MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION
(Autonomous)
(ISO/IEC - 27001 - 2005 Certified)

WINTER– 16 EXAMINATION

Model Answer Subject Code: 17432

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 judgement 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.

Q. No.	Sub Q. N.	Answer	Marking Scheme
1.	 **a)** **Ans:**	**Attempt any TEN of the following:** **List any four object oriented languages.** ***(Any four object oriented languages each ½ marks)*** **Object oriented languages are:** 1) C++ 2) Smalltalk 3) Object Pascal 4) Java 5) Simula 6) Fortran 7) Ada 8) C#	**20**
	b) **Ans:**	**Define class with its syntax.** ***(Definition of class 1 mark, Syntax of class: 1 mark)*** Class is a user defined data type which binds data and functions together. It allows data and functions to be hidden from external use. **Syntax of class:** class class_name { Access specifier:	

```
Declaration of data members;
Declaration of member functions;
};
```

c) **Write any two rules to define friend function.**

(Any two correct rules each 1 mark)

Ans:

Rules to define friend function are:

1) The function declaration should be preceded by the keyword **friend.**
2) The function definition does not use either the keyword friend or scope operator (::).
3) A function can be declared as a friend in any number of classes.
4) A friend function, although not a member function, has full access rights to the private members of the class.
5) It can be declared either in public or the private section of class without affecting its meaning.
6) It has objects as arguments.

d) **What is pure virtual function?**

(Definition of pure virtual function: 2 marks)

Ans:

A pure virtual function is a function which is declared in a base class and which does not have definition relative to the base class. It is either defined in a derived class or is re-declared as pure virtual function.

e) **Define polymorphism. List types of polymorphism.**

(Definition of polymorphism : 1 mark, List of types of polymorphism : 1 mark(each ½ marks))

Ans:

Polymorphism is the ability to take more than one forms. A function may behave differently for performing various tasks.

Functions having same name but different number of arguments/return type in same program is called as polymorphism.

Types of polymorphism:

1) Compile time polymorphism
2) Run time polymorphism

f) **Define pointer variable. Give its syntax**

(Definition of pointer variable : 1 mark, Syntax of pointer variable : 1 mark)

Ans:

Pointer is a variable which stores the address of another variable of same data type.

Syntax:

datatype ***pointer_variable;

g) **Write any two characteristics of static member function.**
(Any two characteristics of static data members each: 1 mark)

Ans: **Characteristics of static data members:**

1) It is initialized to zero when the first object of its class is created. No other initialization is permitted.
2) Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.
3) It is visible only within the class, but its lifetime is the entire program.

h) **What is an abstract base class.**
(Definition of abstract base class : 2 marks)

Ans: An abstract base class is designed only to act as a base class and is not used to create objects. It is a design concept in program development and provides a base upon which other classes may be built.

i) **State any four application of object oriented programming.**
(Any four applications each ½ marks)

Ans: **Applications of object oriented programming:**

1) Real time systems
2) Simulation and modeling
3) Object-oriented databases
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

j) **Define constructor. State any two type of constructor.**
(Definition of constructor: 1 mark, two types of constructor each ½ marks)

Ans: A constructor is a special member function whose task is to initialize the objects of its class. It has the same name as that of class name.

Types of constructor:

1) Default constructor
2) Parameterized constructor
3) Copy constructor

k) **State any two access specifier with example.**
(Any two access specifiers with example each: 1 mark, any other relevant example shall be considered).

Ans:

1) **private**
   Example:
```
class person
{
 private:
char name[20];
int age;
};
```
2) **public**
   Example:
```
class person
{
 public:
 void getdata();
 void putdata();
};
```
3) **protected**
   Example:
```
class item
{
 protected:
 int y;
};
```

l) **What is inheritance? Why inheritance used in C++?**
(Definition of inheritance: 1 mark, use of inheritance : 1 mark)

Ans: The mechanism of deriving a new class from an existing old class is called as an inheritance. Inheritance gives reusability of a code. In this new classes are created by using the properties of the existing classes. It saves the time and money and increases reliability. Hence inheritance is used in C++.

m) **Enlist any four concept of OOP.**
(Any four concepts of OOP each ½ marks)

Ans: **Basic concepts of OOP:**

- Objects
- Classes
- Data abstraction and encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding

- Message passing

n) **State different visibility modes used in inheritance.**
(Any two correct visibility modes each 1 mark)

Ans: **Visibility modes used in inheritance:**

1) private
2) public
3) protected

2. **Attempt any FOUR of the following:** **16**

a) **What are the features of object oriented programming?**
(Any four features each 1 mark)

Ans: **Features of object oriented programming are:**

1) Emphasis is on data rather than procedure.
2) Programs are divided into objects.
3) Data structure designed such that they characterize the objects.
4) Functions that operate on the data of an object are tied together in the data structure.
5) It supports important features like inheritance and polymorphism for reusability.
6) Data is hidden & cannot be accessed by external functions.
7) Objects may communicate with each other through functions.
8) New data and functions can be easily added whenever necessary.
9) Follows bottom-up approach in program designing.

b) **Differentiate between compile time polymorphism and runtime polymorphism**
(Any four points for each point : 1 mark)
{**Note: any other relevant point can be considered**}

Ans:

Compile time polymorphism	**Run time polymorphism**
1. It means that an object is bound to its function call at compile time i.e. linking of function call to its definition at compile time.	**1.** It means that selection of appropriate function is done at run time i.e. linking of function call to its definition at run time.
2. Functions to be called are known well before.	**2.** Function to be called is unknown until appropriate selection is made.
3. This does not require use of pointers to objects.	**3.** This requires use of pointers to object.
4. Function calls are faster.	**4.** Function calls execution is slower.
5. It is also referred as early binding or static binding.	**5.** It is also referred as late binding or dynamic binding.
6. e.g. overloaded function call It is implemented by function overloading or operator overloading	**6.** E.g. virtual function. It is implemented by virtual functions.

c) **Write a program to display largest elements from entered array.**

(correct logic : 2 marks, syntax : 2 marks)

(Any other logic can also be considered)

Ans:

```
#include<iostream.h>
#include<conio.h>
void main()
    {
            int a[10],i,lar=0;
            clrscr();
            cout<<"\nEnter 10 no.";
            for(i=0;i<10;i++)
            {
                    cin>>a[i];
            }
            for(i=0;i<10;i++)
            {
                    if(lar<a[i])
                    {
                            lar=a[i];
                    }
            }
            cout<<"\nThe Largest No is "<<lar;
            getch();
    }
```

d) **Explain various rules for operator overloading.**

(1 mark for each rule; Any 4 Rules)

Ans: ***Rules for Operator Overloading:***

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 follow the syntax rules of the original operators. They cannot be overridden.
5) There are some operators that cannot be overloaded.
6) We cannot use friend functions to overload certain operators. However member functions can be used to overload them.
7) Unary operators, overloaded by means of member function, take no explicit arguments and return no explicit values, but, those overloaded by means of a friend function, take one reference argument (the object of the relevant class).
8) Binary operators overloaded through a member function take one explicit argument and those which are overloaded through 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.

e) **What is destructor? Give it's syntax and example.**
(Description : 2 marks, syntax : 1 mark, Example : 1 mark)

Ans:

Destructor

Description:-

A destructor is used to destroy the objects that are created by a constructor.
It is member function whose name is same as the class name but proceeded by a tilde (~).
A destructor never takes parameters and it does not return any value.
It will be invoked by the compiler upon exit from the program (or block or function) to clean up storage that is no longer accessible.

Syntax:-

```
~destructor_name()
{
}
```

Example:-

```
class student
{
public:
student()
{
cout<<"object is initialized";
}
~student()
{
cout<<"object destroyed";
}
};
```

f) **Write a program to calculate area of circle and rectangle using the concept of function overloading**
(correct logic : 2 marks, syntax : 2 marks)

Ans:

```
#include<iostream.h>
#include<conio.h>
float area(float r)
{
float ar;
ar=3.14*r*r;
return ar;
}
int area(int l, int b)
{
```

```
int ar;
ar=l*b;
return ar;
}
void main()
{
float r,b,l;
float result;
clrscr();


cout<<"\nEnter the Radius of Circle: \n";
cin>>r;
cout<<"\nArea of Circle: "<<area(r);<<endl;
cout<<"\nEnter the Length &Bredth of Rectangle: \n";
cin>>l>>b;
cout<<"\nArea of Rectangle: "<<area(l,b);<<endl;
getch();
}
```

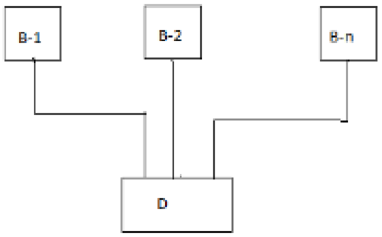
3. **Attempt any FOUR of the following:** **16**

a) **Explain multiple inheritance with suitable example:**
(*Description : 2 marks, example: 2 marks*)

Ans:

Multiple Inheritance:

A class can inherit the attributes of two or more classes as shown in fig .this is known as multiple Inheritance.



```
    cin>>x;
    }
    };
    class B
    {
    protected:
int y;
public:
void getB()
{
```

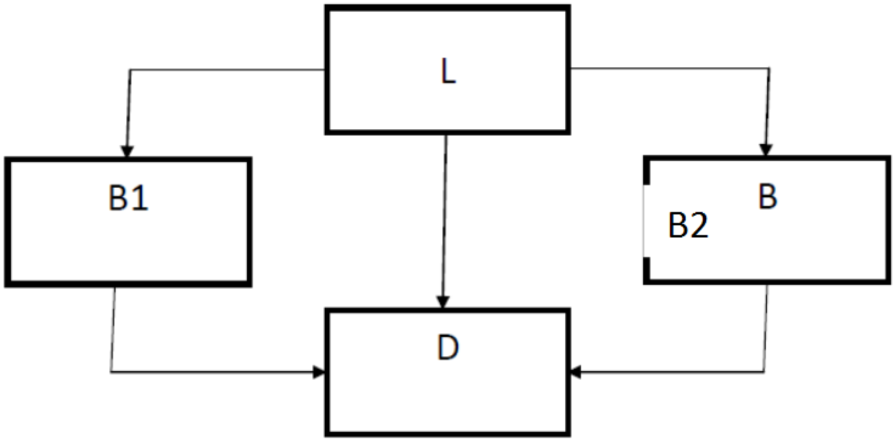
```
cout<<"enter y";
cin>>y;
}
};

class C:public A, public B
{
public:
void display()
{
cout<<"x="<<x;
cout<<"y="<<y;
}
};
void main()
{
C z;
clrscr();
z.getA();
z.getB();
z.display();
getch();
}
```

b) **What is virtual base class? Explain with suitable example.**
(*Description : 2 marks, example : 2 marks*)

Ans:

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.

C++ solves this issue by introducing a virtual base class. When a class is made virtual, necessary care is taken so that the duplication is avoided regardless of the number of paths that exist to the child class. When two or more objects are derived from a common base class, we can prevent multiple copies of the base class being present in an object derived from those objects by declaring the base class as virtual when it is being inherited. Such a base class is known as virtual base class. This can be achieved by preceding the base class' name with the word virtual. In the following example, an object of class D has two distinct sub objects 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 sub object 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 sub object of class L.

Example:

```
#include<iostream.h>
#include<conio.h>
class student
{
int rno;
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<<"\t Marks Obtained \n";
cout<<"\n\t Part1:"<<part1;
cout<<"\n\t Part2:"<<part2;
}
};
class sports: public virtual student
{
public:
int score;
void getscore()
{
cout<<"Enter Sports Score:";
cin>>score;
}
void putscore()
{
cout<<"\n\t Sports Score is:"<<score;
}
};
class result: public test, public sports
{
int total;
public:
void display()
{
total=part1+part2+score;
putnumber();
putmarks();
putscore();
cout<<"\n\t Total Score:"<<total;

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

c) **Write a program to accept string from user and reverse the string using pointer. Display reversed string.**
(correct logic : 2 marks, accept : 1 mark, display of reverse string: 1 mark)

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()
{
char str1[10],*ptr;
int l=0;
cout<<"enter string:";
cin>>str1;
ptr=&str1[0];
while(*ptr!='\0')
{
l++;
ptr++;
}
while(l>0)
{
ptr--;
cout<<*ptr;
l--;
}
getch();
}
```

d) **What is the need of virtual function? Explain with example**.
(*Need of virtual function: 2 marks, example : 2 marks*)

Ans: Polymorphism refers to property by which objects belonging to different classes are classes are able to respond to same message, but in different forms.

- Therefore an essential requirement of polymorphism is ability to refer to object without any regard of their classes. This requires use of single pointer variable to refer to objects of different classes.
- Here we use pointer to base class to refer to all derived objects. Base pointer even when it is made to contain address of derived class, always executes function in base class.
- Compiler simply ignores content of pointer & chooses member function that matches type of pointer. In this case polymorphism is achieved by using virtual functions.
- When we use same function name in both base & derived classes, function in base class is declared as virtual using keyword virtual preceding its normal declaration.
- When function is made virtual C++ determines which function to runtime based on type of object pointed to by base pointer rather than type of pointer.
- Thus by making base pointer to point to different objects different version of virtual function can be executed.
- Runtime polymorphism is achieved only when virtual function accessed through

pointer to base class.

Example:

```
//Virtual function
#include<iostream.h>
#include<conio.h>
class base
 {
 public:
 void display()
 { cout<<"\n Display Base"; }
 virtual void show()
 { cout<<"\n Show Base"; }
 };


 class derived : public base
 {
 public:
 void display()
 {
 cout<<"\n Display Derived";}
 void show()
 {cout<<"\n Show Derived";}
 };
 void main()
 {
 base B;
 derived D;
 base *bptr;
 clrscr();
 cout<<"\n bptr points to base \n";
 bptr=&B;                //pointer to base object
 bptr->display();        //calls base version
 bptr->show();//calls base version
 cout<<"\n bptr points to derived \n";
 bptr=&D;
 bptr->display();        //calls base version
 bptr->show();//calls derived version
 }
```

e) **Give syntax for defining a member function inside and nesting of function in a class with example.**

(syntax for inside definition : 2 marks, example of nesting of function : 2 marks)

Ans:

Syntax of defining member function inside of the class-

```
class classname
{
```

```
public:
return-type  member_function _name(List of parameters)
{
}
};
```

Nesting of function in a class

When a member function can be called by using its name inside another member function of the same class, it is known as nesting of member function.

A member function of a class can be called only by an object of that class using a dot operator. However, there is an exception to this. A member function can be called by using its name inside another member function of the same class. This is known as nesting of member functions.

Example:

```
# include<iostream.h>
class set
{
int m, n;
public:
void input (void);
void display (void);
int largest(void);
};
int set:: largest (void)
{
if (m >= n)
return (m);
else
return (n);
}
void set : : input (void)
{
cout<< “input values of m & n:”;
cin>> m >> n;
}
void set :: display(void)
{
cout<<“largest value = “<<largest();
}
main()
{
set a;
a.input();
a.display();
}
```

f) **Write a program to overload the '-' operator to negate value of variable.**

(creation of class: 1 mark, operator function : 2 mark, operator overloading : 1 mark)

Ans:

```
#include<iostream.h>
#include<conio.h>
class sample
{
int x,y;
public:
void get()
{
cout<<"enter value of x and y";
cin>>x>>y;
}
void display()
{
cout<<x<<y;
}
void operator-()
{
x=-x;
y=-y;
cout<<"x="<<x;
cout<<"y="<<y;
}
};
void main()
{
clrscr();
sample s;
s.get();
s.display();
-s;
getch();
}
```

4. **Attempt any FOUR of the following:** **16**

a) **Explain the concept of constructor with default argument.**

(concept :4 marks, Any relevant description shall be considered)

Ans: **Constructors With Default Argument:**

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

complex (float real , float imag =0);

Default value of argument imag is zero.

Then statement

Complex C (5.0);
Assigns value 5.0 to real variable & 0.0 to imag (by default)

Statement Complex C (2.0,3.0);
Assigns 2.0 to real & 3.0 to imag Actual parameter when specified overrides default value. Missing argument must be trailing arguments“Default constructor”

A :: A() is totally different than “Constructor with default argument”
A :: A (int = 0)
Default argument constructor can be called with either one or no argument. When called with no argument it becomes default constructorWhen both these forms are used in class it causes ambiguity for statement such as A a;34
Ambiguity is whether to call A :: A() or A :: A(int = 0)

b) **What do you mean by inline function? Write its syntax and example.**
(Definition : 1 mark, syntax : 1 mark, example : 2 marks)

Ans: **Inline function**

To eliminate the cost of calls to small functions, c++ proposes a new feature called inline function.

An inline function is a function that is expanded in line when it is invoked. That is the compiler replaces the function call with the corresponding function code.

The inline functions are defined as follows:

Syntax:

```
1)  inline function-header
    {
    function body
    }
2)  inline double cube(double a)
    {
    return(a*a*a);
    }
```

The above inline function can be invoked by statements like

```
c=cune(3.0);
d=cube(2.5+1.5);
```

Example:

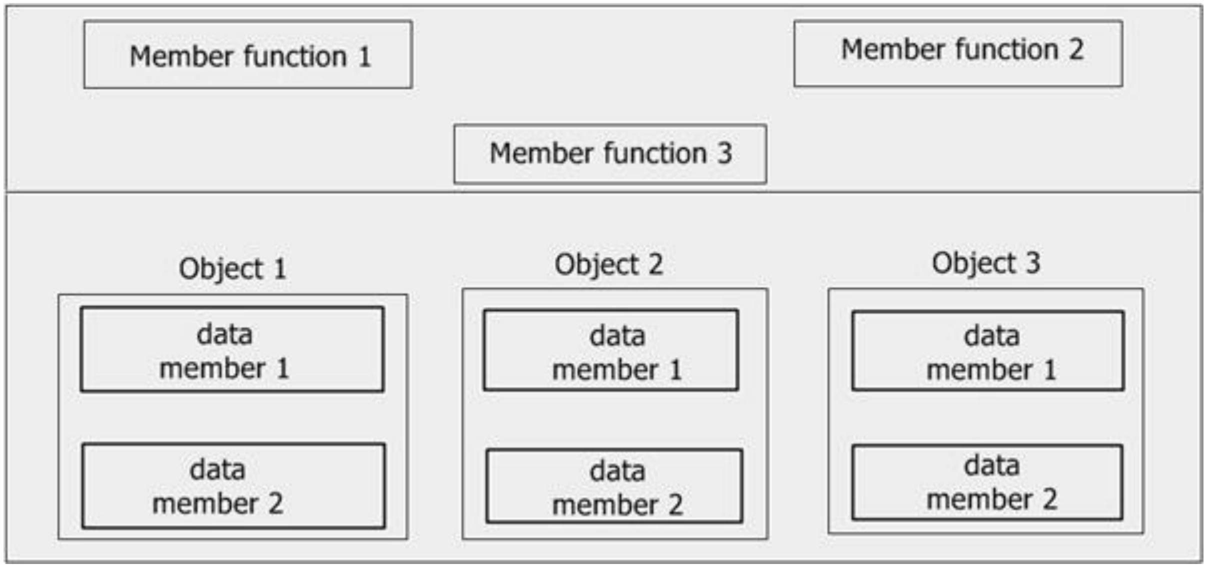
```
#include<iostrem.h>
using namespace std;
inline float mul(float x,float y)
{
return(x*y);
}
inline double div(double p,double q)
{
return(p/q);
}
void main()
{
flaot a=12.345;
float b=9.82;
cout<<mul(a,b)<<"\n";
cout<<div(a,b)<<"\n";
return 0;
}
```

c) **Explain how memory is allocated to objects of a class.**
(Description : 2 marks , Diagram : 2 marks)

Ans:

Description:
Memory space for objects is allocated when they are declared. The member functions are created and placed in the memory space only once when they are defined as a part of class specification. Since all the objects belonging to that class use the same member functions, no separate space is allocated for member functions when the objects are created. Only space for member variables is allocated separately for each object.

Diagram:



d) **Write a program to declare a class ‘student ‘having data members as name and percentage. Write a constructor to initialize these data members accept and display data for one student.**

(Definition of class : 1 mark, defining constructor : 2 marks, display : 1 mark)

Ans:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class student
{
float p;
char name[10];
public:
student(char n[],float per)
{
strcpy(name,n);
p=per;
}
void display()
{
cout<<"\n\n Name of student : "<<name;
cout<<"\n Percentage  of student : "<<p;
}
};
void main()
{
char n[10];
float per;
cout<<”\nEnter Name and Percentage”;
cin>>n>>per;
student s(n,per);
clrscr();
s.display();
getch();
}
```

e) **Write a program to search an element in any array using a pointer.**

(Correct Logic : 2 marks, syntax : 2 marks)

Ans:

```
#include<iostream.h>
#include<conio.h>
void main()
{
int a[5], i,*a1, no, flag=0 ;
clrscr();
a1=&a[0];
cout<<"\n Enter array elements: \n";
for(i=0; i<5; i++)
{
cout<<"Enter "<<i<<" elements: \n";
cin>>*a1;
a1++;
}
cout<<"Enter element to be searched:\n";
cin>>no;
a1=&a[0];
for(i=0; i<5; i++)
{
if(*a1==no)
{
cout<<"Number is present at "<<i+1<<" Position.\n";
flag++;
}
a1++;
}

if(flag == 0)
{
cout<<" Number is not present.\n";
}
getch();
}
```

f) **Write a program to overload “+” operator so that it will perform concatenation of two strings.**

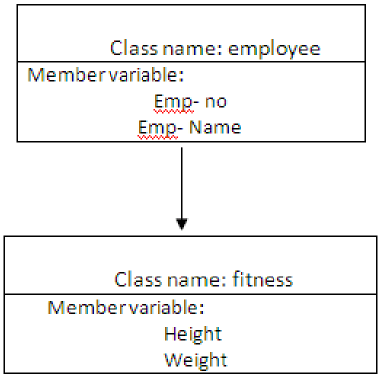
(creation of class : 1 mark, operator function : 2 marks, operator overloading : 1 mark)

Ans:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class sample
{
char str1[10],str2[10];
public:
void get()
{
cout<<"enter first string ";
cin>>str1;
cout<<"enter first string ";
cin>>str2;
}
void sample::operator+()
{
strcat(str1,str2);
cout<<"concatenated String="<<str1;
}
};
void main()
{
clrscr();
sample s;
s.get();
+s;
getch();
}
```

5. **Attempt any FOUR of the following.** 16

a) **Write a program to implement inheritance as shown in Figure No. 1 given below assume suitable Member Function.**



(Implementation of Employee Class : 1 mark; Implementation of Fitness Class with inheritance: 2 marks; Implementation of Main Function :1 mark)

Ans:

```
#include<iostream.h>
#include<conio.h>
class employee
        {
        int emp_no;
        char emp_name [20];
        public:
        void getdata ()
        {
        cout <<"Enter name and number for employee:- \t";
        cin>>emp_name>>emp_no;
        }
        void putdata ()
```

```
                {
                cout <<"Employee details are: - \t"<<emp_no;
                cout<<"\nEmploye name is :-\t"<<emp_name;
                }
                };
        class fitness : public employee
                {
                float height;
                int weight;
                public:
                void accept()
                {
                cout <<"Enter height and weight ";
                cin>>height>>weight;

                }
                void display ()
                {
                cout <<"\nHeight is "<<height <<" and Weight is "<<weight;
                }
        };
        void main()
                {
                fitness f;
                clrscr ();
                f.getdata ();
                f.accept();
                f.putdata ();
                f.display();
                getch ();
                }
```

b) **Write a program to declare class mobile having data members as price and model number. Accept and display this data for ten objects.**

(Implementation Mobile class with data members and member function : 2 marks, Accepting and displaying data for 10 object : 2 marks, Any other relevant logic shall also be considered)

Ans:

```
        #include<iostream.h>
        #include<conio.h>
        class mobile
        {
        int price;
        char model_number[10];
        public:
        void accept()
        {
        cout<<"\nEnter Mobile model number and price";
```

```
cin>>model_number>>price;
}
void display()
{
cout<<"\nModel number is "<<model_number;
cout<<"\nPrice is "<<price;
}
};
void main()
{
mobile m[10];
int i;
clrscr();
for(i=0;i<10;i++)
{
m[i].accept();
}
for(i=0;i<10;i++)
{
m[i].display();
}
getch();
}
```

c) **In procedure oriented programming all data are shared by all functions. Is this statement TRUE? Justify your answer.**

(1 mark for Stating True; 3 marks for Justification)

Ans:

True: In procedure oriented programming all data are shared by all functions.

Justification: In procedure oriented programming most variables are declared as global and hence they can be used by any other function. If a programmer wants to protect data then it has to be constant but then it cannot be changed. So when a function call is made these variables are likely to change its value as there is no protection to these global variables.

d) **Explain overloading of constructor with suitable example.**

(Description : 2 marks, Example : 2 marks)

{**Note: - Any other relevant example shall be considered**}

Ans:

Constructor Overloading is defined as “Having more than one constructor in same class”. Depending upon arguments passed during creation of object specific constructor gets invoked. OOP supports 4 types of constructors as follows.

1. Default Constructor
2. Parameterized Constructor
3. Constructor with Default arguments
4. Copy Constructor

When we use any combination of aforementioned constructors in single class then it is known as Constructor overloading.

Example:

```
#include<iostream.h>
#include<conio.h>
class constov
  {
     int a,b;
    public:
     constov()
           {
             a=5;
             b=6;
           }
     constov(int x,int y)
           {
             a=x;

  b=y;
           }
     constov(int z, float m=5.0)
           {
             a=z;
             b=m;
           }
     void display()
           {
             cout<<endl<<a<<"\t"<<b;
           }
  };
void main()
  {
    clrscr();
    cout<<"\nInvoking Default Constructor";
    constov c1;
    c1.display();
    cout<<"\nInvoking Parameterized Constructor";
    constov c2(10,20);
    c2.display();
    cout<<"\nInvoking Constructor with Default Argument";
    constov c3(76);
    c3.display();
    cout<<"\nInvoking Copy Constructor";
    constov c4=c2;
    c4.display();
    getch();
  }
```

e)

State Characteristics of Static data members of class with suitable example.
(Any 2 Characteristics: 1 mark each, example :2 marks)

Ans:

Static Data Members has following Characteristics

1. Only one copy of static data member is created and it is being shared by all objects.
2. It is initialize to zero.
3. It defined inside class also re-declare outside of class using scope resolution operator (::).
4. It is visible only within the class, but its lifetime is the entire program.
5. Static variables are normally used to maintain values common to the entire class.
6. Static data members are stored separately rather than as a part of an object.
7. They are also known as class variables.

Example:

```
#include<iostream.h>
#include<conio.h>
class simple
{
int p,n;
static float r;
public:
void get()
{
cout<<"enter principle and no of years=";
cin>>p>>n;
}
void put()
{
float si;
si=(p*n*r)/100;
cout<<"simple interest="<<si;
}
};
float simple::r=1.5;
void main()
{
simple s;
clrscr();
s.get();
s.put();
getch();
}
```

f) **Write a program showing use of single inheritance.**

(2 marks for Base class, 2 marks for Derived class)

(Any other example for Single inheritance shall be considered)

Ans:

```
#include<iostream.h>
#include<conio.h>
class employee
        {
                int emp_no;
                char emp_name [20];
                public:
                void getdata ()
                {
                        cout <<"Enter name and number for employee:- \t";
                        cin>>emp_name>>emp_no;
                }
                void putdata ()
                {
                        cout <<"Employee details are: - \t"<<emp_no;
                        cout<<"\nEmploye name is :-\t"<<emp_name;
                }
        };
class fitness : public employee
        {
                float height;
                int weight;
                public:
                void accept()
                {
                        cout <<"Enter height and weight ";
                        cin>>height>>weight;
                }
                void display ()
                {
                        cout <<"\nHeight is "<<height <<" and Weight is "<<weight;
                }
        };
void main()
        {
                fitness f;
                clrscr ();
                f.getdata ();
                f.accept();
                f.putdata ();
                f.display();
                getch ();
        }
```

6. **Attempt any <u>TWO</u> of the following**. **16**

a) ***Write a program to copy the contents of one string to another string using pointer to string. (Accepting String : 1 mark; Creating 2 Pointer variable : 1 mark; Assigning pointer variable at the beginning of both string variable :1 mark; Copy one string into another: 4 marks) {**Note: - Any other relevant logic shall also be considered**}***

Ans:

```
#include<iostream.h>
#include<conio.h>
void main()
  {
    char str1[10],str2[10],*p1,*p2;
    clrscr();
    cout<<"\nEnter a String";
    cin>>str1;
    p1=&str1[0];
    p2=&str2[0];
    while(*p1!='\0')
      {
          *p2=*p1;
          p1++;
          p2++;
      }
    *p2='\0';
    cout<<"Copied String is "<<str2;
    getch();
  }
```

b) **What is “this” point concept? Explain the concept of pointer to object.**
(This pointer Description : 4 marks, Pointer to object : 4 marks, Example optional)

Ans:

This pointer:-

THIS pointer is a default pointer provided by OOP. Every class has its own ***this*** pointer. The advantage of having ***this*** pointer is, a programmer doesn’t need to create it explicitly so no memory is allocated. But drawback of this pointer is that, it has to be used within class only unlike other pointers.

There are two ways of accessing member with this pointer as follows.

```
this->data_member;
this->member_function();
or
*this.(data_member);
```

Pointer to object:-

Pointer to object is a mechanism use when a programmer uses a concept of function overriding. When a program has more than one class having same function with same name one can use pointer to object. In pointer to object, one must always create a pointer of a base class and use it to point objects of same class as well as to the object of a class derived from it.

Syntax:
class_name object_name, ***pointer_variable;
pointer_variable = **&**object name;

Example:
OOP_Pointer obj1, *optr;
Optr=&obj1;

Sample program:

```
#include<iostream.h>
#include<conio.h>
class base
  {
     int a;
    public:
     void virtual getdata()
           {
             cout<<"\nEnter value for A";
             cin>>this->a;
           }
     void virtual putdata()
           {
             cout<<"\nValue for A is"<<this->a;
           }
  };
class derive : public base
  {
     int x;
    public:
     void getdata()
           {
             cout<<"\nEnter value for X";
             cin>>this->x;
           }
     void putdata()
           {
             cout<<"\nValue for X is"<<this->x;
           }
  };
void main()
  {
    base b1,*bptr;
    derive d1;
    clrscr();
    bptr=&b1;
    bptr->getdata();
    bptr->putdata();
    bptr=&d1;
    bptr->getdata();
    bptr->putdata();
    getch();
   }
```

c)

Explain various types of inheritance with example.

(Any 4 types of inheritance : 1 mark for each description, 1 mark for each example/syntax, Example or Syntax shall be considered)

Ans:

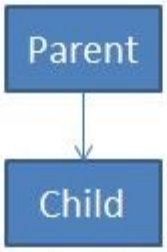
OOP Supports 5 Types of Inheritance as follows.

1. Single Inheritance
2. Multi-level Inheritance
3. Multiple Inheritance
4. Hybrid Inheritance
5. Hierarchical inheritance

Single Inheritance: - A Single inheritance is one, when there exist a single base class and from it exactly one class is derived.

Syntax:-

```
class base
        {
                …
                …
                …
        };
class derive : access_specifier base
        {
                ….
                ….
                ….
        };
```

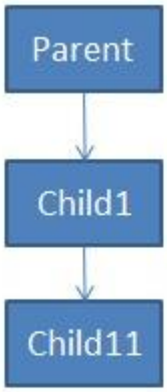


1. Single Inheritance

Multi-level Inheritance: - A Multilevel inheritance is one, when there exist a single base class and from it one class is derived. This derived class act as a base class to other derived class. The class can be derived till any level with condition that only one class has to be derived from each class. All base class from level 2 are known as intermediate class.

Syntax:-

```
class base
        {
                …
                …
                …
        };
class base1 : access_specifier base
        {
                ….
                ….
                ….
        };
class base2 : access_specifier base1
        {
                ….
                ….
                ….
        };
.
.
class derive : access_specifier basen
        {
                ….
                ….
                ….
        };
```

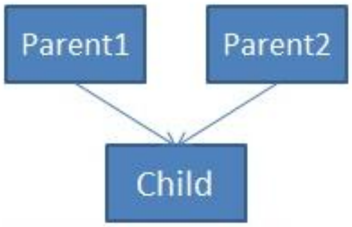


Multi-Level Inheritance

Multiple Inheritance: - It an inheritance type where more than one class give its properties to single class. There can be any number of classes properties can be reuse in one class.

Syntax:-

```
class base1
        {
                …
                …
                …
        };
class base2
        {
                …
                …
                …
        };
class derive : access_specifier base1, access_specifier base2, …access_specifier base_n
        {
                …
                …
        };
```

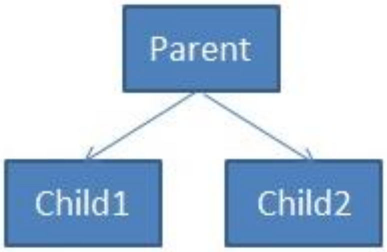


Multiple Inheritance

Hierarchical Inheritance: - In this type of inheritance one class gives its properties to more than one derived class. Further each derived class can give its properties to their respective derived class. This type of inheritance gives tree type structure.

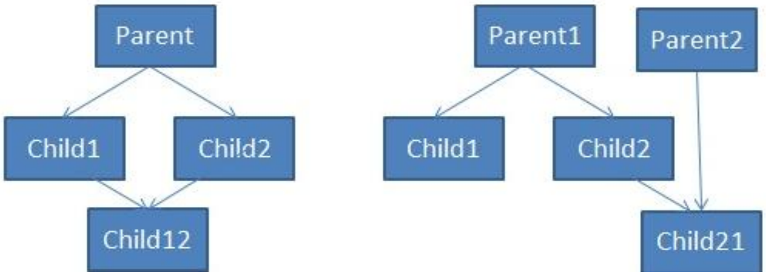
Syntax:

```
class base
        {
                …
                …
                …
        };
class derive : access_specifier base
        {
                …
                …
                …
        };
class derive1 : access_specifier base
        {
                …
                …
                …
        };
```



4. Hierarchical Inheritance

Hybrid Inheritance: - When more than one type of inheritance is use in same program then it forms hybrid inheritance. There is no fix structure available with hybrid inheritance. We can use almost any combination of inheritance type to form hybrid inheritance.



5. Hybrid Inheritance Variations (Mix & Multiple Inheritance)