Roll No :

KLNCIT	**CLASS TEST QUESTION**	Format No.:ACD11A-I
		Issue No. :01
		Rev No. :00

Subject Code/Subject Name : CS6301 – Programming and Datastructures –II **Class test No. : I**

Year and Branch : II & IT **Total marks : 25**

Date : 18.07.2017 **Duration : 50mins**

I. Course outcomes, Question Number, Marks

COs	CO1	CO2	CO3	CO4	CO5
Q.Nos	1,2,3,4,5,6(a)/6(b)				
Marks (Max)	25				

II. Knowledge skill outcomes

Level	Remember (K1)	Understand (K2)	Apply (K3)	Analysis (K4)	Evaluate (K5)	Create (K6)
Q.Nos	1,2,5	3,4,6(a)/6(b)				
Marks (Max)	6	4,15/15 19/19				

PART – A

5 × 2 = 10 Marks

Answer all the questions

1. Define Class and object with its syntax. **(K1)**

2. What are the applications of Object Oriented Programming? **(K1)**

3. Classify the Access Specifiers with an example. **(K2)**

4. Describe the significance of declaring member of a class static. **(K2)**

5. List the features of Object Oriented Programming. **(K1)**

PART – B

1 × 15 = 15 Marks

6. (a) Illustrate with an example about constructor and Destructor. **(K2)**

(OR)

(b) Explain the concept of Pointers with an example. **(K2)**

COURSE COORDINATOR **ACADEMIC COORDINATOR** **HOD\IT**

I – CLASS TEST KEY

Subject Code:CS6301 **Class Test No:I**

Subject Name:Programming and Datastructures II **Total Marks:25**

Year & Branch : II B.Tech IT **Duration: 50mins**

Date:18.07.2017

PART – A

5 × 2 = 10 Marks

Answer all the questions

1. Define Class and object with its syntax. **(K1)**

Class:

Class is a collection of data and member function.Class is C++ is a natural evolution of struct in C.

Syntax:

```
Class classname
 {
        Access specifier:
                       Datatype var1,var2,…..,varn;
        Access specifier:
                      Function declaration or Definition;
 };
```

Object:

Object is a instance of a class.

Objects are the basic run time entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle.

Syntax:

```
Classname Objetname;
```

2. What are the applications of Object Oriented Programming? **(K1)**

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

3. Classify the Access Specifiers with an example. **(K2)**

Public - The members declared as Public are accessible from outside the Class through an object of the class.

Protected - The members declared as Protected are accessible from outside the class but only in a class derived from it.

Private - These members are only accessible from within the class. No outside Access is allowed.

An Source Code Example:

```
class MyClass
{
   public:
      int a;
   protected:
      int b;
   private:
      int c;
};
int main()
{
   MyClass obj;
   obj.a = 10;     //Allowed
   obj.b = 20;     //Not Allowed, gives compiler error
   obj.c = 30;     //Not Allowed, gives compiler error
}
```

4. Describe the significance of declaring member of a class static. (K2)

Static Data Member: It is generally used to store value common to the whole class. The **static** data member differs from an ordinary data member in the following ways:

(i) Only a single copy of the static data member is used by all the objects.
(ii) It can be used within the class but its lifetime is the whole program.

For making a data member static, we require:

(a) Declare it within the class.
(b) Define it outside the class.

5. List the features of Object Oriented Programming. (K1)

- Objects
- Classes
- Data abstraction and encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message passing

PART – B 1 × 15 = 15 Marks

6. (a) Illustrate with an example about constructor and Destructor. (K2)

A **constructor** (having the same name as that of the class) is a member function which is automatically used to initialize the objects of the class type with legal initial values. Destructors are the functions that are complimentary to constructors. These are used to de-initialize objects when they are destroyed. A destructor is called when an object of the class goes out of scope, or when the memory space used by it is de allocated with the help of **delete** operator.

Declaration and Definition of a Constructor:-

It is defined like other member functions of the class, i.e., either inside the class definition or outside the class definition.

For example, the following program illustrates the concept of a constructor :

```
//To demonstrate a constructor
#include <iostram.h>
#include <conio.h>
Class rectangle
{
private :
```

```
float length, breadth;
public:
rectangle ()//constructor definition
{
//displayed whenever an object is created
cout<<"I am in the constructor";
length-10.0;
breadth=20.5;
}
float area()
{
return (length*breadth);
}
};
void main()
{
clrscr();
rectangle rect; //object declared
cout<<"\nThe area of the rectangle with default parameters is:"<<rect.area()<<"sq.units\n";
getch();
}
```

SPECIAL CHARACTERISTICS OF CONSTRUCTORS

These have some special characteristics. These are given below:

(i) These are called automatically when the objects are created.
(ii) All objects of the class having a constructor are initialized before some use.
(iii) These should be declared in the public section for availability to all the functions.
(iv) Return type (not even **void**) cannot be specified for constructors.
(v) These cannot be inherited, but a **derived** class can call the base class constructor.
(vi) These cannot be static.
(vii) Default and copy constructors are generated by the compiler wherever required. Generated constructors are public.
(viii) These can have default arguments as other C++ functions.
(ix) A constructor can call member functions of its class.
(x) An object of a class with a constructor cannot be used as a member of a **union.**
(xi) A constructor can call member functions of its class.
(xii) We can use a constructor to create new objects of its class type by using the syntax.

Name_of_the_class (expresson_list)
For example,
Employee obj3 = obj2; // see program 10.5
Or even
Employee obj3 = employee (1002, 35000); //explicit call

(xiii) The make **implicit** calls to the memory allocation and deallocation operators **new** and **delete.**
(xiv) These cannot be **virtual.**

Declaration and Definition of a Destructor

The syntax for declaring a destructor is :

```
-name_of_the_class()
{
}
```

So the name of the class and destructor is same but it is prefixed with a ~
(tilde). It does not take any parameter nor does it return any value. Overloading a destructor is not possible and can be explicitly invoked. In other words, a class can have only one destructor. A destructor can be defined outside the class. The following program illustrates this concept :

```
//Illustration of the working of Destructor function
```

```
#include<iostream.h>
#include<conio.h>
class add
{
private :
int num1,num2,num3;
public :
add(int=0, int=0); //default argument constructor
//to reduce the number of constructors
void sum();
void display();
~ add(void); //Destructor
};
//Destructor definition ~add()
Add:: ~add(void) //destructor called automatically at end of program {
Num1=num2=num3=0;
Cout<<”\nAfter the final execution, me, the object has entered in the”
<<”\ndestructor to destroy myself\n”;
}
//Constructor definition add()
Add::add(int n1,int n2)
{
num1=n1;
num2=n2;
num3=0;
}
//function definition sum ()
Void add::sum()
{
num3=num1+num2;
}
//function definition display ()
Void add::display ()
{
Cout<<”\nThe sum of two numbers is “<<num3<<end1;
}
void main()
{
Add obj1,obj2(5),obj3(10,20): //objects created and initialized clrscr();
Obj1.sum(); //function call
Obj2.sum();
Obj3.sum();
cout<<”\nUsing obj1 \n”;
obj1.display(); //function call
cout<<”\nUsing obj2 \n”;
obj2.display();
cout<<”\nUsing obj3 \n”;
obj3.display();
}
```

5.6 Special Characteristics of Destructors

Some of the characteristics associated with destructors are :

(i) These are called automatically when the objects are destroyed.
(ii) Destructor functions follow the usual access rules as other member functions.
(iii) These **de-initialize** each object before the object goes out of scope.
(iv) No argument and return type (even void) permitted with destructors.

(v) These cannot be inherited.
(vi) **Static** destructors are not allowed.
(vii) Address of a destructor cannot be taken.
(viii) A destructor can call member functions of its class.
(ix) An object of a class having a destructor cannot be a member of a union.

(OR)

(b) Explain the concept of Pointers with an example. (K2)

<u>Pointers</u>

- Pointers and Character Strings
- Pointer Arithmetic
- Pointers to Function
- Pointer to Object
- Pointer to Constant
- Constant Pointer

<u>Address-of operator (&)</u>

The address of a variable can be obtained by preceding the name of a variable with an ampersand sign (&), known as *address-of operator*. For example:

```
foo = &myvar;
```

<u>Dereference operator (*)</u>

A variable which stores the address of another variable is called a *pointer*. Pointers are said to "point to" the variable whose address they store.

```
baz = *foo;
```

The reference and dereference operators are thus complementary:

& is the *address-of operator*, and can be read simply as "address of"
* is the *dereference operator*, and can be read as "value pointed to by"

<u>Declaring pointers</u>

The declaration of pointers follows this syntax:

```
type * name;
```

<u>Pointers and arrays</u>

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;
```

<u>Pointer initialization</u>

Pointers can be initialized to point to specific locations at the very moment they are defined:

```
int myvar;
int * myptr = &myvar;
```

<u>Pointer arithmetics</u>

To conduct arithmetical operations on pointers is a little different than to conduct them on regular integer types. To begin with, only addition and subtraction operations are allowed; the others make no sense in the world of pointers. But both addition and subtraction have a slightly different behavior with pointers, according to the size of the data type to which they point.

Suppose now that we define three pointers in this compiler:

```
char *mychar;
short *myshort;
long *mylong;
```

and that we know that they point to the memory locations 1000, 2000, and 3000, respectively. Therefore, if we write:

```
++mychar;
++myshort;
++mylong;
```

Pointers and const

Pointers can be used to access a variable by its address, and this access may include modifying the value pointed. But it is also possible to declare pointers that can access the pointed value to read it, but not to modify it. For this, it is enough with qualifying the type pointed to by the pointer as const. For example:

```
int x;
int y = 10;
const int * p = &y;
x = *p;          // ok: reading p
*p = x;          // error: modifying p, which is const-qualified
```

Pointers to functions

C++ allows operations with pointers to functions. The typical use of this is for passing a function as an argument to another function. Pointers to functions are declared with the same syntax as a regular function declaration, except that the name of the function is enclosed between parentheses () and an asterisk (*) is inserted before the name:

```
int (* minus)(int,int) = subtraction;
```