Chapter 6

Constructor & Destructor

Default Constructor

A constructor is a special member function whose task is to initialize the object of its class. It is special because the name of the constructor function and the name of the class is same. The constructor function is automatically called whenever an new object of its associated class is created. It is called constructor because is construct the values of data members of the class.

A constructor is declared & define as follows.

```
class Class-name
{
    private :
                Variable 1;
                Variable 2;
    public :
                Class-name(Parameter List);
                Member Function 1();
                Member Function 2();
                :
                :
};

Class-name : : Class-name(Parameter List)
{
        Statement 1;
        Statement 2;
        :
        :
}
```

e.g. **Inside Function Definition**

```
class book
{
    private :
                int bno;
                float price;
    public :
                book()
                {
                   bno=103;
                   price=55.35
                }
                :
};
```

e.g. **Outside Function Definition**

```
class book
{
    private :
                int bno;
                float price;
    public :
                book();
                :
                :
};

book : : book()
{
    bno=103
    price=55.35
}
```

The statement book b; not only creates an object "b" of type book but also initialize its data members bno & price. There is no need to write any statement to call the constructor function. A constructor that accpts no parameter is called the default constructor.

The constructor function have some special characteristics

1) They should be declared or defined in the public section of the class.
2) They are called automatically when a objects are created.
3) They do not have return types since constructor do not return any value there should not be a keyword void preceded to them.
4) Like other C++ function they can have default arguments.
5) They can not be inherited.

Write a Program to Demonstration of Constructor (Inside Function Definition).

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

class book
{
  private:
     int bno;
     float price;
  public:
     book()
     {
      bno=101;
      price=23.90;
     }
     void getdata();
     void putdata();
};
```

```
void book::getdata()
{
cout<<"Enter the book no & price = ";
cin>>bno>>price;
}

void book::putdata()
{
cout<<"Book no = "<<bno<<endl;
cout<<"Price   = "<<price<<endl;
}

main()
{
book b,a;
clrscr();
b.putdata();
b.getdata();
b.putdata();
a.putdata();
getch();
return(0);
}
```

Write a Program to Demonstration of Constructor (Outside Function Definition).

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

class book
{
   private:
     int bno;
     float price;
   public:
     book();
     void getdata();
     void putdata();
};

book :: book()
{
bno=1;
price=33.33;
}

void book :: getdata()
{
cout<<"Enter book no & price = ";
cin>>bno>>price;
}
```

```
void book::putdata()
{
cout<<"Book no = "<<bno<<endl;
cout<<"Price   = "<<price<<endl;
}

main()
{
book a,b;
clrscr();
b.putdata();
b.getdata();
b.putdata();
a.putdata();
getch();
return(0);
}
```

Create a Class Student with data Members roll no, per. Write a Program to store the information about student using constructor.

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

class stud
{
   private:
      int rno;
      float per;
   public:
      stud()
      {
       rno = 1;
       per = 75.75;
      }
      void getdata();
      void putdata();
};

void stud :: getdata()
{
cout<<"Enter the roll no & per = ";
cin>>rno>>per;
}

void stud :: putdata()
{
   cout<<"Roll no = "<<rno<<endl;
   cout<<"Per = "<<per<<endl;
}
```

```
main()
{
stud a,b;
clrscr();
a.putdata();
b.getdata();
b.putdata();
getch();
return(0);
}
```

Parameterized Constructor

A constructor which does not received any argument is known as default constructor. The constructor which receives the argument is known as parameterized constructor.

We know that the variables of built in data type can be initialized to different values when they are created. One of the aim of C++ is that the variables of user defined data type must be operated similarly to that of built in data type i.e. the variables of user defined data type (Class) must be initialized to different values when they are created.

To achieve this objectives, the concept of parameterized constructor is introduced. If a class contain a parameterized constructor then the statement emp e1; may not work because while creating objects we must pass the arguments to the constructor function.

The pass the argument to the constructor, there are two methods for calling the constructor function.

1) Implicit Call
2) Explicit Call

Implicit call means pass the argument without specifying the name of the constructor function.

e.g. emp e(100,5000.53)

Explicit call means pass the argument by specifying the name of constructor function.

e.g. emp p=emp(200,6660.75)

Write a Program to initialize the class member variable using parameterized Constructor Function.

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

class no
{
   private:
      int x,y;
   public:
```

```
      no(int,int);
      void display();
};

no::no(int a,int b)
{
    x=a;
    y=b;
}

void no::display()
{
    cout<<"Value 1 = "<<x<<endl;
    cout<<"Value 2 = "<<y<<endl;
}

main()
{
no n(41,62);
clrscr();
n.display();
int p,q;
cout<<"Enter the value 1 & value 2 = ";
cin>>p>>q;
no n2=no(p,q);
n2.display();
getch();
return(0);
}
```

Write a Program to implement parameterized Constructor Function.

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

class emp
{
  private:
      int eno;
      float sal;
  public:
      emp(int,float);
      void getdata()
      {
         cout<<"Enter the emp. no & Salary = ";
         cin>>eno>>sal;
      }
      void putdata()
      {
         cout<<"Emp. no = "<<eno<<endl;
         cout<<"Emp. Salary = "<<sal<<endl;
```

```
      }
};
emp :: emp(int a,float b)
{
   eno=a;
   sal=b;
}

main()
{
emp e1(11,4000.5);
clrscr();
e1.putdata();
int p;
float q;
cout<<"Enter eno & Salary = ";
cin>>p>>q;
emp e2=emp(p,q);
e2.putdata();
e2.getdata();
e2.putdata();
getch();
return(0);
}
```

Multiple Constructor

We have to use two kinds of constructor.

e.g.

1) no(); Default Constructor
2) no(int,int); Parameterized Constructor

In first example the constructor itself supplies the data values and no values are passed by the calling program. In second example the function call passes the appropriate values from calling program. C++ permits us to use both these constructor in the same class.

Write a Program to Demonstration of Multiple Constructor.

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

class no
{
     int m,n;
  public:
     no()
     {
     m=10;
     n=5;
     }
```

```
     no(int,int);
     void putdata()
     {
       cout<<"Value 1 = "<<m<<endl
           <<"Value 2 = "<<n<<endl;
     }
};

no :: no(int p,int q)
{
   m=p;
   n=q;
}

main()
{
int x,y;
no a,b(17,35);
clrscr();
cout<<"Constructor 1 Value = "<<endl;;
a.putdata();
cout<<"Constructor 2 Value = "<<endl;
b.putdata();
cout<<"Enter 2 nos = ";
cin>>x>>y;
no c(x,y);
c.putdata();
getch();
return(0);
}
```

Constructor with Default Argument

It is possible to define constructor with default arguments. The default argument constructor can be called either one argument or no arguments. When called with no argument it becomes a default constructor. When both this form are used in a class it causes ambiguity.

Write a Program to demonstration of Constructor with Default Argument.

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

class complex
{
     int real,imag;
  public:
     complex(int,int i=0);
     complex();
     void putdata()
     {
      cout<<"Complex no = ";
```

```
      cout<<real<<"+"<<imag<<"i"<<endl;
     }
};
complex::complex(int r,int i)
{
real=r;
imag=i;
}

complex::complex()
{
real=3;
imag=23;
}

main()
{
complex a,b(5),c(4,7);
clrscr();
a.putdata();
b.putdata();
c.putdata();
getch();
return(0);
}
```

Copy Constructor

A constructor which initialize an object by using a values of some other object is known as copy constructor. A copy constructor is also a parameterized constructor. It receives an argument which is nothing but an object of the class i.e. a constructor receives an object as argument.

In the constructor function the values of data members of an object which is passed as an argument to the constructor are used to initialize the data members of the created object. Since the constructor is coping the values of one object to another object it is known as copy constructor.

Write a Program to demonstration of Copy Constructor Function.

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

class no
{
    int m,n;
  public:
    no(int a,int b)
    {
       m=b;
       n=a;
    }
```

```
    no(no & b)
    {
       m=b.m;
       n=b.n;
    }
    void putdata()
    {
       cout<<"Number 1 = "<<n<<endl;
       cout<<"Number 2 = "<<m<<endl;
    }
};

main()
{
clrscr();
no x(16,42);
no y(x);
no z=x;
x.putdata();
y.putdata();
z.putdata();
getch();
return(0);
}
```

Note 1)

The constructor function can be define as inline function.

e.g.

```
class no
{
   private :
        int m,n;
   public :
        no(int p,int q)
        {
            m=p;
            n=q;
        }
        :
        :
};
```

Note 2)

The parameter of constructor can be of any type except that of the class to which it belongs.

e.g.

```
class no
{
   private :
        int m,n;
   public :
        no(no);
        :
        :
};
```

is illegal.

However a constructor can access a reference to its own class as a parameter.

e.g.

```
class no
{
   private :
        int m,n;
   public :
        no(no &);
        :
        :
};
```

is legal.

In such cases the constructor is called copy constructor. Pass by value is illegal to pass the object in constructor.

Destructor

A destructor is a special member function whose job is to destroy the objects of a class. It is special member function because the name of the class and the name of the destructor function is same. But there is a different between constructor & destructor. A destructor always have a tild (~) symbol preceded to it.

The destructor function gets automatically invoke when the program control comes out of a specific block or function. There is a no necessary to write any statement to call the destructor. When the created object in a program are no longer required, it is necessary to remove those objects from memory and it is a good programming practice to free the memory space occupied by the objects.

Write a Program to demonstration of Destructor Function.

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

class book
{
   private:
      int bno;
```

```
      float price;
   public:
      book()
      {
        bno=11;
        price=44.50;
      }
      book(int a,float b)
      {
        bno=a;
        price=b;
      }
      void putdata()
      {
        cout<<"Book no = "<<bno<<endl;
        cout<<"Price = "<<price<<endl;
      }
      ~book()
      {
      }
};

main()
{
book x,y(8,55.33);
{
  clrscr();
  x.putdata();
  y.putdata();
}
getch();
return(0);
}
```

Write a Program to demonstration of Destructor Function.

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

int n=0;
class pqr
{
   public:
      pqr()
      {
        n++;
        cout<<"Object "<<n<<" Created"<<endl;
      }
      ~pqr()
      {
        cout<<"Object "<<n<<" Destroyed"<<endl;
```

```
        n--;
      }
};

main()
{
pqr a,b,c;
clrscr();
{
  cout<<"Inner block"<<endl;
  pqr x,y;
}
return(0);
}
```

Dynamic Initialization of Objects

Class object can be initialize dynamically too i.e. the initial value of an object may be provided during run time. One advantage of dynamic initialization is that we can provide various initialization format using different constructor.

Write a Program to demonstration of Dynamic Initialization of Object.

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

class no
{
     int m,n;
  public:
     no(int,int);
     void putdata()
     {
       cout<<"Value 1 = "<<m<<endl
           <<"Value 2 = "<<n<<endl;
     }
};

no :: no(int p,int q)
{
   m=p;
   n=q;
}

main()
{
int x,y;
no a(17,35);
clrscr();
cout<<"Constructor 1 Value = "<<endl;;
a.putdata();
```

```
cout<<"Enter 2 nos = ";
cin>>x>>y;
no c(x,y);
cout<<"Constructor 2 Value = "<<endl;
c.putdata();
getch();
return(0);
}
```

Dynamic Constructor

The constructor can also be used to allocate memory while creating objects. This will enable the system to allocate the right amount of memory for each object when the objects are not of the same size, thus resulting in the saving of memory. Allocation of memory to objects at the time of their construction is known as Dynamic Construction of objects. The memory is allocated with the help of new operator.

Write a Program to demonstration of Dynamic Constructor.

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

class string
{
  private:
      char *name;
      int len;
  public:
      string()
      {
       len=0;
       name=new char[len+1];
      }
      string(char *s)
      {
       len=strlen(s);
       name=new char[len+1];
       strcpy(name,s);
      }
      void display()
      {
       cout<<"String = "<<name<<endl;
      }
      void join(string &,string &);
};

void string :: join(string &a,string & b)
{
   len=a.len+b.len;
   delete name;
```

```
   name=new char[len+1];
   strcpy(name,a.name);
   strcat(name,b.name);
}

main()
{
char *str="Amol ";
string x(str),y("Patil"),p;
clrscr();
p.join(x,y);
x.display();
y.display();
p.display();
getch();
return(0);
}
```

Constructing Two-Dimensional Array

We can construct matrix variables using the class type object.

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

class matrix
{
  private:
      int **p;
      int d1,d2;
  public:
      matrix(int,int);
      void getele(int i,int j,int value)
      {
       p[i][j]=value;
      }
      int putele(int i,int j)
      {
       return p[i][j];
      }
};

matrix :: matrix(int x,int y)
{
    d1=x;
    d2=y;
    p=new int *[d1];
    for(int i=0;i<d1;i++)
       p[i]=new int[d2];
}
```

```
main()
{
int m,n;
clrscr();
cout<<"Enter size of Matrix = ";
cin>>m>>n;
matrix x(m,n);
cout<<"Enter Matrix Element ="<<endl;
int i,j,value;
for(i=1;i<=m;i++)
{
   for(j=1;j<=n;j++)
   {
      cin>>value;
      x.getele(i,j,value);
   }
}
cout<<"Matrix Element = "<<x.putele(2,2);
getch();
return(0);
}
```