but public in classes.
C. Members are by default public in structures but private in classes.
D. Members are by default public in structures but protected in classes.

25) Select an option which is true about static Member function,
A. It can access any other member function and member variable.
B. It can access only static member variables and member functions
C. It can be only called through the object of the class
D. It returns only static data

(4marks Questions)

1. Explain the basic concepts of oops.

1.Classes

Class is a template/blue-print representing a group of objects that share common properties and relationships.

The objects can contain data and code to manipulate the data.
The entire set of data and code of an object can be made a user defined data type with the help of a class.
Therefore objects are variables of the type class.

2.**Objects**

Objects are the basic run-time entities in an object oriented system.
An object may represent a person, place, a bank account, a table of data or any item that the program has to handle.
An object is a collection of data members and associated member functions.
Each object is identified by a unique name.
Every object must be a member of a particular class. Ex:
Apple, orange, mango are the objects of class fruit.
Objects take up space in memory and have address associated with them.

3. Data abstraction

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

Data abstraction permits the user to use an object without knowing its internal working.
The attributes are called as data members because they hold information. The functions that operate on these data are called methods or member functions.

4. Data encapsulation

The wrapping up of data and functions into a single unit is known as encapsulation.Data encapsulation combines data and functions into a single unit called class.
Data encapsulation will prevent direct access to data.
The data can be accessed only through methods (function) present inside the class.The data cannot be modified by an external non-member function of a class.
Data encapsulation enables data hiding or information hiding.

5. Inheritance

Inheritance is the process by which objects of one class acquire the properties of objects of another class.

In OOP, the concept of inheritance provides the idea of reusability.

This means that we can add additional features to an existing class without modifying it.Thus the process of forming a new class from an existing class is known as Inheritance.

6. Polymorphism

Polymorphism is a function can take multiple forms based on the type of arguments, numberof arguments and data type of return value.
The ability of an operator and function to take multiple forms is known as polymorphism.
Consider the addition operation. In addition of two numbers the result is the sum of two numbers. In addition of two strings the operation is string concatenation. When an operator behaves differently based on operands, then it is said that operator is overloaded.

7. Dynamic Binding

Binding refers to the linking of a procedure call to the code to be executed in response to thecall. Dynamic binding (also known as late binding) means that the code associated with a given procedure call is not known until the time of call at run-time.
It is associated with polymorphism and inheritance.

2. What are the advantages OOPs?

1. **Simplicity:** software objects model real world objects, so the complexity is reduced and the program structure is very clear;
2. **Modularity:** each object forms a separate entity whose internal workings are decoupled from other parts of the system;
3. **Modifiability:** it is easy to make minor changes in the data representation or the procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods;
4. **Extensibility:** adding new features or responding to changing operating environments can be solved by introducing a few new objects and modifying some existing ones;
5. **Maintainability:** objects can be maintained separately, making locating and fixing problems easier;
6. **Re-usability:** objects can be reused in different programs

3. Write the disadvantages of object oriented programming.

- **Size**: Object Oriented Programs are much larger than other programs.
- **Effort**: Object Oriented Programs require a lot of work to create.
- **Speed**: Object Oriented Programs are slower than other programs, because of their size.
- **Plan before**: A programmer needs to plan beforehand for developing a program in OOP.

4. What is a class? Explain class definition with an example.

The general form of a class declaration is:
Class is a template/blue-print representing a group of objects that share common properties and relationships.
Syntax:
class class_name

{
private:
variable declarations;
function declarations;
public:
variable declarations;
function declarations;
protected:
variable declarations;
function declarations;
};

. The keyword class specifies, that what follows is an abstract data of type class_name. The body of a class is enclosed within braces and terminated by a semicolon. The class body contains the declaration of variables and functions.
These functions and variables are collectively called class members.

They are usually grouped under three sections, namely, private,public and protected.
The keywords private, public and protected are known as visibility labels or accessspecifier.
Keywords are followed by a colon(:).
The class members that have been declared as private can be accessed only from within theclass.
On the other hand, public members can be accessed from outside the class also.

A Simple Class Example

```
class item
{
int   number;
float     cost;
public:
void  getdata(int  a,  float  b);
void putdata(void);
};
```

5. How is a member function of a class defined? Explain.

Member functions can be defined in two places:

- Outside the class definition.
- Inside the class definition.

Outside the Class Definition

Syntax:

return-type class-name : : function-name (argument declaration)
{

Function body

}

The membership label class-name:: tells the compiler that the function function-name belongs to the class class-name. That is, the scope of the function is restricted to the class-name specified in the header line. The symbol :: is called the scope resolution operator.

```
#include  <iostream.h>class
item
{
int   number;
float     cost;
public:
void  getdata(int  a,  float  b);
void putdata(void);
};

void item :: getdata(int a, float b)
{
number  =  a;
cost = b;
}
void putdata(void)
{
```

```
cout<<”Number:”<<number<<”\n”;
cout<<”Cost:”<<cost<<”\n”;
}

int main()
{
item x;
x.getdata(100,          299.5);
x.putdata();
return 0;
}
```

Inside the Class Definition

Another method of defining a member function is to replace the function declaration by theactual function definition inside the class.

Example

```
class item
{
int number:
float cost;
public:
void  getdata(int a,  float b);
void putdata(void)
{
cout << number << "\n":
cout << cost << "\n";
}

};
```

6. Describe Access specifiers in a class.

private

- private access means a member data can only be accessed by the member function. Members declared under private are accessible only within the class.
- If no access specifier is mentioned, then by default, members are private.

```
Private:
int x;
float y;
```

public

public access means that members can be accessed by any function outside the class also.

```
class box
{
int length;int
height;
public : int width;
void set_height ( int i );
void get_height();
};
```

Protected

- The members which are declared using protected can be accessed only by the member functions, friends of the class and also by the member functions derived from this class. The members cannot be accessed from outside.
- The protected access specifier is therefore similar to private specifier.
- protected - members cannot be accessed from outside the class, however, they can be accessed in inherited classes.
- When inheritance or child of a class means we can use protected concept.

7. Illustrate with an example how an array of objects can be defined?

Arrays of variables that are of the type class. Such variables are called arrays of objects.Main purpose is member functions can be call more than one time.

Example

```
#include <iostream.h>class
employee
{
char name[30];float age;
public:
void getdata(void); void
putdata(void);
};


void employee : : getdata(void)
{
cout << "Enter name:"; cin >>
name;
 cout << "Enter age:"
```

```
cin >> age;
}

void employee:: putdata(void)
{
cout <<"Name: “<< name<<”\n”; cout<<
"Age: “<<age<<”\n”;
}

const int size=3;

int main()
{
employee manager[size]; for(int
i=0;i<size; i ++)
{
cout << “\nDetails of manager” << i+1 <<”\n”;
manager[i].getdata();
}
cout << "\n";
for(i=0; i<size; i ++)
{
cout << "\nManager” << i+1 << "\n";
manager[i].putdata();
}
retum O;
}
```

Output:

Detai ls of manager1Enter name: Ram Enter age: 45
Details of manager2Enter name: Shyam Enter age: 37 Details of manager3Enter name: Sam Enter age: 50

Program outputManagerl Name: Ram Age: 45 Manager2 Name: Shyam Age: 37 Manager3 Name: Sam

Age: 50

8. How the objects are passed as function arguments? Explain.

This can be done in two ways:

· A copy of the entire object is passed to the function.

· Only the address of the object is transferred to the function.

The first method is called **pass-by-value**. Since a copy of the object is passed to the function,any changes made to the object inside the function do not affect the object used to call the function.

The second method is called **pass by reference**. When an address of the object is passed, the called function works directly on the actual object used in the call.

This means that any changes made to the object inside the function will reflect in the actual object. The pass-by reference method is more efficient since it requires to pass only the address of the object and not the entire object.

Example:

```
#include <iostream.h>class
time
{
int hours; int minutes;
public:
void gettime(int h, int m)
{
hours =h; minutes= m;
}
void puttime(void)
{
cout << hours <<" hours and ";
cout << minutes << " minutes" << "\ n";
}
void sum(time,time); // declaration with objects as arguments
} ;


void time :: sum(time t1 , time t2) // t l , t2 are Objects
{
minutes = tl.minutes + t2.minutes; hours =
minutes/60; minutes=minute%60 ;
hours = hours + t l.hours + t2.hours;
}


int main()
{
time T1, T2, T3; T1
.gettime(2,45);
T2.gettime(3,30) ;
```

```
T3.sum(Tl,T2);
cout << "Tl =”; T1.puttime();
cout << "T2 =”;T2.puttime();
cout <<'T3=”; T3.puttime();
return 0;
}
```

Output

T1 = 2 hours and 45 minutesT2 = 3 hours and 30 minutes T3 =6 hours and 15 minutes

9. What is a friend function? What are the characteristics of friend functions?

A friend function is a non-member function that is a friend of a class. The friend function is declared within a class with the prefix friend.

Characteristics of friend functions are:

- A friend function although not a member function, has full access right to the privateand protected members of the class.
- A friend function cannot be called using the object of that class. It can be invoked likeany normal function.
- They are normal external functions that are given special access privileges.
- It cannot access the member variables directly and has to use an object
- name.membername. It can be invoked like a normal function without the help of any object.
- The function is declared with keyword friend. But while defining friend function itdoes not use either keyword friend or :: operator.
- It can be declared either in the public or the private part of a class without affecting its meaning.
- Usually, it has the objects as arguments.

10. What you mean by friend function? Explain with example.

- A friend function is a non-member function that is a friend of a class.
- The friend function is declared within a class with the prefix friend.
- But it should be defined outside the class like a normal function without the prefix friend.
- It can access public data members like non- member functions.

Example:

```
#include <iostream.h>
class myclass
{
private:
int a,b;
public:
void set_val(int i, int j);
friend int add(myclass obj);
};
```

```
void myclass::set_val(int i,int j)
{
a = i;
 b = j;
}
int add(myclass obj)
{
return (obj.a+obj.b);
}
int main()
{
myclass object;
 object.set_val(34, 56);
cout << "Sum of 34 and 56 is "<<add(object)<<endl;
 return 0;
}
```

OUTPUT:
Sum of 34 and 56 is 90

11. When do we declare a member of a class static? Explain with example.

STATIC DATA MEMBERS

Hence all the object of the same class share static data member memory is allocated onlyonce to the static data member. It remains common for all the objects of the same class.

Static data member is initialized to 0 when first object of the class is created.

Static data member is declared in the class but it must be defined outside the class using class name and scope resolution operator (::) because memory allocation for static data member of the class is performed different then normal data member of the class and it is notthe part of class object.

STATIC MEMBER FUNCTIONS

A static function can have access to only other static members (functions or variables)declared in the same class.

A static member function can be called using the class name (instead of its objects) asfollows:

class-name:: function -name;

Example:

```
#include    <iostream>   class
Demo
{
private:
static int X;static int Y;

public:
static void Print()
{
cout <<"Value of X: " << X << endl;cout <<"Value of Y: " <<
Y << endl;
}
};
```

```
int Demo :: X =10; int
Demo :: Y =20;

int main()
{
Demo OB; Demo::Print();
return 0;
}
```

Output
Value of X: 10
Value of Y: 20

12. Explain in brief about inline function with example

A function call generally involves the complex process of invoking a function, passing parameters to the function, allocating storage for local variables, thereby using extra time and memory space. It is possible to avoid these overheads of a function by using inline function. The inline function is a short function. Compiler replaces a function call with the body of the function.
The keyword inline is used to define inline functions. The inline function consists of function call along with function code and the process is known as expansion.

- • Inline functions definition starts with keyword inline.
- • The inline functions should be defined before all functions that call it.
- • The compiler replaces the function call statement with the function code itself (expansion) and then compiles the entire code.
- • They run little faster than normal functions as function calling overheads are saved.

Program: Finding the square of a number using inline functions. #include <iostream.h>

```
class sample
{
public: inline int square (int a)
{
return(a*a);
}

int main( )
{
int x, y; x=square(5);
cout<<"Square of 5 = "<<x<<endl; y=square(10);

cout<<"Square of 10 = "<<y<<endl; return 0;
}
```

Output
Square of 5 = 25
Square of 10 = 100

- In the above example square() is an inline function that finds the square of a number.

****

UNIT 2

CONSTRUCTORS, DESTRUCTORS, AND OPERATOR OVERLOADING

Objective Type Questions:

1. What is the keyword used in operator overloading to create an operator function?
   A. Class
   B. Destructor
   C. Constructor
   D. **Operator**

2. Which of the following is the set of operators that can't be overloaded in C++?

   A. +=, ?, :: , >>

   B. >>, <<, ?, *, sizeof()

   C.:: , . , .* , ?:, sizeof()

   D.:: , ->, * , new, delete

3. How many argument/s are required, when overloading unary operators using the Friend function

   A. Zero

   B. **One**

   C. Two

   D. Three.

4. What is a symbol preceding a destructor?

   A.!

   B.?

   C. ~

   D. *

5. How many argument/s are required while overloading binary operators using member function

   A. Zero

   B. **One**

   C. Two

   D. Three

6. The process of making an operator exhibit different behaviors in different instances is known as –

A. Operator overloading

B. Function overloading

C. Dynamic binding

D. Late Binding

7. Which constructor doesn't accept any parameter?

A. Default Constructor

B. Parameterized Constructor

C. Copy Constructor

D. Static Constructor

8. How many default constructors per class are possible?

A. Only one

B. Two

C. Three

D. Unlimited

9. What is the other name for an overloaded casting operator function –

A. Casting function

B. operator function

C. Conversion function

D. overloaded function

10. Can you identify which of the below statement is true about Constructors.

A. construct the data members

B. Both initialize the objects & create new data members

C. Constructs the data type for objects

D. initialize the objects

11. Can you judge which of the following statement is correct?

A. A constructor is called at the time of declaration of an object.

B. A constructor is called at the time of use of an object.

C. A constructor is called at the time of declaration of a class.

D. A constructor is called at the time of use of a class.

12. Which of the following statement is correct about the constructors and destructors?

A. Destructors can take arguments but constructors cannot.

B. Constructors can take arguments but destructors cannot.

C. Destructors can be overloaded but constructors cannot be overloaded.

D. Constructors and destructors can both return a value.

13. Can you judge which of the below can be used to identify the copy constructor of class type X?

A) (X&)

B) **X(&X)**

C) X(X&)

D) X(X)

14. Which of the below is supported for the Constructors declaration.

A. Constructors should be declared in the Private section

B. Constructors should be declared in the Public section

C. Constructors should be declared in the protected section.

D. Constructors should be declared as Static

15. Can you judge which of the below can be said about constructors, “when more than one constructor function is defined in a class”

A. Parameterized

B. Copied

C. Overloaded

D. Inherited

16. Can you Identify the correct method for a copy constructor to receive its arguments

A. either pass-by-value or pass-by-reference

B. only pass-by-value

C. only pass-by-reference

D. only passes by the address

17. Can you judge which of the following statement should be taken into consideration, in case of binary operator overloading with member function,

A. Right-hand operand must be an object.

B. Left-hand operand must be an object.

C. Both the operands must be objects.

D. All of these should be considered.

18. Predict the type conversion where the source type is the basic type and the destination type is the class type.

A. Class type to Basic type

B. Class type to Class type

C. Basic Type to Class Type

D. Basic type to Basic type

19. Examine the below syntax for the conversion function to convert from

class type to basic type and identify the correct way

A. operator typename( )
{

}

B. class typename( )

{

…. .....

}

B. operator operator_name( )
{

}

D. Operator class_name( )

{

....

}

20. Can you assess the following statements and identify the statements which are Not True about the destructor?

1. It is invoked when the object goes out of the scope

2. Like a constructor, it can also have parameters

3. It can be virtual

4. It can be declared in the private section

5. It bears the same name as that of the class and precedes the Lambda sign.

A. Only 2, 3, 5

B. Only 2, 3, 4

C. Only 2, 4, 5

D. Only 3, 4, 5

21. Assume class TEST. Identify which of the following statements is/are responsible to invoke the copy constructor?

A. TEST T2(T1)

B. TEST T4 = T1

C. T2 = T1

D. both A and B

22. Examine the below and identify the correct form used to develop the operator function

A. return_type classname :: operator op (arglist)

{

Function body

}

B. return_type operator :: classname op (arglist)

{

Function body

}

C. classname :: operator op (arglist)

{

Function body

}

D. return_type classname :: op (arglist)

{

Function body

}

23. Predict which of the following needs to be used in the source class, for the conversion from A class to any basic type or any other class

A. casting operator

B. Constructor

C. Class

D. operator function

24. How constructors are different from other member functions of the class?

A. Constructor has the same name as the class itself

B. Constructors do not return anything

C. Constructors are automatically called when an object is created

D. All of the mentioned

25. What is the syntax for defining a destructor of class A?

A. A(){}

B. ~A(){}

C. A::A(){}

D. ~A(){};

Long Answer Questions:

1. **What is the purpose/need for using a Constructor? List the special characteristics of the Constructor.**

A constructor is a special member function for the automatic initialization of an object. Whenever an object is created, the special member function, i.e., the constructor will be executed automatically. A constructor function is different from all other member functions in a class because it is used to initialize the variables of whatever instance is being created.

Special Characteristics:

The constructor functions have some special characteristics. These are:

- 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 therefore, and they cannot return values.
- They cannot be inherited, though a derived class can call the base class constructor.
- Like other C++ functions, they can have default arguments.
- Constructors cannot be virtual.
- We cannot refer to their addresses.
- An object with a constructor (or destructor) cannot be used as a member of a union.
- They make 'implicit calls' to the operators new and delete when memory allocation is
- Required

2. Explain the default constructor with syntax and example program.

constructor that accepts no parameters is called the default constructor. if no such constructor is defined, then the compiler supplies a default constructor. Therefore, a statement such as

A a;

invokes the default constructor of the compiler to create the object a.

Example:

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

```
class x
{
        private:
        int a, b;
        public:
        x( )
        {
                a=10, b=20;
        }
        void display()
        {
                cout<<a<<setw(5)<<b<<endl;
        }
};
void main()
{
        x obj1,obj2;
        obj1.display();
        obj2.display();
}
```

Output:

10 20
10 20

3. Explain parameterized constructor with syntax and example program.

A constructor with one or more parameters is called a parameterized constructor. The advantage of a parameterized constructor is that you can initialize each instance of the class to different values.

The below program demonstrates the passing of arguments to the constructor functions.

```
#include<iostream.h>
class integer
{
        int m,n;
        public:
        integer(int, int); //constructor declared
        void display( )
        {
                cout << " m =" <<m << "\n":
                cout << " n =" << n << "\n";
        }
};
integer :: integer(int x, int y)
{
        m=x;
        n=y;
}
void main( )
```

```
{
        integer int1(0,100); //constructor called implicitly
        integer int2=integer(25,75); //constructor called explicitly
        cout<<"Object1"<<"\n";
        int1.display( );
        cout<<"Object2"<<"\n";
        int2.display( );
}
```

Output:

```
Object1
m=0
n=100
Object2
m=25
n=75
```

4. Explain the features of the copy constructor with an example program.

The constructor which creates an object by copying variables from another object is called a copy constructor. The purpose of a copy constructor is to initialize a new instance to the values of an existing instance.

For example, the statement

integer I1;

integer 12(I1);

would define the object I2 and at the same time initialize it to the values of I2. Another form of this

statement is

integer I2 =I1;

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

Let us consider a simple example of constructing and using a copy constructor as shown in below

program.

```
#include <iostream.h>

class code

{
    int id;
    public:
    code() { } //default constructor
    code(int a) //parameterized constructor
    {
    id = a ;
    }
```

```
    code(code & x) //copy constructor
    {
    id=x. id;
    }
    void display(void)
    {
    cout << id;
    }
};

void main( )

{

    code A (I00); // object A is created and initialized
    code B(A); // copy constructor called
    code C = A; //copy constructor called again
    code D; // D is created, not initialized
    D= A; // copy constructor not called
    cout << "\n id of A: " ;
    A.display( );
    cout << "\n id of B:" ;
    B.display();
    cout << "\n id of C:";
    C.display() ;
    cout <<"\n id or D:";
    D.display();
}
```

The output of Program is shown below:

id of A: 100

id of B: 100

id of C: 100

id of D: 100

5. Explain Destructors with syntax and examples.

A destructor, as the name implies is used to destroy the objects that have been created by a constructor. Like a constructor, the destructor is a member function whose name is the same as the class name but is preceded by a tilde(~). For example, the destructor for the class integer can be defined as shown below:

~integer() { }

- A destructor never takes any argument nor does it return any value.
- It will be invoked implicitly by the compiler upon exit from the program (or block or function as the case may be) to clean up storage that is no longer accessible.
- It is a good practice to declare destructors in a program since it releases memory space for future use.

Example program:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class book
{
        int accno;
        char author_name[20];
        char title[20];
        int year;
        char pub_name[20];
        float cost;
        public:
        book()
        {
                accno=101;
                strcpy(author_name,"balaguruswamy");
                strcpy(title,"oops");
                year=1990;
                strcpy(pub_name,"Mc Graw Hill");
                cost=400;
        }
        void display()
        {
                cout<<"accession number:"<<accno<<endl;
                cout<<"author_name :"<<author_name<<endl;
                cout<<"title :"<<title<<endl;
                cout<<"year :"<<year<<endl;
                cout<<"pub_name :"<<pub_name<<endl;
                cout<<"cost :"<<cost<<endl;
        }
};
void main()
{
~book()
{
        cout<<"object is destroyed";
}
void main()
{
        book b;
        clrscr();
        b.display();
        getch();
}
```

6. Survey on operator overloading and rules for overloading operators.

Operator overloading is one of the main exciting features of the C++ language. It is an important technique that has enhanced the power of extensibility of C++. We have stated more than once that C++ tries to make the user-defined data types behave in much the same way as the built-in types.

The process of making an operator exhibit different behaviors in different instances are known as operator overloading

Operator overloading provides a flexible option for the creation of new definitions for most of the C++ operators. We can overload (give additional meaning to) all the C++ operators except the following:

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

7. Analyze constructors with default arguments with an example program?

It is possible to define constructors with default arguments. For example, the constructor complex() can be declared as follows:

```
complex(float real , float imag=0) ;
```

The default value of the argument imag is zero. Then, the statement

```
complex C(5.0) ;
```

assigns the value 5.0 to the real variable and 0.0 to imag (by default). However, the statement

```
complex C(2.0,3.0);
```

assigns 2.0 to real and 3.0 to imag. The actual parameter, when specified, overrides the default value.

As pointed out earlier, the missing arguments must be the trailing ones. It is important to distinguish between the default constructor A::A() and the default argument constructor A::A(int). The default argument constructor can be called with either one argument or no arguments. When called with no arguments, it becomes a default constructor. When both these forms are used in a class,

it causes ambiguity for a statement such as

A a;

The ambiguity is whether to 'call' A::A() or A::A(int = 0).

8. Analyze overloading of Unary operators with an example program.

Let us consider the unary minus - operator. A minus operator when used as a unary, takes just one operand. We know that this operator changes the sign of an operand when applied to a basic data item. We will see here how to overload this operator so that it can be applied to an object in much the same way as is applied to an int or float variable. The unary minus when applied to an object should change the sign of each of its data items

Below program shows how the unary minus operator is overloaded.

```
# include <iostream.h>
class space
{
        int x;
        int y;
        int z;
```

```
        public:
        void getdata(int a, int b, int c);
        void display(void);
        void operator-(); //overload unary minus

};
void space ::getdata (int a, int b, int c)
{
        x = a;
        y = b;
        z = c;
}
void space :: display(void)
{
        cout <<" x ="<< x;
        cout <<" y ="<<y;
        cout <<" z ="<< z;
}
void space :: operator-()
{
        x = -x;
        y = -y;
        z = -z;
}
void main{)
{
        space S;
        S.getdata(10, -20, 30);
        cout<<"S:";
        S.display();
        -S; //activates operator-() function
        cout<<"-S:";
        S.display();
}
```

The above program produces the following output:

S : x=10 y=- 20 z=30

S: x = - 10 y=20 z=-30

9. **Analyze overloading of Binary operators using member function with an example program.**

A statement like,

C = sum(A, B); // functional notation can be used. The functional notation can be replaced by a natural looking expression

C = A + B; // arithmetic notation

by overloading the + operator using an operator+() function. The Program below illustrates how this is accomplished.

```
#include
class complex
{
        float x;
        float y;
        public:
        complex() { }
        complex(float real, float imag)
        {
                x = real;
                y=imag;
        }
        complex operator+(complex);
        void display(void);
};
complex complex :: operator+(complex c)
{
        complex temp;
        temp.x = x + c.x;
        temp.y = y + c.y;
        return(temp);
}
void complex :: display(void)
{
        cout << x << " + " << y << "i\n";
}
void main()
{
        complex C1, C2 , C3 ;
        C1 = complex(2 .5, 3.5) ;
        C2 = complex(l.6, 2.7);
        C3 = C1 + C2;
        cout << "Cl =" ;
        C1.display();
        cout << "C2 = ";
        C2.display();
        cout << "C3=";
        C3.display():
}
```

The output of Program would be:

C1 = 2.5 + 3.5i
C2 = 1.6 + 2.7i
C3 = 4.1 + 6.2i

10. Analyze overloading of Binary operators using friend function with an example program.

The friend function is also used to overload binary operators in the place of member functions, the only difference being that a friend function requires two arguments to be explicitly passed to it, while a member function requires only one.

Example:

```
#include<iostream.h>
class Complex
{
	int num1, num2;
	public:
	void accept()
	{
		cout<<"\n Enter Two Complex Numbers : ";
		cin>>num1>>num2;
	}
	//Overloading '+' operator using Friend function
	friend Complex operator+(Complex c1, Complex c2);
	void display()
	{
		cout<<num1<<"+"<<num2<<"i"<<"\n";
	}
};

Complex operator+(Complex c1, Complex c2)
{
	Complex c;
	c.num1=c1.num1+c2.num1;
	c.num2=c1.num2+c2.num2;
	return(c);
}
int main()
{
	Complex c1,c2, sum; //Created Object of Class Complex i.e c1 and c2
	c1.accept(); //Accepting the values
	c2.accept();
	sum = c1+c2; //Addition of object
	cout<<"\n Entered Values : \n";
	cout<<"\t";
	c1.display(); //Displaying user input values
	cout<<"\t";
	c2.display();
```

```
        cout<<"\n Addition of Real and Imaginary Numbers : \n";
        cout<<"\t";
        sum.display(); //Displaying the addition of real and imaginary numbers
        return 0;
}
```

11. Explain string manipulation using operator overloading.

Strings are also can manipulated using operator overloading function. It permits us to create

our own definitions of operators that can be used to manipulate the strings are very much similar to the decimal numbers. For example, we shall be able to use statements like

strtng3 = string1 + string2;

if(string1 >=string2) string = string1

Example:

```
#include<iostream>
#include<string.h>
class String
{
        char str[20];
        public:
        void accept_string()
        {
                 cout<<"\n Enter String ";
                cin>>str
        }
        void display_string()
        {
                cout<<str;
        }
        String operator+(String x) //Concatenating String
        {
                String s;
                strcat(str,x.str);
                strcpy(s.str,str);
                return s;
        }
};
int main()
{
        String str1, str2, str3;
        str1.accept_string();
        str2.accept_string();
        cout<<"\n ----------------------------------------";
        cout<<"\n\n First String is : ";
        str1.display_string(); //Displaying First String
```

```
    cout<<"\n\n Second String is : ";
    str2.display_string(); //Displaying Second String
    cout<<"\n ----------------------------------------";
    str3=str1+str2; //String is concatenated. Overloaded '+' operator
    cout<<"\n\n Concatenated String is : ";
    str3.display_string();
    return 0;
}
```

12. What are the rules for overloading an operator?

- Only existing operators can be overloaded. New operators cannot be created.
- The overloaded operator must have at least one operand that is of the user-defined type.
- Overloaded operators follow the syntax rules of the original operators. They cannot be
- overridden.
- There are some operators that cannot be overloaded. (See Table 1.)
- We cannot use friend functions to overload certain operators. (See Table 2.) However
- member functions can be used to overload them.

UNIT-III
FUNCTION OVERLOADING AND INHERITANCE

Multiple choice questions

1. What is the correct syntax for function declaration

A. Function_name(argument-list);

B. Return_type function_name(argument-list);

C. Class_name function_name(argument-list);

D. operator_name function-name(argument-list);

2. Which of the following permits function overloading on C++?

A. type

B. number of arguments

C. type & number of arguments

D. different function name

3. What is Overloaded functions in oops ?

A.Functions preceding with virtual keywords.

B. Functions inherited from base class to derived class.

C.Two or more functions having same name but different number of parameters or type.

D. Two or more functions having different name but same type of parameters.

4. How many types of inheritance are provided as OOP feature?**A.4**

B.3

C.2

D.5

5. Which among the following best defines single level inheritance?

A. A base class inheriting a derived class

B. A derived class inheriting a base class

C. A derived class inheriting a nested class

D. A class which gets inherited by 2 classes

6. Which access type data gets derived as private member in derived class:

A. Private

B. Public

C. Protected

D. Protected and Private

7. What is meant by multiple inheritance?

A.Deriving a base class from derived class

B. Deriving a derived class from base class

C. Deriving a derived class from more than one base class

D. None of the mentioned

8. Which symbol indicates derivation of child class from base class ?
A.::
B.:
C.;
D.|

9. A derived class with only one base class is called inheritance.
A.Single
B.Multiple
C.Multilevel
D.Hierarchical

10. **What facilitates Reusability of the code achieved in OOPs**
A.Dynamic binding
B.Encapsulation
C.Inheritance
D.Message Passing

11. Which of the below is the error code?

A. **int test(int a) { }**
double test(int b){ }

B. int exam(int a) { }
double test(int b){ }

C. int test(int a) { }
double test(float b){ }

D. int test(int a) { }
double test(float b){ }

12. Which among the following is true , if a base class is inherited in protectedaccess mode?
A.Public and Protected members of base class becomes protected members of derived class
B. Only protected members become protected members of derived class
C. Private, Protected and Public all members of base, become private of derivedclass
D. Only private members of base, become private of derived class

13. Which is the appropriate way to declare members in the class which are notintendedto be inherited
A. Public members
B. Protected members
C. Private members

D. Private or Protected members

14. While inheriting a class, if no access mode is specified, then which among the following is true? (in C++)

A. It gets inherited publicly by default
B. It gets inherited protected by default
C. It gets inherited privately by default
D. It is not possible

15. Which among the following is correct for multiple inheritance?

A. class student{ }; class test{ };
class topper : public student, public test{ };

B. class student{ }; class test{ };
class topper: public student{ };
C. class student{ };
class test : public student{ };
D.class student{ }; class test{ };
class topper:public topper{};

16. Which among the following is correct for hierarchical inheritance?

A. Two base classes can be used to be derived into one single class
B. Two or more classes can be derived into one class
C. One base class can be derived into two derived classes or more
D. One base class can be derived into only 2 classes

17. Which is the correct syntax of inheritance?

A. class derived_classname : base_classname
{
/*define class body*/
};
B. class base_classname : derived_classname
{
/*define class body*/
};

C. class derived_classname : visibility_mode base_classname
{
/*define class body*/
};

D. class base_classname :visibility_mode derived_classname
{
/*define class body*/
};

18. If a derived class object is created, which constructor is called first?

A. Base class constructor

B. Derived class constructor

C. Depends on how we call the object

D. Not possible

19. How can you make the private members inheritable?

A. By making their visibility mode as public only

B. By making their visibility mode as protected only

C. By making their visibility mode as private in derived class

D. It can be done both by making the visibility mode public or protected

20. Which kind of inheritance supports a child class to inherit properties from morethan one base class.

A. Hierarchical

B. Hybrid

C. Multilevel

D. Multiple

21. Which among the below statements is not correct about private inheritance

A. The private members of a base class cannot be inherited to the derived class.

B. The protected members of a base class become private in a derived class.

C. The protected members of a base class stay protected in a derived class.

D. The public members of a base class become the private members of the derivedclass.

22. Class X, class Y and class Z are derived from class BASE. This is

A. Multiple

B. Multilevel

C. Hierarchical

D. Single

23. Which is the appropriate statement among the below mentioned?

A. Base class inherits some or all of the properties of the derived class.

B. Derived class inherits some or all of the properties of the base class.

C. Virtual class inherits some or all of the properties of the abstact class.

D. Base class inherits some or all of the properties of the virtualclass.

24. Which among the below inheritance type , has the constructors executed in theorder of inheritance.

A. Multipath

B. Multiple

C. Multilevel

D.Hierarchical

25. What will be the order of execution of base class constructors in the followingmethod of inheritance?

class P: public M, public N {….};

A. M(); N(); P();
B. N(); M(); P();
C. P(); M(); N();
D. M(); P(); N();

4Marks Question and Answers

1. **What is function overloading? What are the different ways to overload a function? Explain with an example program.**

Function overloading means two or more functions having same name, but differ in the number of arguments or data type of arguments.

Different ways to overload a function

1. By changing number of Arguments.
2. By having different types of argument.

Function overloading: different number of arguments

In this type of function overloading we define two functions with same names but different number of parameters of the same type. For example, in the below mentioned program we have made two sum() functions to return sum of two and three integers.

```
int sum (int x, int y)
{
        cout << x+y;
}
// second overloaded defintion
int sum(int x, int y, int z)
{
        cout << x+y+z;
}
```

Here sum() function is said to overloaded, as it has two definition, one which accepts two arguments and another which accepts three arguments. Which sum() function will be called, depends on the number of arguments.

```
int main()
{
         sum (10, 20); // sum() with 2 parameter will be called
         sum(10, 20, 30); //sum() with 3 parameter will be called
```

```
}
```

Function overloading: different type of arguments

In this type of overloading we define two or more functions with same name and same number of parameters, but the type of parameter is different. For example in this program, we have two sum() function, first one gets two integer arguments and second one gets two double arguments.

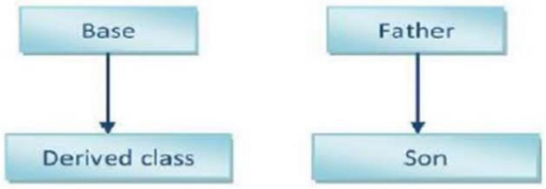
```
int sum(int x, int y) // first definition
{
        cout<< x+y;
}
double sum(double x, double y) // second overloaded definition
{
        cout << x+y;
}
```

2. What is inheritance? Discuss any three types with examples.

The mechanism of deriving a new class from an old one is called inheritance. [Base Class: It is the class whose properties are inherited by another class. It is also called Super Class. Derived Class: It is the class that inherits properties from base class or classes. It is also called Sub Class.]

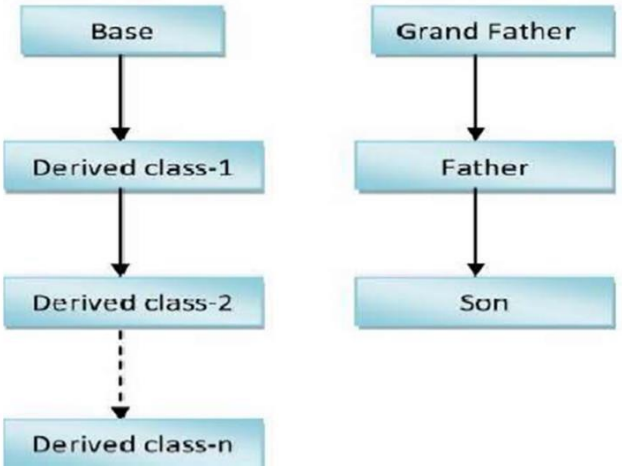
SINGLE INHERITANCE

If a class is derived from a single base class, it is called as single inheritance. In other words a derived class with only one base class, is called singleinheritance.



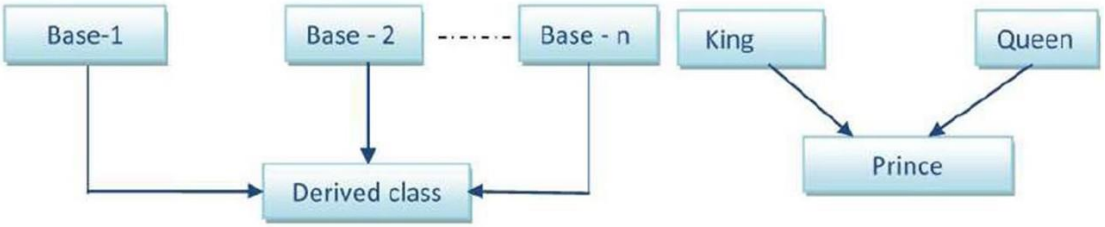
MULTILEVEL INHERITANCE

The mechanism of deriving a class from another 'derived class' is known as multilevel inheritance. The classes can also be derived from the classes that are already derived. This type of inheritance is called multilevel inheritance.



MULTIPLE INHERITANCE

If a class is derived from more than one base class, it is known as multiple inheritance. Multiple inheritance allows us to combine the features of several existing classes as a starting point for defining new classes.



3. Differentiate between public and protected visibility mode by giving suitable examples for each.

Public Inheritance

This is the most used inheritance mode. In this

- The public members of a base class become public members of the derived class.
- The private members of a base class cannot be inherited to the derived class.
- The protected members of a base class stay protected in a derived class.Example:

```
#include <iostream.h>
class A {
        public:
           int x;

        protected:
           int y;
```

```
        private:
            int z;
};
// Class B will inherit Class A and using Public Visibility mode
class B : public A
{
};

int main()
{
    B b;
    // x is public and it will remain public
    // so its value will be printed
    cout << b.x << endl;

    // y is protected and it will remain protected
    // so it will give visibility error
    cout << b.y << endl;

   // z is not accessible from B as
    // z is private and it will remain private
    // so it will give visibility error
    cout << b.z << endl;
};
```

Protected Inheritance

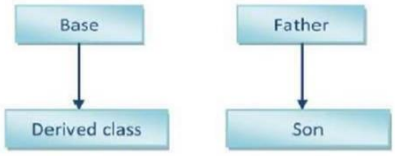
- The public members of a base class become protected in a derived class.
- The private members of a base class cannot be inherited to the derived class.
- The protected members of a base class stay protected in a derived class.Example:

```
#include <iostream.h>
class A {
        public:
            int x;

        protected:
            int y;

        private:
            int z;
};
// Class B will inherit Class A & using Protected Visibility mode
```

```
class B : protected A
{
};

int main()
{
   B b;

   // x is public and it will become protected
   // so it will give visibility error
   cout << b.x << endl;

   // y is protected and it will remain protected
   // so it will give visibility error
   cout << b.y << endl;

   // z is not accessible from B as
   // z is private and it will remain private
   // so it will give visibility error
   cout << b.z << endl;
};
```

4. What is Single inheritance? Explain with an example program.

If a class is derived from a single base class, it is called as single inheritance. In other words a derived class with only one base class, is called single inheritance.



Example:

```
#include<iostream.h>
#include<conio.h>
class rectangle
{
      protected:
      float length, breadth, area;
      public:
      void accept()
      {
```

```
            cout<<"Enter the length:";
            cin>>length;
            cout<<"Enter the breadth:";
            cin>>breadth;
      }
      void calculate_area()
      {
            area=length*breadth;
            cout<<"The area of rectangle:"<<area<<endl;
      }
};

class cuboid : public rectangle
{
            float height, volume;
            public:
            void  getdata()
            {
                  accept();
                  cout<<"Enter the height:";
                  cin>>height;
            }
            void calculate_volume()
            {
                  volume=length*breadth*height;
                  cout<<"The volume of cuboid is:"<<volume<<endl;
            }
};

void main()
{

      cuboid c;
      clrscr();
      c.getdata();
      c.calculate_area();
      c.calculate_volume();
      getch();
}
```

Output:

Enter Length: 5

Enter breadth :4

Enter height: 5

The area of rectangle: 20

The volume of cuboid is: 100

5. How does the visibility mode control the access of members in the derived class? Explain.

Table showing all the Visibility Modes

Base class	Derived Class		
	Public mode	Private Mode	Protected Mode
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

Public Inheritance

This is the most used inheritance mode. In this

- The public members of a base class become public members of the derived class.
- The private members of a base class cannot be inherited to the derived class.
- The protected members of a base class stay protected in a derived class.

Private Inheritance

- The public members of a base class become the private members of the derived class.
- The private members of a base class cannot be inherited to the derived class.
- The protected members of a base class become private in a derived class.

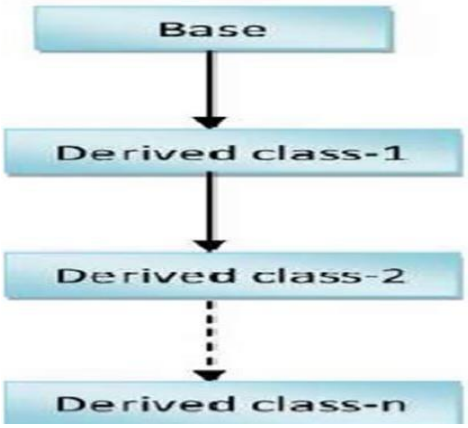
Protected Inheritance

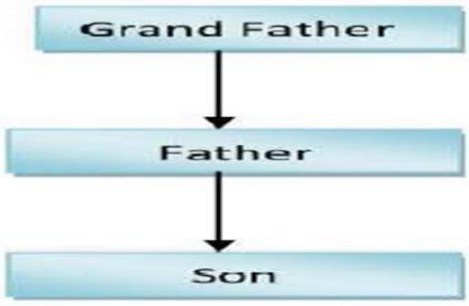
- The public members of a base class become protected in a derived class.
- The private members of a base class cannot be inherited to the derived class.
- The protected members of a base class stay protected in a derived class.

6. Explain multilevel inheritance with an example program.

The mechanism of deriving a class from another 'derived class' is known as multilevel inheritance. The classes can also be derived from the classes that are already derived. This type of inheritance is called multilevel inheritance.

(The class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. The class B is known as intermediate base class since it provides a link for the inheritance between A and C. The chain ABC is known as inheritancepath.)





Example:

Example:

```
#include <iostream.h>
class student
{
        protected: int roll_number;
        public: void getnumber(int);
        void putnumber( );
};
void student : :getnumber(int a)
{
         Roll_number= a;
}
void student :: putnumber()
{
         cout << “Roll number=”<<roll_number;
}

class test : public student
{
protected:
        float sub1;
        float sub2;
        public:
        void get_marks(float, float);
        void put_marks( );
};
void test :: get_marks(float x, float y)
{
        sub1= x;
        sub2 = y;
}
```

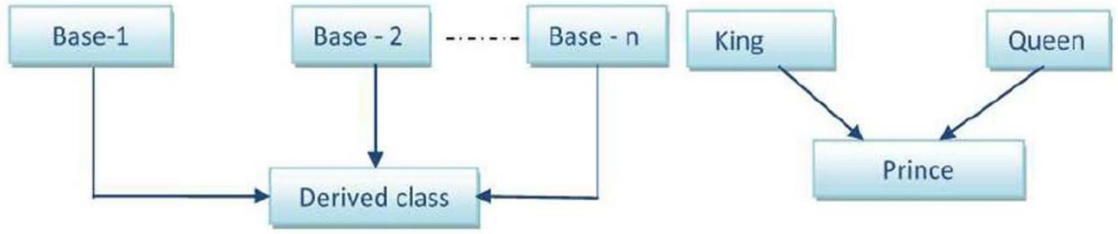
```
void test : : put_marks()
{
        cout << “Marks in Sub1=” << sub1<< “\n” ;
        cout << “Marks in Sub2 =” << sub2 << “\n”;
}

class result : public test
{
        float total;
        public:
        void display( );
};

void result :: display( )
{
         total = sub1+sub2;
         putnunber();
         put_marks();
         cout <<"Total =”<<total ;
}
void main()
{
         result student1; // student1 created
         student1.getnumber(101);
         student1.get_marks(75.0, 59.5);
         studentl.display();
}
```

output:
Roll Humber: 101
Marks in Sub1 = 75
Marks in Sub2 = 59 .5
Total = 134.5

7. Describe the syntax and example of multiple inheritance. When do we use such inheritance?

If a class is derived from more than one base class, it is known as multiple inheritance.

Multiple inheritance allows us to combine the features of several existing classes as a starting point for defining new classes.



Syntax:

```
class new_class_name: visibility baseclass1_name,visibility baseclass2_name…
{
        body of new_class
};
```

Example:

```
class M
{
      protected:
      int m;
      public:
      void get_ m(int);
};
class N
{
      protected:
      int n;
      public:
      void get_n(int);
};
```

class P : public M, public N
{

public:
void display(void);

} ;

```
void M :: get_m(int x)
{
        m =x;
}
void N :: get_n(int y)
{
        n =y;
```

```
}
void P :: display()
{
        cout << "m=" << m<< "\n";
        cout << "n=" << n << "\n";
        cout << "m*n=" << m*n << "\n";
}
void main( )
{
        P p;
        p .get_m(10) ;
        p.get_n(20) ;
        p.display() ;
}
```

Output:
m =10
n = 20
m*n = 200

8. When does ambiguity arise in multiple inheritance? How can one resolve it?

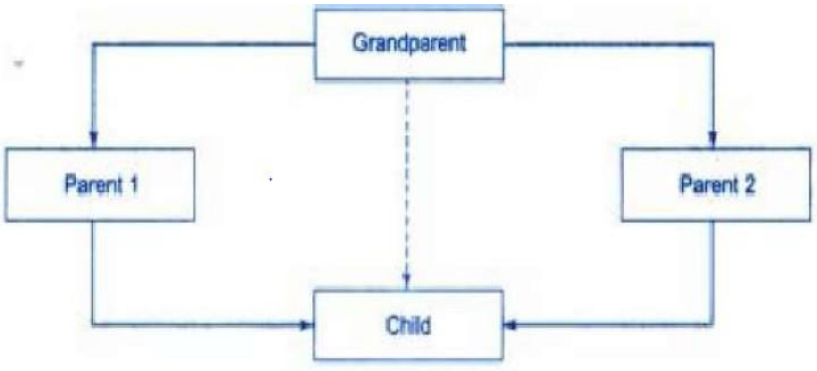
Ambiguity arise in multiple inheritance when a function with the same name appears in more than one base class.
We can solve this problem by defining a named instance within the derived class, using the class resolution operator(::) with the function name as shown below:

```
class M
{
      public:
      void display(void)
      {
      cout<<"Class  M";
      }
} ;
class N
{
      public:
      void display(void)
      {
      cout << "Class N\n";
```

```
        }
};

class P : public M, public N
{
        public:
        void display(void)
        {
        M::display();
        }
} ;
void  main()
{
P p;
p.display() ;
}
```

9. What are virtual base classes? What is their significance?

An ambiguity can arise when several paths exist to a class from the same base class. This means that a child class could have duplicate sets of members inherited from a single base class.



Consider the situation that The 'child' has two direct base classes 'parent1' and 'parent2' which themselves have a common base class 'grandparent'. The 'child' inherits the properties of 'grandparent' via two separate paths. It can also inherit directly as shown by the broken line. The 'grandparent' is sometimes referred to as indirect base class.

Inheritance by the 'child' as shown in above figure might pose some problems. All the public and protected members of 'grandparent' are inherited into 'child' twice, first via 'parent1' and again via 'parent2'. This means, 'child' would have duplicate sets of the members inherited from 'grandparent'. This introduces ambiguity and should be avoided.

The duplication of inherited members due to these multiple paths can be avoided by making the common base class (ancestor class) as virtual base class while declaring the direct or intermediate base classes as shown below.

```
class A // grandparent
{
……..
…….
};
class B1 : virtual public A        // parent1
{
……
……
} ;
class B2 : public virtual A        // parent2
{
…….
…….
};
class C : public B1, public B2 //child
{
…….. // only one copy of A
……. // will be Inherited
};
```

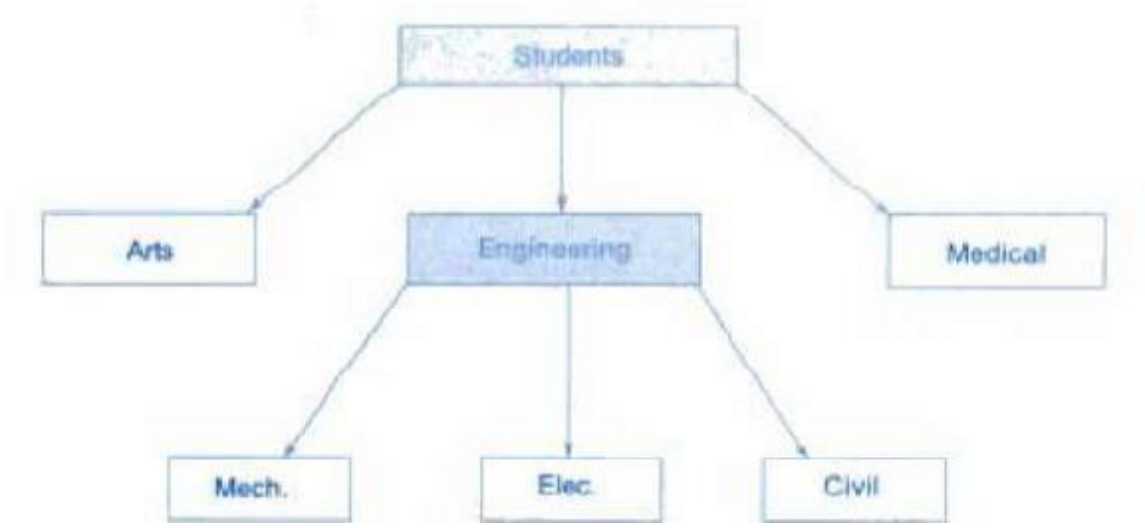
When a class is made a virtual base class, C++ takes necessary care to see that only one copy of that class is inherited, regardless of how many inheritance paths exist between the virtual base class and a derived class.

10. What is hierarchical inheritance? Give two examples.

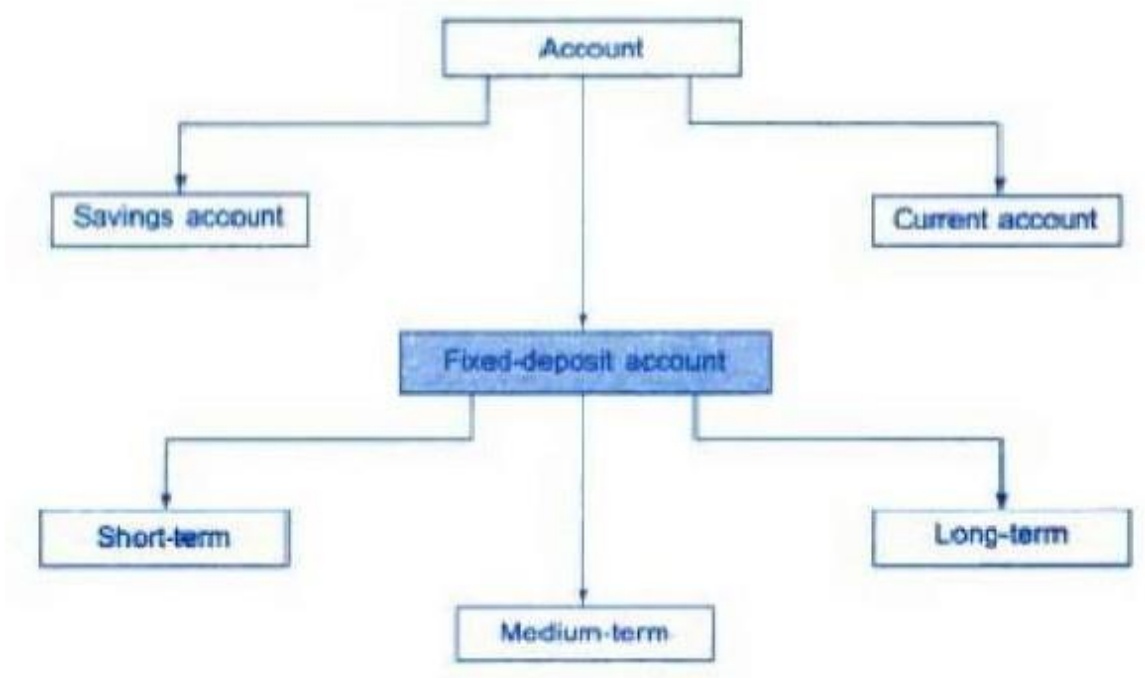
If a number of classes are derived from a single base class, it is called as hierarchical inheritance.

A subclass can be constructed by inheriting the properties of the base class. A subclass can serve as a base class for the lower level classes and so on.

Example:1 Classification of students in a university.



Example:2 Classification of accounts in a commercial bank.



11. Explain the concept of abstract base classes with example

An abstract class is one that is not used to create objects. An abstract class is designed only to act as a base class (to be inherited by other classes).

It is a design concept in program development and provides a base upon which other classes may be built.

the class A is an abstract class since it is not used to create any objects.

However, one should remember that an abstract class must have a pure virtual function.

- **The general form of using Abstract base class is shown below:**

```
class base //abstract base class
{
public: virtual void xyz()=0; //pure virtual function
};
class derived : public base
{
public: void xyz()
{
Cout<<"Abstract base class with pure virtual function";
}
};
```

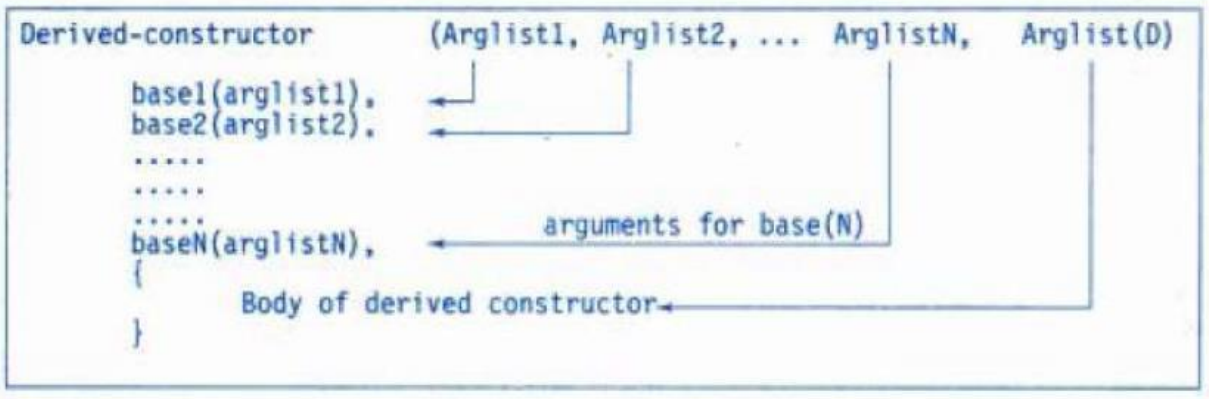
```
Void main()
{
base b ; //Error :no object for abstract class
derived ob;
ob.xyz();
}
```

Output:

Abstract base class with pure virtual function

12. Write a note on constructors in derived class.

The general form of defining a derived constructor is:



The header line of derived-constructor function contains two parts separated by a colon(:).The first part provides the declaration of the arguments that are passed to the derived-constructor and the second part lists the function calls to the base constructors. base1(arglistl), base2(arglist2) ... are function calls to base constructors base1(), base2(), ... and therefore arglistl, arglist2, ... etc. represent the actual parameters that are passed to the base constructors. Arglist1 through ArglistN are the argument declarations for base constructors base1 through baseN. ArglistD provides the parameters that are necessary to initialize the members of the derived class.

D(i nt a1 , int a2, float b1 , float b2, int d1) : A(a1, a2), B(b1, b2) // (call to constructor A first then constructor B)
{
d = d1; // executes its own body
}
A(al, a2) invokes the base constructor A() and B(b1, b2) invokes another base constructor B(). The constructor D() supplies the values for these four arguments. In addition, it has one argument of its own. The constructor D() hasa total of five arguments. D() may be invoked as follows:
D objD (5, 12, 2.5, 7.54, 30);
These values are assigned to various parameters by the constructor D() as follows:
5 -> a1
12 -> a2
2.5 -> b1
7.54 -
> b2
30 -
>d1

QUESTION BANK IV UNIT

Multiple Choice questions

1. which symbol is used to declare the pointer variable.

A. @ **B. *** C. # D. ()

2. Polymorphism means

A. **Many forms** C. Hiding data

B. Only one form D. None of them

3. Compile time polymorphism is also known as.....

A. late binding B. Dynamic binding

B. **Static binding** D. Fixed binding

4. The pointers which are not initialized in a program are called ……..

A. void pointers C. this pointer

B. **Null pointers** D. base pointer

5. which pointers are known as generic pointers.

A. This pointer **C. Void pointer**

B. Null pointer D. Base pointer

6. Which pointer is useful in creating objects at run time.

A. void pointer C. this pointer

B. null pointer D**. object pointer**

7. A pure virtual function is initialized by

A.**0** B. 1 C. 2 D. 3

8. Which of the following keyword is used before a function in a base class to be overridden in derived class in C++

A. Override C. void

B. **Virtual** D. none

9. Which type of function among the following shows polymorphism?

A. Inline function **C. Virtual function**

B. Undefined functions D. Class member functions

10. A pointer can be initialized with

A. Null C. Address of an object of same type

B. Zero **D. All of the above**

11. Examine the below instructions and identify the correct initialization statement in pointers?

A. ptr=&a; B. ptr=a; C. ptr=*a D. ptr=@a;

12. can you analyze the following statements about virtual functions and state Whether True or False.

i) A virtual function, equated to zero is called pure virtual function.

ii) A class containing pure virtual function is called an abstract class

A. True, True C. False, True

B. True, False D. False, False

13. Can you judge which is the correct declaration of pure virtual function in C++

A. virtual void func = 0; C. virtual void func(){0};

B. **virtual void func() = 0;** D. void func() = 0;

14. Can you identify which pointer denotes the object calling the member function?

A. Variable pointer C. Null pointer

B. this pointer D. Zero pointer

15. Select the mechanism which means that an object is bound to its function call at the compile time.

A. Inheritance B. Message Passing

C. Compile-time polymorphism D. Function Overriding

16. Judge on which of the following statements are true according virtual functions?

i) The virtual functions must be members of some class.

ii) A virtual function can be a friend of another class

A. Only i **B. Both i and ii** C. Both are wrong D. Only ii

17. Select the polymorphism mechanism involving Function overriding

A. **run-time** B. compile-time C. Static D. Object

18. Select the function declared in a base class that has no definition relative to the base class.

A. member function C. virtual function

B. **pure virtual function** D. pure function

19. Suppose a polymorphism is supported in C++ with the help of virtual function, then it is termed as?

A. **Dynamic binding** C. early binding

B. run time D. static

20. In compile-time polymorphism, a compiler is able to select the appropriate function for a particular call at the compile time itself, which is known as ……

A. early binding C. static linking

B. static binding **D. All of the above**

21. Can you assess which among the following best describes polymorphism?

A. It is the ability for a message/data to be processed in more than one form

B. It is the ability for a message/data to be processed in only 1form

C. C. It is the ability for many messages/data to be processed in one way

D. It is the ability for undefined message/data to be processed in at least one way

22. Assess which among these is true about abstract base class?

A. It can have One or more pure virtual member functions

B. It can have One or more child classes

C. It can have One or more normal functions

D. It can have one or more objects

23. Suppose derived class defines same function as defined in its base class, then such a situation is termed as?

A. Function overloading C. Member function

B. Inheritance **D. Function Overriding**

24. Examine the given code and Choose the right option from below? string *x, y;

A. x is a pointer to a string, y is a string

B. y is a pointer to a string, x is a string

C. both x and y are pointer to string types

D. both x and y are string to pointer types

Long Answer questions

1. Make use of the concept of pointers and show how to declare and initialize a pointer?

We can declare a pointer variable similar to other variables in C++. Like C, the declaration is based on the data type of the variable it points to. The declaration of o pointer variable takes the following form:

data-type *pointer-variable;

Here, *pointer-variable* is the name of the pointer, and the *data_type* refers to one of the valid C++ data types, such as int., char, float, and so on. The *data_type* is followed by an asterisk(*) symbol known **as indirection operator or dereferencing operator**, which distinguishes a pointer variable from other variables to the compiler.

We can locate asterisk (*) immediately before the pointer variable, or between the data type and the pointer variable, or immediately after the data type. It does not cause any effect in the execution process.

The asterisk you used to declare a pointer is the same asterisk that you use for multiplication.

However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration –

```
int    *ip;    // pointer to an integer double
*dp;           // pointer to a double float *fp;
// pointer to a float

 char  *ch    // pointer to character
```

2. Define a pointer? Write the syntax of using the pointer.

A pointer is a variable whose value is the address of another variable. Like any variable or constant, you must declare a pointer before you can work with it. We know that a pointer is a derived data type that refers to another data variable by storing the variable's memory address rather than data.

We can declare a pointer variable similar to other variables in C++. Like C, the declaration is based on the data type of the variable it points to. The declaration of o pointer variable takes the following form:

data-type *pointer-variable;

Here, *pointer-variable* is the name of the pointer, and the *data_type* refers to one of the valid C++ data types, such as int., char, float, and so on. The *data_type* is followed by an asterisk(*) symbol known **as indirection operator or dereferencing operator**, which distinguishes a pointer variable from other variables to the compiler.

3. Develop a program to explain the concept of pointers to object.

```
#include<iostream.h>
class item
{
int code; float price; publ ic:
void getdata(int a, float b)
{
code = a; price = b;
void show(void)
{

cout << “code : " << code <<”\n”;
cout<< “Price: " << price << "\n”;

};
const int size=2;

cout << “code : " << code <<”\n”; cout<< “Price: " << price << "\n”;
}

void main()
```

```
{
item *p =new item [size]; item *d=p;
int x, i; float y;
for(i=0; i<size; i ++)
{

cout << “Input code and price for item” << i +1; cin>>x>>y;
p->getdata(x,y); P++
for(i=0; i<size; i ++)
{
cout<<”Item:”<<i+1; d->show();
d++;
}
}
```

The output of Program will be:

Input code and price for item1: 40 500 Input code and price for item2: 50 600 Item: 1

Code : 40

Price: 500

ltem:2 Code : 50

Price: 600

4. Analyze on what does ‘this’ pointer points to?

Every object in C++ has access to its own address through an important pointer called this pointer. The’ this’ pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object.

Friend functions do not have a 'this' pointer, because friends are not members of a class. Only member functions have a 'this' pointer.

The 'this' pointer holds the address of current object, in simple words you can say that this pointer points to the current object of the class.

One of the important applications of using this pointer is to return the object it points. For example, the statement,

return *this;

inside a member function will return the object that calls the function.

5. Elaborate on how polymorphism can be achieved at i) compile time ii) run time?

COMPILE-TIME POLYMORPHISM

- Means that an object is bound to its function call at the compile time.
- No ambiguity at the compile time about which function is to be linked to a particular function's call.
- Also known as early binding or static binding or static linking.

It is achieved in 2 ways:

- Function overloading and operator overloading.

RUN TIME POLYMORPHISM

- The uses of scope resolution operator in resolving ambiguity works fine in smaller programs.

b; b.display(); b.A::display();

- But in large applications, it would be tedious to modify the code which implements static binding.
- To overcome this problem, run-time polymorphism is very useful.
- It links the function call to a particular class at run-time.
- Thus, it is not known which function will be invoked till an object actually makes the function call during the program's execution.
- This process is known as late binding or dynamic binding.
- Run-time polymorphism is achieved through virtual functions.

6. Elaborate on function overriding in c++ with an example.

```
#include <iostream.h> class BaseClass
{
public: void disp()
{

cout<<"Function of Parent Class";
}
};
class DerivedClass: public BaseClass
{
public: void disp()
{

 cout<<"Function of Child Class";



};
void main() {
DerivedClass obj ;
obj.disp();

}
```

Output: Function of Child Class

- A virtual function a member function which is declared within base class and is re-defined (Overriden) by derived class.
- When we refer to a derived class object using a pointer or a reference to the base class, we can call a virtual function for that object and execute the derived class's version of the function.
- Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.
- They are mainly used to achieve run-time polymorphism.
- Functions are declared with a virtual keyword in base class.
- The resolving of function call is done at Run-time. The keyword virtual is used for defining virtual function.

7. Define the meaning and need of virtual function?

There is a necessity to use the single pointer to refer to all the objects of the different classes. So, we create the pointer to the base class that refers to all the derived objects. But, when base class pointer contains the address of the derived class object, always executes the base class function. This issue can only be resolved by using the 'virtual' function. When the function is made virtual, C++ determines which function is to be invoked at the runtime based on the type of the object pointed by the base class pointer.

8. Can you identify and determine any four rules for virtual functions.

1. The virtual functions must be members of some class.
2. They cannot be static members.
3. They are accessed by using object pointers.
4. A virtual function can be a friend of another class.
5. A virtual function in a base class must be defined, even though it may not be used.
6. The prototypes of the base class version of a virtual function and all the derived class versions must be identical. If two functions with the same name have different prototypes, C++ considers them as overloaded functions, and the virtual function mechanism is ignored.
7. We cannot have virtual constructors, but we can have virtual destructors.

8. While a base pointer can point to any type of the derived object, the reverse is not true. That is to say, we cannot use a pointer to a derived class to access an object of the base type.
9. When a base pointer points to a derived class, incrementing or decrementing it will not make it to point to the next object of the derived class. It is incremented or decremented only relative to its base type. Therefore, we should not use this method to move the pointer to the next object.
10. If a virtual function is defined in the base class, it need not be necessarily redefined in the derived class. In such cases, calls will invoke the base function.

9. What is the motive of making a virtual function 'pure'? Explain.

- if we have not defined any object of class media and therefore the function display() in the base class has been defined 'empty'.
- It can be also written as: virtual void display() =0;
- Such functions are called pure virtual functions.
- A pure virtual function is a function declared in a base class that has no definition relative to the base class.
- In such cases, the compiler requires each derived class to either define the function or redeclare it as a pure virtual function.
- Remember that a class containing pure virtual functions cannot be used to declare any objects or its own.

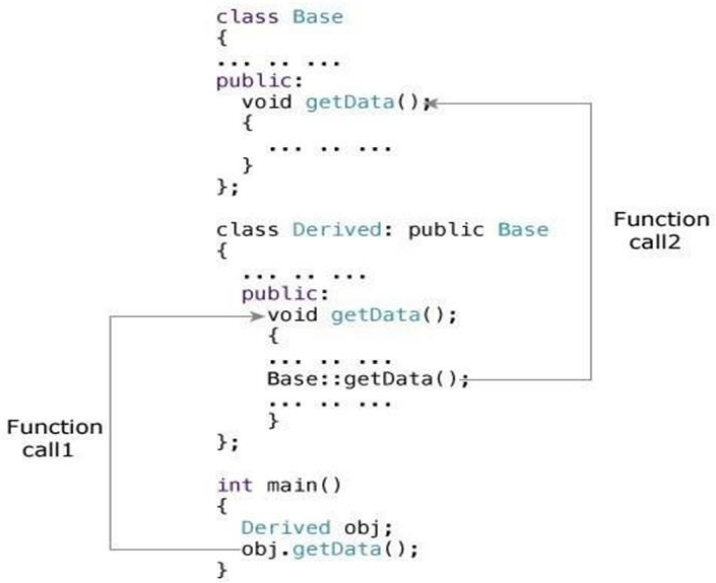
10. We cannot use a pointer to a derived class to access an object of the base type. Elaborate.

While a base pointer can point to any type of the derived object, the reverse is not true. That is to say, we cannot use a pointer to a derived class to access an object of the base type.

When a base pointer points to a derived class, incrementing or decrementing it will not make it to point to the next object of the derived class. It is incremented or decremented only relative to its base type. Therefore, we should not use this method to move the pointer to the next object.

11.Plan on how to access the overridden function in the base class from the derived class?

- To access the overridden function of the base class from the derived class, scope resolution operator :: is used. For example,
- If you want to access getData() function of the base class, you can use the following statement in the derived class.
- Base::getData();



12. Analyze on how to use the pointers to objects of the derived class?

We can use pointers not only to the base object but also to the objects of derived classes.

Pointers to objects of a base class are type-compatible with pointers to objects of derived class.

Therefore, a single class pointer variable can be made available to point to objects belonging to different classes.

For example, if B is a base class and D is a derived class from B, then a pointer declared as a pointer to B can also be a pointer to D.

B *cptr; B b;

D d;

cptr =&b;

We can make cptr to point the object d as follows:

cptr=&d; // this is valid because d is an object derived from B.

Question Bank UNIT V

Multiple Choice Questions

1. What consists in the family that is created by using a template?

A. Classes and functions B. Structures C. Variables D. Keywords

2. Which of the following can be handled by family of classes generated ,using a template

A. Different data types C. Same data types

B. Different variables D. Different identifiers

3. Which is the most significant feature that arises by using template classes?

A. Code readability **C. Code reusability**

B. Ease in coding D. Modularity in code

4. What can be passed by non-type template parameters during compile- time?

A. int C. float

B. constant expression D. char

5. What is a template?

A. A template is used to manipulate the class

B.A template is a formula for creating a generic class

C.A template is used for creating the attributes

D.A template is a data type

6. How many types of templates are there in C++?

A. 1 **B. 2** C. 3 D. 4

7. What is an exception?

A. Problem arising during compile time C. Problem in syntax

B. Problem arising during runtime D. Problem in IDE

8. What is a function template?

A. Creating a function without having to specify the exact type.

B. Creating a function with an exact type.

C. Creating a variable with exact type.

D. None of the above.

9. Which are the two blocks that are used to check error and handle the error?

A. Try and catch C. Do and while

B. Trying and catching D. TryDo and Check

10. Which keyword can be used in a template ?

A. class **C. Both A and B**

B. typename D. Function

11. How many kinds of exceptions are present in C++

A. 4 **B. 2** C. 3 D. 1

12. Which of the below is not a keyword in exception handling mechanism?

A. Try **B. stop** C. catch D. Throw

13. What can a template class have?

A. It can have More than one generic data type

B. It can have Only one generic data type

C. It can have At most two data types

D. It can have Only generic type of integers and not characters

14. How to declare a template?

A. template ( )
C. template < >
B. template{ }
D. template[ ]

15. If template class is defined, is it necessary to use different types of data for each call?

A. No, not necessary

B. No, but at least two types must be there

C. Yes, to make proper use of template

D. Yes, for code efficiency

16. What may be the name of the parameter that the template should take?

A. same as variable
C. same as function
B. same as class
D. same as template

17. Which is the statement used to catch all the exceptions.

A. catch{…}
C. catch(…)
B. catch[…]
D. catch(void x)

18. Which is the exception type for Error such as “out-of-range index”.

A. Synchronous B. Asynchronous C. type-safe D. functional

19. Which block must be immediate after the try block without any code between them.

A. Throw B. Stop **C. Catch** D. type

20. Choose which is the correct syntax of throw statement from below

A. Throw( ) B. throw[exception] C. throw { } **D. throw(exception)**

21. Which block can have the code of statements which may cause abnormal termination of the program

A. Try C. Finally

B. Catch D. Throw

22. Which of the following statements are true?

1. Catch should always be placed last in the list of exception handlers.

2. C++ compiler does not check if any exception is caught or not.

A. Only 1 **B. both 1 and 2** C. Only 2 D. Both are wrong

23. What is the validity of template parameters?

A. inside that block only C. whole program

B. inside the class D. any of the mentioned

24. Why do we need to handle exceptions?

A. To avoid syntax errors **C. To prevent abnormal termination of program**

B. To avoid semantic errors D. To save memory

25.Can you predict when can an exception arise?

A. When Input is fixed **C. When Input given is invalid**

B. When Input is some constant value of program D. When Input is valid

Long Answer questions

1. Plan on how to declare the class templates? Explain with syntax and example.

Syntax:

template <class T>

class className

{

... .

public: T a;

T someOperation(T arg);

... .

};

The class template definition is similar to an ordinary class definition except the prefix template<class T> and the use of type T.

• In the above declaration, T is the template argument which is a placeholder for the data type used.

• Inside the class body, a member variable a and a member function someOperation() are both of type T.

• A class created from a class template is called template class. The syntax for defining an object of a template class is:

classname <data type> objectname(arglist);

This process of creating a specific class from a class template is called instantiation.

For example,

className<int> classObject;

className<float> classObject;

className<string> classObject;

Example:

Program to add, subtract, multiply and divide two numbers using class template.

```
#include <iostream.h>
template <class T>
class Calculator
{
private: T num1, num2;
public: Calculator(T n1, T n2)
{
num1 = n1;
num2 = n2;
}
void displayResult()
{
cout << "Numbers are: " << num1 << " and " << num2 << endl;
cout << "Addition is: " << add() << endl;
cout << "Subtraction is: " << subtract() << endl;
cout << "Product is: " << multiply() << endl;
cout << "Division is: " << divide() << endl;
}
T add() { return num1 + num2; }
T subtract() { return num1 - num2; }
T multiply() { return num1 * num2; }
T divide() { return num1 / num2; }
};
void main()
{
Calculator<int> intCalc(6, 2);
Calculator<float> floatCalc(2.4, 3.2);
cout << "Int results:" << endl;
```

```
intCalc.displayResult();
cout << endl << "Float results:" << endl;
floatCalc.displayResult();
}
```

Output:

Int results:

Numbers are: 6 and 2

Addition is: 8

Subtraction is 4

Product is: 12

Division is: 3

Float Results

Numbers are 2.4 and 3.2

Addition is: 5.6

Subtraction is :-0.8

Product is: 7.68

Division is: 0.75

2. Plan on how to define the function templates? Explain with syntax and example.

• A function template starts with the keyword template followed by template parameter/s inside < > which is followed by function declaration.

• In the above code, T is a template argument that accepts different data types (int, float), and class is a keyword.

• You can also use keyword typename instead of class in the above example.

• When, an argument of a data type is passed to someFunction( ), compiler generates a new version of someFunction() for the given data type.

Example 1: Function Template to find the largest number

• Program to display largest among two numbers using function templates.

// If two characters are passed to function template, character with larger ASCII value is displayed.

```
#include <iostream.h>
template <class T>
T Large(T n1, T n2)
{
return (n1 > n2) ? n1 : n2;
}
void main()
{
int i1, i2;
float f1, f2;
char c1, c2;
cout << "Enter two integers:\n";
cin >> i1 >> i2;
cout << Large(i1, i2) <<" is larger." << endl;
cout << "\nEnter two floating-point numbers:\n";
cin >> f1 >> f2;
cout << Large(f1, f2) <<" is larger." << endl;
cout << "\nEnter two characters:\n";
cin >> c1 >> c2;
cout << Large(c1, c2) << " has larger ASCII value.“;
}
```

Output

Enter two integers: 5 10

10 is larger.

Enter two floating-point numbers: 12.4 10.2

12.4 is larger.

Enter two characters: z Z

z has larger ASCII value.

Example 2:

• Program to swap data using function templates.

```
#include <iostream.h>
template <typename T>
void Swap(T &n1, T &n2)
{
T temp;
temp = n1;
n1 = n2;
n2 = temp;
}
void main()
{
int i1 = 1, i2 = 2;
float f1 = 1.1, f2 = 2.2;
char c1 = 'a', c2 = 'b';
cout << "Before passing data to function template.\n";
cout << "i1 = " << i1 << "\ni2 = " << i2;
cout << "\nf1 = " << f1 << "\nf2 = " << f2;
cout << "\nc1 = " << c1 << "\nc2 = " << c2; Swap(i1, i2);
Swap(f1, f2);
Swap(c1, c2);
cout << "\n After passing data to function template.\n";
```

```
cout << "i1 = " << i1 << "\ni2 = " << i2;
cout << "\nf1 = " << f1 << "\nf2 = " << f2;
cout << "\nc1 = " << c1 << "\nc2 = " << c2;
}
```

Output:

Before passing data to function template

i1=1 i2=2

f1=1.2 f2=2.2

c1 = 'a', c2 = 'b';

After passing data to function template

i1=2 i2=1

f1=2.2 f2=1.2

c1 = 'b', c2 = 'a';

3. Construct a program and illustrate how to throw an exception? Explain

```
#include<iostream.h>
void main()
{
int a,b;
cout<<“Enter the values of a and b\n”;
cin>>a>>b;
int x=a-b;
try
{
if(x!=0)
{
cout<<“Result (a/x)=“<<a/x;
}
```

```
else
throw(x);
}
catch(int i)
{
cout<<“Exception caught: Divide by Zero\n”;
}
cout<<“End”;
}
```

Output:

First Run

11

Enter the values of a and b: 20 15

Result(a/x)=4

Second Run

Enter the values of a and b: 10 10

Exception caught: Divide by Zero

4. Experiment with the use of multiple catch statements in a program? Explain.

It is possible that a program segment has more than one condition to throw an exception. In such

cases, we can associate more than one catch statement with a try as shown below:

```
try
{
//try block
}
catch(type1 arg)
{
```

```
// catch block1
}
catch(type2 arg)
{
// catch block2
}
…….
…….
catch(typeN arg)
{
// catch blockN
}
```

• When an exception is thrown, the exception handlers are searched in order for an appropriate match.

• The first handler that yields a match is executed.

• After executing the handler, the control goes to the first statement after the last catch block for that try.

• When no match is found, the program is terminated.

• It is possible that arguments of several catch statements match the type of an exception. In such cases, the first handler that matches the exception type is executed.

Exception Handling with Multiple Catch Example Program:

```
#include<iostream.h>
#include<conio.h>
void test(int x)
{
try
{
}
```

```
if (x > 0) throw x; //int
else throw 'x'; //char
catch (int x)
{
cout << "Catch a integer and that integer is:" << x;
}
catch (char m)
{
cout << "Catch a character and that character is:" << m;
}
}
void main()
{
cout << "Testing
multiple catches\n:";
test(10);
test(0);
getch();
}
```

Output:

Testing multiple catches

Catch a integer and

that integer is: 10

Catch a character and

that character is: x

5. Construct a program to demonstrate how to re-throw an exception? Explain

• An exception handler may decide to throw the exception caught without processing it.

• In such situations, we invoke throw without any arguments as shown below:

throw;

16

• This causes the current exception to be thrown to the next enclosing try/catch sequence and is caught by catch statement listed after that enclosing try block.

```
#include<iostream.h>
void divide(double x, double y)
{
cout<<“Inside function\n”;
try
{
if(y==0.0)
throw y; //throwing double
else
cout<<“Division =“<<x/y;
}
catch(double)
{
cout<<“Caught double inside function”;
throw; //rethrowing double
}
cout<<“ end of function”;
}
void main()
{
```

```
cout<<“Inside main”;
try
{
divide(10.5,2.0);
divide(20.0,0.0);
}
catch(double)
{
cout<<“caught double inside main”;
}
cout<<“End of main”;
}
```

Output:

Inside main

Inside function

Division=5.25

End of function

Inside function

Caught double inside function

Caught double inside main

End of main

6. Discuss the Importance of templates in programming Explain with the help of an Example.

Templates are powerful features of C++ which allows us to write generic classes and functions thus provide support for generic programming.

• Generic programming is an approach where generic types are used as parameters in algorithms so that they work for a variety of suitable data types and data structures.

• In simple terms, we can create a single function or a class to work with different data types usingtemplates.

• A template can be used to create a family of classes or functions. For example, a class template for an array class would enable us to create arrays of various data types such as int array and float array.

• Similarly we can also define a template for a function, say mul(), that would help us create various versions of mul() for multiplying int, float and double type values.

• Templates are often used in larger codebase for the purpose of code reusability and flexibility of the programs.

• Templates are of great utility to programmers in C++, especially when combined with multiple inheritance and operator overloading.

• The C++ Standard Library provides many useful functions within a framework of connected templates.

7. Elaborate on basics of exception handling having two kinds of exceptions.

There are two kinds of exceptions:

• SYNCHRONOUS EXCEPTIONS: The exceptions which occur during the program

execution due to some fault in the input data or technique that is not suitable to handle current class of data, within the program are known as synchronous exceptions.

Errors such as out-of-range index, “over-flow” and underflow belong to this. Exception handling is designed to support only synchronous exceptions, such as array range checks. The term synchronous exception means that exceptions can be originated only from throw Expressions. The C++ standard supports synchronous exception handling with a termination model. Termination means that once an exception is thrown, control never returns to the throw point.

• ASYNCHRONOUS EXCEPTIONS: The exceptions are caused by events or faults

unrelated (External) to the program and beyond the control of the program are called

asynchronous exceptions. For example, errors such as keyboard interrupts, hardware

malfunctions, disk failure etc. Exception handling is not designed to directly handle asynchronous exceptions such as keyboard interrupts. However, we can make exception handling work in the presence of asynchronous events if we are careful.

For instance, to make exception handling work with signals, we can write a signal

handler that sets a global variable, and creates another routine that polls the value of that variable at regular intervals and throws an exception when the value changes. We cannot throw an exception from a signal handler.

8. Elaborate about Class templates with multiple parameters.

We can use more than one generic data types in a class template.

They are declared as,

```
template <class T1,class T2, ….. >
class className
{
... .. ...
……..(Body of the class)
……..
};
```

Example:

```
#include<iostream.h>
Template<class T1, class T2>
class Test
{
T1 a;
T2 b;
public: Test(T1 x, T2 y)
{
}
void show()
{
a=x;
b=y;
```

```
cout<<a<<" and" <<b<<"\n";
4
}
};
void main()
{
Test<float, int> test1(1.23, 8);
Test<int,char> test2(100, 'w');
test1.show();
test2.show();
}
```

Output:

1.23 and 8

100 and w

9. Analyze overloading of template functions? Explain with example.

• A template function can be overloaded either by template functions or ordinary functions of its name.

An overloading is accomplished as follows:

1. Call an ordinary function that has an exact match.

2. Call a template function that could be created with an exact match.

3. Try normal overloading to ordinary functions and call the one that matches.

An error is generated if no match is found.

Example program:

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

```
template<class T1>
void display(T1 x)
{
cout<< “Overloaded Template display 1:” <<x<<“\n”;
}
template <class T1 x, T2 y)
void display(T1 x, T2 y)
{
cout<<“Overloaded template display 2:”<<x<<“and” <<y;
}
void display(int x)
{
cout<< “Explicit display” <<x<<“\n”;
}
void main()
{
display(100); display(10, 20);
display(12.34); display(2.45, 5.5);
display(‘C’); display(‘A’, ‘B’);
}
```

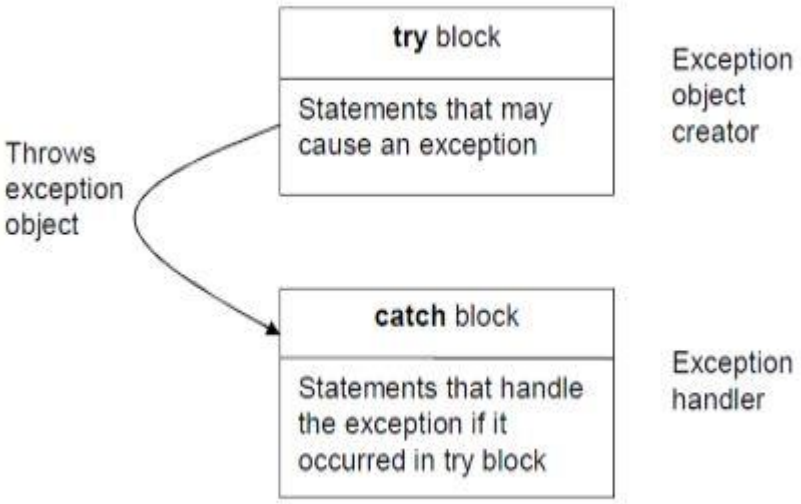
10. Elaborate on Exception Handling mechanism with example.

C++ exception handling mechanism built upon three keywords: try, throw and catch.

The keyword try is used to preface a block of statements which may generate exceptions. This block of statements is known as try block.

When an exception is detected, it is thrown using a throw statement in the try block.

A catch block defined by the keyword catch ‘catches’ the exception ‘thrown’ by the throw statement in the try block and handles it properly.



The catch block that catches an exception must immediately follow try block that throws the exception. The general form of these two blocks is as follows:

```
.....
.....
try
{
      .....
      throw exception;          // Block of statements which
      .....                     // detects and throws an exception
      .....
}
catch(type arg)                 // Catches exception
{
      .....
      .....                     // Block of statements that
      .....                     // handles the exception
      .....
}
.....
.....
```

• When the try block throws an exception, the program control leaves the try block and enters the catch statement of the catch block.

• Note that exceptions are objects used to transmit information about a problem.

• If the type of object thrown matches the arg type in the catch statement, then catch block is executed for handling the exception.

• If they do not match, the program is aborted with the help of abort() function which is involved by default. When no exception is detected and thrown, the control goes to the statement immediately after the catch block. i.e the catch block is skipped.

```
#include<iostream.h>
void main()
{
int a,b;
cout<<“Enter the values of a and b\n”;
cin>>a>>b;
int x=a-b;
try
{
if(x!=0)
{
cout<<“Result (a/x)=“<<a/x;
}
else
throw(x);
}
catch(int i)
{
cout<<“Exception caught: Divide by Zero\n”;
}
cout<<“End”;
}
```

Output:

First Run

Enter the values of a and b: 20 15

Result(a/x)=4

Second Run

Enter the values of a and b: 10 10

Exception caught: Divide by Zero

• The output of the first run shows successful execution. When no exception is thrown, the

catch block is skipped and execution resumes with the first line after catch.

• In second run, the denominator x becomes zero and therefore a division-by-zero situation

occurs.

• This exception is thrown using the object x.

• Since the exception object is an int type, the catch statement containing int type argument

catches the exception and displays necessary message.

• Exceptions are thrown by functions that are invoked from within the try blocks.

• The point at which throw is executed is called the throw point.

• Once an exception is thrown to the catch block, control cannot return to the throw point.

The general format of code for this kind of relationship is:

```
Type function(arg list)                    // Function with exceptions
{
        ……..
        ……..
        throw( object);                    // Throws exception
        ……..
        ……..
}
……..
……..

try
```

```
{
        …….
        ……. Invoke function here
        …….
}
catch(type arg)
{
        …….
        ……. Handles exception here
        …….
}
……..
```

Note: The try block is immediately followed by the catch block, irrespective of the location of the

throw point.

Below program demonstrates how a try block invokes a function that generates an exception.

11. Analyze about specified exception in detail

It is possible to restrict a function to throw only certain specified exceptions. This is

achieved by adding a throw list clause to the function definition.

The general form of using an exception specification is

type function(arglist) throw (type-list)

{

……

…… //function body

}

The type-list specifies the type of exceptions that may be thrown.

Throwing any other type of exception will cause abnormal program termination. If we

wish to prevent a function from throwing any exception, we may do so by making the

type-list empty.

throw( ); //empty list

Example program:

```
#include<iostream.h>
#include<conio.h>
void test(int x) throw(int, double)
{
if (x==0) throw ‘x’; //char
else if(x==1) throw x; //int
else if (x==-1) throw 1.0; //double
cout<<“End of function block”;
}
void main()
{
try
{
cout << "Testing throw restrictions";
test(0);
test(1);
test(-1);
}
catch(char c)
{
```

```
cout<<“caught a character”;
}
catch( int m)
{
cout<<“caught integer”;
}
Catch(double d)
{
cout<<“caught a double”;
}
Cout<<“End of main”;
}
```

Output:

Testing rethrowing exception

Caught a character

End of main