Constructor and Destructor

-by Praveen Choudhary

CONSTRUCTORS

- It is a special member function of a class , which is used to construct the memory of object & provides initialization.
- Its name is same as class name.
- It must be declared in public part otherwise result will be error.

- It does not return any value not even void otherwise result will be error.
- It can be defined by inline /offline method.
- Does not need to call because it get call automatically whenever object is created.
- It can be called explicitly also.
- It can take parameters.
- Constructer can be overloaded.
- It does not inherit.

Destructors

- It is a special member function of a class , which is used to destroy the memory of object
- Its name is same as class name but till sign followed by destructor.
- It must be declared in public part otherwise result will be error.
- It does not return any value not even void otherwise result will be error.
- It can be defined by inline /offline method.

- Does not need to call because it get call automatically whenever object destroy from his scope.
- It can be called explicitly also using delete operator.
- It does not take parameters.
- Destructor can not be overloaded , only one destructor is possible in a class.
- It does not inherit.

Types of constructor

i. default constructor
ii. parameterized constructor
iii. default parameterized constructor
iv. copy constructor
v. dynamic constructor

Their is no type of destructor.

Example 1-

```
#include<iostream.h>
class demo
{
        int x,y;
};
void main()
{
        demo d1,d2,d3;
}
```

***Note**- if we are not defining user define constructor and destructor then compiler create its own constructor and destructor to create the memory of object and to destroy the memory of object but we cannot initialize our object in compiler constructor.*

Example of default constructor.

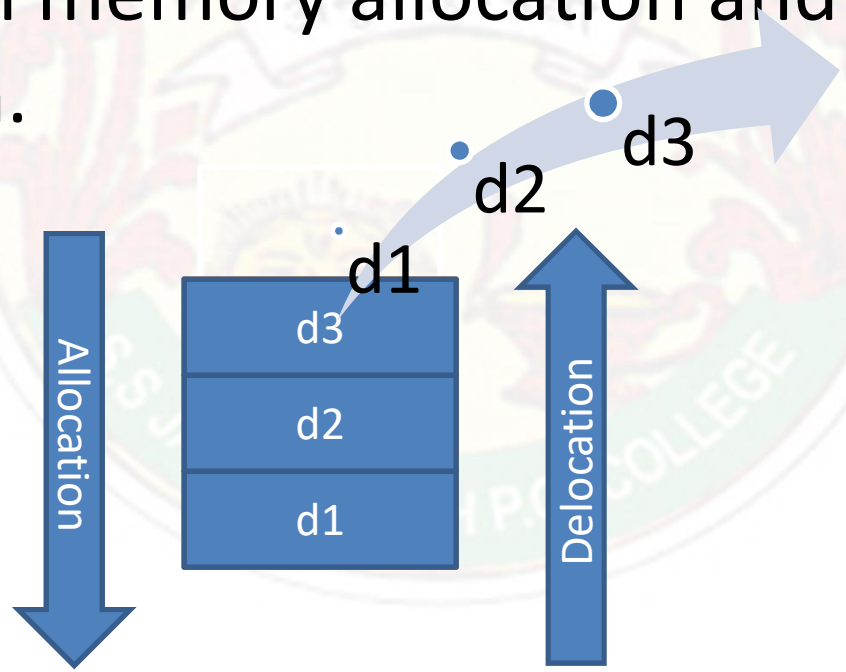
```
#include<iostream.h>
class demo
{int x,,y;
public:
demo()
{
x=0,y=0;
cout<<"\n dc"<<(unsigned int)this;
}
~demo(){
cout<<"\n od"<<(unsigned int)this;
}};
```

```
void main()
{
demo d1,d2,d3;//implicit call of
    default constructor
}
```

output
-
dc-65119
dc-65125
dc-65129
od-65129
od-65125
od-65119

NOTE-

- In this scope compiler create the memory of object sequentially and destroy in reverse order because "c++"compiler uses the concept of stack in memory allocation and de allocation.



Example of parameterized constructor

```
#include<iostream.h>
class demo
{int x,,y;
public:
demo()
{
x=0,y=0;
cout<<"\n dc";}
~demo()
{
cout<<"\n od";}
demo(int a, int b)
{
x=a,y=b;
cout<<"\n pc";
}};
```

```
void main()
{
demo d1,d2;//implicit call of default
    constructor
demo d3(5,7);// implicit call of
    parameterized constructor
demo d4=demo(10,12);//explicit call of
    parameterized constructor
demo d5;
}
```

output-

dc
dc
pc
pc
dc
od
od
od
od
od

Example of default & default parameterized construct

```
#include<iostream.h>
class demo
{int x,,y,z;
public:
demo()
{
x=0,y=0,z=0;
cout<<"\n dc";}
~demo()
{
cout<<"\n od";}
demo(int a, int b,int c=10)
{
x=a,y=b,z=c;
cout<<"\n dpc";}};
void main()
{
demo d1,d2;//implicit call of default
    constructor
demo d3(5,7);// implicit call of default par.
    constructor
demo d4(1,2,3);//explicit call of default
 par. constructor
demo d5=demo(10,12);
demo d6=demo(1,8,2);
d1=demo(9,9);//explicit call for existing
    object
d2=demo(1,7,1);//explicit call for existing
    object
}
```

output

```
Dc
Dc
Dpc
Dpc
Dpc
dpc
dpc
Od
Dpc
Od
Od
Od
Od
Od
Od
Od
```

Invalid definition of default parameterized constructor

Case 1-

```
Demo(int a,int b,int c=8)
{
----------;
----------;
----------;
}
```

Demo(, ,8)

Case2-

```
Demo(int a,int b=8,int c)
{
----------;
----------;
----------;
}
```

Demo(,8,)

INVALID

Copy constructor

Its a kind of constructor n gets called in following cases-

case 1-

when ever we are going to initialize any new object by existing object.

Ex-

demod4(d2);

demod4=d2;

case2-

when ever we pass object as an arguments a function compiler create the memory of formal parameter(object)using copy constructor.

EX-void call(demo a, demo b)

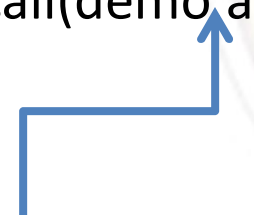
{

demo c;

}

called by-

call(d4 ,d5)



case 3-

when ever any function return object as a return value compiler create a temporary object using copy constructor to return this value to one fn to another fn.

Ex-

```
demo call(_ _ _)
{
demo c;
.
.
.
return c;

}
```

NOTE-Compiler create one temporary object using copy constructor to return this value.

syntax:-

```
class_name(const class_name & ref_object)
{
---
---
---
}
```

NOTE-

if '&' is not used in copy constructor compiler will create a new object in place of sharing the memory and constructor is used to create an object so its a recursive process and const keyword is used so that no change can be made in existing object.

Example-

```
#include<iostream.h>
class demo
{int x,,y;
public:
demo()
{
x=0,y=0;
cout<<"\n dc";}
~demo()
{
cout<<"\n od";}
demo(int a, int b)
{
x=a,y=b;
cout<<"\n pc";}
```

```
demo (const demo &d)
{
x=d.x;
y=d.y;
cout<<"\n cc";
}};
void main()
{
demo d1,d2;//implicit call of default
    constructor
demo d3(5,7);
demo d4=(10,12);
demo d5(D3);
demo d6=d4;
}
```

output-

dc,dc
pc,pc
CC,CC,
od,od,
od,od,
od,od

WAP of sum using copy constructor

```
#inckude<iostream.h>
class demo
{
int x,,y;
public:
demo()
{x=0,y=0;
cout<<"\n dc";}
~demo()
{cout<<"\n od";}
demo(int a, int b)
{x=a,y=b;
cout<<"\n pc";}
demo (const demo &d)
{x=d.x;
y=d.y;
cout<<"\n cc";}};
```

```
demo sum(demo a, demo b)
{
demo c;
c.x=a.x+b.x;
c.y=a.y+b.y;
return c;
}
void main()
{
demo d1,d2;
demo d3(5,7);
demo d4=demo(10,12);
d2=d1.sum(d3,d4);
}
```

Output-
Dc,dc
Pc,pc
Cc,cc
dc
Cc,od
Od,od,od
Od,od,
Od,od

DRAWBACK OF USER DEFINE CONSTRUCTOR

In user define ***Constructors & Destructors*** *programming, we can not create any other type of object if its constructor definition is not define otherwise result will be compile time error.*