C++ and Object Oriented Programming

Unit – 2 (Topic – 2)

Constructors & Destructors

Prepared By
Prof. Shah Brijesh

Constructors

Constructors:

-> A constructor is a “special” member function whose task is to initialize the objects of its class.

-> It is special because its name is the same as the class name.

-> The constructor is invoked whenever an object of its associated class is created.

-> It is called constructor because it constructs the values of data members of the class.

-> Characteristics of Constructor

1. They should be declared in the public section
2. They are invoked automatically when the objects are created.
3. They do not have return types, not even void and therefore, they cannot return values.
4. They cannot be inherited, through a derived class can call the base class constructor.
5. Like other C++ functions, they can have default arguments.
6. Constructors cannot be virtual.
7. We cannot refer to their addresses.
8. An object with a constructor cannot be used as a member of a union.
9. They make ‘implicit calls’ to the operators new and delete when memory allocation is required.

Constructors

Constructors:

For Example:

```
class A
{
        int m, n;
        public:
                A() // Constructor
                {
                        m=0;
                        n = 0;                          }
                void disp()
                {
                        cout<<“\n m=“<<m;
                        cout<<“\n n=“<<n;  }
};
void main()
{
        A x;        // Create an object x and call the Constructor function.
        x.disp();
}
```

Constructors

Types Of Constructors:

-> Following are the types of the constructor

1. Default Constructor
2. Parameterized Constructor
3. Multiple Constructors / Constructor Overloading
4. Dynamic Constructor
5. Copy Constructor

1. Default Constructor:

-> A constructor without any argument is called Default Constructor.

-> When we create default constructor, it is guaranteed that an object created by the class will be initialized automatically.

-> For Example:

```
A x;
```

-> not only creates the object x of the type A but also initialize its data member m and n to zero.

-> There is no need to write any statement to invoke the constructor functions.

Constructors

Types Of Constructors:

2. Parameterized Constructor:

-> The constructor A(), defined above, initializes the data members of all the objects to zero.

-> However, in practice it may be necessary to initialize the various data elements of different objects with different values when they are created.

-> C++ permits us to achieve this objective by passing arguments to the constructor function when the objects are created.

-> The constructors that can take arguments are called parameterized constructors.

-> The constructor A() may be modified to take arguments as shown below

```
class A
{           int m, n;
            public:
                        A(int x, int y)  {
                                    m=x;
                                    n=y;                    }
                        ....................
};
```

Constructors

Types Of Constructors:

2. Parameterized Constructor:

-> When the constructor has been parameterized, the object declaration statement such as

A x;

-> may not work.

-> We must pass the initial values as arguments to the constructor function when an object is declared.

-> This can be done in two ways:

1. By calling the constructor explicitly.

A x = A(10,20);

-> This statement creates an A object x and passes the values 10 and 20 to it.

2. By calling the constructor implicitly.

A x(10,20);

-> This method, sometimes called shorthand method, is used very often as it is shorter, looks better and is easy to implement.

Note : When the constructor is parameterized, we must provide appropriate arguments for the constructor.

Constructors

Types Of Constructors:

2. Parameterized Constructor:

```
class A
{
        int m, n;
        public:
                A(int x, int y) // Parameterized Constructor
                {
                        m=x; n=y;                       }
                void disp()
                {
                        cout<<“\n m=“<<m;
                        cout<<“\n n =“<<n;              }
};
void main()
{
        A x(100,200;
        x.disp();
}
```

Constructors

Types Of Constructors:

3. Multiple Constructors In A Class (Constructor Overloading):

-> So far we have used two kinds of constructors. They are..

```
A();                    // No Arguments
A(int x, int y);        // Two Arguments
```

-> In first case, the constructor itself supplies the data values and no values are passed by the calling program.

-> In the second case, the function call passes the appropriate values from main().

-> C++ permits us to use both these constructors in the same class.

```
class A
{       int m, n;
        public:
                A()
                {
                        m=0;    n=0;            }
                A(int x, int y)  {
                        m=x;    n=y;            }
                ..................
};
```

Constructors

Types Of Constructors:

3. Multiple Constructors In A Class (Constructor Overloading):

-> Above code declares two constructors for an A object.

-> The first constructor receives no arguments and the second receives two integer arguments.

-> For example, The declaration

A x;

-> would automatically invoke the first constructor and set both m and n of x to 0. The statement

A y(10,20);

-> would call the second constructor which will initialize the data members m and n of y to 10 and 20.

-> Here we create two constructor functions in a same class.

-> So, it is known as constructor overloading.

```
class A
{       int m, n;
        public:
        A()         // Default Constructor
        {
                cout<<"\n Default Constructor";
                m=0;      n=0;                  }
```

Constructors

Types Of Constructors:

3. Multiple Constructors In A Class (Constructor Overloading):

```
        A(int x, int y)          // Parameterized Constructor
        {       cout<<"\n Parameterized Constructor";
                m=x;
                n=y;
        }
        void display()
        {
                cout<<"\n m ="<<m;
                cout<<"\n n ="<<n;
        }
};
void main()
{
        A x;
        A y(100,200);
        x.display();
        y.display();
}
```

Constructors

Types Of Constructors:

4. Dynamic Constructors:

-> 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 the new operator.

```
class String
{
        char *name;
        int length;
        public:
                String()    // Default Constructor
                {           length = 0;
                            name = new char[length+1];          }
                String(char *s)       // Dynamic Constructor
                {           length = strlen(s);
                            name = new char[length+1];
                            strcpy(name,s);                     }
```

Constructors

Types Of Constructors:

4. Dynamic Constructors:

```
                    void join(String &a, String &b);
                    void disp()
                    {
                              cout<<"\n Name ="<<name;
                    }
};
void String :: join(String &a, String &b)
{
          length = a.length + b.length;
          name = new char[length+1];
          strcpy(name, a.name);
          strcat(name, b.name);
}
void main()
{
          String s;
          String s1("Shah"), s2("Brijesh");
          s.join(s1,s2);
          s1.disp();
          s2.disp();
          s.disp();
}
```

Constructors

Types Of Constructors:

4. Dynamic Constructors:

-> Constructing Two Dimensional Array:

```
class Matrix
{
        int **p, d1, d2;
        public:
                Matrix(int x, int y);
                void disp();
};
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];
        for(i=0; i<d1; i++)
        {
                for(int j=0; j<d2; j++)
                {       cout<<"\nEnter value:";
                        cin>>p[i][j];
                }
        }
}
```

Constructors

Types Of Constructors:

4. Dynamic Constructors:

```
void Matrix :: disp()
{
        for(int i=0; i<d1; i++)
        {
                for(int j=0; j<d2; j++)
                {
                        cout<<“\t”<<p[i][j];
                }
                cout<<endl;
        }
}
void main()
{
        int r,c;
        cout<<“Enter no. of rows and columns:”;
        cin>>r>>c;
        Matrx m(r,c);
        m.disp();                       }
```

Constructors

Types Of Constructors:

5. Copy Constructors:

-> A copy constructor is used to declare and initialize an object from another object.

-> The process of initializing through copy constructor is known as copy initialization.

-> A copy constructor takes a reference to an object of the same class as itself as an argument.

-> For Example:

```
class A
{       int a, b;
        public:
                A(A &x) // Copy Constructor
                {
                        a = x.a;
                        b=x.b;          }
                ....................................
};
```

-> The statement

A y = x; // A y(x); Where x is an object of class A

-> Will define the object y and at the same time initialize it to the values of x.

Constructors

Types Of Constructors:

5. Copy Constructors:

Notes:

-> A reference variable has been used as an argument to the copy constuctor.

-> We cannot pass the argument by value to a copy constructor.

-> The statement:

y = x;

will not invoke the copy constructor.

-> However, if x and y are objects, this statement is legal and simply assigns the values of x to y member by member.

-> For Example:

```
class A
{       int a, b;
        public:
                A(A &x) // Copy Constructor
                {
                        a = x.a;
                        b=x.b;                  }
                A() // Default Constructor
                {
                        a=0;
                        b=0;
                }
```

Constructors

Types Of Constructors:

5. Copy Constructors:

```
                    A(int x, int y) // parameterized Constructor
                    {
                            a = x;
                            b = y;                  }
                    void disp()
                    {
                            cout<<"\n a ="<<a;
                            cout<<"\n b ="<<b; }
};
void main()
{       A x;        // Default Constructor
        A y(10,20);         // Parametrized Constructor
        A z = y;            // Copy Constructor
        x.disp();
        y.disp();
        z.disp();
        x = z;      // Simply assign value of object z to x.
        x.disp();           }
```

Constructors

Dynamic Initialization Of Objects:

-> Class Object can be initialized too.

-> That is, the initial value of an object may be provided during run time.

-> One advantage of dynamic initialization is that we can provide various initialization formats, using overloaded constructors.

-> This provides the flexibility of using different format of data at runtime depending upon the situation.

-> For Example:

```
class FixedDeposit
{
        long int p;
        int y;
        float r, amount;
        public:
                FixedDeposit() // Default Constructor
                {           }
                FixedDeposit(long int P, int Y, float R=0.12);
                FixedDeposit(long int P, int Y, int R);
                void disp();
};
```

Constructors

Dynamic Initialization Of Objects:

```
void FixedDeposit ::display()
{
        cout<<"\n P ="<<p<<"\n Y="<<y;
        cout<<"\n R ="<<r<<"\n Amount="<<amount;
}
FixedDeposit :: FixedDeposit(long int P, int Y, float R)
{
        p=P;    y=Y;    r=R;    amout=P;
        for(int i=0;i<y;i++)
                amount *= (1.0+r);
}
FixedDeposit :: FixedDeposit(long int P, int Y, int R)
{
        p=P;    y=Y;    r=R;    amout=P;
        for(int i=0;i<y;i++)
                amount *= (1.0+float®/100);
}
```

Constructors

Dynamic Initialization Of Objects:

```
void main()
{       FixedDeposit fd1, fd2, fd3;
        long int p;
        int y;
        float r;
        int R;
        cout<<“\nEnter amount, period, rate(in percent)”;
        cin>>p>>y>>r;
        fd1 = FixedDeposit(p, y, r);
        cout<<“\nEnter amount, period, rate(decimal form)”;
        cin>>p>>y>>R;
        fd2 = FixedDeposit(p, y, R);
        cout<<“\nEnter amount, period”;
        cin>>p>>y;
        fd3 = FixedDeposit(p, y);
        cout<<“\nDeposit 1“;        fd1.disp();
        cout<<“\nDeposit 2“;        fd2.disp();
        cout<<“\nDeposit 3“;        fd3.disp();                    }
```

Constructors

Const Objects:

-> We may create and use constant objects using const keyword before object declaration.

-> For Example , we may create m as a constant object of the class Matrix as follows:

```
const Matrix m(r, c);
```

-> Any attempt to modify the values of r and c will generate compile time error.

-> Further object can call only const member functions.

-> As we know, a const member is a function prototype or function definition where the keyword const appears after the function's signature.

-> Whenever const objects try to invoke non-const member functions, the compiler generates errors.

Destructors

Destructors:

-> A destructor as the name suggest, is used to destroy the objects that have been created by a constructor.

-> Like a constructor, the destructor is a member function whose name is the same as the class name but is preceded by a tilde(~).

-> For example, the destructor for the class A can be defined as shown below:

```
~A() {                    }
```

-> A destructor never takes any argument nor does it return any value.

-> It will be invoked implicitly by the compiler upon exit from the program(or block or function as case may be) to clean up storage that is no longer accessible.

-> It is a good practice to declare destructors in a program since it releases memory space for future use.

-> Whenever new is used to allocate memory in the constructors, we should use delete to free that memory in destructors.

-> For example:

```
Matrix :: ~matirx()  {
            for(int i=0;i<d1;i++)
                        delete p[i];
            delete p;              }
```

-> This is required because when the pointers to objects go out of scope, a destructor is not called implicitly.

Destructors

Destructors:

```
class A
{           public:
            A()
            {       cout<<"\n Object Created";          }
            ~A()
            {       cout<<"\n Object Destroyed";        }
};
void main()
{
            A a1;
            A a2;
            {
                        A a3;
            }
            A a4;
            {
                        A a5;
            }
            A a6;                   }
```