CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

Internal Assesment Test I- November 2019

Sub:	Object Oriented Programming Using C++							Code:	18MCA11
Date:	16.11.2019	Duration:	90 mins	Max Marks:	50	Sem:	I	Branch:	MCA

		Marks	OBE CO	RBT
	Part - I			
1.	What is Object Oriented Programming? Bring out the underlying concepts of Oops?	[10]	CO1	L2
	OR			
2.	What is a class? Explain the structure of a class with an example.	[10]	CO2	L1
	Part - II			
3 (a)	Explain the concepts of constructors and destructors with example for each.	[5]	CO2	L2
(b)	What is copy Constructor? Explain with one suitable example.	[5]	CO2	L2
	OR			
4 (a)	Write a program to show the concept of Constructor Overloading.	[6]	CO3	L2
(b)	What is reference variable? Explain with an example.	[4]	CO3	L2
5 (a)	Explain function overloading in detail with an example.	[4]	CO2	L2
(b)	Write a program to overload a function area to display the area of different shapes like cube, cylinder and sphere.	[6]	CO3	L3
	OR			
6.	What is static data member and static member function? Explain with example.	[10]	CO2	L2
	Part - IV			
7.	Describe inline functions with an example. Write a program to find the largest, smallest & second largest of three numbers. (use inline function MAX and MIN to find largest & smallest of 2 numbers)	[10]	CO3	L3
	OR			
8.	What is function template? Write a program using function template to swap integers and floats.	[10]	CO3	L2
	Part - V			
9 (a)	Write a program to pass object as an argument to function.	[6]	CO3	L3
(b)	Write a program to add two numbers using default arguments.	[4]	CO3	L3
	OR			
10.	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.	[10]	CO4	L3

Internal Assesment Test I- November 2019
Answer Key

Sub: Object Oriented Programming Using C++ Code: 18MCA11

1. What is Object Oriented Programming? Bring out the underlying concepts of Oops? [10]

Ans: 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

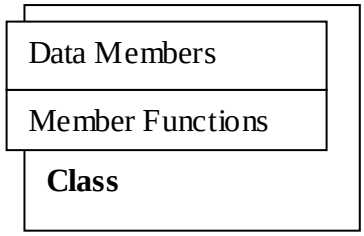
objectname.functionname(message);

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. What is a class? Explain the structure of a class with an example. [10]

Ans: 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.

3 (a)Explain the concepts of constructors and destructors with example for each. [5]

Ans: 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;
}
```

(b)What is copy Constructor? Explain with one suitable example. [5]

Ans: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.

4a)Write a program to show the concept of Constructor Overloading. [6]
Ans:

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

class Human{
int Head,Legs,Hands;

 public: Human()
        {
                Head=1;
                Legs=2;
                Hands=2;

        }
        Human(int n1,int n2,int n3=5)
        {
                Head=n1;
                Legs=n2;
                Hands=n3;


        }
```

```
        Human(Human&);
        void Display()
        {
                cout<<"Head="<<Head<<endl;
                cout<<"Legs="<<Legs<<endl;
                cout<<"Hands="<<Hands<<endl;
        }
        ~Human()
        {
                 cout<<"Distructor Executing"<<endl;
                 cout<<"Human Died"<<endl;
        }
};
 Human :: Human(Human &h1)
{
        Head=h1.Head;
        Legs= h1.Legs;
        Hands=h1.Hands;
}
int main()
{
        Human Human1;
        Human Human2(2,4,2);
        Human Human3(3,6);
        Human Human4(Human1);
        cout<<"Object with Default Constructor"<<endl;
        Human1.Display();
        cout<<"Object with Parameterized Constructor"<<endl;
        Human2.Display();
        cout<<"Object with Parameterized Constructor with default argument "<<endl;
        Human3.Display();
        cout<<"Copy Constructor"<<endl;
        Human4.Display();

}
```

4.b) What is reference variable? Explain with an example. [4]

Ans: A reference is essentially an implicit pointer. There are three ways that a reference can be used: as a function parameter, as a function return value, or as a stand-alone reference.

(i) As a function parameter:Probably the most important use for a reference is to allow you to create functions that automatically use call-by-reference parameter passing.

Call-by-reference passes the address of the argument to the function.

Example:

```
// Use a reference parameter.
#include <iostream>
using namespace std;
void neg(int &i); // i now a reference
int main()
{
int x;
x = 10;
cout << x << " negated is ";
neg(x); // no longer need the & operator
cout << x << "\n";
```

```
return 0;
}
void neg(int &i)
{
i = -i; // i is now a reference, don't need *
}
```

5 (a)Explain function overloading in detail with an example [4]

Ans: 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);
}
```

(b)Write a program to overload a function area to display the area of different shapes like cube, cylinder and sphere. [6]

Ans:

```
#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(double ra)
{
        return 0.33*3.14*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 radius in Double to calculate the volume of Sphere:\n";
        cin>>ra>>rb;
        cout<<" Volume of Sphere is :"<<volume(ra)<<endl;
        return 0;
}
```

6. What is static data member and static member function? Explain with example. [10]

Ans: Static Data Members

When you precede a member variable's declaration with static, you are telling the compiler that only one copy of that variable will exist and that all objects of the class will share that variable. Unlike regular data members, individual copies of a static member variable are not made for each object. No matter how many objects of a class are created, only one copy of a static data member exists. Thus, all objects of that class use that same variable. All static variables are initialized to zero before the first object is created. When you declare a static data member within a class, you are not defining it. (That is, you are not allocating storage for it.) Instead, you must provide a global definition for it elsewhere, outside the class. This is done by redeclaring the static variable using the scope resolution operator to identify the class to which it belongs. This causes storage for the variable to be allocated. (Remember, a class declaration is simply a logical
construct that does not have physical reality.)

Example:

```
#include <iostream>
using namespace std;
class shared {
static int a;
int b;
```

```
public:
void set(int i, int j) {a=i; b=j;}
void show();
} ;
int shared::a; // define a
void shared::show()
{
cout << "This is static a: " << a;
cout << "\nThis is non-static b: " << b;
cout << "\n";
}
int main()
{
shared x, y;
x.set(1, 1); // set a to 1
x.show();
y.set(2, 2); // change a to 2
y.show();
x.show(); /* Here, a has been changed for both x and y
because a is shared by both objects. */
return 0;
}
```

This program displays the following output when run.

```
This is static a: 1
This is non-static b: 1
This is static a: 2
This is non-static b: 2
This is static a: 2
This is non-static b: 1
```

Static Member Function:

Member functions may also be declared as static. There are several restrictions placed on static member functions. They may only directly refer to other static members of the class. (Of course, global functions and data may be accessed by static member functions.)

A static member function does not have a this pointer. There cannot be a static and a non-static version of the same function. A static member function may not be virtual. Finally, they cannot be declared as const or volatile.

Example:

```
 #include <iostream>
using namespace std;
class cl {
static int resource;
public:
static int get_resource();
void free_resource() { resource = 0; }
};
int cl::resource; // define resource
int cl::get_resource()
{
if(resource) return 0; // resource already in use
else {
resource = 1;
return 1; // resource allocated to this object
}
```

```
}
int main()
{
cl ob1, ob2;
/* get_resource() is static so may be called independent
of any object. */
if(cl::get_resource()) cout << "ob1 has resource\n";
if(!cl::get_resource()) cout << "ob2 denied resource\n";
ob1.free_resource();
if(ob2.get_resource()) // can still call using object syntax
cout << "ob2 can now use resource\n";
return 0;
}
```

7. Describe inline functions with an example. Write a program to find the largest, smallest & second largest of three numbers. (use inline function MAX and MIN to find largest & smallest of 2 numbers) [10]

Ans: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;
}
```

Program:

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

inline int Max(int a,int b)
{
        if(a>b)
        return a;
        else
        return b;
}
inline int Min(int a,int b)
{
```

```
        if(a<b)
        return a;
        else
        return b;
}
int main()
{
        int a,b,c;
        cout<<" Enter three numbers:";
        cin>>a;
        cin>>b;
        cin>>c;
        int Res= Max(a,b);
        int Res1=Max(Res,c);
        cout<<"Largest Number is :"<<Res1<<endl;
        Res= Min(a,b);
        int Res2=Min(Res,c);
        cout<<"Smallest Number is :"<<Res2<<endl;
        //if(a==Res1 || a==Res2)
        //{
        //if(b==Res1 || b==Res2)
        cout<<"Second Largest Number is:"<<(a+b+c)-Res1-Res2<<endl;
        //else
        //cout<<"Second Largest Number is:"<<b<<endl;
        //}
        //else
        //cout<<"Second Largest Number is:"<<a<<endl;
        return 0;
}
```

8. What is function template? Write a program using function template to swap integers and floats. [10]

Ans: 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 a parameter. Through a generic function, a single general procedure can be applied to a wide range of data. Many algorithms are logically the same no matter what type of data is being operated upon. For example, the Quick sort sorting algorithm is the same whether it is applied to an array of integers or an array of floats. It is just that the type of the data being sorted is different. By creating a generic function, you can define the nature of the algorithm, independent of any data. Once you have done this, the compiler will automatically generate the correct code for the type of data that is actually used when you execute the function. In essence, when you create a generic function you are creating a function that can automatically overload itself. A generic function is created using the keyword template. The normal meaning of the word "template" accurately reflects its use in C++. It is used to create a template (or framework) that describes what a function will do, leaving it to the compiler to fill in the details as needed. The general form of a template function definition is shown here:

```
template <class Ttype> ret-type func-name(parameter list)
{
// body of function
}
#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;
}
```

9 (a)Write a program to pass object as an argument to function. [6]

Ans: Objects may be passed to functions in just the same way that any other type of variable can. Objects are passed to functions through the use of the standard call-by value mechanism.

```
// Passing an object to a function.
#include <iostream>
using namespace std;
class myclass {
int i;
public:
myclass(int n);
~myclass();
void set_i(int n) { i=n; }
int get_i() { return i; }
};
myclass::myclass(int n)
{
i = n;
cout << "Constructing " << i << "\n";
}
myclass::~myclass()
{
cout << "Destroying " << i << "\n";
}
void f(myclass ob);
int main()
{
myclass o(1);
f(o);
cout << "This is i in main: ";
cout << o.get_i() << "\n";
return 0;
}
void f(myclass ob)
{
ob.set_i(2);
cout << "This is local i: " << ob.get_i();
cout << "\n";
```

```
}
```
This program produces this output:
```
Constructing 1
This is local i: 2
Destroying 2
This is i in main: 1
Destroying 1
```
(b)Write a program to add two numbers using default arguments. [4]

```
#include <iostream>
using namespace std;
  void sum(int a,int b= 6);
  int main( )
  {
     int a,b;
     cout<<"enter any two numbers\n";
     cin>>a>>b;
     sum(a) ; // sum of default values
     sum(a,b);
     sum(b);
     return 0;
  }
  void sum (int a1, int a2)
  {
     int temp;
     temp = a1 + a2;
     cout<<"a="<<a1<<endl;
     cout<<"b="<<a2<<endl;
     cout<<"sum="<<temp<<endl;
  }
```

O/P
```
enter any two numbers
2 3
a=2
b=6
sum=8
a=2
b=3
sum=5
a=3
b=6
sum=9
```

10 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. [10]

```
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;
}
```