# CONSTRUCTORS

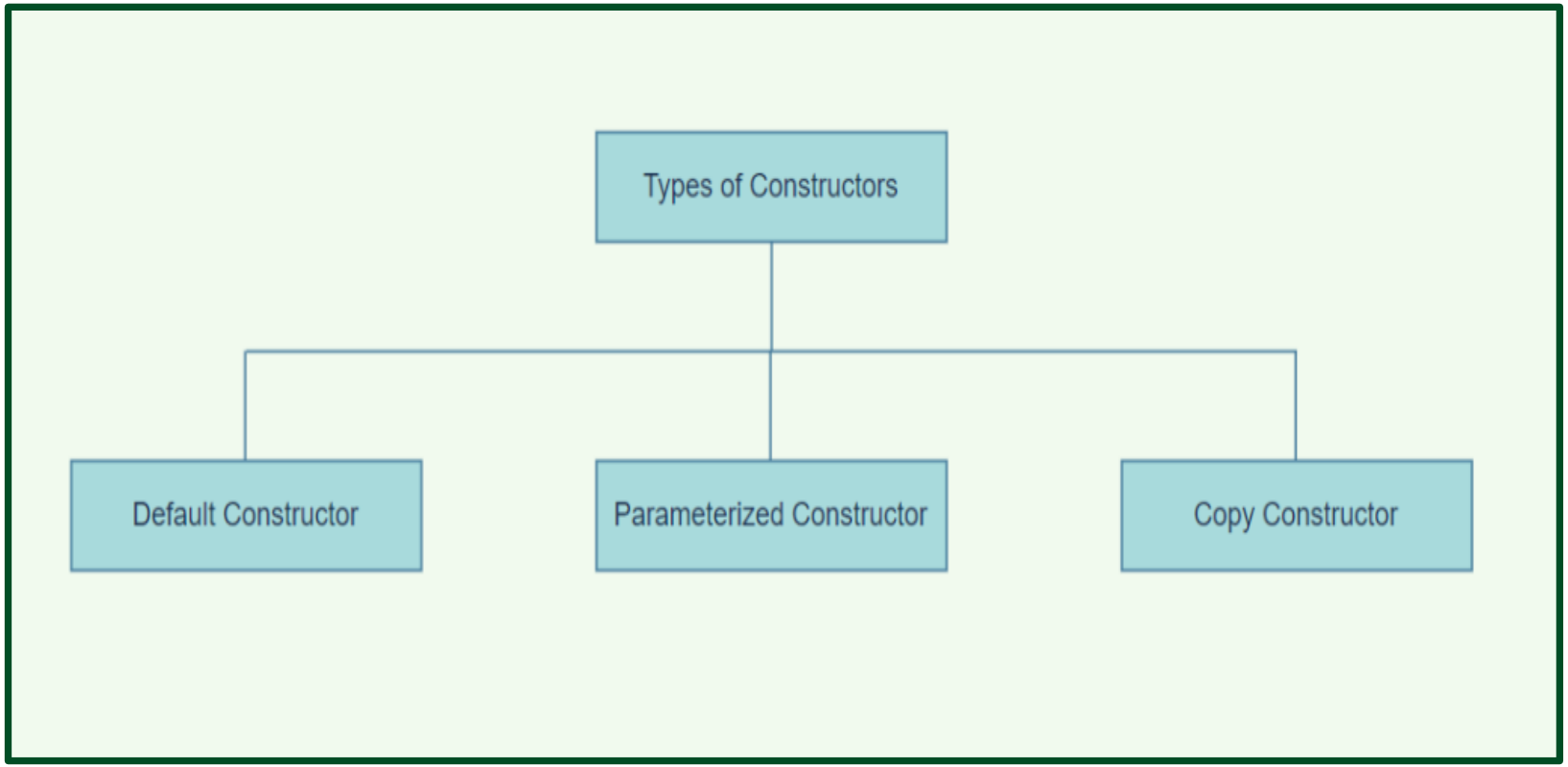
- **A constructor is a member function that is invoked automatically when an object is declared.**
- **A constructor function must have the same name as the class itself, and it is declared without return type.**
- ***To illustrate this, consider the following program***

```
#include <iostream>
using namespace std;
class Employee
 {
   public:
        Employee()
        {
            cout<<"Default Constructor Invoked"<<endl;
        }
};
int main(void)
{
    Employee e1; //creating an object of Employee
    Employee e2;
    return 0;
}
```

**Here, the function Employee() is a constructor of the class Employee.**

***Notice that the constructor***

- **has the same name as the class,**
- **does not have a return type, and**
- **is public**


Types of Constructors
Default Constructor
Parameterized Constructor
Copy Constructor

- **A constructor with no parameters is known as a *default constructor*. For example,**

```
#include <iostream>
using namespace std;
class  Test {          // declare a class
    private:
        double length;
    public:
     // default constructor to initialize variable
     Test()
     {
         length = 5.5;
         cout << "Creating a wall." << endl;
         cout << "Length = " << length << endl;
     }
};
int main() {
     Test T1;
     return 0;
}
```

- **The run of this program will be as follows:**

```
Creating a wall.
Length = 5.5
```

- **Here, when the T1 object is created, the Test() constructor is called.**

- **This sets the length variable of the object to 5.5.**

## EX. Consider the following code:

```
#include <iostream>
using namespace std;
class Counter
{
   private:
     unsigned int count;     //count
   public:
  Counter() : count(0)    //constructor
  {
     /*empty body*/
  }
void inc_count()         //increment count
{
   count++;
}
int get_count()          //return count
{
   return count;
}
};
```

## *Continued*

```
int main()
{
    Counter c1, c2; //define and initialize
    cout << "\nc1 ="<<  c1.get_count(); //display
    cout << "\nc2 ="<<  c2.get_count();
    c1.inc_count(); //increment c1
    c2.inc_count(); //increment c2
    c2.inc_count(); //increment c2
    cout << "\nc1 =" << c1.get_count(); //display again
    cout << "\nc2 =" << c2.get_count();
    cout << endl;
    return 0;
}
```

**Sample Run**

```
c1 =0
c2 =0
c1 =1
c2 =2
```

- **As can be noted, the Counter class has one data member: count, of type unsigned int (since the count is always positive).**

- **The statement `Counter c1, c2;` creates two objects of type Counter.**

- **As each is created, its *constructor*, Counter(), is executed.**

- **This function sets the count variable to 0, so the effect of this single statement is to not only create two objects, but also to initialize their count variables to 0.**

## Parameterized Constructor

- A constructor with parameters is known as a parameterized constructor. This is the preferred method to initialize member data. For example,

```
// C++ program to calculate the area of a wall
#include <iostream>
using namespace std;
class Wall {          // declare a class
private:
    double length;
    double height;
public:
    // parameterized constructor to initialize variables
    Wall(double len, double hgt) {
        length = len;
        height = hgt;
    }
    double calculateArea() {
        return length * height;
    }
};
```

## *Continued*

```
int main() {
  // create object and initialize data members
  Wall wall1(10.5, 8.6);
  Wall wall2(8.5, 6.3);
  cout << "Area of Wall 1:" << wall1.calculateArea() << endl;
  cout << "Area  of Wall 2: " << wall2.calculateArea();
  return 0;
}
```

**Sample Run**

```
Area of Wall 1: 90.3
Area of Wall 2: 53.55
```

- **Here, a parameterized constructor Wall() is created that has two parameters: double len and double hgt.**

- **The values contained in these parameters are used to initialize the member variables length and height.**

- **When we create an object of the Wall class, we pass the values for the member variables as arguments.**

- **With the member variables thus initialized, we can now calculate the area of the wall with the calculateArea() function.**

**Ex.**

```
#include <iostream>
using namespace std;
class Ratio
{
  private:
    int num, den;
  public:
  Ratio(int n, int d)   //flarameterized Constructor
  {
       num = n; den = d;
  }
  void print()
  {
     cout << num << '/' << den;
  }
};
```

## *Continued*

```
int main()
{
    Ratio x(-1, 3), y(22, 7);
    cout << "x = ";
    x.print();
    cout << " and y = ";
    y.print();
    return 0;
}
```

**Sample Output:**

```
x = -1/3 and y = 22/7
```

## ➢ Ex: Adding More Constructors to the Ratio Class

```
#include <iostream>
using namespace std;
class Ratio
{
    private:
    int num, den;
    public:
    Ratio()
    {
      num = 0; den = 1;
    }
    Ratio(int n)
    {
      num = n; den = 1;
    }
    Ratio(int n, int d)
    {
      num = n; den = d;
    }
    void print() { cout << num << '/' << den; }
};
```

## *Continued*

```
int main()
{
Ratio x, y(4), z(22, 7);
cout << "x = ";
x.print();
cout << "\ny = ";
y.print();
cout << "\nz = ";
z.print();
}
```

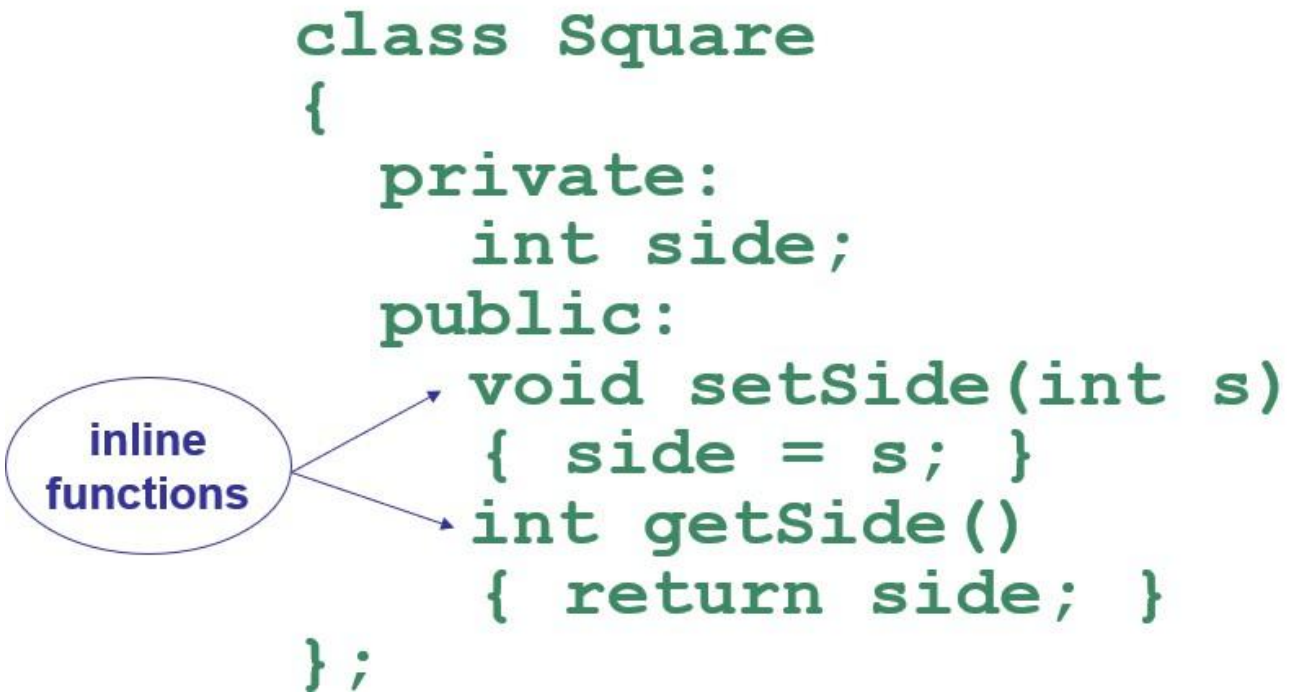
**Sample Run**

```
x = 0/1
y = 4/1
z = 22/7
```

# Inline Member Function

- **Member functions defined inside the class declaration are called *inline functions***

***Ex.***



- *Note that, constructor can be defined outside the class as follows:*

**Inline:**

```
class Square
{
   . . .
  public:
     Square(int s)
     { side = s; }
   . . .
};
```

**Declaration outside the class:**

```
Square(int);   //prototype
               //in class

Square::Square(int s)
{
    side = s;
}
```

# Overloading Constructors

- **As discussed earlier, a class can have more than 1 constructor**

- ***Overloaded constructors*** **in a class must have different parameter lists**

```
Square();
Square(int);
```

## Copy Constructor

- **The copy constructor in C++ is used to *copy data from one object to another*. Simply, you can initialize object with another object of the same type.**

- **The following ECOPYCON program shows how copy constructor is used.**

```
// flrogram ecopycon
// initialize objects using default copy constructor
#include <iostream>
using namespace std;
class Distance          //English Distance class
{
private:
int feet;
float inches;
public:
//constructor (no args)
Distance() : feet(0), inches(0.0)
{ }
//Note: no one-arg constructor
//constructor (two args)
Distance(int ft, float in) : feet(ft), inches(in)
{ }
```

## *Continued:*

```
    void getdist() //get length from user
    {
      cout <<"\nEnter  feet : " ;
      cin >> feet;
      cout << "Enter inches : ";
      cin >> inches;
    }
    void showdist() //display distance
    {
      cout << feet << "'- " << inches << "''";
    }
};
```

## *Continued:*

```
int main()
{
    Distance dist1(11, 6.25);//two-arg constructor
    Distance dist2(dist1);   //one-arg constructor
    Distance dist3 = dist1; //also one-arg constructor
    //display all lengths
    cout << "\n dist1 = "; dist1.showdist();
    cout << "\n dist2 = "; dist2.showdist();
    cout << "\n dist3 = "; dist3.showdist();
    cout << endl;
    return 0;
}
```

```
dist1 = 11'- 6.25''
dist2 = 11'- 6.25''
dist3 = 11'- 6.25''
```

- **Based on the program given above, we initialize dist1 to the value of 11'-6.25" using the two-argument constructor.**

- **Then we define two more objects of type Distance, dist2 and dist3, initializing both to the value of dist1.**

- **You might think this would require us to define a one-argument constructor, but initializing an object with another object of the same type is a special case.**

- **These definitions both use the default copy constructor. The object dist2 is initialized in the statement**

```
Distance dist2(dist1);
```

- **This causes the default *copy constructor* for the Distance class to perform a *member-by-member copy* of dist1 into dist2.**

- **A different format has exactly the same effect, causing dist1 to be copied member-by-member into dist3:**

```
Distance dist3 = dist1;
```

- **Although this looks like an assignment statement, it is not. *Both formats invoke the default copy constructor, and can be used interchangeably.***