20MCA11C OBJECT ORIENTED PROGRAMMING WITH C++

UNIT III: Constructors

FACULTY

Dr. K. ARTHI MCA, M.Phil., Ph.D.,

Assistant Professor,

Postgraduate Department of Computer Applications,

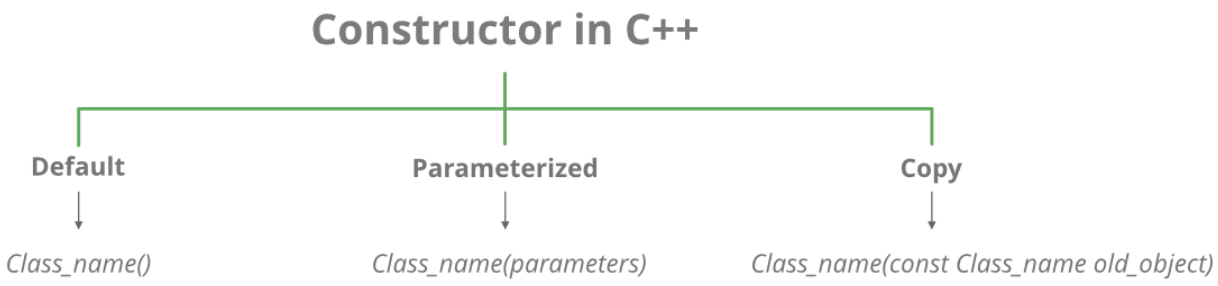
Government Arts College (Autonomous),

Coimbatore-641018.

constructors

A **constructor** is a member function of a class which initializes objects of a class. In C++, **Constructor** is automatically called when object(instance of class) create. It is special member function of the class

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



C++ Default Constructor

A constructor with no parameters is known as a **default constructor**.

- **Parameterized constructor :-**

In default constructor every objects are initialized with the same values but in many cases, it may be necessary to initialize the various data elements of different object with different values when they are permits us to achieve this objective by passing argument to the construct function. When the objects are created, the constructors that can take arguments are called parameterized constructors.

E.g.
```
Class integer
{
        int m,n;
Public:
        integer(int x, int y)          // parameterized constructor
        {
        m=x;
        n=y;
        }
};
```

When a constructor has been parameterized, the object of declaration statement as

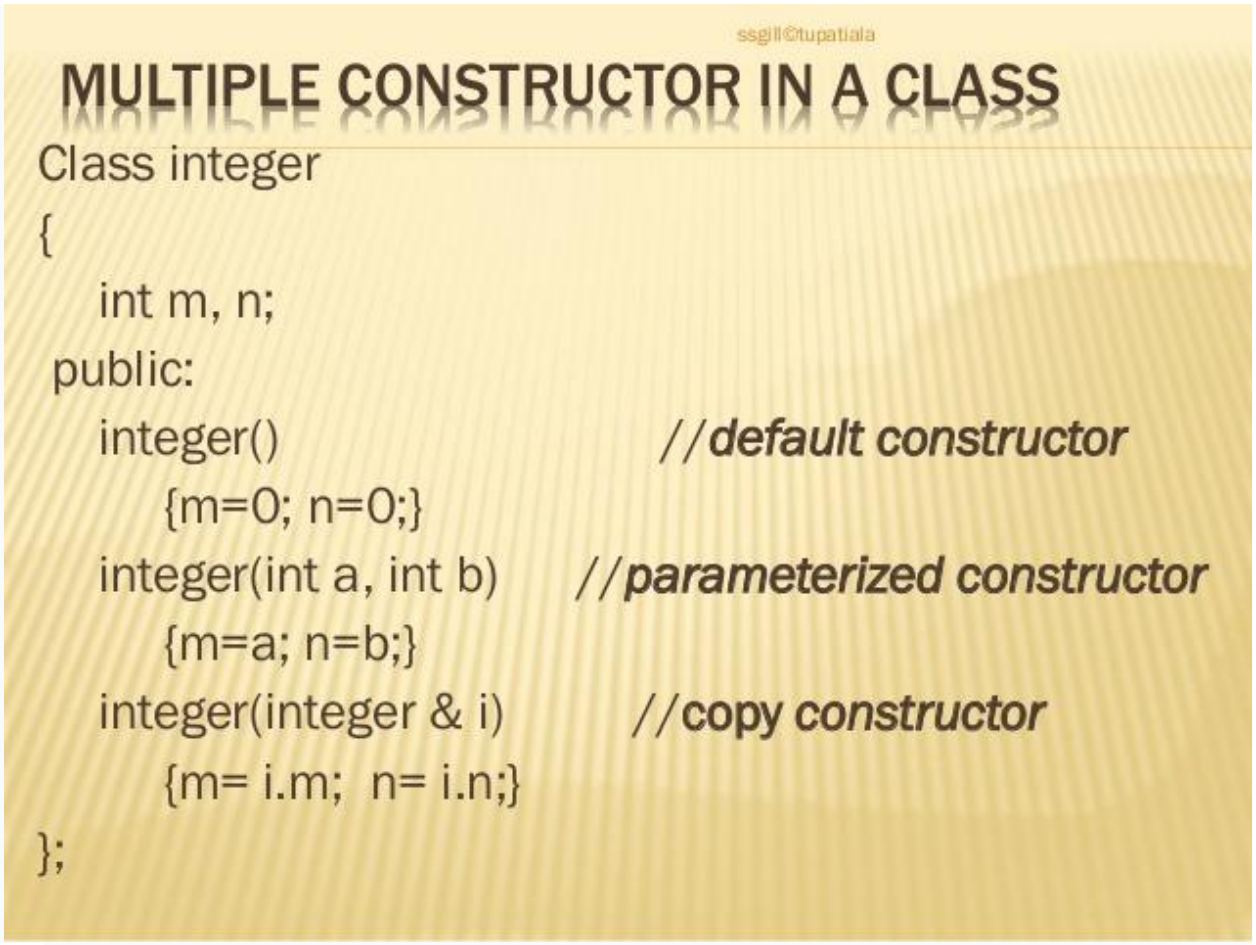
```
integer int1;
integer int1=integer(0,100);
```

This statement creates an integer objects int1 & passes the values 0 and 100 to it.

```
integer int1(0,100);
```

Copy Constructor

- A copy constructor is called whenever a new variable is created from an object
  - `Point p(5,5);`
  - `Point a = p; // Copy constructor call!`
  - `Point b(p); // Copy constructor call!`
- C++ creates **default copy constructor** automatically



- Destructors
  - Special member function
  - Same name as class
    - Preceded with tilde (~)
  - No arguments
  - No return value
  - Cannot be overloaded
  - Before system reclaims object's memory
    - Reuse memory for new objects

Mainly used to de-allocate dynamic memory locations

Introduction to Operator Overloading in C++

```
a1 = a2 + a3;
```

The above operation is valid, as you know if a1, a2 and a3 are instances of in-built *Data Types*. But what if those are, say objects of a *Class*; is the operation valid?

Yes, it is, if you overload the '+' *Operator* in the class, to which a1, a2 and a3 belong.

Operator overloading is used to give special meaning to the commonly used operators (such as +, -, * etc.) with respect to a class. By overloading operators, we can control or define how an operator should operate on data with respect to a class.

Operators are overloaded in c++ by creating operator functions either as a member or a s a Friend Function of a class. Since creating member operator functions are easier, we'll be using that method in this article.

As I said operator functions are declared using the following general form:

```
ret-type operator#(arg-list);
```

and then defining it as a normal member function.

Here, ret-type is commonly the name of the class itself as the operations would commonly return data (object) of that class type.

is replaced by any valid operator such as +, -, *, /, ++, -- etc.

Now that you have understood the theory, let's have a look at an example program:

```
// Example program to illustrate
// operator overloading
#include <iostream.h>

class myclass
{
  int sub1, sub2;

public:
  // default constructor
  myclass(){}

  // main constructor
  myclass(int x, int y){sub1=x;sub2=y;}

  // notice the declaration
  myclass operator +(myclass);

  void show(){cout<<sub1<<endl<<sub2;}
};

// returns data of type myclass
```

```
myclass myclass::operator +(myclass ob)
{
  myclass temp;

  // add the data of the object
  // that generated the call
  // with the data of the object
  // passed to it and store in temp
  temp.sub1=sub1 + ob.sub1;
  temp.sub2=sub2 + ob.sub2;

  return temp;
}

void main()
{
  myclass ob1(10,90);
  myclass ob2(90,10);

  // this is valid
  ob1=ob1+ob2;

  ob1.show();
}
```

At this stage many of you might be wondering why the operator function is taking only one argument when it's operating on two objects (i.e. it's a binary operator).

To understand this, first have a look at this line of code:

```
ob1 = ob1 + ob2;
```

Now assume that 'operator+' function is just a regular function which is called as below when the respective operator ('+' in this case) is encountered with respect to the objects of its class.

```
ob1 = ob1.operator+(ob2);
```

THANK YOU

This content is taken from the text books and reference books prescribed in the syllabus.