MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION
(Autonomous)
(ISO/IEC - 27001 - 2005 Certified)
WINTER-15 EXAMINATION
Model Answer Paper

Subject Code: 17432 **Subject Name: Object Oriented Programming**

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) List two memory management operators available in C++ and state its use in one line
(1- Mark for new; 1- Mark for delete)
Ans: There are two types of memory management operators in C++:
New:- use for allocating memory;
Delete:- use for deleting/ free allocated memory space.

(ii) How address of (&) operator is used in pointers, explain with example.
(1- Mark for use of & operator; 1- Mark for example)
Ans : &:- & (The address of operator) is a unary operator that returns the memory address of its operand. For eg. if var is an integer variable, then &var is its address.
It is used as: **ptr=&a;** // pointer ptr starts pointing to address of a;

(iii) Write only errors if any from following code.

```
class abc
{ private : int x ;
 public :  void getdata();
{ cout<<"Enter data";
  cin<<x; }
        };
```

(1- Mark for each error)

Ans:

Error:

a) Declaration terminated incorrectly
b) Inappropriate operator used with cin;

(iv) List four types of constructors.

(1- Mark for any two constructors)

Ans :

1) Default constructor
2) Parameterized constructor
3) Copy Constructor
4) Dynamic constructor

(v) Give any example where runtime polymorphism can be used.

(Any example - 2 Marks.)

Ans:

For example: There is a base class "shape" with a virtual function draw(). It has 2 (or more) child classes "square", "circle" etc. which have implementations of draw().

(vi) State general format of defining derived class.

(format - 2 Marks)

Ans: General Format for defining Derived class is

```
class base-class-name
{
…..
…..
};
class derived_class_name:visibility-mode base-class-name

{
…….        // members of derived class
…….         // members of derived class
……..        // members of derived class
};
```

(vii) Describe 'this' pointer with respect to its use only
(Description - 1 Mark; use - 1 Mark)

Ans :

This pointer: A unique keyword called *this* to represent an object that invokes a member function. It is predefined pointer in any class.

```
class abc
{
int a;
…..
…..
};
```

The private variable 'a' can be directly inside a member function,like
a=123;
Using 'this' can do the same job like:
this->a=123;
this-getdata( ); // call getdata function;

(viii) Write example code of constructor with default argument
(Example code of constructor only, not whole program)
(Any example - 2 Marks)

Ans:

```
Class  complex
    {
        ……….……..
        ………………..
complex (float real,float imag=0);
    };
```

b) Attempt any TWO of the following : 8

(i) Explain concept of overloaded constructor in a class with example.

(Description of Overloaded constructor - 1 Mark; Example - 3 Marks)

Ans:

Overloaded Constructor:

When more than one constructor are present in a class then it is a example of constructor overloading. By using this concept in program user can create different object with various methods & initialization.

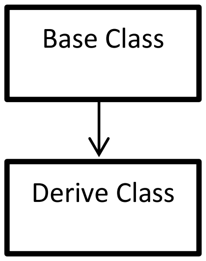
```
#include<iostream.h>
#include<conio.h>
class integer
 {
int m,n;
public:
 integer()
{m=0;n=0;} //constructor1
 integer(int a, int b) //constructor2
{
m=a;n=b;
}
integer(integer &i) //constructor3
{
m=i.m;
n=i.n;
 }
 void display()
{
cout<<"m="<<m<<"\t"; cout<<"n="<<n<<"\n";
}
};
void main()
{
 integer I1;
integer I2(10,20);
 integer I3(I2);
cout<<"object I1";
I1.display();
cout<<"\nobject I2";
I2.display();
cout<<"\nobject I3";
I3.display();
getch();
}
```

(ii) State different types of inheritances and describe any one.

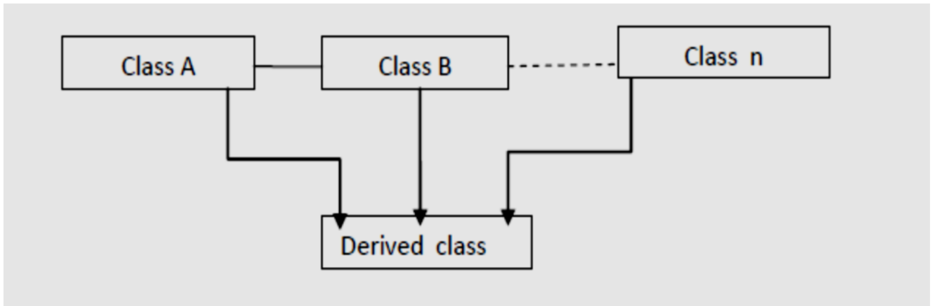
(List of different types of inheritance - 2 Marks; Description of any one - 2 Marks)

Ans :

Single inheritance: It includes single base class which can allow only one derived class to inherit its properties.

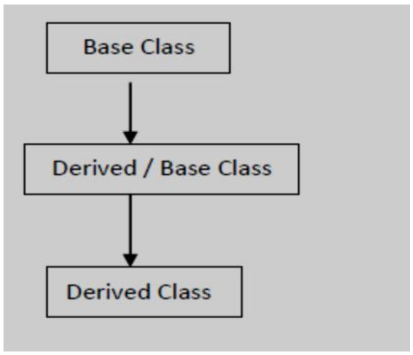


1. Multiple inheritance: In this inheritance, a single derived class can inherit properties of more than one base class

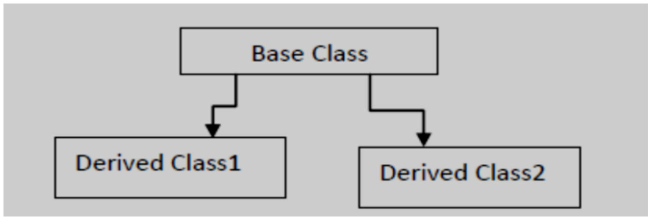


Multiple inheritance

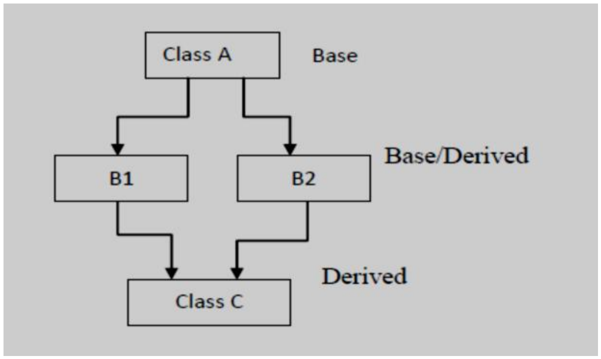
2. Multi-level inheritance: A single class can be derived from a single base class. We can derive a new class from as already derived class. It includes different levels for defining class. A child class can share properties of its base class (parent class) as well as grandparent class



3. Hierarchical inheritance: In this inheritance, multiple classes can be derived from one single base class. All derived classes inherit properties of single base class



4. Hybrid Inheritance: In this inheritance, it combines single inheritance, multiple inheritances, multi – level inheritance & hierarchical inheritance.



(iii) Differentiate between constructor and destructor.
(Any - 4 , Each - 1 Mark)

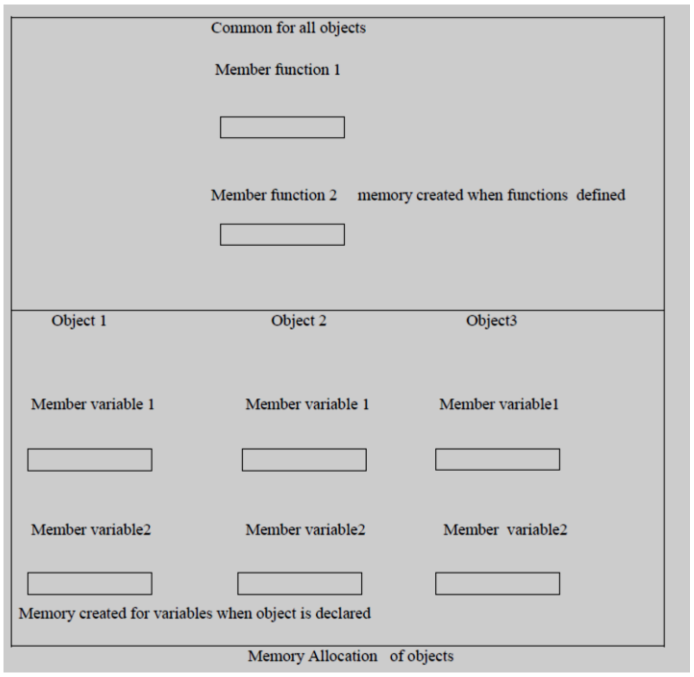
Ans:

Constructor	**Destructor**
Constructor is Used to Initialize the Object.	Destructor is used to destroy the object that are created by the constructor
Constructor can takes arguments.	Destructor cannot take any arguments
Constructor has same name as class name.	Destructor has same name as class name with tiled operator.
Constructor overloading can be possible means more than one constructor can be defined in same class.	Destructor has no any types.
Constructors can be used to dynamically initialize the memory.	Destructor can be used to deallocate the memory.

Syntax of constructor:
class class_name
{
class_name() {}
class_name(argulist){ }
} ;
	Syntax of Destructor
	class class_name
	{
	
	~class-name(void){}
	};

2. Attempt any FOUR of the following: Marks 16

a) How memory is allocated to the objects of class, explain with diagram
(Description - 3 Marks; diagram - 1 Mark)
Ans:



The memory space for object is allocated when they are declared & not when the class is specified. This statement is partly true. Actually, the member functions are created &placed in memory space only once when they are defined as a part of a class definition 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 variable is allocated separately for each object. Separate memory locations for the objects are essential because the member variables will hold different data values for different objects this is shown in fig

b) What is need of virtual function? Explain with example.
(Need of virtual function – 2 Marks; example - 2 Marks)
Ans:
Polymorphism refers to the property by which objects belonging to different classes are able to respond to the same message, but in the different forms.
1. Therefore an essential requirement of polymorphism is the ability to refer to objects without any regard of their classes. This requires the use of a single pointer variable to refer to the objects of different classes.
2. Here, we use the pointer to base class to refer to all the derived objects.
3. The compiler simply ignores the contents of the pointer and chooses the member function that matches the type of the pointer. In this case polymorphism is achieved by using virtual functions.
4. When we use the same function name in both the base and derived classes, the function in the base class is declared as virtual using the keyword virtual preceding its normal declaration.
5. When a function is made virtual, C++ determines which function to use at runtime based on the type of object pointed to by the base pointer, rather than the type of the pointer.
6. Thus, by making the base pointer to point to different objects, different versions of the virtual functions can be executed.
7. Runtime polymorphism is achieved only when a virtual function is accessed through a pointer to the base class.

Example
```
#include<iostream.h>
class Base
{
public:
virtual void show( )
{
cout <<"\nshow base";
}
    };
    class Derived : public Base
```

```
{
public:
void show( )
{
cout<<"\nshow derived";
}
};
void main( )
{
Base B;
Derived D;
Base *bptr;
bptr=&B
bptr→ show( );
bptr=&D;
bptr→ show( );
}
```

c) How protected access specifier is different from private?
(Any Description regarding differentiation between private and protected shall be considered; 1 Mark for each point)
Ans:

1) The members declared as "protected" cannot be accessed from outside the class, but can be accessed from a derived class. This is used when inheritance is applied to the members of a class.
2) The members declared as "private" can be accessed only within the same class and not from outside the class.
3) A member declared as a protected is accessible by the member functions within its class and any class immediately derived from it. A member declared as a private is accessible within class only;
4) When a protected member is inherited in public mode, it becomes protected in the derived class and therefore it is accessible by the member functions of the derived class.
5) If a private member is inherited in public mode, it becomes private in the derived class and therefore it is not accessible by the member functions of the derived class.

d) State any four applications of OOP.
(**Any four applications; each carry - 1 Mark**)
Ans:
Applications Object oriented programming are as follows

- Real time systems
- Simulation and modeling
- Object oriented databases
- Hypertext, hypermedia and expertext
- AI and expert systems
- Neural networks and parallel programming
- Decision support and office automation systems
- CIM/CAM/CAD systems

e) State any four characteristics of static data members
(Any four characteristics - 1 Mark each)
Ans:

- If a static data member is declared as a private category of a class then non-member functions can't access these members.
- If a static member is declared as public then any member of class can access. Whenever a static data member is declared and it has only a single copy, it will be shared by all instances of class.
- The main advantage of using a static member is to declare global data which should be updated while program lives in the memory.

f) Write a program to declare class product having data members as product-id and price. Accept and display data for one object using pointer to the object.
(Creating class - 2 Marks; creating object - 1 Mark; calling functions - 1 Mark)
Ans:
```
 #include<iostream.h>
#include<conio.h>
class product
{
```

```
private:
int product_id;
float prod_price;
public:
void getdata()
{
cout<<"enetr product id"<<'\n';
cin>>product_id;
cout<<"enter product price"<<'\n';
cin>>prod_price;
}
void display()
{
cout<<" product id is::"<<'\n'<<product_id<<'\n';
cout<<"product price is::"<<prod_price;
}
};
void main()
{
product p;
product *ptr;
clrscr();
ptr=&p;
ptr->getdata();
ptr->display();
getch();
}
```

3. Attempt any FOUR of the following: 16

a) Describe syntax of 'cin' and 'cout' with example.
(cin syntax - 1 Mark, example - 1 Mark; cout syntax - 1 Mark, example - 1 Mark)

Ans:
cin = cin statement is use to accept input from user.
Syntax:-

cin>>variable_name

Example:

cin>>i;

cout = cout statement is use to display output processed by computer or to display simple messages to user.
Syntax:-

cout<<variable_name
cout<<**"Message";**

Example:

cout<<"Hello world";
cout<<x;

b) What is overloaded constructor in a class? Explain with example.

(Overloaded Constructor - 1 Mark; example - 3 Marks; any relevant example shall consider)

Ans: Overloaded constructor is one where we can have more than one constructor in same class. Appropriate constructor will be called as per object creation.

Example:

```
#include<iostream.h>
#include<conio.h>
class demo
{
int num;
public:
demo()          //Default Constructor
{
  num=10;
  cout<<"\nValues from Default Constructor"<<num;
}
demo(int x)     //Parameterized Constructor
```

```
  {
    num = x;
    cout<<"\nValues from Parameterized Constructor"<<num;
  }
  void display()
  {
    cout<<"\nValues of are"<<num;
  }
  };
 void main()
{
 clrscr();
demo d1,d2(30);
demo d3=d2; //Copy constructor
d3.display();
getch();
}
```

In above example class demo has three constructors. Default constructor, Parameterized constructor and copy constructor. Object d1 invokes default constructor, object d2 invokes parameterized constructor, whereas d3 invokes copy constructor.

c) What is Virtual Base class? Describe with suitable diagram.

(Virtual base class - 1 Mark; diagram - 1 Mark; description of diagram - 2 Marks)

Ans:

Consider a situation where all three kinds of inheritance, namely, multilevel, multiple, hierarchical inheritance, are involved. This illustrated in fig a. the child has two direct base classes, “parent1” & “parent2” which themselves have a common base class “grandparent”. The child inherits the traits of “grandparent” via two separate path .it can also inherit directly as shown by broken line. The “grandparent” sometimes referred to as indirect base class.

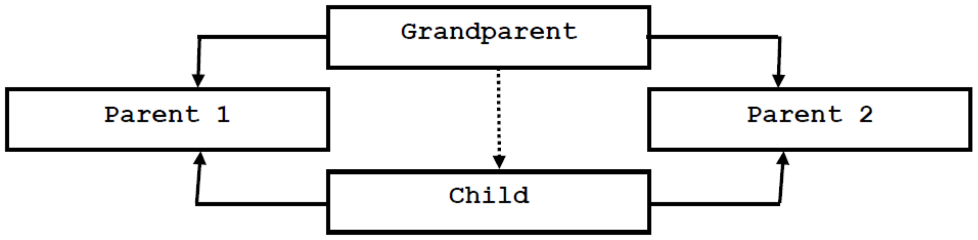


Fig. a: Virtual Base Class

Inheritance by the "child" as shown in fig a might pose some problems. All the public & protected members of "grandparent" is inherited into "child" twice, first via "parent1" & again via "parent 2". This means, "child" would have duplicate sets of the members inherited from "grandparent. This introduces ambiguity & should be avoided. The duplication of inherited members due to these multiple paths can be avoided by making the common base class as virtual base class while declaring the direct or intermediate base classes as shown below.

d) Write a program using function overloading to calculate addition of ten integer numbers and five float numbers.

(Any other logic can also be consider; correct program - 4 Marks)

Ans:

```
#include<iostream.h>
#include<conio.h>
int add(int in[ ]);
float add(float fl[ ]);
void main()
  {
    int a[10],int_sum,i;
    float b[5],flo_sum;
    clrscr();
    cout<<"\nEnter 10 integer numbers";
    for(i=0;i<10;i++)
    {
          cin>>a[i];
    }
 cout<<"\nEnter 5 Float numbers";
```

```
    for(i=0;i<5;i++)
    {
            cin>>b[i];
     }
    int_sum = add(a);
    flo_sum = add(b);
    cout<<"Total of Integer number is "<<int_sum<<endl;
    cout<<"Total of Float number is "<<flo_sum;
  getch();
  }
int add(int x[10])
  {
    int i,res=0;
    for(i=0;i<10;i++)
     {
            res=res+x[i];
             }
    return res;
  }
float add(float j[5])
  {
   float res=0;
   int i;
   for(i=0;i<5;i++)
    {
            res=res+j[i];
  }
 return res;
  }
```

e) What is pointer to array? Explain with example.
(Pointer to array - 2 Marks; Example - 2 Marks)

Ans:

The concept of arrays is related to that of pointers. In fact, arrays work very much like pointers to their first elements, and, actually, an array can always be implicitly converted to the pointer of the proper type. For example, consider these two declarations:

```
int myarray [20];
int * mypointer;
```

The following assignment operation would be valid:

```
mypointer = myarray;
```

one can also use

```
mypointer = &myarray[0];
```

After that, mypointer and myarray would be equivalent and would have very similar properties. The main difference being that mypointer can be assigned a different address, whereas myarray can never be assigned anything, and will always represent the same block of 20 elements of type int. Therefore, the following assignment would not be valid:

```
 myarray = mypointer;
```

Let's see an example that mixes arrays and pointers:

```
#include <iostream>
int main ()
{
  int numbers[5];
  int * p;
  p = numbers;
*p = 10;
  p++;
*p = 20;
  p = &numbers[2];
*p = 30;
  p = numbers + 3;
*p = 40;
  p = numbers;
*(p+4) = 50;
 for (int n=0; n<5; n++)
    cout << numbers[n] << ", ";
  return 0;
```

}

Pointers and arrays support the same set of operations, with the same meaning for both. The main difference being that pointers can be assigned new addresses, while arrays cannot.

f) How function is defined outside of class, write general syntax and example of same
(Description - 1 Mark; syntax - 1 Mark ; example - 2 Marks)

Ans:

Outside class definition:

Member functions that are declared inside class have to be defined separately outside class. Their definitions are as simple as normal function definition. They should have function header & body.

Difference between member function & normal function is that member function incorporates membership "identity label‟ in header

General syntax of member function definition is as:

```
class class_name
    {
            …
            …
            …
            public:
            return_type function_name(argument(s));
            };
            return-type class-name:: function-name(argument(s))
            {
            function body
            }
```

Membership label class-name:: tell complier that function-name belongs to class class-name. This is scope of function restricted to class-name specified in header line. Scope resolution operator (::) is used.

Example:

```
class demo
{
public:
void print( )'
};
```

```
void demo :: print( )
{
cout<<"\nHello";
}
void main( )
{
demo d;
d.print( );
}
```

4. Attempt any FOUR of the following: Marks 16

a) Write a program to implement single inheritance from following data. Accept and display data for one object.

Data ;
Base class_name = Furniture
Data_mem = material , price
Derived class_name = Table
Data-mem = height, surface-area

(Creating base class - 1 Mark; creating Derive class - 1 Mark; creating main function - 2 Marks)

Ans:

```
#include<iostream.h>
#include<conio.h>
class Furniture
  {
    char material[10];
    int price;
   public:
    void accept()
    {
      cout<<"\nEnter Material and Price for Furniture";
      cin>>material>>price;
    }
    void display()
```

```
      {
        cout<<"\nValues for Material and Price of Furniture\t"<<material<<"\t"<<price;
      }
  };
class Table : public Furniture
  {
      int height,surface_area;
    public:
      void accept1()
      {
        cout<<"\nEnter height and surface area for table";
        cin>>height>>surface_area;
      }
      void display1()
      {
        cout<<"\nValues for height and surface area of Table\t"<<height<<"\t"<<surface_area;
      }
  };
void main()
  {
    Table t1;
    clrscr();
    t1.accept();
    t1.accept1();
    t1.display();
    t1.display1();
    getch();
  }
```

b) State any four characteristics of constructor.

(Any four characteristics - 1 Mark for each characteristics)

Ans:

1. Constructors should be declared in the public section.
2. They are invoked automatically when the objects are created.
3. They do not have return type, not even void and therefore they cannot return values.

4. They can accept arguments.
5. They cannot be inherited, though a derived class can call the base class constructor.
6. They cannot be virtual.
7. One cannot refer to their addresses.
8. An object with a constructor cannot be used as a member of a union.
9. They make implicit calls to the operators new and delete when memory allocation is required.

c) With suitable diagram describe structure of C++ program.
(Diagram - 1 Mark; Description - 3 Marks)
Ans:
General C++ program has following structure.

INCLUDE HEADER FILES
DECLARE CLASS
DEFINE MEMBER FUNCTIONS
DEFINE MAIN FUNCTION

Description:-
1. Include header files
In this section a programmer include all header files which are require to execute given program. The most important file is *iostream.h* header file. This file defines most of the C++ statements like *cout* and *cin*. Without this file one cannot load C++ program.
2. Declare Class
In this section a programmer declares all classes which are necessary for given program. The programmer uses general syntax of creating class.
3. Define Member Functions
This section allows programmer to design member functions of a class. The programmer can have inside declaration of a function or outside declaration of a function.
4. Define Main Functions
This section the programmer creates objects and calls various functions writer within various class.

d) Define friend function. Write syntax of declaring it.

(Definition - 2 Marks; Syntax - 2 Marks)

Ans:

Friend function: - The private members of a class cannot be accessed from outside the class but in some situations two classes may need access of each other‟s private data. So a common function can be declared which can be made friend of more than one class to access the private data of more than one class. The common function is made friendly with all those classes whose private data need to be shared in that function. This common function is called as friend function. Friend function is not in the scope of the class in which it is declared. It is called without any object. The class members are accessed with the object name and dot membership operator inside the friend function. It accepts objects as arguments.

Syntax:- friend return_type function_type(parameter1,parameter2,…,parameter n);

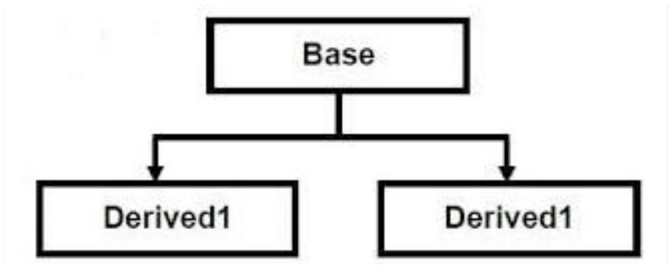
Syntax for calling friend function: - function_name(parameter1,parameter2,,parameter n);

e) How hierarchical inheritance is achieved, explain with example.

(Description- 2 Marks; Example - 2 Marks)

Ans:

Hierarchical Inheritance is a method of inheritance where one or more derived classes is derived from common base class.



```
#include <iostream.h>
class Side
{
protected:
int l;
public:
void set_values (int x)
{ l=x;}
};
class Square: public Side
{
public:
int sq()
{ return (l *l); }
};
class Cube: public Side
{
public:
int cub()
{ return (l *l*l); }
};
int main ()
{
Square s;
s.set_values (10);
cout << "The square value is::" << s.sq() << endl;
Cube c;
c.set_values (20);
cout << "The cube value is::" << c.cub() << endl;
return 0;
}
```

f) Write a program to display elements of array using pointer to array of integers.
(Correct program - 4 Marks; any other logic can also be considered)
Ans:

```
#include<iostream.h>
#include<conio.h>
void main()
  {
    int x[10],*xptr,i;
  clrscr();
  cout<<"\nEnter 10 Numbers";
  for(i=0;i<10;i++)
    {
    cin>>x[i];
    }
  xptr=&x[0];
  while(*xptr!='\0')
    {
    cout<<*xptr<<" ";
    xptr++;
    }
  getch();
}
```

5. Attempt any FOUR of the following : 16

a) State any four rules for operator overloading.
(Any four rules; each - 1 Mark)
Ans:
Rules for overloading operators:

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 data type.
3. We can't change the basic meaning of an operator. That is to say, we can't redefine the plus(+) operator to subtract one value from other.
4. Overloaded operators follow the syntax rules of the original operators. They can't be overridden.
5. There are some operators that can't be overloaded.
6. We can't 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 the 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.

b) Define Structure. Give syntax of defining structure.
(Definition - 2 Marks, Syntax - 2 Marks)
Ans:
Structure is a collection of different data types written under a common name. It is a user defined data type.
Syntax:-
```
struct structure_name
{
data_type variable 1;
data_type variable 2;
•
•
•
data_type variable n;
};
```

c) Create class shape. Derive two class triangle and rectangle from class shape .Write appropriate functions in both classes to accept dimensions and calculate area. Here make area () function virtual which is common to all and will calculate area of both rectangle and triangle. Display area of both.
(Declaring class shape, triangle, rectangle with proper functions and data members - 1 Mark each, main functions - 1 Mark)

Ans

```
#include<iostream.h>
#include<conio.h>
class shape
{
public:
int l,b,h;
virtual void area()=0;
};
class triangle:public shape
{
public:
void getdata()
{
cout<<"\n enter dimensions of triangle:\n length:";
cin>>l;
cout<<"\nheight:";
cin>>h;
}
void area()
{
int a=0.5*l*h;
cout<<"\n area of triangle:"<<a;
}
};
class rectangle:public shape
{
public:
void getdata()
{
cout<<"\n enter dimensions of rectangle:\n length:";
cin>>l;
cout<<"\n breadth:";
cin>>b;
}
void area()
{
int a=l*b;
cout<<"area of rectangle:"<<a;
}
};
void main()
{
clrscr();
shape *p;
triangle t;
rectangle r;
p=&t;
t.getdata();
p->area();
p=&r;
r.getdata();
p->area();
getch()
};
```

d) List any four features of POP.

(Any four points; each - 1 Mark)

Ans:

1. More emphasis is given in doing things.
2. Large program are divided into small modules class functions.
3. Most of functions show global data.
4. Does not support Abstraction, Inheritance, Encapsulation and polymorphism.
5. Employs top – down approach
6. Data moves openly around stem to system from one function to another.
7. New data and functions cannot be easily added whenever required.

e) Define pointer. Give syntax of declaration and initialization of pointer.
(Definition – 1 Mark, Syntax – 3 Mark)
Ans:
Definition:
Pointer is variable used to store the memory address.
Declaration Syntax:
data_type *pointer_variable_name;

Initilization Syntax:
Pointer_variable_name= &variable_name

f) What is difference between compile time polymorphism and run time polymorphism?
(Any four points each - 1 Mark)
Ans:

Compile time Polymorphism	Runtime Polymorphism
It simply means that an object is bound to its function call at compile time.	It simply means that selection of appropriate function is done at run time
Functions to be called are know well before	Function to be called is unknown until appropriate selection is made.
This does not require use of pointers to objects	This requires use of pointers to object
Function calls are faster	Function calls execution are slower
Also called as early binding	Also called as late binding
e.g. overloaded function call	e.g. virtual function
It also referred as Static Binding	It also referred as Dynamic Binding

6. Attempt any TWO of the following: Marks 16

a) Write a program to declare class 'staff' having data members as name and post. Accept this data for 5 staffs and display name of staff who are HOD.
(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],post[5];
public:
void accept()
{
cout<<"\nEnter Name and post";
cin>>name>>post;
}
void display()
{
if(strcmp(post,"HOD")==0)
{
cout<<"\nHOD: "<<name;
}
}
};
void main()
{
staff    s[5];
int i;
clrscr();
for(i=0;i<5;i++)
{
s[i].accept();
}
for(i=0;i<5;i++)
{
s[i].display();
}
getch();
}
```

b) Write a program to demonstrate constructor in derived class with respect to order of calling constructor and passing parameters to base class constructor.

(Declaring base class with proper functions and data members - 3 Marks, Declaring derived class with proper functions and data members - 3 Marks, Main functions - 2 Marks)

Ans:

```
        #include <iostream.h>
        class alpha
        (
   private:
       int x;
   public:
       alpha(int i)
       {
          x = i;
          cout << "\n alpha initialized \n";
            }
            void show_x()
            {
          cout << "\n x = "<<x;
           }
            );
            class beta
            (
               private:
           float y;
                       public:
            beta(float j)
           {
           y = j;
                   cout << "\n beta initialized \n";
            }
           void show_y()
           {
          cout << "\n y = "<<y;
              }
          );
          class gamma : public beta, public alpha
          (
  private:
    int n, m;
  public:
     gamma(int a, float b, int c, int d):: alpha(a), beta(b)
   {
m = c;
n = d;
cout << "\n gamma initialized \n";
   }
void show_mn()
  {
cout << "\n m = "<<m;
cout << "\n n = "<<n;
   }
);

void main()
{
    gamma g(5, 7.65, 30, 100);
    cout << "\n";
   g.show_x();
   g.show_y();
   g.show_mn();
}
```

c) Write a program to concatenate two strings using pointer to string.
(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=&str[0]; q=&str2[0];
while(*p!='\0')
{

p++;
 }
while(*q!='\0')
{
*p=*q;
  p++;
  q++;
   }
    cout<<"\nConcatenated String is:-\t"<<str1;
    getch();
    }
```