Model Answer
BSC-V - OOP – Object Oriented Concepts (PCSC-502)

Ans. 1

(a) 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
}
```

(b) Protected members: The member whose visibility modifier option is declared as protected is called protected member. It can be either data member or member function. These protected members are accessible within the class member functions only and they are inheritable. They remain protected in the derived class when they inherited publicly or protectedly but become private when they inherited privately.

Example:

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

(c) Private and public member functions: In C++ the visibility option of a member function of a class can be set as private or public. Member functions having visibility option as private are called private member function and having visibility option as public called public member functions. Private member function can be accessed within the class member functions only and they are not inheritable. Public member functions can be accessed within the class member function and outside the class and require the class object to be accessed from outside the class. Public member functions are inheritable.

```
class test
{
        private: //by default all members are private until they are specified by some other visibility options
        void fun(void){ //private members are accessible }
        public:
        void fun1(){ //private members are accessible}
};
Void main()
{
        test t1;
        [t1.fun();////private members are not allowed]
        t1.fun1();
}
```

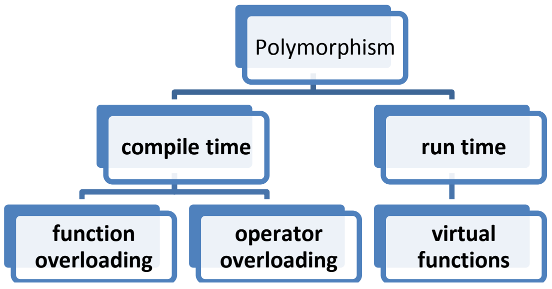
(d) Member functions declared outside the class are called explicit declaration of member function. For the explicit declaration of member functions we require the scope resolution operator i.e. "::".

```
Class test
{
        int fun(int);//only declaration of the member function without body
};
int test::fun(int)
{
        //body of function declared explicitly
}
```

(e) Structure, union, class and enumeration are user defined data types in C++. To declare a data type which can hold heterogeneous data items within it we can use user defined data types. Keyword to declare user defined data types are struct, union, class and enum.

```
class test
{
        Int x;
        char y;
        float z;
        public:
        void fun1(int, float){}
};
```

2. Polymorphism means single name multiple forms. It can be achieved by two different ways, one is compile time which is a result of early binding and another one is run time which is a result of late binding. We can chart different forms of polymorphism as given below:



Polymorphism achieved by function overloading as the example given below:

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
int computeArea(int x, int y)
{
        return x*y; //returns area of a rectangle (or square having same length arms)
}
float computeArea(int x)
{
        return (22/7)*x*x; //returns the area of a circle
}
float computeArea(int x, int y, int z)
{
        float s=(x+y+z)/2;
        return sqrt(s*(s-x)*(s-y)*(s-z));//returns the area of a triangle
}
Void main()
{
        clrscr();
        cout<<"area of the rectangle of arms 5cm and 6cm is: "<<computeArea(5,6);
        cout<<"area of the square of all arms 5cm is: "<<computeArea(5,5);
        cout<<"area of the circle of radious 7cm is: "<<computeArea(7);
        cout<<"area of the triangle of arms 5cm, 6cm and 7 is: "<<computeArea(5,6,7);
        getch();
}
```

Here we overloaded the function computeArea(). The names of the functions are same but they are returning areas of different shapes such as rectangle, circle and triangle. In This example, polymorphism is done by using different set of passing arguments for each function. The compiler can differentiate the calling of function by the difference in passing arguments of the function. That's why it is assumed as polymorphism achieved at compile time or early binding.

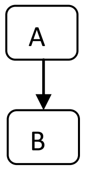
3.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class string
{
        char *x;
        public:
        string(void){}
        string(char *in)
        {
```

```
                    x=in;
            }
            string operator+(string s)
            {
                    string temp;
                    temp.x=strcat(x,s.x);
                    return temp;
            }
            void display(void)
            {
                    cout<<x<<"\n";
            }
    };
    void main(void)
    {
            clrscr();
            string s1("India is "),s2("great");
            string s3;
            s3=s1+s2;
            s3.display();
            getch();
    }
```

4. 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:

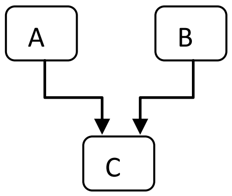


Here class A is the base class and class B is the derived class. Properties of class A can be visible in class B (along with its own properties if any) as per the visibility mode used at the time of inheritance. In single inheritance there is one level of inheritance. For example:

```
#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
        }
```

5. Rules to declare constructor and destructor of a class:

<u>Constructor</u>

- 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

<u>Destructor</u>

- 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.

6.

```
class  A
{
        //body of class A
};
class  B : public A
{
        //body of class B
};
class  C : public B
{
        //body of class C
};
class  D : public C
{
        //body of class D
};
class  E : public C
{
        //body of class E
};
class  F : private D , private E
{
        //body of class F
};
```

It is an example of hybrid inheritance as the total inheritance contains multilevel and multiple inheritance both.

7.

Features of C++:

The C++ programming language is based on the C language.

In C++, you can develop new data types that contain functional descriptions (member functions) as well as data representations. These new data types are called *classes*. The work of developing such classes is known as *data abstraction*. You can work with a combination of classes from established class libraries, develop your own classes, or derive new classes from existing classes by adding data descriptions and functions. New classes can contain (*inherit*) properties from one or more classes. The classes describe the data types and functions available, but they can hide (*encapsulate*) the implementation details from the client programs.

You can define a series of functions with different argument types that all use the same function name. This is called *function overloading*. A function can have the same name and argument types in base and derived classes.

Declaring a class member function in a base class allows you to override its implementation in a derived class. If you use virtual functions, class-dependent behavior may be determined at run time. This ability to select functions at run time, depending on data types, is called *polymorphism*

You can redefine the meaning of the basic language operators so that they can perform operations on user-defined classes (new data types), in addition to operations on system-defined data types, such as int, char, and float. Adding properties to operators for new data types is called *operator overloading*.

The C++ language provides templates and several keywords not found in the C language. Other features include *try-catch-throw* exception handling, stricter type checking and more versatile access to data and functions compared to the C language.

Advantages of C++: (you can refer benefits of OOP from Balaguruswami)

1: Stronger Type Checking - the use of classes, inheritance, automatic type conversions.

2: Type safe linkage - you can't accidentally call a routine from another module with the wrong type and/or number of arguments.

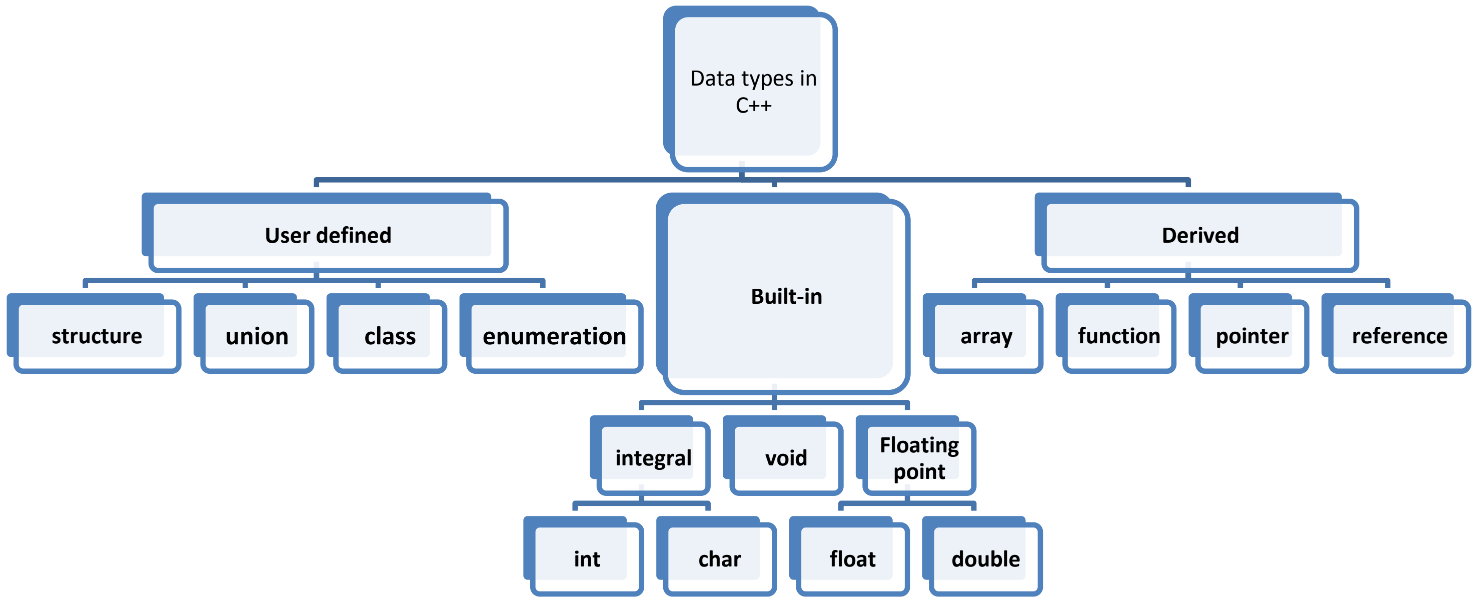
3: A complex data type is provided. It includes all the standard arithmetic operations, implemented as operators, not function calls.

4: User-defined operators and function overloading are supported. When you design a data type you can specify which operators and functions are provided.

5: You can define automatic type conversions to convert between data types.

6: Provides inline functions which combine the efficiency of using macros with the safety of using functions - simply prepend the word 'inline' in front of the function - if the compiler can inline it, it will.

8.


Data types in C++
User defined
Built-in
Derived
structure
union
class
enumeration
array
function
pointer
reference
integral
void
Floating point
int
char
float
double