CHAPTER 2

CONSTRUCTORS AND DESTRUCTORS:

Marks 14

Objectives:

- State Concepts of constructor & destructor, types of constructor.
- Programs based on constructor & destructors

3.1 Concepts of Constructors, Types of constructors: Default, Parameterized, Copy.
3.2 Overloaded Constructors: Multiple Constructors in a Class, Constructors with default arguments.
3.3 Destructors.

3.1 Concept of Constructors:

- "A constructor is a special member function **that** is executed automatically whenever the object of a class is created."
- The main task of the constructor is to initialize data members of the object when it is created. It is always good ,if we can initialize data members of an object without making a call to a member function
- The constructor has the same name that of the class.
- Constructors do not have return data type.

```
Class employee
{   int a, b;
   float salary;
public :
 employee (void ) ;        // declaration of constructor
};
employee :: employee (void )   // definition of constructor
{
 a =0 ;
 b= 0;
}
```

- Note that the constructor has the same name that of the class and it does not have any return type.
- When a class with constructor is defined, the constructor is automatically invoked when object is created.

```
 Employee e1;   // constructor is invoked.
```

- The above statement creates the object e1 of type employee and at the same time its data members a and b are initialized to 0.

Defining constructor:

Like member functions, constructors can be defined either inside or out side the class.

Defining Inside the class:

Syntax:

```
Class-name (argument list)
{
          ---
}
```

Example:

```
Class student
{
 private : int rollno;
             char name[20];
public :  student (int  r, char  n )
            {
            }
          void putdata(void)
            {
            }
};
```

Defining outside the class:

syntax :

```
class-name :: class-name ( argument list )
{
}
```

- The constructor defined out side the class must be declared in side the class.

```
Class student
{
 private : int rollno;
```

```
        char name[20];

public: student();          //  member function declaration
void putdata(void);         //  member function declaration
};
void student ::student()  / / member function out side the class
{
        cin>> rollno >> name;
}
void student:: putdata(void)        // member function out side the class
{
 cout << rollno << name;
}
```

CHARACTERICTICS OF CONSTRUCTORS

1. Constructor has the same name that of the class
2. Constructor is invoked automatically when the object of the class is created.
3. Constructor must be declared / defined in the public section of a class.
4. They do not have return data type.
5. Constructors can not be inherited, but a derived class can call the constructor of the base class.
6. Constructors can have default values.
7. Constructors can take parameters.
8. Constructor can be virtual.
9. One cannot refer the address of constructor.
10. Constructor makes implicit call to the operators new and delete when memory allocation is required.

TYPES OF CONSTRUCTORS

1) **Default Constructor**
2) **Parameterized constructor**
3) **Copy constructors**

DEFAULT CONSTRUCTOR:

- The constructor with no arguments is called default constructor.

Example:

```
Class employee
{
  int a , b;
 public :
        employee()       // default constructor
      {
        a=0;
        b =0 ;
       }
};
```

PARAMETERIZED CONSTRUCTOR:

- It may be necessary to initialize the various data elements of different objects with different values when they are created.
- This objective can be achieved by passing parameters (arguments) to the constructor when the object is created.
- "The constructors can take arguments are called parameterized constructors".

```
Class employee
{
  int a , b;
public :
 employee ( int ,int);      //  constructor declaration
};
employee :: employee ( int aa , int bb)    // parameterized constructor
{
 a =aa ;
 b = bb;
}
```

- When a constructor has been parameterized the object declaration such as

  Employee e1

may not work. This is because we have to pass initial values as arguments to the constructor when object is declared.

- Passing the initial values to the constructor can be done in two ways

1) By calling the constructor implicitly.
2) By calling the constructor explicitly.

```
employee e1 = employee ( 10 ,30 );        // explicit call
employee e1 ( 10,30 ) ;                   // implicit call
```

Both the statements creates object e1 and initialize data members a to 10 and b to 30.

Example:

```
include <iostream.h>
class integer
{
int m, n;
public: integer (int, int) ; // constructor declared
void display (void)
{
cout << . m = << m << .\n.;
cout <<  n = << n << .\n.;
}
};
integer : : integer (int x, int y) // constructor defined
{
m = x; n = y ;
}
main()
{
integer int1 (0, 100) // IMPLICIT call
integer int2 = integer (25, 75); // EXPLICIT call
cout << .\nOBJECT1. <<. .\n;
int1.display();
cout << .\nOBJECT2. << .\n.;
int2.display();
}
```

Output

```
OBJECT1
m = 0
n = 100
OBJECT2
m = 25
n = 75
```

COPY CONSTRUCTOR:

- A copy constructor is used to declare and at the same time initialize an object from another object.
- For example,

```
Complex c2 (c1); // copy constructor invoked
OR
Complex c2 =c1;  // copy constructor invoked
```

- In both the cases copy constructor will be invoked.
  Both the statements would define object c2 and at the same time initialize it to the value of c1.
- Copy constructor copies data member –by-member from one object to another.
  In the above case data would be copied member –by-member from c1 to c2.
- A copy constructor takes a reference to an object of the same class.

```
Class code
{
        int id;
        public:
        code () {   }       // default constructor

        code (int a)
        { id =a;}
        code ( code & x) // copy constructor takes object of the same class by reference
        {
          this ->id = x.id;          // copy data member- by- member from x to current object
```

```
        }
        void display()
        {
                cout <<" ID = " <<id <<endl;
        }
};
void main()
{
        code c1(3);
        code c2 (c1); // copy constructor is invoked
        code c3 = c1;     // copy constructor is invoked
        c1.display();
        c2.display();
        c3.display();
}
```

OUTPUT
ID = 3
ID = 3
ID = 3

PROGRAMS BASED ON CONSTRUCTORS:

- **Define a class fraction, which has the numerator and denominator as data members. Write the constructor and display fraction.**

```
# include <iostream.h>
class fraction
{
        float n , d ;
public : fraction ( float  nn , float dd)       // constructor - to initialize values
           {        n = nn ;
                    d = dd ;
           }
        void display_fraction()
        {
         cout << n / d;
       }
};
void main(void)
{
  fraction f1 (5,2);
 f1.display_fraction ();
}
```

OUTPUT:
2.5

- **Define a class to represent a Bank account. Include the following members.**

Name of Depositor, Account Number, and balance amount. Initialize the values using constructor. And write member functions for

1) **to deposit an amount**
2) **To withdraw an amount after checking a balance**
3) **To display name & balance**

```
# include <iostream.h>
# include < string.h>
class account
{       int acctno;
        char name [30];
        float balance;
public : account ( int a, char *s , float b )
        {
        acctno = a;
        strcpy ( name ,s);
        balance = b;
        }
        void deposit (float amt)
```

```
        {
           balance = balance + amt;
        }
        void withdraw ( float amt)
        {
            if ( balance < amt)
            cout << " SORRY U CANNOT WITHDRAW "<<endl;
            else
           balance = balance – amt;
       }
        void display(void)
        {
        cout << "NAME = " << name<< endl;
        cout << "YOUR CURRENT BALANCE IS  "<< balance << "Rs." <<endl
        }
};
void main ( void)
{
  account a ( 34567 , " Deepali" , 10000);
 a.display();
a.deposit ( 2000);
a.display();
a.withdraw( 5000);
a.display();
}
```

OUTPUT

NAME = Deppali
YOU CURRENT BALANCE IS 10000 Rs.
NAME = Deepali
YOU CURRENT BALANCE IS 12000 Rs.
NAME = Deepali
YOU CURRENT BALANCE IS 7000 Rs.

Write a program to declare a class 'student' having data members as rollno and percentage. Write constructor to initialize these data members. Accept and display this data for one object of a class.

```
# include < iostream.h>
Class student
{
 private : int rollno;
        float per;
public : student ( int r , float p)   // constructor – to initialize rollno and per
        {
        rollno = r;
        per =p;
        }
        void student :: showdata (void)        // member function definition out side the class
        {
          cout<<" Roll No =  " <<  roll-no << endl;
          cout<<"  Marks  = " <<  per << endl;
        }
};
void main(void)
{
        student s1 ( 102, 89);`    // constructor is invoked
        s1.showdata();
}
```

3.2 MULTIPLE CONSTRUCTORS IN A CLASS (OR Overloaded constructors)

- A class can have more than one constructor. In other words constructors can be overloaded. When overloaded, compiler selects an appropriate version of constructor depending upon the number and type of arguments passed to it.
- By using this concept in program user can create different object with various methods & initialization.
- The following example shows how constructors can be overloaded.

```
Class employee
{   int a , b;
public:
        employee ( )              // (1) default constructor
        {  a =0 ;  b =0;  }
        employee ( int x )        // (2) constructor with one arguments
        {
                a =x ;
                b = x;
        }
        employee ( int x ,int y ) // (2) constructor with two arguments
        {
                a =x;
                b =y;
        }
};
```

The above class has three constructors.

- The first constructor receives no arguments
- The second constructor receives one integer
- The third constructor receives two integers.

While executing the constructor, compiler matches number and types of arguments passed to it and accordingly it executes respective constructor.

For example for statements

1) employee e1;
   It would invoke the first constructor and set both a's and b's values of e1 to 0.
2) employee e2(10);
   It would invoke the second constructor and set both a's and b's values of e2 to 10
3) employee e3 (10,20);
   It would invoke the third constructor and set a's value to 10 and b's values to 20.

CONSTRUCTOR WITH DEFAULT PARAMETERS

- It is possible to define a constructor with default arguments.
- We can assigned default values for one or more parameters at the time of function declaration, which is used when those corresponding parameters are omitted in the call to the constructor.

For example the constructor complex ( ) can be declared as follows: **Complex (float real, float image=0);**
Here the image argument has the default value 0. Then the statement : **Complex (5.0)**
Assigns value 5.0 to the real variable and 0.0 to the image (by default). However the statement **Complex(5.0, 4.5)** will assign 5.0 to the real and 4.5 to the image variables.
The actual parameters when specified override the default value.

```
Class Complex
{    private:      float real , image;
     public:    Complex( float r, float i=0.0)
                    { real = r;   image = i ;
                    }
      void show()
{     cout << "Real :" <<real;
     cout << " Imaginary :" << image<< endl;
}
};
   void main()
{    Complex c1(2.3);
     Complex c2(2.3,3.4);
     C1.show();
     C2.show();
```

```
}
```
output:
Real : 2.3 Image : 0.0
Real : 2.3 Image : 3.4

Note : Once you assign a default value for any parameter in a constructor , all subsequent parameters must also be assigned the default values.

For example following constructor will not work.

Complex (float real = 0.0 , float image)

To avoid error you must assign a default value to image variable.

1) complex (float real , float image = 0.0) OK
2) complex (float real = 0.0 , float image) will not work
3) complex (float real =0.0 , float image = 0.0) OK

❖ What will be the out put of following program

```
# include <iostream.h>
class sample
{
  private : int a, b;
  public :
  sample ( )        // default constructor
  sample ( int x )            // constructor with one argument
 {          a = b = x;
 }
 sample ( int x, int y = 0)            // constructor with default values
{ a = x;
 b=y;
 }
void sum ()
{ cout << a+b;
}
};
void main()
{
 sample s1();
sample s2(10);   // error on this line
}
```

Ans: Error

Ambiguity between 'sample**::** sample (int)' and 'sample **::** sample (int ,int)'

PROGARM BASED ON CONSTRUCTOR WITH DEFAULT VALUES

❖ **Define a class employee which will contain member variables basic, TA, DA, HRA. Write a program using constructor with default values of HRA and DA and calculate the Gross salary of employee.**

```
# include <iostream.h>
class employee
{          float basic , TA, DA , HRA;
public: employee ( float b , float T , float H =3500 , float D = 5000)
                    {   basic = b;
                        TA =T;
                        HRA = h;
                        DA = D;
                    }
          void  display_salary (void)
          {          float GS;
                     GS = basic + TA + DA + HRA;
                     cout << GS << endl;
          }
}
void main ()
{
  employee e1 ( 8000,2000);
cout << " Gross Salary of e1 = " ;
e1.display_salary();
 employee e2 ( 8000 , 2000 , 5000 , 10000);
```

```
 cout << " Gross Salary of e2 = " ;
 e2. Display_salary();
}
```

OUTPUT:

```
cout << " Gross Salary of e1 = 18500" ;
cout << " Gross Salary of e2 = 25000" ;
```

- **Define class student which will contain member variables rollno , name and course. Write a program- using constructor with default value for course as 'computer engg'. Accept this data for two objects and display accepted data.**

```
 # include <iostream.h> # include < string.h>
 class student
 {        int rollno;
          char name [20];
          char course [20];
   public:student ( int r, char *n , char * c = " computer engg")
            { rollno = r;
             strcpy ( name , n);
            strcpy (course, c);
             }
          void display()
          { cout << name << endl;
           cout <<rollno<<endl;
           cout<<course <<endl;
          }
 };
void main()
{           student s1( 102 , "Deepali" );
           s1.display();
           student s2 ( 103 ," Rohit", "Information Tech")
          s2.display();
}
```

DESTRUCTORS:

- Destructor is used to destroy the object that have been created by a constructor
- It's a special member function having the same name that of the class. And is preceded by tilde (~) sign.
- Like constructors, Destructors do not have return data type and does not take any argument.
- This member function in public section only.
- A destructor will be invoked implicitly by clean up storage that is no longer accessible.
- It can"t be declared static, const or volatile.
- Definition of destructor does not have any meaningful contents.

  Ex:- Class abc Will have destructor function like this:

```
 ~abc()
{
cout<<"Destructor invoked";
}
```

- **Write a program to find the sum of numbers between 1 to n suing constructor where value of n will be passed as parameter to the constructor. The program should display the sum.**

```
# include <iostream.h>
class add
{ private : int n;
public: add ( int a)         // constructor to initialize two numbers
        {            n =a;
        }
void sum(void)
{   int result =0 ;
          for (int I =1 ; I<=n ; I++)
          {  result = result + I;
          }
  cout <<  "THE SUM  = "<<result;
}
};
void main(void)
```

```
{
  add a (4);
  a.sum();
}
```

OUTPUT

THE SUM = 10