

## CONSTRUCTORS

Why required?

*After initialization of class, we set value in next line, using put member function.*

E.g.: -

Student Rahul;

Rahul.put(1, 'Rahul Sharma');

**C++ Aims** to bring user defined datatypes as close as built-in datatypes

Hence just like built in datatypes:

Int x = 20;

Char z = 'k';

*[Declaration → init]*

We need Constructor – which constructs the data member value while initialization.

E.g.:

Class Code

{

int data;

int data2;

public:

Code()

{

data = 0;

data2 = 0;

}

};

## DESTRUCTORS

Why required?

Eg:- we will be seeing this in last of pdf

In built-in datatype -> variable goes out of scope -> compiler destroys it to free memory.

BUT we don't have anything like that in class.

Hence, we need DESTRUCTOR.

CONSTRUCTOR: For automatic initialization of objects

DESTRUCTOR: For Automatic Destruction of objects.

## SPECIAL CHARACTERISTICS OF CONSTRUCTOR

- Declare in PUBLIC Section only
- No RETURN type
- They CANNOT be INHERITED, but can be called from base class. (In inheritance)
- They CANNOT be VIRTUAL (in polymorphism we will see)
- They are invoked automatically

## TYPES OF CONSTRUCTORS

- |                             |                                 |
|-----------------------------|---------------------------------|
| - Default Constructor       | - No arguments                  |
| - Parameterized Constructor | - Yes arguments                 |
| - Copy Constructor          | - Only Class Itself as argument |

code() →  
code(int c, ...)  
code(code obj)

Note: If you declare a constructor, initialization becomes mandatory.

## HOW WE CAN CALL CONSTRUCTOR

- Explicitly -> ( Student Rahul = Student(1, 'Rahul Sharma'); )
- Implicitly -> ( Student Rahul(1, 'Rahul Sharma'), )

Note: Constructor can be INLINE too -> Define inside class, for NON-INLINE Define outside class.

Similar to functions -> Constructor can be Overloaded.

Like having multiple constructors together, default, parameterized and copy constructors together.

NOTE: Constructor can be inline if (Defined inside class), Non-Inline (Defined outside)

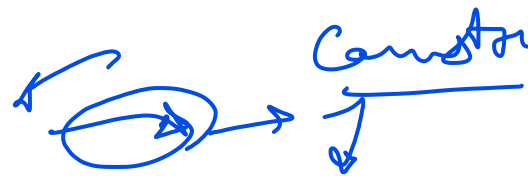
class  
{  
...  
}  
23

## COPY CONSTRUCTOR

A(A)



copy by value -



A(A&)



Constructor can take any argument except its own class data. Hence, we pass reference

Reason: Because if it's not by reference, it's by value. To do that you make a copy, and to do that you call the copy constructor. But to do that, we need to make a new value, so we call the copy constructor, and so on...

(You would have infinite recursion because "to make a copy, you need to make a copy".)

## DYNAMIC INITIALIZATION

As we have seen Implicit, and explicit initialization,

If we have runtime data to be given to class, we have dynamic initialization.

Eg:

Int m,n; cin>>m>>n; // taking values at runtime

Code A = Code(m,n); // passing it to constructor, and initialisation is dynamic now

Code A(m,n)

## COPY INITIALIZATION (MOST IMPORTANT)

Process of Initialization using copy constructor

Code C=A; //Using Copy Constructor

Code B(A); //Using Copy Constructor

Code D; // Declaration

D=A; // Using Operator Overloading

NOTE: Copy Constructor is provided by Default

Code (Code B A)  
data : A data  
data2 : A data2

class default

Code ( ) { }

## CONST OBJECT

Const code(m, n);

Can't change the value, and only const member function will be call for them

## DESTRUCTOR

How it looks! ->

~Code () {....}

No args, No return type.

Note: Good to declare as it frees memory.

Code

~~private: Code() {}~~  
Code(int x, int y)  
{  
    d1 = x;  
    d2 = y;  
}

Code