Lab Sheet 4
Additional Components of a Class

Constructor

A constructor is basically a function used for initialization of the class data members, allocation of memory, and so on. It is convenient if an object can initialize itself when it is first created, without the need to make a separate call to a member function. It has the same name as the class in which they are members. No return type is used for constructors. Since the constructor is called automatically by the system there is no reason for it to return anything. Compiler knows that they are constructors by looking at their name and return type. Let us take an example that each time an object of the date class is created, it is to be initialized with the date 2/02/2011.

Examples:

```
class date{

 private:

 int dd,yy;

 char mm[3];

 public:

      date()         // constructor

      {     dd=2,mm[0]='0',mm[1]='2';

            mm[2]='\0', yy=2011;

      }

      showdate()  //display

      {

            cout<<"\n"<<dd<<"/"<<mm<<"/"<<yy;

      }

 };

 int main()

 {

     date d1;  //define and initialize

     clrscr();

     d1.showdate();

     return 0;
 }
```

Here as soon as an object d1 is created, the constructor will be involved and the values dd, mm, and yy initialized to the required date- 2/02/2011. Note that the constructor here neither takes any parameters nor does it return values. So it is called default constructor. Data members of object can be initialized by passing arguments to the constructor when the object is created. The constructor that takes arguments is called parameterized constructor. Following example adds parameterized constructor in the above example program.

```
class date{

 //....

public:

      date()        // constructor

      {    dd=2,mm[0]='0',mm[1]='2';

           mm[2]='\0', yy=2011;

      }

      date(int d, char m[], int y)        //parameterized constructor

      {    dd=d,mm[0]=m[0],mm[1]=m[1];

           mm[2]='\0', yy=y;

      }

    //......

 };

int main()

{

     date d1(2,"02",2011);  //define and initialize

     d1.showdate();

     return 0;

 }
```

Copy Constructor

We already know that no argument constructor can initialize data members to constant values, and parameterized constructor can initialize data members to values passed as arguments. There is also another way to initialize an object with another object of the same type. It is called the *default copy constructor*. It is a one-argument constructor whose argument is an object of same class.

Example:

```
class date

{

 private:

     int dd,yy;

     int mm;

 public:

    date()

    { dd=26, mm=1,yy=2004; }

    date(int ee,int nn,int zz): dd(ee),mm(nn),yy(zz) //parameterized
Constructor

       { }

    showdate()

    { cout<<"\n"<<dd<<"/"<<mm<<"/"<<yy;}

};

int main()

{

   date d1(26,2,2004);

   date d2(d1);                 //copy constructor call: it can also be
written as date d2=d1;

   d1.showdate();

   d2.showdate();

   return 0;

}
```

Destructor

Constructors serve to automatically initialize data members of an object at the point of creation. Destructors are complimentary to constructors. They serve to cleanup objects when they are destroyed.

A destructor may be called either when the object goes out of scope or when it is destroyed explicitly using the `delete` operator. A destructor, like a constructor has the same name as that of the class, but is prefixed by the tilde ('~') symbol. A class cannot have more than one destructor. And destructor can't take arguments or specify a return value. The most common use of destructor is to de-allocate memory that was allocated for the object by the constructor. Destructor for above program is

```
~date()  //destructor

{   }
```

Constant Member Function

A function is made a constant function by placing the keyword `const` after the function header before function body. Member function that do not change the value of its object but acquire data from their object are obvious candidates for constant function.

General syntax:

```
return_type func_name(argument list) const; //const function declaration

//.....

return_type func_name(argument list) const       //const function
definition

{

    //Function body;

}
```

Constant Object

When an object is declared as constant, we can't modify it. The constant objects can only use constant member functions with it, because they are the only ones that guarantee not to modify its value.

General syntax:

```
const class_name object_name;    //creation of constant object
```

Constant Reference Argument

When we don't want to modify the arguments passed to the function, the reference parameters are made const in the function declaration and definition.

General syntax:

```
return_type func_name(const int&, float&);      //function declaration with
const member function arguments

//......

return_type func_name(const int&, float&)      //function definition with
const member function arguments

{

    //function body; //here int type can't be modify but float can be.

}
```

Static Data Members

If a data item in class is declared as static, then only one copy of that item is created for the entire class, no matter how many objects there are. All the objects share a common item of information. Once the first object is created its value is initialized to zero. And separate definition for static data member is necessary outside the class

General syntax:

```
class class_name

{

    //......

    static data_type variable_name;                //declaration of static
data member inside the class

    //.....

};

data_type class_name :: variable_name;                 //definition of
static data member outside the class and here value can
```

```
                                                            //be assign to
         variable.
```

Static Member Function

A static member function can have access to only other static members declared in the same class and it can be called using the class name

General Syntax:

```
static return_type func_name (argument list);         //declaration of
static member function

//....

class_name:: fun_name(argument passed)                              //
static member function call
```

Exercises

1. Write a program that has a class to represent time. The class should have constructors to initialize data members hour, minute and second to 0 and to initialize them to values passed as arguments. The class should have member function to add time objects and return the result as time object. There should be another function to display the result in 24 hour format.
2. Write a program that has a class with a dynamically allocated character array as its data member. One object should contain "Engineers are" and another should contain " Creatures of logic". Member function `join()` should concatenate two strings by passing two objects as arguments. Display the concatenated string through a member function. Use constructors to allocate and initialize the data member. Also, write a destructor to free the allocated memory for the character array. Make your own function for concatenation of two strings.
3. Write a class that can store Department ID and Department Name with constructors to initialize its members. Write destructor member in the same class and display the message "Object *n* goes out of the scope". Your program should be made such that it should show the order of constructor and destructor invocation.
4. Assume that one constructor initializes data member say num_vehicle, hour and rate. There should be 10% discount if num_vehicle exceeds 10. Display the total charge. Use two objects and show bit-by-bit copy of one object to another (make your own copy constructor).
5. Write a program that illustrate the following relationship and comment the relationships.

   ```
   i) const_object.non_const_mem_function
   ii) const_object.const_mem_function
   iii) non_const_object.non_const_mem_function
   iv) non_const_object.const_mem_function
   ```

6. Create a class with a data member to hold "serial number" for each object created from the class. That is, the first object created will be numbered 1, the second 2 and so on by using the basic concept of static data members. Use static member function if it is useful in any of the member functions declared in the program. Otherwise make separate program that demonstrate the use of static member function.