Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills).
4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgment on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

1. a) Attempt any SIX of the following: Marks 12

i) What is Data abstraction?

(Definition of data abstraction – 2 Marks)

Ans: Abstraction refers to the act of representing essential features without including the background details or explanation.

ii) Define the following terms:

1) Object

2) Class

(*Definition of Object - 1 Mark, Definition of Class- 1 Mark)*

Ans: Object:

Objects are run-time entities used to represent or model a particular piece of the system.

OR

An object is the instance of the class.

OR

An object is like a compound variable of the user defined type.

Class:

A class is the collection of related data and function under a single name.

OR

A class is collection of object of similar type.

iii) State the characteristics of destructor.

(Any two characteristics – 2 Marks)

Ans: Following are characteristics of destructor

1. They should be declared in public section.
2. They are invoked automatically when object goes out of scope.
3. They have same name as class name, But preceded by tild (~) symbol.
4. They can not be inherited.
5. They can not have default arguments.
6. They do not have return type, not even void.
7. They can be virtual.
8. We can not refer to addresses of destructors.

iv) Define inheritance and enlist it's types.

(*Definition of inheritance -1 Mark, List of types -1 Mark)*

Ans: Definition:

The mechanism of deriving new classes from an old one is called inheritance.

OR

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

Types of Inheritance:

1. Single Inheritance

2. Multiple Inheritance

3. Multilevel Inheritance

4. Hierarchical Inheritance

5. Hybrid Inheritance

v) State any two pointer operator.

(List of two operator -1 Mark each)

Ans: Following are pointer operator:

1. & (address of operator)

2. * (Value of operator)

vi) What is a pointer? Write down the general syntax of its declaration.

(Definition – 1 Mark, Syntax – 1 Mark)

Ans: Definition:

Pointer is variable used to store the memory address.

Syntax:

data_type *pointer_variable_name;

vii) Enlist any four operators which cannot be overloaded.

(For each operator (any four) - 1/2 Marks each)

Ans:

Size of	Size of Operator
.	Membership operator
.*	Pointer-to-member operator
::	Scope resolution operator
?:	Conditional operator

viii) List types of polymorphism.

(List of each type – 1 Mark)

Ans: Types of polymorphism are as follows:

1. Compile time polymorphism
2. Run time polymorphism

b) Attempt any TWO of the following: Marks 8

i) Differentiate between POP and OOP. (Any four points)

(Any four points each point - 1 Mark)

Ans:

POP	OOP
(1) In pop, more emphasis is given in doing things.	(1) In oop, more comphasis is given on data rather than procedures.
(2) Large program are divided into small modules class functions.	(2) Large program are divided into what are known as objects.
(3) Most of functions show global data.	(3) Data is hidden and cannot be accessed by external functions.
(4) Does not supports Abstraction, Inheritance , Encapsulation and polymorphism .	(4) Supports Abstraction, Inheritance, Encapsulation and polymorphism.
(5) Employs top – down approach	(5) Employs bottom-up approach
(6)Data moves openly around stem to system from one function to another.	(6) Functions that operate on data of an object are tied together in data structure.
(7)Examples :- PASCAL, COBOL,FORTRAN, C.	(7) Examples :- C++, JAVA , VB ,small talk ,effiel , c# , .net . etc .
(8) New data and functions cannot be easily added whenever required.	(8)New data and functions can be easily added whenever required .

ii) Write a program to define a structure 'Tender' having data members tender-no., cost and company-name. Accept & display this data two variable of this structure.

(Structure Declaration: 2 Marks, Accept data for 2 variables: 1 Mark, Display data for 2 variables: 1 Mark)

Ans:

```
# include <conio.h>
#include<iostream.h>
struct tender
{
 int tender_no;
float cost;
char company_name[20];
} t1,t2;

void main()
{
 clrscr();
cout <<"Enter values for tender_number, cost and company for first tender";
cin>>t1.tender_no>>t1.cost>>t1.company_name;
cout <<"Enter values for tender_no,cost and company for second tender";
cin>>t2.tender_no>>t2.cost>>t2.company_name;
cout<<"\n Details of tender1 are:";
cout<<"\n tender no="<<t1.tender_no;
cout<<"\n tender cost="<<t1.cost;
```

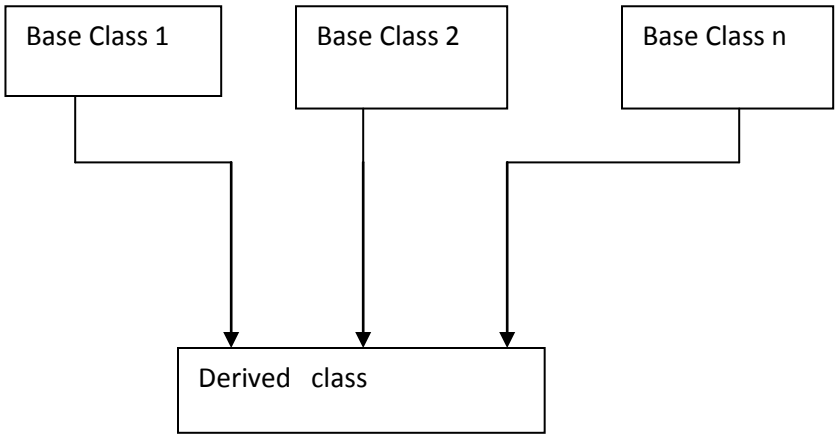
```
cout<<"\n company name="<<t1.company_name;
cout<<"\n Details of tender2 are:";
cout<<"\n tender no="<<t2.tender_no;
cout<<"\n tender cost="<<t2.cost;
cout<<"\n company name="<<t2.company_name;
getch();
}
```

iii) Describe multiple inheritance with suitable example.

(Description – 2 Marks, Example - 2 Marks)

Ans: A derived class with multiple base classes is called as multiple inheritance .

Diagram:



Syntax :-

```
class   base _class _name1
        {   ____  } ;
class   base _class _name2
        {   ____  } ;
```

class derived_class_name :visibility_mode base_class_name_1 ,…, visibility mode

base _class _name_n

{ ____ } ;

Note: Any Example *(2 Marks)

```
#include<conio.h>
#include<iostream.h>
class base1
 {
  public:
   int b1;
    void get()
     {
      cout<<"\n Enter a number";
      cin>>b1;
     }
   void put()
     {
      cout<<"\n b=="<<b1;
     }
 };


 class base2
 {
  public:
   int b2;
```

```
  void get1()
   {
    cout<<"\n Enter a number";
    cin>>b2;
   }
  void put1()
   {
    cout<<"\n b=="<<b2;
   }
};

class derived : public base1,public base2
 {
  public:
   void get2()
    {
     get();
     get1();
    }
   void put2()
    {
     put();
     put1();
    }
  };
```

```
void main()
{
clrscr();
derived d;
d.get2();
d.put2();
getch();
}
```

2. Attempt any four of the following: Marks 16

a) Describe any four basic concepts of OOP.

(Any 4 concepts each - 1 Mark)

Ans: Basic Concepts of Object Oriented Programming

1. Objects

Objects are the basic run time entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle.

An object is the instance of the class.

When a program is executed, the objects interact by sending messages to one another.

2. Classes

A class is the collection of related data and function under a single name.

A class is collection of object of similar type. The entire set of data and code of an object can be made a user-defined data type with the help of class. Once a class has been defined, we can create any number of objects belonging to that class. Classes are user-defined that types and behave like the built-in types of a programming language.

3. Data Abstraction and Encapsulation

The wrapping up of data and function into a single unit (called class) is known as encapsulation. The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it. This insulation of the data from direct access by the program is called data

hiding or information hiding. Abstraction refers to the act of representing essential features without including the background details or explanation. Classes use the concept of abstraction ,they encapsulate all the essential properties of the object that are to be created.

4. Inheritance

Inheritance is the process by which objects of one class acquired the properties of objects of another classes. 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. This is possible by deriving a new class from the existing one. The new class will have the combined feature of both the classes.

5. Polymorphism

Polymorphism means the ability to take more than on form. An operation may exhibit different behavior is different instances. For example, consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation.

6. Dynamic Binding

Binding refers to the linking of a procedure call to the code to be executed in response to the call.

7. Message Passing

An object-oriented program consists of a set of objects that communicate with each other. Objects communicate with one another by sending and receiving information.

Example:

Employee.Salary (name);

b) Explain memory allocation for objects.

(*Explanation- 2 Marks, Figure and its explanation- 2 Marks*)

Ans: Memory Allocation for objects

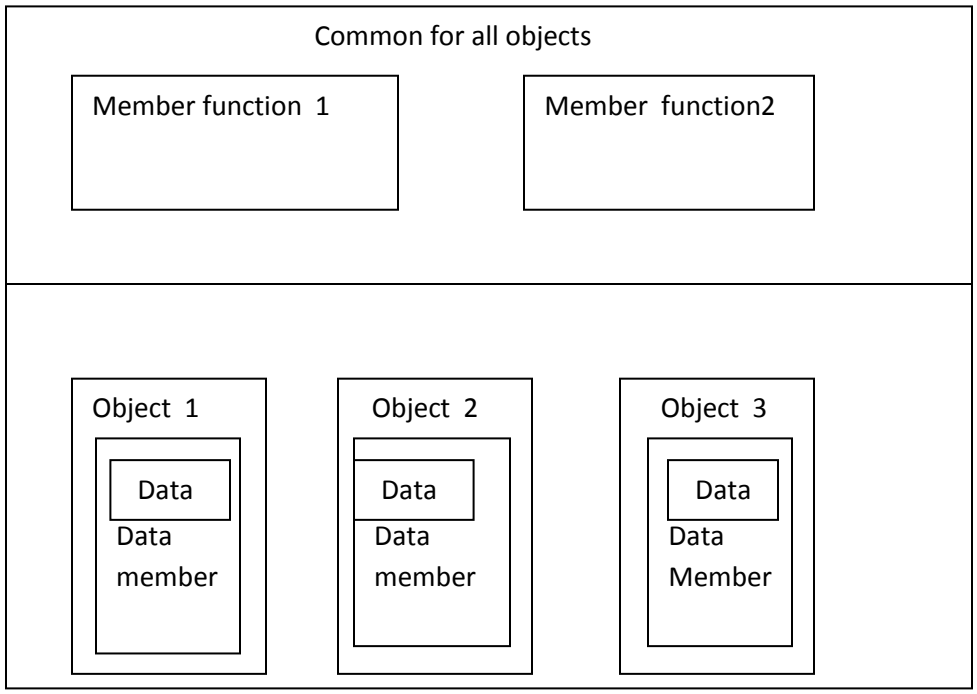
Memory space for objects is allocated when they are declared and not when the class is partly true.

Actually, the memory functions are created and placed in memory space only once when they are defined as part of a class specification.

Since all objects of that class shares the same memory functions, no separate space is allocated for member functions when objects are created.

only space for member variables is allocated separately for each object as member variables will hold different values.

fig. Memory allocation for object



As in diagram only one copy of each function is created and shared by all objects and separate copy for each variable is created.

c) Write a program to declare a class 'Journal' having data members as journal-name, price & no-of-pages. Accept this data for two objects & display the name of journal having greater price.

(Class Declaration - 1 Mark, Creating two objects – 1 Mark, Accept and display data - 2 Marks)

Ans:

```
#include<conio.h>
#include<iostream.h>
class  Journal
{
public:
```

```
char journal_name[20];
int no_of_pages;
float price;
void get()
{
        cout<<"\n Enter value of journal name and no. of pages and price";
        cin>>journal_name>>no_of_pages>>price;
}
void put()
{
        cout<<"\n Journal Name="<<journal_name;
        cout<<"\n No. of pages="<<no_of_pages;
         cout<<"\n Price="<<price;
}
};

void main()
{
Journal J1,J2;
J1.get();
J2.get();
cout<<"\n details of journal having greater price";
if(J1.price > J2.price)
J1.put();
else
```

```
J2.put();
getch();
}
```

d) What is parameterized constructor? Give the syntax & example of it.

(Definition - 1 Mark, Syntax - 1 Mark, Example - 2 Marks)

Ans: **Definition:** constructor which accepts one or more arguments are called as parameterized constructor.

Syntax:

```
Constructor_name( list of parameters)
{
Constructor body
}
```

Example:

```
class Number
 {
public:
int m,n;
Number(int x, int y) //Parameterized constructor
 {
  n=x;
 m=y;
}
};
void main()
{
clrscr();
```

```
Number  N2(10,20);
getch();
}
```

e) Explain the concept of pointer to object with suitable example.

(Explain- 2 Marks, Example/Program - 2 Marks)

Ans: When address of an object of a class is stored into the pointer variable of the same class type then it is pointer to object.

This pointer can be used to access the data member and member functions of same class.

Following example illustrate use of pointer to object

```
#include<conio.h>
#include<iostream.h>
class product
{
private:
int code;
float  price;
public:
void getdata(void)
  {
        cout<<"\n Enter code";
        cin>>code;
        cout<<"\n Enter price";
        cin>>price;
  }
void display(void)
```

```
 {
cout<<"\n Code="<<code<<"\n Price="<<price;
 }
};
void main()
{
product  p1;        //create object of product
product *ptr;       //creat pointer of type product
ptr=&p1;            //ptr points to object p 1
ptr->getdata ();   // Invoking getdata()using pointer to object
p1.display();   // Invoking putdata()using  object
}
```

f) Distinguish between run-time polymorphism & compile-time polymorphism.

(*Any four points – 1 Mark each*)

Ans:

Sr No	**Compile tile polymorphism**	**Run time polymorphism**
1	it is also called as early binding or static binding	it is also called as late binding or dynamic binding
2	In Compile tile polymorphism the linking between function call and function definition is done at compile time only.	In Run time polymorphism the linking between function call and function definition is done at run time only
3	Faster execution of code	Slower execution of code.
4	e.g. function overloading and operator overloading	e.g. function overriding; virtual function

3. Attempt any FOUR of the following: Marks 16

a) State any four features of OOP.

***(List of features (Any four) – 1 Mark each*)**

Ans: Some of the features of object oriented programming are:

1. Emphasis is on data rather than procedure.
2. Programs are divided into what are known as objects.
3. Data structures are designed such that they characterize the objects.
4. Functions that operate on the data of an object are ties together in the data structure.
5. Data is hidden and cannot be accessed by external function.
6. Objects may communicate with each other through function.
7. New data and functions can be easily added whenever necessary.
8. Follows bottom up approach in program design.

b) Write a program to implement single inheritance from following Figure No.1 accepts & display the data for one table.

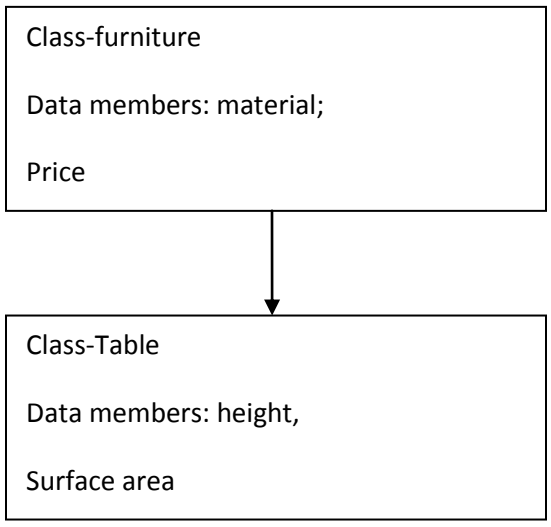


Fig. No. 1

(Class Declaration -2 Marks, Accept and display data - 2 Marks)

Ans:

```
#include<stdlib.h>
#include<iostream.h>
class furniture
        {
        public:
        char matrial[20];
        int price;
        };
class table :public furniture
{
public:
int height ;
float sur_area;
};
void main()
{
table t1;
cout<<"enter material";
cin>>t1.matrial;
cout<<"enter price";
cin>>t1.matrial;
cout<<"enter height";
cin>>t1.height;
```

```
cout<<"enter surface area";
cin>>t1.sur_area;
cout<<" material is<<t1.matrial"<<endl;
cout<<"price is"<<t1.matrial<<endl;
cout<<" height is "<<t1.height<<endl;
cout<<"surface area is "<<t1.sur_area<<endl;
}
```

***NOTE: Any similar correct program can be considered.**

c) Explain different visibility modes used in inheritance.

(State visibility modes - 1 Mark, explanation each - 1 Mark)

Ans: Following are different visibility modes in C++

1. Public
2. Private
3. Protected

Following Table illustrates use of Visibility modes.

Base class access specifies Base Class visibility mode PUBLIC visibility mode	DERIVED CLASS VISIBILITY		
	PUBLIC INHERITANCE	PRIVATE INHERITANCE0	PROTECTED INHERITANCE
PUBLIC	PUBLIC	PRIVATE	PROTECTED
PRIVATE	NOT INHERITED	NOT INHERITED	NOT INHERITED
PROTECTED	PROTECTED	PRIVATE	PROTECTED

when base class is inherited with private visibility mode by derived class, public and protected ember of base class becomes private members of the derived class and Private data of base class will not inherited.

When base class is publicly Inherited then Public members of base class becomes public members of derived class, Private data member will not be inherited and protected data of base class become protected in derived class.

When base class inherited protected, then public and protected members of base class becomes protected in derived class, Private data member will not be inherited

d) Write a program to find length of a string using pointer to the string.

(Declaration of string and assigning pointers to string - 2 Marks; calculating length - 2 Marks)

Ans:

```
#include<iostream.h>
#include<conio.h>
void main()
{
char s[10];
char *ptr;
int count=0;
cout<<"enter string"<<endl;
cin>>s;
ptr=&s;
while(*ptr!='\0')
{
count++;
ptr++;
}
cout<<"length of string is"<<count;
getch();
}
```

e) State any four rules for operator overloading.

(For each rule - 1 Mark, any four rules are expected)

1. Only existing operators can be overloaded. New operators cannot be created.
2. The overloaded operator must have at least one operand that is of user-defined type.
3. We cannot change the basic meaning of an operator. That is to say, we cannot redefine the plus(+) operator to subtract one value from the other.
4. Overloaded operators follows the syntax rules of the original operators. They cannot be overridden.
5. There are operators that cannot be overloaded.

Size of	Size of Operator
.	Membership operator
.*	Pointer-to-member operator
::	Scope resolution operator
?:	Conditional operator

6. We cannot use friend functions to overload certain operators.

   = Assignment operator

   ( ) Function call operator

   [ ] Subscripting operator

   - > Class member access operator
7. Unary operators, overloaded by means of a member function, take no explicit arguments and return no explicit values, but those overloaded by means of friend function, take one reference argument (the object of the relevant class).
8. Binary operators overloaded through a member function take one explicit arguments and those which are overloaded thorough a friend function take two explicit arguments.
9. When using binary operators overloaded through a member function, the left hand operand must be an object of the relevant class.
10. Binary arithmetic operators such as +,-,*, and / must explicitly return a value.

    They must not attempt to change their own arguments

f) Write a program using function overloading to swap 2 integer numbers & swap 2 float numbers.

(Function for swapping two integers – 2 Marks, Function for swapping two Float numbers – 2 Marks)

Ans:

```
#include<conio.h>
#include<iostream.h>
void swap(int a, int b)
 {
int temp;
temp=a;
  a=b;
  b=temp;
cout<<"\n After integer swapping";
cout<<"\n A=="<<a<<"\nB=="<<b;
 }
void swap(float x, float y)
 {
float temp1;
temp1=x;
  x=y;
  y=temp1;
cout<<"\n After float swapping";
cout<<"\n X=="<<x<<"\nY=="<<y;
 }
void main()
 {
clrscr();
swap(10,20);
```

```
swap(10.10,20.10);
getch();
 }
```

4. Attempt any FOUR of the following: Marks 16

a) Enlist the applications of OOP.

(List of applications (Any four) – 1 Mark each)

Ans: Following are applications of OOP:

1. Real-time system
2. Simulation and modeling
3. Object-oriented data bases
4. Hypertext, Hypermedia, and expertext
5. AI and expert systems
6. Neural networks and parallel programming
7. Decision support and office automation systems
8. CIM/CAM/CAD systems

b) Write a program to define a class having data members principle, duration & rate-of-interest. Declare rate-of- interest as static member variable. Calculate the simple interest & display it for one object.

(*Class declaration – 2 Marks, Calculation Function – 1 Mark, Display data function – 1 Mark*)

Ans:

```
#include<iostream.h>
#include<conio.h>
class SI
{
float principle;
int duration;
```

```
static int rate_of_interest;
public:
void get()
 {
cout<<"\n Enter the principle: ";
cin>> principle;
cout<<"\n Enter the duration: ";
cin>>duration;
cout<<"\n.....\n";
 }
void put()
 {
cout<<"\n The principle is: "<< principle <<"Rs \n";
cout<<"\n The duration is: "<<duration<<"yrs \n";
cout<<"\n The rate of interest is: "<<rate_of_interest<<"% \n";
cout<<"\n The simple interest is: "<<(principle*rate_of _interest*duration)/100<<"Rs \n";
cout<<"\n_____\n";
 }
 };
int SI:: rate_of_interest =10;
void main()
 {
 SI s;
clrscr();
s.get();
```

```
cout<<"\n The simple interest is @10% \n";
s.put();
getch();
 }
```

c) Illustrate the concept of constructor with default argument with suitable example.

(Explanation – 2 Marks, Example – 2 Marks)

Note: Any other relevant example should be considered.

Ans: Constructor:

- It as possible to define the constructors with default arguments e.g.
  The constructor antiques () can be declared as –
  integers' (int x, int y =20);
- The default value of argument is20.The the statement integer I *(30)
- Rules for constructor with default argument are sane function with default argument.
- Eg. Programs:

```
class integer
{
int m,n ;
public:
void display ()
{
cout<< "\n"<<m;
cout<<"\t"<<n;
}
integer (int x, int y=100)
{
```

```
m=x;
n=y;
}
};
void main ()
{
integer i1 (200);
i1.display ();
getch ();
}
```

d) State any four characteristics of constructor.

(Each characteristics - 1 Mark; any four)

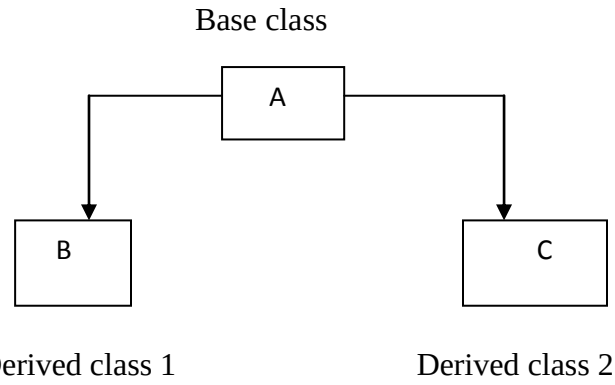
Ans: The constructor functions have some special characteristics. These are:

- 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 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.

e) Illustrate the hierarchical inheritance.

(Definition: 1 Mark, Diagram -1 Mark, Program/syntax/example 2 Marks)

Ans: One base class with many derived classes is called Hierarchical Inheritance.




```
#include<iostream.h>
#include<conio.h>
class emp
{
int id;
char name[20];
int sal;
public:
void accept()
{ cout<<"enter id, name and salary"<<endl;
cin>>id>>name>>sal;
}
void dis()
{
```

```
cout<<"id"<<id;
cout<<"name--"<<name<<endl;
cout<<"salary"<<sal;
}};
class manager : public emp
{ public:
acc1()
{
accept();
dis();
}} ;
class clerk :public emp
{public:
acc2()
{accept();
dis() ;
}};
void main()
{
manager m;
clerk c;
m.acc1();
c.acc2();
getch();
}
```

f) Explain the concept of 'this' pointer.

(Explanation- 2 Marks, Example/Program - 2 Marks)

Ans: this pointer:

1. C++ uses a unique keyword called 'this' to represent an object that invokes a member function.

2. This unique pointer is automatically passed to a member function when it is invoked.

3. 'this ' is a pointer that always point to the object for which the member function was called .

4. For example, the function call A.max () will set the pointer 'this' to the address of the object A. Next time suppose we call B.max(), the pointer 'this' will store address of object B.

Consider the following example:

```
#include<conio.h>
#include<iostream>
class sample
 {
int a;
public:
void setdata(int x)
                {
this ->a=x;
              }
void putdata()
                {
cout<<this ->a;
              }
};
```

```
void main()
 {
clrscr();
sample s;
s.setdata(100);
s.putdata();
getch();
}
```

5. Attempt any FOUR of the following: Marks 16

a) Explain the concept of friend function.

(*Explanation- 2 Marks; Syntax/ program- 2 Marks*)

Ans: Friend functions

Private and protected members of a class cannot be accessed from outside the same class in which they are declared. However, this rule does not affect friends.

A function that is not a member of a class but has access to the class's private and protected members.

They are normal external functions that are given special access privileges.

Friend function is declared by the class that is granting access. The friend declaration can be placed anywhere in the class declaration. It is not affected by the access control keywords (public, private and protected.)

```
#include <iostream>
class myclass {
int num;
public:
myclass(int x) {
num = x;
 }
```

```
friend int isneg(myclassob);
};

int isneg(myclassob) //friend function
{
return (ob.num< 0) ? 1 : 0;
}

int main()
{
myclass a(-1), b(2);

cout<<isneg(a) << ' ' <<isneg(b);
cout<<endl;

return 0;
}
```

b) Write a program to accept string from user & count number of vowels in the string using pointer to string.

(*Accepting String - 1 Mark, positioning pointer variable - 1 Mark, Counting vowels - 2 Marks*)

Ans:

```
#include<iostream.h>
#include<conio.h>
void main()
 {
char string[20], *p;
int count=0;
clrscr();
cout<<"\nEnter a string:-\t";
cin>>string;
```

```
    p=&string[0];
while(*p!='\0')
    {
if(*p=='a'||*p=='e'||*p=='i'||*p=='o'||*p=='u')
 {
count++;
 }
else if(*p=='A'||*p=='E'||*p=='I'||*p=='O'||*p=='U')
 {
count++;
 }
p++;
    }
cout<<"\nNumbers of vowels characters in given string are "<<count;
getch();
 }
```

c) Explain the concept of overloaded constructors in a class with suitable example.

(Explanation- 2 Mark, Program- 2 Marks)

(Note: Any other relevant example should be considered)

Ans: Overloaded constructor is one where we have more than one constructor present in a class. It may have any combination of constructors like default-parameterized, default – copy or copy – parameterized.

Example:-

```
#include<iostream.h>
#include<conio.h>
class ovldcnst
 {
inta,b;
public:
ovldcnst()
```

```
        {
          a=0;
          b=0;
        }
ovldcnst(int x,int y)
        {
          a=x;
          b=y;
        }
void display()
        {
        cout<<"\n"<<"\t"<<a<<"\t"<<b<<"\n";
        }
 };
void main()
 {
ovldcnst o,o1(10,20);
clrscr();
o.display();
o1.display();
ovldcnst o2 = o1;
o2.display();
getch();
 }
```

d) Identify the type of inheritance and implement it by writing a program for the following figure no. 2. Assume suitable member functions.

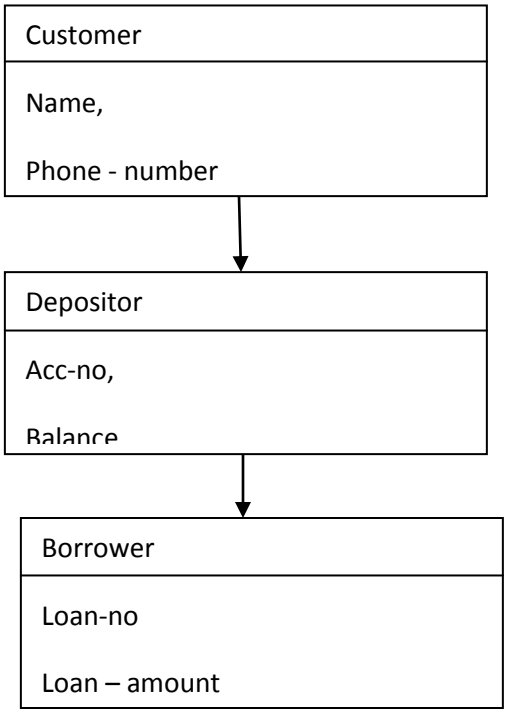


Fig. No. 2

(Type identification- 1 Mark, Program - 3 Marks)

Ans: Figure 2 gives the diagrammatical representation of multilevel inheritance type.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class Customer
 {
char name[20];
long phone_number;
public:
void accept_c()
```

```
        {
        cout<<"\nEnter Name and Phone number:-\t";
        cin>>name>>phone_number;
        }
void display_c()
        {
        cout<<"\nThe Name is:-\t"<<name;
        cout<<"\nPhone Number is:-\t"<<phone_number;
        }
 };
class Depositor:public Customer
 {
int acc_no, balance;
public:
void accept_d()
        {
        cout<<"\nEnter Account Number and Balance:-\t";
        cin>>acc_no>>balance;
        }
void display_d()
        {
        cout<<"\nAccount Number is:-\t"<<acc_no;
        cout<<"\nBalance is:-\t"<<balance;
        }
 };
class Borrower : public Depositor
 {
int Loan_no, Loan_amount;
public:
void accept_b()
        {
```

```
        cout<<"\nEnter Loan Number and Loan Amount:-\t";
        cin>>Loan_no>>Loan_amount;
        }
void display_b()
        {
        cout<<"\nLoan Number is:-\t"<<Loan_no;
        cout<<"\nLoan Amount is:-\t"<<Loan_amount;
        }
 };

void main()
 {
  Borrower b1;
  clrscr();
  b1.accept_c();
  b1.accept_d();
  b1.accept_b();
  b1.display_c();
  b1.display_d();
  b1.display_b();
 }
```

e) Explain virtual function with suitable example.

(Explanation - 2 Marks, Example - 2 Marks)

Ans: Virtual function is a mechanism in Object oriented programming which allows the programmer to have more than one class to have a function with same name, return type and arguments i.e. their function signature can be same.

In Virtual function one defines a function with keyword “Virtual” at the beginning. This ensures that only one copy will be exist at any given period of time.

While accessing these functions the developer will first creates the pointer variable of class type i.e. a pointer of object for the class which holds the virtual function in it. Then while run time one can call required function by adjusting the pointer location.

This is also known as run time polymorphism or late binding or dynamic binding as selection of the functions takes place at run time.

Example of Virtual function:

```
#include<iostream.h>
#include<conio.h>
class student
{
        public:
        int roll;
        float per;
        char name[10];
        virtual void getdata()
        {
        cout<<"\n Enter the Rollno, Percentage, Name:";
        cin>>roll>>per>>name;
        }
        virtual void putdata()
        {
        cout<<"Rollno"<<roll<<"has percentage"<<per<<"And Name is:"<<name;
        }
};
class student1:public student
{
        char branch[10];
        public:
        void getdata()
        {
```

```
        cout<<"\n Enter the branch:";
        cin>>branch;
        }
        void putdata()
         {
        cout<<"The Branch is:"<<branch;
        }
};
void main()
{
        student s,*sptr;
        student1 s1;
        clrscr();
        sptr=&s;
        sptr->getdata();
        sptr->putdata();
        sptr=&s1;
        sptr->getdata();
        sptr->putdata();
        getch();
}
```

f) Write a program to declare a class distance having data members feet & inches. Overload unary '_' operator so that when it is used with object of this class, it will decrement values of inches by 1.

(Class declaration - 1 Mark, Operator function - 2 Marks, Accept & display value -1 Mark)

Ans:

```
#include<iostream.h>
#include<conio.h>
class distance
 {
```

```
int feet,inches;
public:
void accept()
        {
        cout<<"\nEnter distance in feet and inches";
        cin>>feet>>inches;
        }
void display()
{
cout<<"\nDistance is:-";
cout<<"\nFeet = "<<feet;
cout<<"\nInches = "<<inches;
}
void operator -()
 {
        feet--;
        inches--;
    }
 };
void main()
 {
distance d1;
clrscr();
d1.accept();
cout<<"\nDistance before Operator overloading";
d1.display();
   -d1;
cout<<"\nDistance after Operator Overloading";
d1.display();
getch();
 }
```

6. Attempt any TWO of the following: Marks 16

a) Write a program to declare a class 'staff 'having data member as name & department. Accept this data for 10 staffs & display names of staff that are in cm department.
(Declaring class with proper functions and data members - 3 Marks, creating ten objects - 1 Mark, accepting values- 1 Mark, displaying requiring data - 3 Marks)

Ans:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class staff
 {
char name[20],department[5];
public:
void accept()
        {
        cout<<"\nEnter Name and Department";
        cin>>name>>department;
        }
void display()
        {
        if(strcmpi(department,"cm")==0)
           {
        cout<<"\nStaff "<<name<<" works in cm department";
           }
        }
 };
void main()
 {
staff s[10];
int i;
```

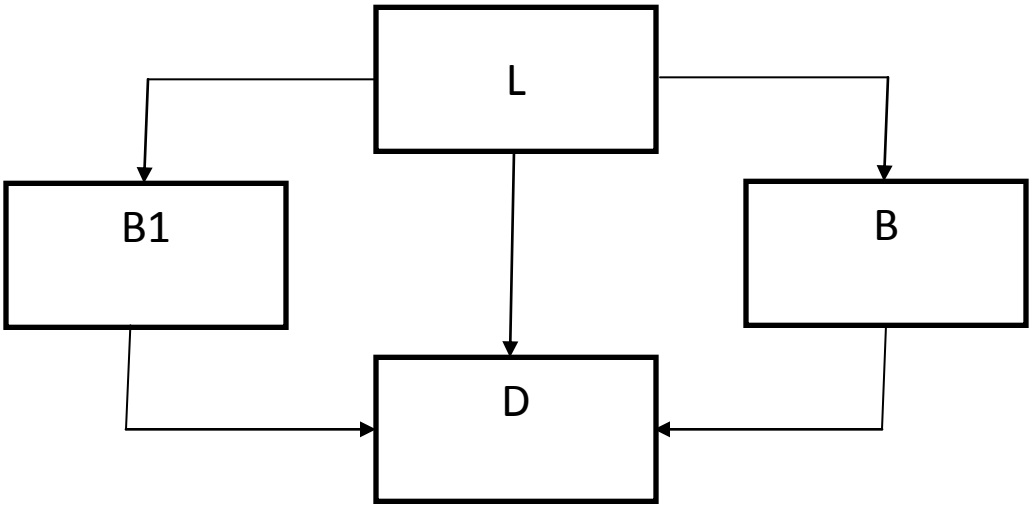
```
clrscr();
for(i=0;i<10;i++)
    {
        s[i].accept();
}
for(i=0;i<10;i++)
    {
        s[i].display();
    }
getch();
 }
```

b) Explain the concept of virtual base class with its general syntax & suitable example.

(Explanation - 2 Marks, Diagram- 1 Mark, Syntax -2 Marks, Program -3 Marks)

Ans: Suppose you have two derived classes B and C that have a common base class A, and you also have another class D that inherits from B and C. You can declare the base class A as virtual to ensure that B and C share the same subobject of A.

In the following example, an object of class D has two distinct subobjects of class L, one through class B1 and another through class B2. You can use the keyword virtual in front of the base class specifiers in the base lists of classes B1 and B2 to indicate that only one subobject of type L, shared by class B1 and class B2, exists.



Syntax of Virtual Base Class

```
class L { /* ... */ }; // indirect base class

class B1 : virtual public L { /* ... */ };

class B2 : virtual public L { /* ... */ };

class D : public B1, public B2, public L { /* ... */ }; // valid
```

Using the keyword virtual in this example ensures that an object of class D inherits only one subobject of class L.

```
#include<iostream.h>
#include<conio.h>
class student
{
intrno;
public:
void getnumber()
  {
cout<<"Enter Roll No:";
cin>>rno;
  }
void putnumber()
  {
cout<<"\n\n\tRoll No:"<<rno<<"\n";
  }
};

class test:virtual public student
{
public:
```

```
int part1,part2;
void getmarks()
  {
cout<<"Enter Marks\n";
cout<<"Part1:";
cin>>part1;
cout<<"Part2:";
cin>>part2;
  }
void putmarks()
  {
cout<<"\tMarks Obtained\n";
cout<<"\n\tPart1:"<<part1;
cout<<"\n\tPart2:"<<part2;
  }
};

class sports:public virtual student
{
public:
int score;
void getscore()
   {
cout<<"Enter Sports Score:";
cin>>score;
   }
void putscore()
   {
cout<<"\n\tSports Score is:"<<score;
   }
};
```

```
class result:public test,public sports
{
int total;
public:
void display()
  {
total=part1+part2+score;
putnumber();
putmarks();
putscore();
cout<<"\n\tTotal Score:"<<total;
  }
};

void main()
{
result obj;
clrscr();
obj.getnumber();
obj.getmarks();
obj.getscore();
obj.display();
getch();
}
```

c) Write a program to concatenate two strings by using pointers.

(Accepting string -1 Mark, pointing to string- 2 Marks, navigating to the end of first string -1 Mark, coping sting -3 Marks, displaying string -1 Mark)

Ans:

```
#include<iostream.h>
#include<conio.h>
void main()
 {
char str1[20],str2[20],*p,*q;
clrscr();
cout<<"\nEnter two string for concatenation";
cin>>str1>>str2;
   p=&str1[0];
   q=&str2[0];
while(*p!='\0')
    {
        p++;
    }
while(*q!='\0')
    {
        *p=*q;
        p++;
        q++;
    }
cout<<"\nConcatenated String is:-\t"<<str1;
getch();
 }
```