CMR
INSTITUTE OF
TECHNOLOGY

Scheme Of Evaluation
Internal Assessment Test II – May. 2017

Sub:	Programming in C++	Code:	10EC665
Date: 10/05/2017	Duration: 90mins	Max Marks: 50	Sem: VI	Branch: ECE/TCE

Note: Answer Any Five Question. All questions carry equal marks

Question #		Description	Marks Distribution		Max Marks
1	a)	**Q:** Define **inline** function with example.. Definition of inline function Example code Drawbacks of inline function	2 M 2 M 1 M	5 M	10 M
	b)	**Q:** Define **local class, global class, local object & global object.** Definition of all 4 = 1x4 Elaboration/Example	4 M 1 M	5 M	
2	a)	**Q:** What is static data member and static member function? When do we declare member of class static?**.** Definition 2 + 2 Application	4 M 1 M	5 M	10 M
	b)	**Q:** Explain with example, friend function acting as a bridge: friend function example code	2 M 3 M	5 M	
3		**Q:** Define a class of employee with members empid, name and salary. Create **array of objects** for 100 employees & write user defined function to read and display their records **.** Definition of class of employee creating array of object Step-wise correctness	4 M 2 M 4 M	10 M	10 M
4	a)	What is **this pointer**? Explain this pointer with example program Definition Correctness of example program	2 M 3 M	5 M	10 M
	b)	What is recursive function? Write a recursive function to find fibonacci series. Definition of recursive function Correctness of function	1 M 4 M	5 M	
5		**Q:** Write a C++ program using **function overloading** to overload area function to find area of circle, rectangle & triangle (by 3 sides)**.**	5 M	10 M	10 M

		Correctness of logic (step-wise) Correctness of program (step-wise)	5 M		
6		**Q:** Explain default **constructor, parameterized constructor & copy constructor** with example program**.** Definition/comments in program Copy constructor definition & call Step-wise correctness of program/logic	3 M 3 M 4 M	10 M	10 M
7		**Q:** What do you mean by **operators overloading**? Write a program to explain **overloading of + operator** to add two complex numbers**.** Definition of Operator Overloading Correctness of adding complex numbers Correctness of logic/program (step-wise)	2 M 2 M 6 M	10 M	10 M
8		**Q:** Write a C++ program to overload ++ increment operator (**pre & post increment both**) so that when it is used with object, all data of the object should be incremented by 1**.** Correctness of logic (step-wise) Correctness of program (step-wise)	5 M 5 M	10 M	10 M

CMR
INSTITUTE OF
TECHNOLOGY

Solution/Model Answer of IAT-2 (May 2017)
Programming in C++ – 10EC665/61
VI sem – ECE/TCE Elective
Dr. P. N. Singh, Professor(CSE)

1a) Define **inline** function with example. 5
Ans:

Inline function tells compiler to replace each call to the function with explicit code for the function.
During execution the function call is replaced by actual function definition(code). When the compiler replaces the inline function call with the function's actual code, we say that it **expands** the inline function. Function code, defined within a class are by default inline.

But there are some drawbacks of inline functions.

We can't use looping statements while, do..while and for statements in inline function.
goto, break and continue are also not supported by inline functions.
It is not useful where the function is called more than once. If an inline function has 40 line code and it is called 26 times, it would add 1000 lines of unnoticed source code of the program.

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

inline cube(int x)
{
    return (x)*(x)*(x);
}

main( )
{

    cout << "Cube of 5 = " << cube(5)<<endl;
    cout << "Cube of (5+2) = " << cube(5+2)<<endl;
    return 0;
}

/*The program will actually be compiled as though it were in the main( ) */

main( )
{

    cout << "Cube of 5 = " << (5) * (5) *(5)<<endl;
    cout << "Cube of (5+2) = " << (5+2)*(5+2)*(5+2)<<endl;
    return 0;
}
```

1b) Define **local class, global class, local object & global object** **5**

Ans:

Local class : A class declared inside of a function is known as local class. Its object can be declared within the function only. (Locally available to the function only)

In the given example Test is class declared within function fun.

```
#include<iostream>
using namespace std;
void fun()
{
     class Test  // local to fun
     {
       /* members of Test class */
     };
}
int main()
{
     // code
    return 0;
}
```

Global class : A class declared outside of a function is known as global class. Its object can be declared globally outside of functions or locally within the function.

In Here class C1 is declared outside of the main()

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

class C1
{
       /* members of Test class */
};

C1 globalobject;
int main()
{
    C2 localobject;
   // code
    return 0;
}
```

Local & global object: Objects of global class may be declared globally outside of a function (global object) or locally inside a function (local object). Global objects can be accessed by any of the function and local objects are locally available to the function only in which it is declared.

In the previous/given example *globalobject* is a global object and *localobject* is local.

```
#include<iostream>
using namespace std;
class C1
{
       /* members of C1 class */
};

C1 globalobject;  //global object
int main()
{
   C2 localobject;  //local object
   // code
   return 0;
}
```

2a) What is static data member and static member function? When do we declare member of class static? 5

Ans:

Static members (data or function):

- ➔ static members are class variables
- ➔ static members are class specific & not object specific
- ➔ static members are declared before any/all objects (or no objects are there) so content does not depend on object
- ➔ Unique value of static members are shared by all objects(member of any object)
- ➔ static members have properties like public/global variables but it enjoys class scope.
- ➔ Prototype of static members are declared within class but definition/initialization is done outside of the class
- ➔ **Object.static member is not good. Classname::staticmember is elegant**
- ➔ **static member function can refer/use only static data mebers**
- ➔ **static member function is called classname::staticmemberfunction() and not (should not) by object using member operator . .**

```
/* Example program – static data and static member function */
#include <iostream.h>

class box {

    double length,breadth,height;
    public:
    static int count; // static member
    box(double l, double b, double h) {   // constructor
        length = l;     breadth = b;  height = h;
        count++;  // as the objects are declared count is incremented
    }
    static void objects() {   // static member function
        cout << "Total objects = "<<objectCount<<"\n";
    }
    void volume() {
        cout << "from static member function Volume = "
            << length * breadth * height << " cubic unit\n";
    }

  };

int box::count = 0; // Initialize static member of class Box

int main( ) {
  box b1(3.3, 1.2, 1.5), b2(8.5, 6.0, 2.0);  // b1 & b2 are objects of class box
  b1.Volume();   b2.Volume();
  box::objects();   // calling static member function to print total number of objects.
  // or calling like a global
  cout << "from main Total objects = " << box::objectcount<<"\n";
  return 0;
}
```

2b) Explain with example, friend function acting as a bridge **5**

Ans:

- ➔ A friend function is a non-member function which is used to access private data members of a class.
- ➔ A friend function is declared within a class with address arguments of class objects and defined as general function outside of the class to access private members.
- ➔ A friend function has all the privileges of a member function without actually being a member function of the class.
- ➔ This attribute of friend function is used mostly with overloaded operators.
- ➔ The keyword to declare a friend function is **friend**.

Prototype to declare a friend function

friend reurntype functionname(class1name &, type, …);

A friend may be friend of two classes. In this situation it acts as a bridge between two classes to access or modify or swap private data members of two classes.

/*example program – friend function of two classes */

```
#include<iostream.h>

class class_2;

class class_1
{
         int value1;
         public:
        void indata(int a) {  value1 = a;     }
         void display() {     cout << value1 << "\n"; }
        friend void exchange(class_1 &, class_2 &); // friend function of class_1 & class_2
  };

  class class_2
  {
          int value2;
          public:
          void indata(int a) {   value2 = a;  }
          void display() { cout << value2 << "\n";}
            friend void exchange(class_1 & , class_2 &); // friend function of class_1 & class_2

    };

    void exchange(class_1 &x, class_2 &y)
      {
         int temp = x.value1;
         x.value1 = y.value2;
         y.value2 = temp;
      }
    main()
    {
         int ss,yy;
         class_1 c1;
         class_2 c2;
         cout << "Enter first & second number : " ;
         cin >> ss >> yy;
         c1.indata(ss);
         c2.indata(yy);
         cout << "value before exchange = \n" ;
         c1.display();
         c2.display();
         exchange(c1,c2);   // calling friend function
         cout << " Value after exchange = \n";
         c1.display();
         c2.display();
         return(0);
     }
```

3 Define a class of employee with members empid, name and salary. Create array of objects for 100 employees & write user defined function to read and display their records 10

Ans:

```
//Array of employee objects
#include <iostream>
using namespace std;
class employee
{
        int empid;
        char name[30];
        long int salary;
        public:
        void getdata()
        {
                cout << “Enter employee id, name and salary :”;
                cin >> empid >> name >> salary;
        }
        void display()
        {
                cout << empid << “ “ << name << salary <<”\n”;
        }
};

int main()
{
        employee e[100]; // array of objects
        for(int x = 0; x<100; x++)
                e[x].getdata();
        for( x = 0; x<100; x++)
                e[x].display();
        return 0;
}
```

4a) What is *this* pointer? Explain this pointer with example program. 5

Ans:

this is a pointer that keeps the address of the object which calls the member function. All the member function has one argument by default i.e. address of the object which calls the member function.

“this pointer points to owner object.”

It is beneficial to overload operators with one less argument because address of calling object is already there in this pointer.

```
//Example program – this pointer
#include <iostream>
using namespace std;

class c1
{
        //private members
        public:
          // other member function
        void func()     { cout << “This time called by “ << this << “\n”; }
};
int main()
{
        c1 ob1, ob2;
        ob1.func();
        ob2.func();
        // other part of code
        return 0;
}
```

4b) What is recursive function? Write a recursive function to find fibonacci series. 5

Ans:

Recursion is a process when a function calls itself. The function has recursion is known as recursive function. One simple example for recursive function is to find factorial as
factorial(n) = n*factorial(n-1);

```
//recursive function to find Fibonacci series
int fibo(int n)
{
        if(n==1) return 0;
        else if(n==2) return 1;
        else return fibo(n-1)+fibo(n-2);
}
```

5) Write a C++ program using **function overloading** to overload area function to find area of circle, rectangle & triangle (by 3 sides) **10**

```
#include <iostream.h>
#include <math.h>

class shape
{
        float l,w,rad,a,b,c;
        public:
        void area(float len, float wid)
        {
                l=len;
                w=wid;
                cout << “Area of rectangle = “ << l*w << “ square unit\n”;
        }
        void area(float radius) // area function is overloaded
        {
                r=radius;
                cout << “Area of circle = “ << 3.14*r*r << “ square unit\n”;
        }
        void area(float sidea, float sideb, float sidec) // area function is overloaded
        {
                a=sidea;
                b=sideb;
                c=sidec;
                float s=(a+b+c)/2.00;
                if(s<=a || s <=b || s<=c)
                        cout << “Wrong measurement\n”;
                else
                        cout << “Area of triangle = “ << sqrt(s*(s-a)*(s-b)*(s-c)) << “ square unit\n”;
        }
};

int main()
{
        shape rectangle, circle, triangle;
        rectangle.area(6.7,3.5);
        circle.area(8.97);
         triangle.area(6.7,5.4,4.5);
        return 0;
 }
```

6. Explain default **constructor, parameterized constructor & copy constructor** with example program.10
Ans:

Constructor:

- ➔ Constructor is a member function having the same name as class.
- ➔ Constructor member function is called automatically as the objects are declared in memory
- ➔ There is no return type of constructor (It does not return any value)
- ➔ Constructor is used to initialize the object's data

Default constructor: Default constructor does not take any argument.
Parameterized constructor: Parameterized constructor has arguments. Some of the objects may have existing data.
Copy constructor: Copy constructor creates an object by initializing it with an object of the same class, which has been created previously. Default constructor & Copy constructor are default/in-built feature of C++ but programmer can also define a copy constructor to initialize one object from another of the same type.

```
//Example program with default constructor, parameterized constructor & copy constructor
#include <iostream.h>
#include <conio.h>

class rectangle
{
	 int l,w;
	public:
	rectangle(){} // default constructor
	rectangle(int len, int wid) // parameterized constructor
	{
		l=len;
		w=wid;
	}
	rectangle(rectangle &ob)  // copy constructor
	{
		l=ob.l;
		w=ob.w;
	}
	void getdime()
	{
		cout << "Enter length and width : ";
		cin >> l >> w;
	}
	void area()
	{
		  cout << "Area = " << l*w << " sq. unit\n";
	}
};

main()
{
	rectangle r1,r2(7,5),r3;
	r1.getdime();
	r1.area();
	r2.area();
	r3.getdime();
	r3.area();
	rectangle r4(r3);
	 r4.area();
	return 0;
 }
```

7) What do you mean by operators overloading? Write a program to explain overloading of + operator to add two complex numbers. 10

Ans:
Operators can be overloaded for extra task. Most of the operators can be overloaded in C++. They will also perform its built-in tasks.
To overload an operator keyword is **operator** followed by operator symbol in operator overloaded member function as:

```
type operator symbol(type arg1, type arg2,..)
 {
        // code
}
```

```
// program to overload + to add two complex numbers

#include<iostream.h>

class sample
{
        float real, imag;
        public:

        void getdata ()
        {
                cout<<"\n Enter real and imaginary value of complex number";
                cin>>real>>imag;
        }

        sample operator + (sample obj) // Operator overloading
        {
                sample temp;
                temp.real= real+obj.real;
                temp.imag =imag+obj.imag;
                return temp;
        }

        void display ()
        {
                cout<<"\n Additional of two complex numbers = ";
                cout<<real<<"+i"<<imag;
        }
};

void main ()
{
        Sample obj1, obj2, obj3;
        obj1. getdata ();
        obj2.getdata();
        obj3=obj1+obj2;
        obj3.display ();
}
```

8) Write a C++ program to overload ++ increment operator (pre & post increment both) so that when it is used with object, all data of the object should be incremented by 1. **10**
Ans:

```
// overloading ++ increment operator
#include <iostream.h>

class abc
{
         int x;
        public:
        abc(){x=1;}
        void operator ++()
        {
                x = x+1;
        }

        void operator ++(int) // for postfix increment
        {
                x = x+1;
        }

        void display()
        {
                cout << "x = " << x << "\n";
        }
 };

 main()
 {
        abc o1;
        o1.display();
        ++o1;
        o1.display();
        o1++;
        o1.display();
        return 0;
  }
```