

## Constructor

---

### Q. What is the constructor ?

---

Constructor is function same name as class name but without return type.

Syntax:

---

```
class A
{
    public:
    A() → constructor
    {
    }
}
```

---

### Q. Why use the constructor or what is the benefit of constructor

---

1) Constructor is used for initialize the object.

**Note:** in the case of C++ if we want initialize the value to the data member or variable in class it is not possible

So better way we can declare the constructor and we can initialize the value in it.

2) Constructor is used for execute the logic automatically when we create object of the class.

Means we can say constructor call automatically when we create the object of the class.

## Example

---

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

class Square
{ int no;
  public:
  Square()
  { no =5;
    cout<<"No is " <<no<<"\n";
  }
};

void main()
{
  clrscr();
  Square s;

  getch();
}
```

The diagram illustrates the relationship between the `Square` class constructor and the variable `s` in the `main` function. A curved arrow points from the `Square()` constructor to the `Square s;` line in `main`. Another arrow points from the `s` in `Square s;` to a box containing the text `no 5`, which is itself inside a larger box. This represents the memory state of the object `s` after the constructor has executed.

### Q. What is the diff between constructor and function?

---

- 1) Constructor name and classname must be same and function not same as classname.
- 2) Constructor cannot use return type and function must have return type

3) Constructor call automatically when we create object of class and function need to call manually as per our need.

4) Constructor cannot support recursion and function can work with recursion.

5) Constructor cannot declare as static and function may be static.

Because static member allocate memory before object and constructor need object for calling.

6) Constructor cannot be declared as abstract and function may be abstract

7) Constructor cannot be override and function may be override.

### **Types of Constructor**

---

1) Default constructor

2) Argumented constructor or parameterized constructor

3) overloaded constructor

4) copy constructor

**Note:** if we not declare the constructor within class then every class having default constructor internally called as implicit constructor.

### **Default constructor**

---

If we declare the constructor within class without parameter called as default constructor.

## Syntax:

---

```
class A
{
    public:
    A() //default constructor
    {
    }
};
```

## Q. What is the diff between implicit and default constructor

---

If use define constructor without parameter called as default and if user not define and compiler added constructor internally without parameter called as implicit.

## Program for Default constructor

---

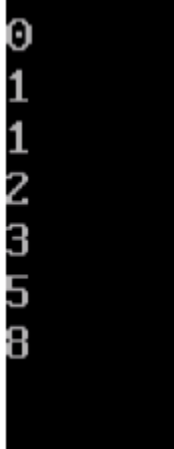
WAP to print the Fibonacci series using default constructor

```
#include<iostream.h>
#include<conio.h>
class Fibo
{
    int f1,f2,fib,i;
    Fibo()
    { f1=0;
      f2=1;
    }
    void showFibo()
    {
        for(i=1; i<=5; i++)
        { fib =f1+f2;
          f1=f2;
          f2=fib;
          cout<<fib<<"\n";
        }
    }
};

void main()
{
    clrscr();
    Fibo f;
    f.showFibo();
    getch();
}
```

Output:

---



0  
1  
1  
2  
3  
5  
8

## Parameterized constructor

---

If we pass the parameter to constructor called as parameterized constructor.

**syntax**

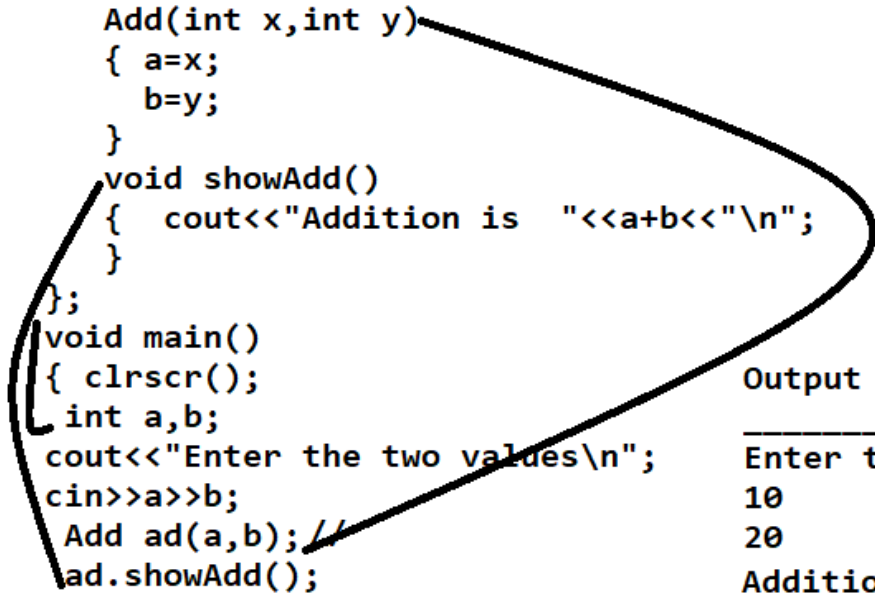
```
class classname
{
    public:
        classname(datatype variablename,datatype variablename)
        {
            write here logics of application
        }
};
```

**Note:** when we have parameterized constructor then we can pass parameter where we create the object of the class.

## Example of Parameterized constructor

---

```
#include<iostream.h>
#include<conio.h>
class Add
{
    int a,b;
public:
    Add(int x,int y)
    {
        a=x;
        b=y;
    }
    void showAdd()
    {
        cout<<"Addition is "<<a+b<<"\n";
    }
};
void main()
{
    clrscr();
    int a,b;
    cout<<"Enter the two values\n";
    cin>>a>>b;
    Add ad(a,b);
    ad.showAdd();
    getch();
}
```

A large hand-drawn black oval encircles the `Add(int x, int y)` constructor, the `showAdd()` method, and the `ad.showAdd();` call in the `main()` function. A line extends from the right side of this oval towards the output section.

### Output

```
Enter the two values
10
20
Addition is 30
```

## Overloaded constructor

---

### What is the constructor overloading?

---

Constructor overloading means if we define the same name constructor with different parameter with different sequence with different data type called as constructor overloading.

### Example

---

```
class Square {
public :
    Square (int x)
```

```

{
}

Square (float x)

{
}

};

```

**Note:** in the case of constructor overloading which constructor get executed is depend on how much parameter pass in it and its sequence.

```

#include<iostream.h>
#include<conio.h>
class Square
{
    public:
    Square(int x)
    { cout<<"Square of integer is  "<<x*x<<"\n";
    }
    Square(float x)
    { cout<<"Square of float is  "<<x*x<<"\n";
    }
};
void main()
{
    clrscr();
    Square s1(5); //call integer constructor
    Square s2(5.5f); //call float constructor
    getch();
}

```

## Output

---

```
Square of integer is 25  
Square of float is 30.25
```

## Copy Constructor

---

Copy constructor is used for copy the content of one object in to another object. Copy constructor normally used for perform object cloning concept.

### How to define the copy constructor

---

```
class ABC  
{  
    public:  
    ABC(int x)//argument constructor  
    {  
    }  
    ABC(ABC &a) //copy constructor  
    {  
    }  
};
```

Copy constructor can call by using two ways

---

#### I) using implicit copy constructor:

---

**Classname object in which content want to copy (object whose content want to copy);**



## Example

---

Suppose consider we have the two objects of class ABC name as a1 and a2 and we want to copy the content of a2 in a1

```
ABC a2(100);
```

```
ABC a1(a2); // we copy the content of a2 in a1
```

### ii) Using explicit copy constructor :

Suppose consider we have the two objects of class ABC name as a1 and a2 and we want to copy the content of a2 in a1

```
ABC a2(100);
```

```
ABC a1=a2; //we copy the content of a2 in a1
```

Following Diagram shows the working of copy constructor

---

```
#include<iostream.h>
#include<conio.h>
class ABC
{
    int no;
public:
    ABC(int x) //argument constructor
    { no=x;
    }
    ABC(ABC &a)
    { a1.no = a2.no
      this ->no = a.no;
    }
};
void main()
{
    clrscr();
    ABC a2(100);
    ABC a1(a2);
    getch();
}
```

no 100

no 100

no 100

a2

a1

In Above code we have the class name as ABC with two constructor ABC(int x) normal parameterized constructor and ABC(ABC &a) copy constructor.

In main function we have the statement ABC a2(100) using this statement we call the argument constructor of type integer.

Means we created object in memory by name a2 and in that a2 contain no variable which contain 100 value

In main function we have the next statement ABC a1(a2) using this statement we call the copy constructor and in copy constructor we have the statement this->no = a.no here this reference points to current running object and our current running object is a1 means this points to a1 and a2 we pass as parameter and our parameter name is a means a point to a2 means here we copy the content a2 object in to the a1 object called as copy constructor or in short called as object cloning concept.

Source Code

---

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

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

```
class ABC
```

```
{ int no;
```

```
public:
```

```
ABC(int x)
```

```

{
    no=x;
}
ABC(ABC &a)
{
    this -> no = a.no;
}
void show()
{
    cout<<no<<"\n";
}
};

void main()
{ clrscr();
  ABC a2(100);
  ABC a1(a2);
  cout<<"With First A2 Object"<<"\n";
  a2.show();
  cout<<"After copy in A1 object"<<"\n";
  a1.show(); getch(); }

```

## Output

---

```
With First A2 Object
100
After copy in A1 object
100
```

## Pointer with Object

---

It is used for allocate dynamic object at run time.

### Syntax:

---

classname \*objectname;

e.g Employee \*emp;

When we use the pointer with object then we use the -> operator for access the class member.

### Example

---

```
#include<iostream.h>

#include<conio.h>

class Employee
{ private:
    int id;
    char name[90];
    int sal;
    public:
```

```

void setData()
{
cout<<"Enter the name id and salary of employee\n";
    cin>>name>>id>>sal;
}

void showData()
{
    cout<<name<<"\t"<<id<<"\t"<<sal<<"\n";
}

};

void main()
{
    clrscr();
    Employee *emp;
    emp->setData(); //call setData() function
    emp->showData(); //call showData()
    getch();
}

```

## Source Code For Dyanmic object creation

---

```
#include<iostream.h>

#include<conio.h>

class Employee
{ private:
    int id;

    char name[90];

    int sal;

public:
    void setData()
    {
cout<<"Enter the name id and salary of employee\n";

        cin>>name>>id>>sal;

    }

    void showData()
    { cout<<name<<"\t"<<id<<"\t"<<sal<<"\n";

    }

};

void main()
{ clrscr();
```

```
Employee *emp;  
emp->setData(); //call setData() function  
emp->showData(); //call showData()  
getch();  
}
```

## Output

---

```
Enter employee count  
3  
Enter the name id and salary of employee  
a  
1  
1000  
Enter the name id and salary of employee  
b  
2  
2000  
Enter the name id and salary of employee  
c  
3  
3000  
Display Employee Record  
a      1      1000  
b      2      2000  
c      3      3000
```

