`CMR
INSTITUTE OF
TECHNOLOGY

USN

Internal Assessment Test 1 – March 2016

Sub:	Object Oriented Programming Using C++							Code:	13MCA 22
Date:	*14/03/2016*	*Duration:*	*90 mins*	*Max Marks:*	*50*	*Sem:*	*2*	*Branch:*	*MCA*

Answer Key

1. What is Object Oriented Programming?Bring out the underlying concepts of Oops?

To overcome the flaws in procedural approach such as global variables,emphasis on functions rather than data,top down approach in program design etc,object oriented approach is introduced.

Oops treat data as a critical element in the program development and does not allow it to flow freely around the system.The data of an abject can be accessed only by the functions associated with that object.

Object oriented programming can be defined as “an approach that provides a way of modularizing programs by creating partitioned mmory area for both data and functions that can be used as templates for creating copies of such modules on demand”.

Concepts of Oops:

Class:

1. Class is user defined data type and behave like the built-in data type of a programming language.

2. Class is a blue print/model for creating objects.

3. Class specifies the properties and actions of an object.

4. Class does not physically exist.

5. Once a class has been defined, we create any number of objects belonging to that class. Thus, class is a collection of objects of similar type.

Object:

1. Object is the basic run time entities in an object oriented system.

2. Object is the basic unit that are associated with the data and methods of a class.

3. Object is an instance of a particular class.

4. Object physically exists.

5. Objects take up space in memory and have an associated address.

6. Objects communicate by sending messages to one another.

Data Abstraction and Encapsulation:

Abstraction refers to the act of representing essential features without including the background details. In programming languages, data abstraction will be specified by abstract data types and can be achieved through classes.

The wrapping up of data and functions into a single unit is known as encapsulation. It keeps them safe from external interface and misuse as the functions that are wrapped in class can access it. The insulation of the data from direct access by the program is called data hiding.

Inheritance:

1. It provides the concept of reusability.

2. It is a mechanism of creating new classes from the existing classes.

3. It supports the concept of hierarchical classification.

4. A class which provides the properties is called Parent/Super/Base class.

5. A class which acquires the properties is called Child/Sub/Derived class.

6. A sub class defines only those features that are unique to it.

Polymorphism:

1. Polymorphism is derived from two greek words Poly and Morphis where poly means many and morphis means forms.

2.Polymorphism means one thing existing in many forms.

3.Polymorphism plays an important role in allowing objects having different internal structures to share the same external interfaces.

4.Polymorphism is extensively used in implementing inheritance.

5.Function overloading and operator overloading can be used to achieve polymorphism.

Dynamic Binding:

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

2.If the binding occurs at runtime then it is called as dynamic binding.

3.It is also called as late binding as binding of a call to the procedure is not known until runtime.

4.Dynamic Binding is associated with polymorphism and inheritance.

Message Binding:

1.Objects communicate with each other by sending and receiving information using functions.

2.The basic steps to perform message passing are

* Creating classes that define objects and their behaviour.

* Creating objects from class definitions.

* Establishing communication among objects.

3. Message passing involves specifying the name of the object, name of the function and the information to be sent as

objectname.functionname(message);

4. A message for an object is a request for execution of a procedure and therefore

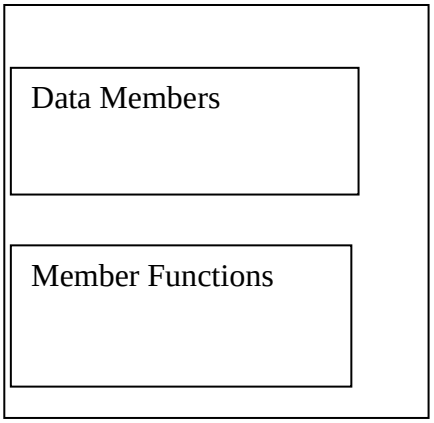
will invoke a function in receiving object that generates the desired result.

5. Communication with an object is feasible as long as it is alive.

2. a)What is a class? Explain the structure of a class with an example.

Class :

A class is the collection of related data and function under a single name. A C++ program can have any number of classes. When related data and functions are kept under a class, it helps to visualize the complex problem efficiently and effectively.



Defining the Class in C++:

Class is defined in C++ programming using keyword class followed by identifier(name of class). Body of class is defined inside curly brackets an terminated by semicolon at the end in similar way as structure.

```
class class_name
  {
  // some data
  // some functions
```

```
};
```

Example of Class in C++

```
class temp
  {
    private:
      int data1;
      float data2;
    public:
      void func1()
        {   data1=2;  }
     float func2(){
          data2=3.5;
          retrun data;
        }
};
```

Explanation

As mentioned, definition of class starts with keyword class followed by name of class(temp) in this case. The body of that class is inside the curly brackets and terminated by semicolon at the end. There are two keywords: private and public mentioned inside the body of class.

b)Differentiate between class and structure.

Class	Structure
1.Classses are reference types	1.Structures are reference types.

2.Classes will have data members and member functions bind together	2.Structures will not have member functions that bind together with data members
3.Class can inherit another class	3.Structures cannot provide inheritance
4.Members of a class are private by default	4.Members of a structure are public by default
5.Class can have all types of constructors and destructors	5.Structure can have all types of constructors and destructors

3. a Explain the concepts of constructors and destructors with example for each.

Constructor is a special member of a class which is used to initalise the objects.Constructors will be called whenever we create objects in the program.

Characteristics of a constructor:

* Constructor will be having the same class name.

* Constructor will not have any return type.It will not be specified even with void.

* Constructor may or may not have parameters.

* Constructor should be declared in public section of a class.

* Constructor can also be overloaded.

Types of constructors:

There are two types of constructors.They are

* Default Constructors

* Parameterised Constructors

Default Constructor:

Constructor with no parameters is called default constructor.

Syntax:

```
classname()
{
    body of the constructor
}
```

Parameterised Constructors:

Constructor with parameters is called parameterised constructor.

Syntax:

```
classname(parameter list)
{
    body of the constructor
}
```

Ex: /*Source Code to demonstrate the working of constructor in C++ Programming */

```
/* This program calculates the area of a rectangle and  displays it. */
#include <iostream>
using namespace std;
class Area
{
   private:
     int length;
     int breadth;

   public:
     Area()
          {
                Length=5;
                Breadth=2;
           }   /* Constructor */
     void GetLength()
     {
        cout<<"Enter length and breadth respectively: ";
        cin>>length>>breadth;
     }
     int AreaCalculation() {  return (length*breadth);  }
     void DisplayArea(int temp)
     {
        cout<<"Area: "<<temp;
     }
```

```
};
int main()
{
   Area A1,A2;
   int temp;
   A1.GetLength();
   temp=A1.AreaCalculation();
   A1.DisplayArea(temp);
   cout<<endl<<"Default Area when value is not taken from user"<<endl;
   temp=A2.AreaCalculation();
   A2.DisplayArea(temp);
   return 0;
}
```

Destructor:

A **destructor** is a special member function of a class that is executed whenever an object of it's class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class.

A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters. It should be declared in public section of a class. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc.

Ex:

```
#include<iostream>
Using namespace std;
Class Test
{
        int *a;
public:
      Test(int size)
      {
            a=new int[size];
```

```
        cout<<"Constructor created";
    }
    ~Test()
    {
        delete  a;
        cout<<"Destructor created";
    }
};
int main()
{
    int s;
    cout<<"Enter the size of an array";
    cin>>s;
    Test t(s);
    return 0;
}
```

3 b: What is copy Constructor? Explain with one suitable example.

The parameters of a constructor can be of any of the data types except an object of its own class as a value parameter.

Hence declaration of the following class specification leads to an error:

```
class x
{
        private:
            ……
        public:
            x( x obj);
            ……..
        };
```

A class's own object can be passed as a reference parameter.

Ex:

class X

{ ……..

public:

X()

X(X &obj);

X(int a);

};

is valid

Such a constructor having a reference to an instance of its own class as an argument is known as copy constructor.

Ex.

bag b3=b2; // copy constructor invoked

bag b3(b2); // copy constructor invoked

b3=b2; // copy constructor is not invoked.

A copy constructor copies the data members from one object to another.

4. a) What is friend function and friend class? Why it is used?

A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.

To declare a function as a friend of a class, precede the function prototype in the class definition with keyword **friend** as follows:

```
class Box {
   double width;
public:
   double length;
   friend void printWidth( Box box );
   void setWidth( double wid );
```

```
};
```

To declare all member functions of class ClassTwo as friends of class ClassOne, place a following declaration in the definition of class ClassOne:

```
friend class ClassTwo;
```

Consider the following program:

```
#include <iostream>

using namespace std;

class Box {
   double width;
public:
   friend void printWidth( Box box );
   void setWidth( double wid );
};

// Member function definition
void Box::setWidth( double wid ) {
   width = wid;
}

// Note: printWidth() is not a member function of any class.
void printWidth( Box box ) {
   /* Because printWidth() is a friend of Box, it can
   directly access any member of this class */
   cout << "Width of box : " << box.width <<endl;
}

// Main function for the program
int main( ) {
   Box box;
```

```
   // set box width with member function
   box.setWidth(10.0);

   // Use friend function to print the wdith.
   printWidth( box );

   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Width of box : 10
```

4. b) Find the mean value of two given number using friend function. (Create a class base with two data member and calculate mean of that data member using friend function.)

```
#include<iostream.h>
#include<conio.h>
class  base
{
    int val1,val2;
   public:
    void get()
    {
       cout<<"Enter two values:";
       cin>>val1>>val2;
    }
    friend float mean(base ob);
};
float mean(base ob)
{
   return float(ob.val1+ob.val2)/2;
}
void main()
{
    clrscr();
    base obj;
    obj.get();
    cout<<"\n Mean value is : "<<mean(obj);
    getch();
}
```

Enter two values: 10, 20
Mean Value is: 15

5. a) Explain the dynamic memory allocation operator in C++. Explain proper syntax and example.

Dynamic memory allocation operators: new and delete.

New: To allocate the memory.

Syntax:

Ptr_var = new vartype;

e.g: ptr = new int;

Ptr_var = new vartype(initial_value);

The type of initial value should be same as the vartype;

e.g: ptr = new int(100);

Delete: To free the memory.

delete ptr_var;

If there is insufficient memory then the exception bad_alloc will be raised. This exception is defined in the header <new>. It is available in standard C++.

Advantages of new and delete:

new and delete operators are like malloc() and free() in C Language. But they have more advantages.

1. **new automatically allocates enough memory to hold the object.**

(No need to use sizeof operator).

2. new automatically returns the pointer to the specified type.

It is not needed to typecast it explicitly.

Allocating Arrays:

ptrvar = new arrtype[size];

Delete [ ] ptrvar;

***** The initial values can't be given during the array allocation.**

5. b) Write a program to dynamically allocate memory for object (using constructor).

Dynamic Object allocation is done by using "new" operator. Then the object is created and a pointer is returned to it. When the object is created, its constructor is called and when it is freed, its destructor is executed. The Parameterized constructors can also be used.

The array of objects can also be allocated. But it can't have initialized values. So if the constructor is created for this , it should be a parameter-less constructs.

6. Define a STUDENT class with USN, Name, and Marks in 3 tests of a subject. Declare an

array of 10 STUDENT objects. Using appropriate functions, find the average of the two better marks for each student. Print the USN, Name and the average marks of all the students.

```
#include <iostream>

using namespace std;

// Declare a class with appropriate member functions and member variables as in the problem statement

class Student

{

        char usn[11];

        char name[15];

        int m1,m2,m3;

        float avg;

        public :

                void readstudent();

                void calcavg();

                void display();

};

//Function to read the student details from the user

void Student :: readstudent()

{

        cout<<"Enter the usn\n";

        cin>>usn;

        cout<<"Enter the name\n";

        cin>>name;

        cout<<"enter the marks of 3 subjects\n";

        cin>>m1>>m2>>m3;
```

```
}
// Function to calculate better of two better marks and to find the average of them
void Student::calcavg()
{
        float small;
        small = ((m1<=m2)?((m1<=m3)?m1:m3):((m2<=m3)?m2:m3));
        avg = (float)((m1+m2+m3) - small)/2;
}
// Function to display the student details
void Student :: display()
{
        cout<<usn<<"\t"<<name<<"\t"<<avg<<endl;
}
int main()
{
        Student st[10];
        int i,n;
        cout<<"Enter the number of students\n";
        cin>>n;
        for(i=0;i<n;i++)
        {
                st[i].readstudent();
        }
        for(i=0;i<n;i++)
        {
                st[i].calcavg();
```

```
        }
        cout<<"USN \t Student Name \t Average \n";
        for(i=0;i<n;i++)
        {
                st[i].display();
        }
        return 0;
}
```

7a)Describe function templates and inline functions with an example for each

A generic function defines a general set of operations that will be applied to various types of data. The type of data that the function will operate upon is passed to it as parameter.

A generic function is created using the keyword “template”. It is used to create a framework that describes what a function will do, leaving it to the compiler to fill in the details as needed.

Syn: template<class T>

Return_type func_name(parameter list)

{

Body of function

}

Ex:

```
template <class A_Type> class calc{
 public:
A_Type multiply(A_Type x, A_Type y);
A_Type add(A_Type x, A_Type y);
};
template <class A_Type>
A_Type calc<A_Type>::multiply(A_Type x,A_Type y)
```

```
{
return x*y;
}
template <class A_Type>
A_Type calc<A_Type>::add(A_Type x, A_Type y)
{
return x+y;
}
```

Inline Functions:

If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.

To inline a function, place the keyword **inline** before the function name and define the function before any calls are made to the function. The compiler can ignore the inline qualifier in case defined function is more than a line.

Ex:

```
#include <iostream>

using namespace std;

inline int Max(int x, int y)
{
  return (x > y)? x : y;
}

// Main function for the program
int main( )
{

  cout << "Max (20,10): " << Max(20,10) << endl;
  cout << "Max (0,200): " << Max(0,200) << endl;
  cout << "Max (100,1010): " << Max(100,1010) << endl;
  return 0;
}
```

7. b)Write a program using function template to swap integers and floats.

```
#include<iostream>
using namespace std;
template <class X> void swapargs(X &a, x &b)
{
    X temp;
    temp=a;
    a=b;
    b=temp;
}
int main()
{
    int i=10,j=20;
    double x=10.5,y=90.8;
    cout<<"Original i,j:"<<i<<' '<<j<<'\n';
    cout<<"Original x,y:"<<x<<' '<<y<<'\n';
    swapargs(i,j);
    swapargs(x,y);
    cout<<"Swapped  i,j:"<<i<<' '<<j<<'\n';
    cout<<"Swapped x,y:"<<x<<' '<<y<<'\n';
    return 0;
}
```

6.

8. a) Explain function overloading in detail with an example.

A function with same name and multiple definitions is called function overloading.

Function overloading is usually used to enhance the readability of the program. If we have to perform one single operation but with different number or types of arguments, then we can simply overload the function.

There are two ways to overload a function

*By change in number of arguments

* By change in type of arguments

***By change in number of arguments:**

In this type of function overloading we define two functions with same names but different number of parameters of the same type

Ex:

```
int sum (int x, int y)
{
cout << x+y;
}
int sum(int x, int y, int z)
{
cout << x+y+z;
}
int main()
{
sum (10,20);  // sum() with 2 parameter will be called
sum(10,20,30);  //sum() with 3 parameter will be called
}
```

*** By change in type of arguments**

In this type of overloading we define two or more functions with same name and same number of parameters, but the type of parameter is different.

Ex:

```
int sum(int x,int y)
{
cout<< x+y;
```

```
}
double sum(double x,double y)
{
cout << x+y;
}
int main()
{
sum (10,20);
sum(10.5,20.5);
}
```

8. b) Write a program to overload a function volume to display the volume of different shapes like cube, cylinder and rectangular box

```
#include<iostream>
using namespace std;

int volume(int n)
{
        return n*n*n;
}
double volume(double r, double h)
{
        Return3.14*r* r*h;
}
double volume(int ra,int rb)
{
        return ra*ra*ra;
}
int main()
{
        int s,ra,rb;
        double r,h;
        cout<<" Enter the value of side in Integer to calculate the volume of cube:\n";
        cin>>s;
        cout<<" Volume of Cube is :"<<volume(s)<<endl;
        cout<<" Enter the value of radius and height in Double to calculate the volume of
cylinder:\n";
        cin>>r>>h;
        cout<<" Volume of Cylinder is :"<<volume(r,h)<<endl;
        cout<<" Enter the value of sides in Double to calculate the volume of Rectangle:\n";
```

```
        cin>>ra>>rb;
        cout<<" Volume of Ractangle is :"<<volume(ra,rb)<<endl;
        return 0;
}
```