C++

Constructor-Destructor

1. Constructor

- A constructor is a special member function of a class which is automatically invoked at the time of creation of an object to initialize or construct the values of data members of the object.
- The name of the constructor is the same as that of the class to which it belongs.
  - A constructor must be declared in the public section.
  - It should not be explicitly called because a constructor is automatically invoked when an object of a class is created.
  - A constructor can never return any value; therefore, unlike a normal function, a constructor does not have any return value (not even void).
  - A constructor cannot be inherited and virtual.
  - The address of a constructor cannot be referred to in programs. Therefore, pointers and references do not work with constructors.

1. Constructor

- **A constructor cannot be declared as static, volatile, or const.**

- **Like a normal function, a constructor function can also be overloaded.**

- **Like a normal function, a constructor function can also have default arguments.**

- **Like a normal class member function, a constructor can be either defined inside the class or outside the class.**

1. Constructor - Classification

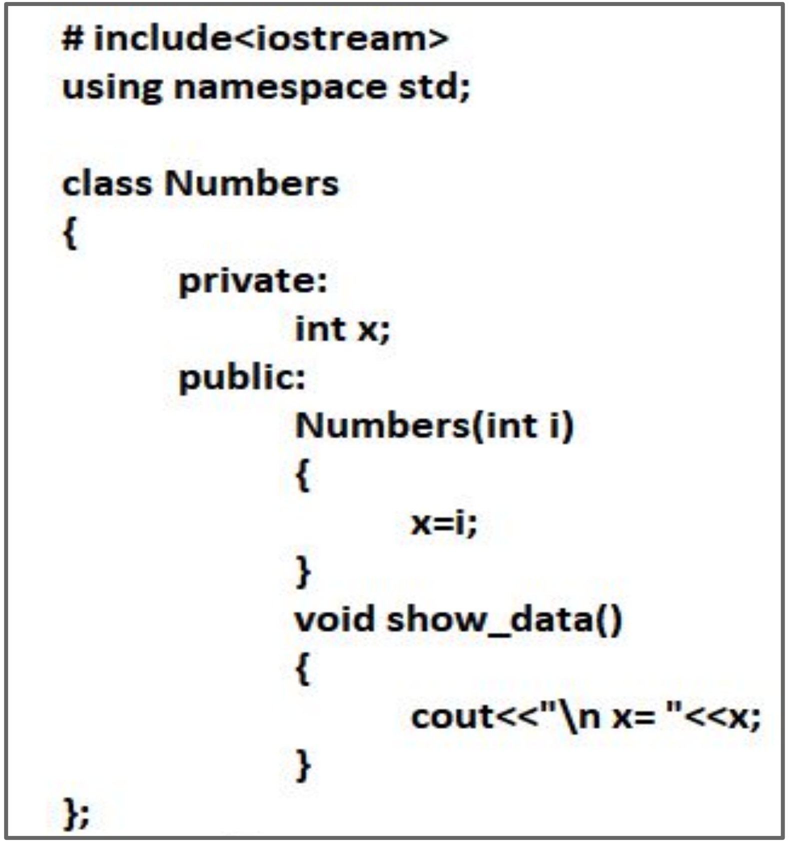
- **Parameterized**
- **Copy**
- **Dynamic**
- **Dummy**
- **Default**

1. Constructor - Parameterized

- A constructor that accepts one or more parameters is called a parameterized constructor.

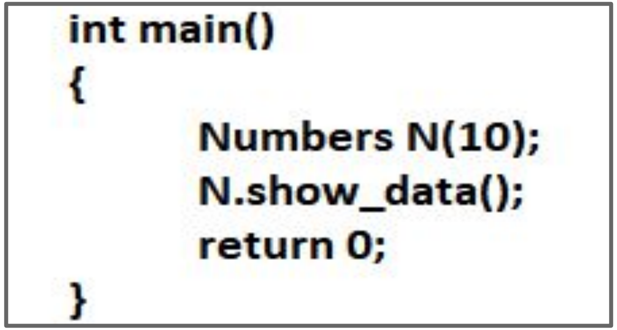
- The program code given here uses a parameterized constructor to initialize the data member of the class.

1. Constructor - Parameterized



```
# include<iostream>
using namespace std;

class Numbers
{
        private:
                int x;
        public:
                Numbers(int i)
                {
                        x=i;
                }
                void show_data()
                {
                        cout<<"\n x= "<<x;
                }
};
```



```
int main()
{
        Numbers N(10);
        N.show_data();
        return 0;
}
```

1. Constructor - Copy Constructor

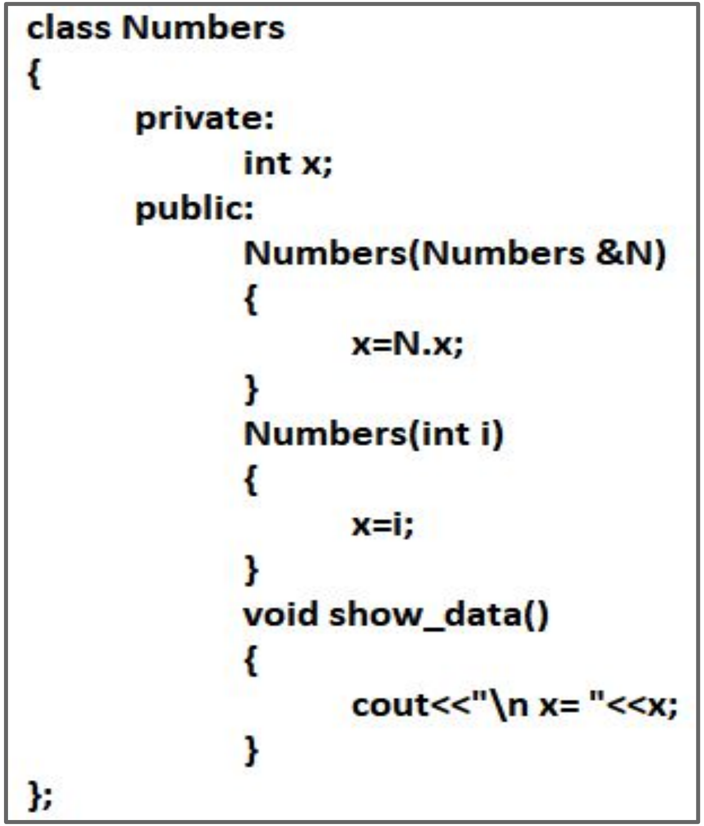
- A copy constructor takes an object of the class as an argument and copies data values of members of one object into the values of members of another object.

- Since it takes only one argument, it is also known as a one argument constructor.

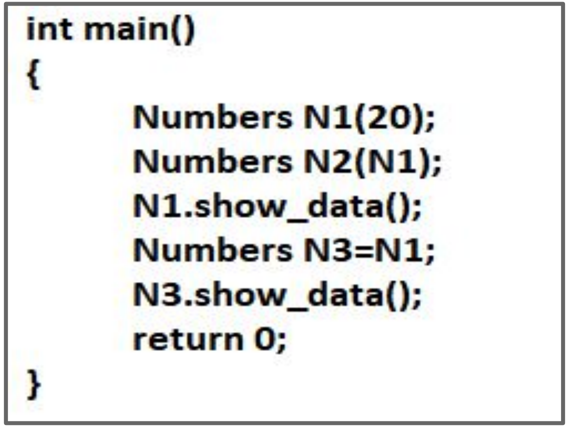
- The primary use of a copy constructor is to create a new object from an existing one by initialization.

- For this, the copy constructor takes a reference to an object of the same class as an argument.

1. Constructor - Copy Constructor



```
class Numbers
{
        private:
                int x;
        public:
                Numbers(Numbers &N)
                {
                        x=N.x;
                }
                Numbers(int i)
                {
                        x=i;
                }
                void show_data()
                {
                        cout<<"\n x= "<<x;
                }
};
```



```
int main()
{
        Numbers N1(20);
        Numbers N2(N1);
        N1.show_data();
        Numbers N3=N1;
        N3.show_data();
        return 0;
}
```

1. Constructor - Dynamic Constructor

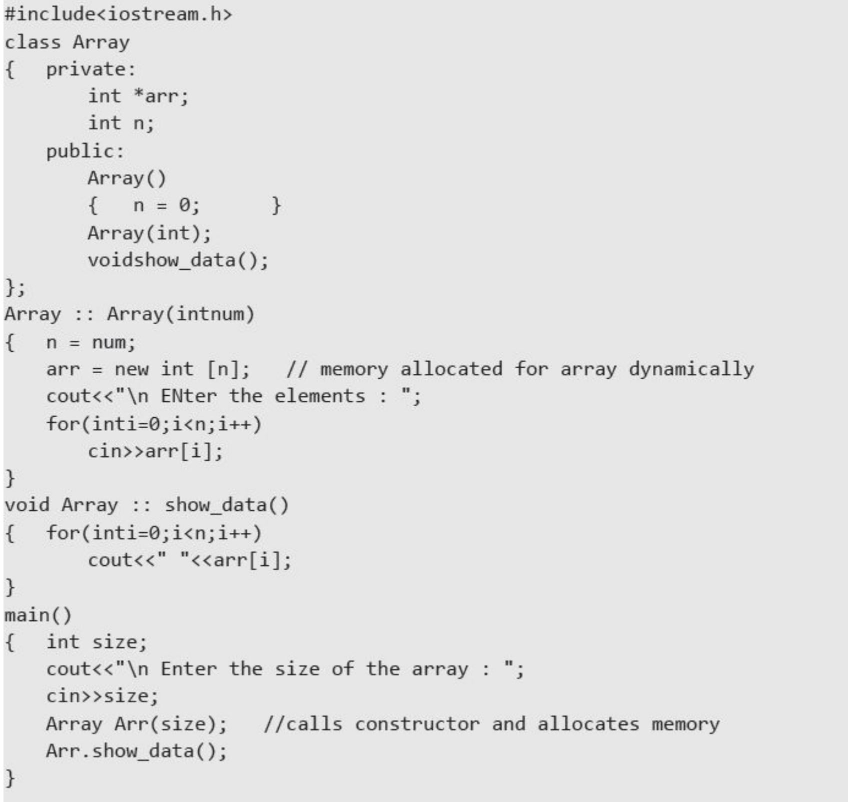
- Dynamic constructors, as the name suggests, are those constructors in which memory for data members is allocated dynamically.

- Dynamic constructor enables the program to allocate the right amount of memory to data members of the object during execution.

- This is even more beneficial when the size of data members is not the same each time the program is executed.

- The memory allocated to the data members is released when the object is no longer required and when the object goes out of scope.

1. Constructor - Dynamic Constructor

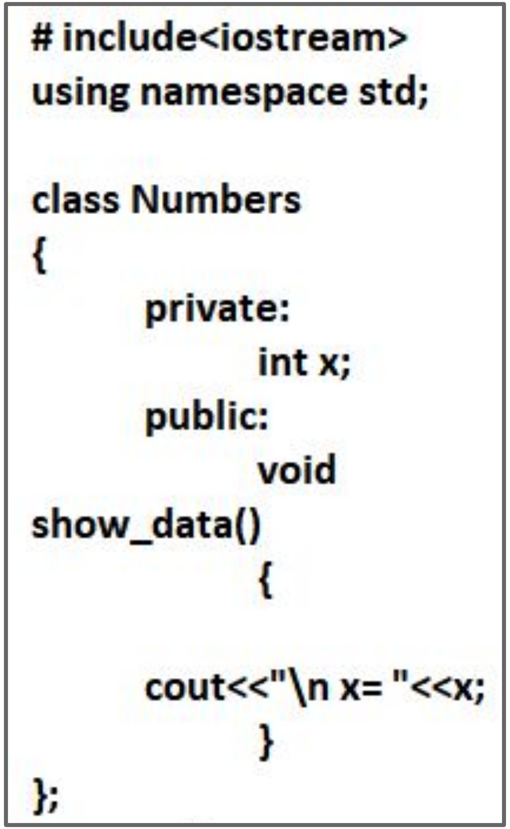


```
#include<iostream.h>
class Array
{   private:
        int *arr;
        int n;
    public:
        Array()
        {   n = 0;      }
        Array(int);
        voidshow_data();
};
Array :: Array(intnum)
{   n = num;
    arr = new int [n];   // memory allocated for array dynamically
    cout<<"\n ENter the elements : ";
    for(inti=0;i<n;i++)
        cin>>arr[i];
}
void Array :: show_data()
{   for(inti=0;i<n;i++)
        cout<<" "<<arr[i];
}
main()
{   int size;
    cout<<"\n Enter the size of the array : ";
    cin>>size;
    Array Arr(size);   //calls constructor and allocates memory
    Arr.show_data();
}
```

1. Constructor - Dummy

In Chapter 9, we had been writing programs without any constructor. In such cases, the C++ run time mechanism calls a dummy constructor which does not perform any action. Here, action means does not initialize any data member and thus, the variables acquire a garbage (irrelevant) value.

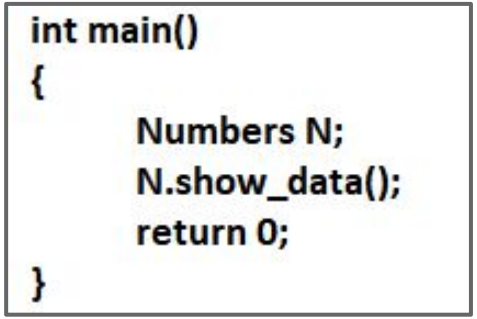
1. Constructor - Dummy



```
# include<iostream>
using namespace std;

class Numbers
{
        private:
                int x;
        public:
                void
show_data()
                {

        cout<<"\n x= "<<x;
                }
};
```



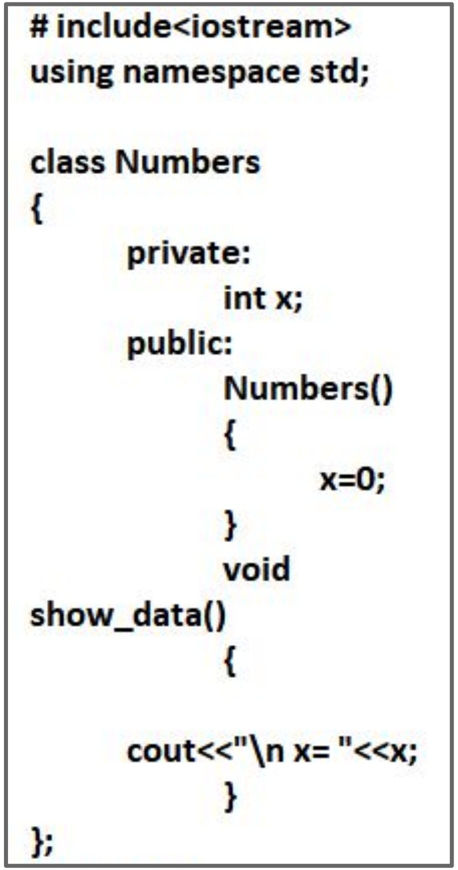
```
int main()
{
        Numbers N;
        N.show_data();
        return 0;
}
```

1. Constructor - Default

- A constructor that does not take any argument is called a default constructor.

- The default constructor simply allocates storage for the data members of the object.

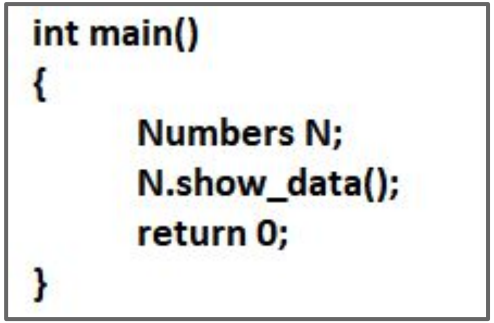
1. Constructor - Default



```
# include<iostream>
using namespace std;

class Numbers
{
        private:
                int x;
        public:
                Numbers()
                {
                        x=0;
                }
                void
show_data()
                {

        cout<<"\n x= "<<x;
                }
};
```



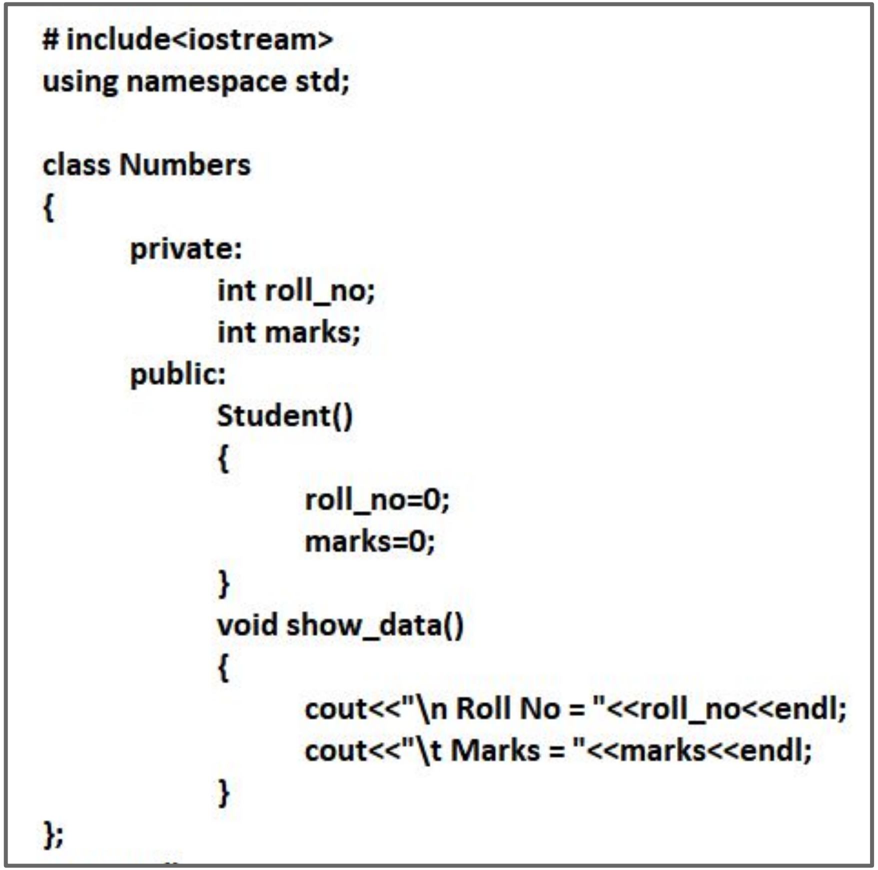
```
int main()
{
        Numbers N;
        N.show_data();
        return 0;
}
```

1. Constructor - Default Arguments

- **Like other functions, constructors can also have default arguments.**

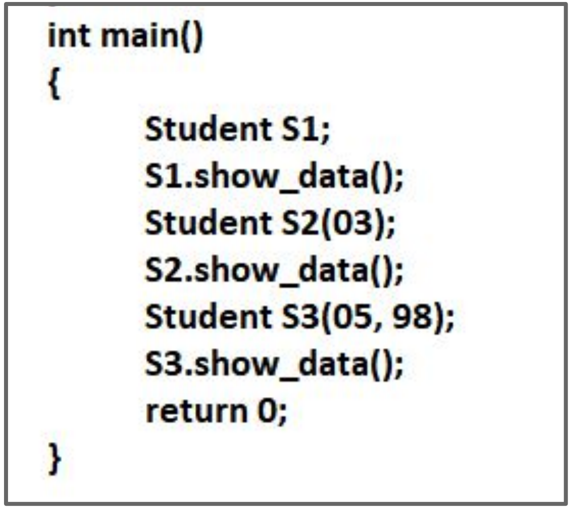
- **When an object of a class is created, the C++ compiler calls the suitable constructor for initializing that object.**

1. Constructor - Default Arguments



```
# include<iostream>
using namespace std;

class Numbers
{
        private:
                int roll_no;
                int marks;
        public:
                Student()
                {
                        roll_no=0;
                        marks=0;
                }
                void show_data()
                {
                        cout<<"\n Roll No = "<<roll_no<<endl;
                        cout<<"\t Marks = "<<marks<<endl;
                }
};
```



```
int main()
{
        Student S1;
        S1.show_data();
        Student S2(03);
        S2.show_data();
        Student S3(05, 98);
        S3.show_data();
        return 0;
}
```

2. Constructor - Array Object

```
ClassName    ObjectName[number of objects];
```

2. Constructor - Array Object (Function Call)

```
class Test
{
    private:
        int x, y;
    public:
        Test(int cx, int cy)
        {
            x = cx;
            y = cy;
        }
        void add()
        {
            cout << x + y <<endl;
        }
};
```

```
int main()
{
    Test obj[] = { Test(1, 1), Test(2, 2), Test(3, 3) };
    for (int i = 0; i < 3; i++)
    {
        obj[i].add();
    }

    return 0;
}
```

2. Constructor - Array Object (New-Delete)

```
# define N 5
class Test
{
        int x, y;
        public:
            Test() {}
            Test(int x, int y)
            {
                this->x = x;
                this->y = y;
            }
            void print()
            {
                cout << x << " " << y << endl;
            }
};
```

```
int main()
{
    Test* arr = new Test[N];

    for (int i = 0; i < N; i++)
    {
        arr[i] = Test(i, i + 1);
    }
    for (int i = 0; i < N; i++)
    {
        arr[i].print();
    }
    return 0;
}
```

2. Constructor - Overloading

- When a class has multiple constructors, they are called overloaded constructors. Some important features of overloaded constructors are as follows:

- They have the same name; the names of all the constructors is the name of the class.

- Overloaded constructors differ in their signature with respect to the number and sequence of arguments passed.

- When an object of the class is created, the specific constructor is called.

2. Constructor - Overloading

```
class construct
{
        public:
                float area;
                construct()
                {
                        area = 0;
                }
                construct(int a, int b)
                {
                        area = a * b;
                }
                void disp()
                {
                        cout<< area<< endl;
                }
};
```

```
int main()
{
        construct o;
        construct o2(10, 20);

        o.disp();
        o2.disp();
        return 0;
}
```

2. Constructor - Overloading

```
class Person
{
   private:
          int age;

   public:
          Person()
          {
                   age = 20;
          }
          Person(int a)
          {
                   age = a;
          }
          int getAge()
          {
                   return age;
          }
};
```

```
int main()
{
    Person person1, person2(45);

    cout << "Person1 Age = " << person1.getAge() << endl;
    cout << "Person2 Age = " << person2.getAge() << endl;

    return 0;
}
```

2. Destructor - I

- Like a constructor, a destructor is also a member function that is automatically invoked.

- However, unlike the constructor which constructs the object, the job of destructor is to destroy the object.

- For this, it de-allocates the memory dynamically allocated to the variable(s) or perform other clean up operations.

2. Destructor - II

- The name of the destructor is also the same as that of the class. However, the destructor's name is preceded by the tilde symbol '~'.

- A destructor is called when an object goes out of scope.

- A destructor is also called when the programmer explicitly deletes an object using the delete operator.

- Like a constructor, a destructor is also declared in the public section.

- The order of invoking a destructor is the reverse of invoking a constructor.

- Destructors do not take any argument and hence cannot be overloaded.

2. Destructor - III

- A destructor does not return any value.

- A destructor must be specifically defined to free (deallocate) the resources such as memory and files opened that have been dynamically allocated in the program.

- The address of a destructor cannot be accessed in the program.

- An object with a constructor or a destructor cannot be used as a member of a union.

- Constructors and destructors cannot be inherited. On Inheritance , unlike constructors, destructors can be virtual.

2. Destructor - Example

```
class Student
{
        private:
                int roll_no;
                int marks;
        public:
                Student(int x, int y)
                {
                        roll_no=x;
                        marks=y;
                }
                void show_data()
                {
                        cout<<"\n Roll No = "<<roll_no<<endl;
                        cout<<"\n Marks = "<<marks<<endl;
                }
                ~Student()
                {
                        cout<<"Destructor Called for student roll no : "<< roll_no;
                }
};
```

```
int main()
{
        Student S1(101,10);
        S1.show_data();

        return 0;
}
```

3. Problem - I

- Write a C++ program to deposit or withdraw money in a bank account using constructor-destructor.

- Write a C++ program to display the cheapest book available on a subject.

- Write a C++ program to add two binary numbers of four digits.

- Write a C++ program to add two matrix with dynamic allocation & destructor concept.

3. Problem - II

- Write a C++ program that dynamically allocates memory to a string. Encrypt this string and deallocate the memory.

- Write a C++ program to sort an array that has been allocate memory dynamically.

- Write a C++ menu driven program to add or delete items from your inventory of stationary range.

- Write a C++ program to display the details the of a student. The details should be given according to sorted names of the students.