Department of Computer Science & Information technology
Guru Ghasidas University

Subject: Object Oriented Concepts (PCSC-502)
Class: BSC-V

[Max Marks: 30]

[Duration: 3 Hrs]

Instruction: Q.1 is compulsory and any FOUR questions from Q.2 to Q.8.

Section - A (5x2=10 Marks)

Q1.

a) Define Parameterized Constructor with example.
b) Explain different type of visibility mode.
c) Define class and object with example.
d) Explain static members with the help of example.
e) Explain Run time polymorphism.

Section - B (4x5=20 Marks)

Q2. What is inheritance? Describe single and multiple inheritance with the help of example.

Q3. Explain unary operator overloading using member function and friend function (give code).

Q4. Write point wise difference between Procedural Oriented Programming and Object Oriented Programming (at least 6).

Q5. Explain Union and Structure with example and compare union and structure with class.

Q6. What is the difference between arguments passed by value or passed as reference? Explain the difference with the help of an example.

Q7. What are Constructors and Destructors? Explain with the help of an example.

Q8. Explain data abstraction with example .

AV-8929

B.Sc. (Fifth Semester) Esamination, 2015-16

Object Oriented Concepts

Paper : PCSC-502

Answer 1 (i)

Definition

In C++,Constructor is automatically called when object(instance of class) create.It is special member function of the class.Which constructor has arguments thats called **Parameterized Constructor.**

- It has same name of class.
- It must be a public member.
- No Return Values.
- Default constructors are called when constructors are not defined for the classes.

Syntax

```
class class-name
{
    Access Specifier:
        Member-Variables
        Member-Functions
    public:
        class-name(variables)
        {
            // Constructor code
        }

        ... other Variables & Functions
}
```

Example Program

```
/*  Example Program For Simple Example Program Of
Parameterized Constructor In C++
*/

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

```
using namespace std;

class Example        {
    // Variable Declaration
    int a,b;
    public:

    //Constructor
    Example(int x,int y)            {
    // Assign Values In Constructor
    a=x;
    b=y;
    cout<<"Im Constructor\n";
    }

    void Display()    {
    cout<<"Values :"<<a<<"\t"<<b;
    }
};

int main()                {
        Example Object(10,20);
        // Constructor invoked.
        Object.Display();

        // Wait For Output Screen
        getch();
        return 0;
}
```

Sample Output

Im Constructor
Values :10 20

Answer 1(ii)

You should write three type of visibility mode

Public

Private

Protected

and explain them with visibility chart of inheritance..

(Refer:- page no. 99,100, 212 Book name :- Object Oriented Programming C++ fourth Edition)

Answer 1(iii)

Object - Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors - wagging, barking, and eating. An object is an instance of a class.

☐ **Class -** A class can be defined as a template/blueprint that describes the behaviors/states that object of its type support. Example car is a class maruti is a object of class car.

Answer 1 (IV)

Static members: A member of a class can be qualified as static using static keyword. Therefore, static members can be either static data member or static member functions.

You are instructed to give an example of either static data member or static member function:

For static data member, it is initialized to zero when the first object of its class is created. No other initialization is permitted. Only one copy of the member will be created and that copy will be shared by the all objects of its class. Its lifetime is throughout the program.

For static member function, it can have access to only static members declared in the same class. It can be invoked using its classname and scope resolution operator instead of its class object.

Example for declare a static data member and static member function:

```
class test
{
        static int x;
};
int test::x;

class test
{       public:
        static void fun(void)
        {}
};
```

```
void main()
{
        test t1;
        test::fun();  //calling of static member function
}
```

Answer 1(V)

Run time polymorphism:

C++ allows binding to be delayed till run time. When you have a function with same name, equal number of arguments and same data type in same sequence in base class as well derived class and a function call of form: base_class_type_ptr->member_function(args); will always call base class member function. The keyword ***virtual*** on a member function in base class indicates to the compiler to delay the binding till run time. The overloaded member function are selected for invoking by matching argument, both type and number this information is known to the compiler much later after compilation.

Looking at the content of base class type pointer it will correctly call the member function of one of possible derived / base class member function.

Example of run time polymorphism:

```
class Base
{
public:

    virtual void display(int i)
    { cout<<"Base::"<<i; }
};

class Derv: public Base
{
public:

    void display(int j)
    { cout<<"Derv::"<<j; }
};

int main()
{
    Base *ptr=new Derv;
    ptr->display(10);
```

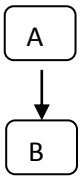
```
    return 0;
}
```

Output: Derv::10

Answer 2.

Reusability is an important feature of object oriented concepts. C++ strongly supports reusability. Once a class is designed, compiled and applied successfully then that class can be reused by any other programmer at any time. Reusability of coding can be achieved by many ways in C++; one of them is inheritance. This is basically creating new classes, reusing the properties of existing classes or extending the existing classes. Here the existing classes can be termed as “base class” and the new classes can be termed as “derived class”. There are different forms of inheritance like single inheritance, multilevel inheritance, multiple inheritance and hybrid inheritance.

Single inheritance:

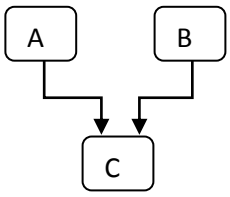


Here class A is the base class and class B is the derived class. Properties of class A can be visible in class B (along with its own properties if any) as per the visibility mode used at the time of inheritance. In single inheritance there is one level of inheritance. For example:

```
#include<iostream.h>
Class base  //base class
{
        int x; //private not inheritable and not accessible outside the class
        public: // members are inheritable and also accessible outside the class
                int y;
                void fun(int a, int b)
                {//body of fun}
};
Class derived : public base  //derived class inheriting the base class publicly
{
        int p; //private not inheritable and not accessible outside the class
```

```
        public: // members are inheritable and also accessible outside
the class
                int q;
                void fun1()
                {//body of fun1}
};
Void main()
{
        derived d1;
        [d1.x=10// access denied as private members can't be
inherited]
        d1.y=20// access of base class public property through derived
class object
        d1.fun(10,20)// access of base class public property through
derived class object
        [d1.p=10// accsess denied as private members can't accessed
outside the class]
        d1.q=10// access of own public members
        d1.fun1()// access of own public members
}
```

Multiple inheritance:



Here the class C inherits the attributes of two classes A and B. In case of multiple inheritance a class can inherit the attributes of two or more classes. It allows us to combine the features of several existing classes as a starting point for defining a new class. For example:

```
Class base1  //base class
{
        int x; //private not inheritable and not accessible outside the
class
        public: // members are inheritable and also accessible outside
the class
                int y;
                void fun(int a, int b)
                {//body of fun}
};
Class base2  //base class
{
        int p; //private not inheritable and not accessible outside the
class
```

```
        public: // members are inheritable and also accessible outside
the class
                int q;
                void fun1(int a, int b)
                {//body of fun1}
};
Class derived : public base1, public base2  //derived class inheriting
two base classes publicly
{
        int m; //private not inheritable and not accessible outside the
class
        public: // members are inheritable and also accessible outside
the class
                int n;
                void fun2()
                {//body of fun1}
};
Void main()
{
        derived d1;
        [d1.x=10// access denied as private members can't be
inherited]
        d1.y=20// access of base class public property through derived
class object
        d1.fun(10,20)// access of base class public property through
derived class object

        [d1.p=10// access denied as private members can't be
inherited]
        d1.q=10// access of base class public property through derived
class object
        d1.fun1()//access of base class public property through derived
class object

        d1.m=10//private not inheritable and not accessible outside
the class
        d1.n=10// access of own public members
        d1.fun2()// access of own public memberss
}
```

Answer 3.

You should write a program for unary operator with friend function and member function and explain that program.

(Refer:- page no. 173-175 Book name :- Object Oriented Programming C++ fourth Edition)

Answer 4

- **Difference Between Procedure Oriented Programming (POP) & Object Oriented Programming (OOP)**

	Procedure Oriented Programming	Object Oriented Programming
Divided Into	In POP, program is divided into small parts called **functions**.	In OOP, program is divided into parts called **objects**.
Importance	In POP,Importance is not given to **data** but to functions as well as **sequence** of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a **real world**.
Approach	POP follows **Top Down approach**.	OOP follows **Bottom Up approach**.
Access Specifiers	POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
Data Moving	In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.
Data Access	In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from function to function,it can be kept public or private so we can control the access of data.
Data Hiding	POP does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
Overloading	In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Examples	Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

Answer 5

You should explain union and structure with simple example and also explain memory allocation in both for data member and compare both with class as discussed in class.

Answer 6

You should write program for both arguments passed by value and passed as reference and explain that program.

(Refer:- page no. 123,124,130,131 Book name :- Object Oriented Programming C++ fourth Edition)

Answer 7

Rules to declare constructor and destructor of a class:

Constructor	Destructor
• Constructor name must be same as class name • It has no return type, not even void • It has to be declared in the public section • It can't be virtual • It is invoked automatically with the • Multiple constructors is allowed • Like other function, it can have arguments • It is invoked automatically when the object is created • One can overload constructor	• Destructor name must be same as class name • ~ symbol before the name of destructor is must. • It never take any argument • It never return any value • It is invoked automatically when the object is destroyed

Example:

```
#include<iostream.h>
class  test
{
        int x;
        public:
        test(int a) //constructor with one argument
        {
                        x=a;
```

```
        }
        ~test() //destructor
        {
                cout<<"Object has been destroyed";
        }
};
void main()
{
        test t1(5); //implicit call to the constructor
        test t2=test(10); //explicit call to the constructor

} //destructor invoked automatically twice with the destruction of the object t1 and t2
```

Relevancy and utility:

Though any object creation invokes default constructor, we can create our customized constructor as per the necessity of the program and it may perform any startup job specifically assignment of data members. As per the example, we have created one parameterized constructor which can initialize the value of private data member x of the class test. Using that constructor we can assign the value of x at the time of declaration of test class object.

For destructor, we can be sure the destruction of the object immediately after the expiration of the objects by customizing the destructor. In the example we would receive two messages of confirmation of destruction of objects t1 and t2.

Answer 8

Data abstraction refers to, providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.

Data abstraction is a programming (and design) technique that relies on the separation of interface and implementation.

Let's take one real life example of a TV, which you can turn on and off, change the channel, adjust the volume, and add external components such as speakers, VCRs, and DVD players, BUT you do not know its internal details, that is, you do not know how it receives signals over the air or through a cable, how it translates them, and finally displays them on the screen.

Thus, we can say a television clearly separates its internal implementation from its external interface and you can play with its interfaces like the power button, channel changer, and volume control without having zero knowledge of its internals.

Now, if we talk in terms of C++ Programming, C++ classes provides great level of **data abstraction**. They provide sufficient public methods to the outside world to play with the functionality of the object and to manipulate object data, i.e., state without actually knowing how class has been implemented internally.

For example, your program can make a call to the **sort()** function without knowing what algorithm the function actually uses to sort the given values. In fact, the underlying implementation of the sorting functionality could change between releases of the library, and as long as the interface stays the same, your function call will still work.

In C++, we use **classes** to define our own abstract data types (ADT). You can use the **cout** object of class **ostream** to stream data to standard output like this:

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

int main( )
{
   cout << "Hello C++" <<endl;
   return 0;
}
```

Here, you don't need to understand how **cout** displays the text on the user's screen. You need to only know the public interface and the underlying implementation of cout is free to change.

Access Labels Enforce Abstraction:

In C++, we use access labels to define the abstract interface to the class. A class may contain zero or more access labels:

- Members defined with a public label are accessible to all parts of the program. The data-abstraction view of a type is defined by its public members.
- Members defined with a private label are not accessible to code that uses the class. The private sections hide the implementation from code that uses the type.

There are no restrictions on how often an access label may appear. Each access label specifies the access level of the succeeding member definitions. The specified access level remains in effect until the next access label is encountered or the closing right brace of the class body is seen.

Benefits of Data Abstraction:

Data abstraction provides two important advantages:

- Class internals are protected from inadvertent user-level errors, which might corrupt the state of the object.
- The class implementation may evolve over time in response to changing requirements or bug reports without requiring change in user-level code.

By defining data members only in the private section of the class, the class author is free to make changes in the data. If the implementation changes, only the class code needs to be examined to see what affect the change may have. If data are public, then any function that directly accesses the data members of the old representation might be broken.

Data Abstraction Example: Any C++ program where you implement a class with public and private members is an example of data abstraction. Consider the following example:

```
#include <iostream>
using namespace std;
class Adder{
   public:
      // constructor
      Adder(int i = 0)
      {
        total = i;
      }
      // interface to outside world
      void addNum(int number)
      {
          total += number;
      }
      // interface to outside world
      int getTotal()
      {
          return total;
      };
   private:
      // hidden data from outside world
      int total;
};
int main( )
{
   Adder a;

   a.addNum(10);
   a.addNum(20);
   a.addNum(30);

   cout << "Total " << a.getTotal() <<endl;
```

```
   return 0;
}
```

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

```
Total 60
```

Above class adds numbers together, and returns the sum. The public members **addNum** and **getTotal** are the interfaces to the outside world and a user needs to know them to use the class. The private member **total** is something that the user doesn't need to know about, but is needed for the class to operate properly.