Sameer Sir Classes, Jabalpur
Auth Exam Center Oracle, Microsoft
9407077858

CONSTRUCTOR

PROPERTIES OF CONSTRUCTOR

1. SPECIAL MEMBER FUNCTION
2. AUTOMATIC CALL WHEN OBJECT IS CREATED
3. SAME NAME AS CLASS NAME
4. DECLARE IN PUBLIC SECTION // definition
5. INITIALIZE DATA MEMBER
6. DYNAMIC INITIALIZATION // uses
7. PASSING ARGUMENT ✓
8. NO RETURN (even void X)
9. OVERLOADING ✓
10. DEFAULT ARGUMENT ✓ // function
11. VIRTUAL X
12. INHERIT X

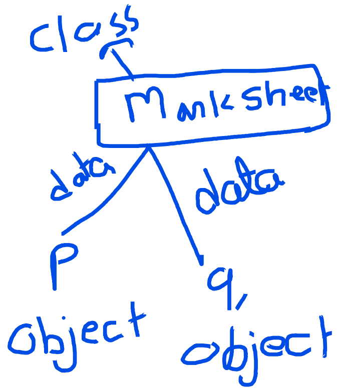
TYPES OF CONSTRUCTOR

1. DEFAULT OR NO ARGUMENT CONSTRUCTOR

2. PARAMETERIZED OR ARGUMENT CONSTRUCTOR

3. COPY CONSTRUCTOR

WAP TO FIND FACTORIAL



```
#include<iostream>
using namespace std;

  class  fact
  {
        private :   int  n , f ;  // only declaration

        public  :
                  fact ()      // special member function ( constructor )
                  {
                       f = 1;  // initialize data member
                  }
                  void   get()
                  {
                          cout<< " ENTER NO " << endl;
                          cin >> n;
                  }
                  void  cal()
                  {
                          for( int i = 1 ; i <= n ; i++)
                          {
                                f = f * i;
                          }
                  }
                  void  out()
                  {
                          cout << " FACT = " << f << endl;
                  }
          };
          int     main()
          {
                  fact   p ;   // f = 1

                  p.get();  //  n = 4 , f = 1

                  p.cal();  //  n = 4 , f = 24

                  p.out(); //   fact = 24
          }
```

```
// CONSTRUCTOR  OVERLOADING

//   PARAMETERIED  CONSTRUCTOR

              //  INITIALIZE AND PRINT COMPLEX NO.

#include<iostream>
using namespace std;

  class   complex
  {
            private : int  a , b;

            public  :
                 complex()
                 {
                       a = 0 ; b = 0;
                 }
                 complex( int  x )
                 {
                       a = x ; b = 0 ;
                 }
                 complex ( int  x  , int  y   )
                 {
                        a =  x ; b = y ;
                 }
```

```
               void   out()
               {
                     cout<< a << "+i" << b << endl;
              }
};
int    main()
{
      complex  p;   // IMPLICIT CALLING  CONSTRUCTOR
      complex  q (2);
      complex  t (3,2);

      p . out(); // 0 +i 0
      q . out(); // 2 +i 0
      t . out(); // 3 +i 2
}
```

or

```
int    main()
{
        complex  p = complex();    // EXPLICIT  CALLING  CONSTRUCTOR
        complex  q = complex(2);
        complex   t = complex( 3 ,2 );

        p. out(); // 0 +i 0
        q. out(); // 2 +i 0
        t. out(); // 3 +i 2
}
```

CONSTRUCTOR WITH DEFAULT ARGUMENT

INITIALIZE AND PRINT COMPLEX NO.

```
#include<iostream>
using namespace std;

  class   complex
  {
          private : int  a , b;
          public  :
  complex ( int  x = 0  , int  y = 0  ) // cons.  with default argument
  {
      a =  x ; b = y ;
  }
  void   out()
  {      cout<< a << "+i" << b << endl;
  }
};
int   main()
{
      complex  p;
      complex  q (2);
      complex  t (3,2);
      p . out(); // 0 +i 0
      q . out(); // 2 +i 0
}      t . out(); // 3 +i 2
```

```
}
//  this  POINTER

              //  ADDRESS  OF  THE  CURRENT OBJECT  IN  CLASS

#include<iostream>
using namespace std;

  class   test
  {
            private :  int  a;

            public  :
 test()
 {
             a = 5 ;          // implicit way   OR
      this -> a = 5;     // explicit way
 }

 void    out()
  {
         cout << "   " << a << endl;           //   5

         cout << "    " << this->a << endl;  //    5

         cout<<" ADDRESS OF THE OBJECT = " << this << endl; // 100
  }
```

```
};
int     main()
{
        test  p;
        p . out();
}
```

DESTRUCTOR

1. SPECIAL MEMBER FUNCTION

2. AUTOMATIC CALL WHEN OBJECT IS DESTROYED

3. SAME NAME AS CLASS NAME (~ :- tilde)

   ```
   e.g  ~test()
        {
        }
   ```

4. DECLARE IN PUBLIC SECTION

CALLING CONSTRUCTOR AND DESTRUCTOR

```
#include<iostream>
using namespace std;

  class  test
  {
         public :
         test()
         {
           cout<< " CONSTRUCTOR CALLED = " << this<<endl;
         }

        ~test()
         {
                cout<< " DESTRUCTOR  CALLED = " << this <<endl;
         }
  };
  int    main()
  {
          {
                test  p ;
          }
  }
```

ans :-

```
constructor  called   65524
```

```
    destructor    called   65524
```

SEQUENCE OF CONSTRUCTOR AND DESTRUCTOR

```
int   main()
{
      {
            test  p  ;
            test  q  ;
            test   r ;
      }
}
```

```
CONSTRUCTOR  CALLED      100 → p
CONSTRUCTOR  CALLED      200 → q
CONSTRUCTOR  CALLED      300 → r

DESTRUCTOR    CALLED     300 → r
DESTRUCTOR    CALLED     200 → q
DESTRUCTOR    CALLED     100 → p
```

CONSTRUCTOR	DESTRUCTOR
1. PASSING AGRUMENT ✓	1. NO ARGUMENT X
2. OVERLOADING ✓	2. OVERLOADING X

3. VIRTUAL X **3. VIRTUAL** ✓

```
                    //  COPY CONSTRUCTOR
               // INITIALIZE AND COPY COMPLEX NO.
#include<iostream>
using namespace std;

  class   complex
  {
         private : int  a , b;

         public  :
            complex()   //   DEFAULT CONST.
            {
                 a = 0 ; b = 0 ;
            }
            complex ( int  x , int  y ) // PARA.  CONST.
            {
                 a = x ; b = y ;
```

```
        }
        complex ( complex  &x ) // COPY  CONST.
        {
               a = x . a;
               b = x . b;
        }
       void  out()
       {
               cout << a << "+i" << b <<endl;
        }

   ~complex()  // DESTRUCTOR
     {
     }
};
      int    main()
      {
         complex  p (3,2);    // PARA.  CONSTRUCTOR

         complex  q (p) ;    // call COPY  CONSTRUCTOR

         p. out(); // 3 +i 2

        q. out(); // 3 +i 2
     }

/*
---------------------------------------------------------------------------------
```

int a = 5 ; // int a(5) ;

complex q (p); // complex q = p;

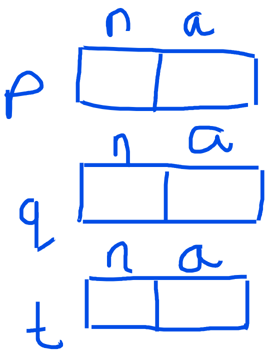
--

STATIC DATA MEMBER AND STATIC MEMBER FUNCTION

1. NORMAL DATA MEMBER

private : int n , a ;

test p , q , t; (OBJECT)

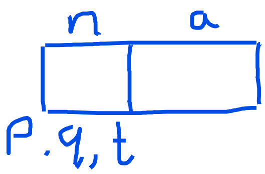


seperate memory

2. STATIC DATA MEMBER

private : static int n , a;

test p , q , t; (OBJECT)



common memory

```
#include<iostream>
using namespace std;

  class  test
  {
         private :     static  int  n , a;  // static data member

         public :
                      test()   // CONSTRUCTOR
                      {
                           n++;
                           a++;
                      }
static  void   out()   // STATIC  MEMBER  FUN.
{
       cout<< " TOTAL NO. OF OBJECT = "  << n << endl;
       cout<< " NO. OF ALIVE OBJECT  = "  << a << endl;
}
 ~ test()    // destructor
   {
        a--;
   }
};

int  test :: n = 0 ;  // DECLARE  STATIC  DATA  MEMBER
int  test :: a = 0 ;
int   main()
{
      test  p , q , t ;
      test :: out();    // n = 3 , a = 3
      {
            test  u , v;
            test :: out(); // n = 5 , a = 5
      }
      test :: out();  // n = 5 , a = 3
 }

---------------------------------------------------------------------
               // ARRAY  OF   OBJECT

              //  INPUT AND  PRINT N RECORDS

#include<iostream>
using namespace std;

  class   student
  {

      private:   char  name[10];
                 int    roll;
                 static  int  n;

      public :         student()
                       {
```

```
                    n = 100;
                }
                void  get()
                {
                        cout<<" ENTER NAME " << endl;
                        cin >>  name;
                        roll = n;
                        n++;
                }
                void  out()
                {
                        cout<<" NAME = " << name << endl;

                        cout<<" ROLL = " << roll << endl;

                }
};

int  student :: n  ;

int  main()
{
      student  p[10] ;  // object
      int  i , n ;

      cout<< " ENTER SIZE " << endl;
      cin >> n;
```

```
        for( i = 0 ; i < n ; i++ )
        {
              p[i] . get();  // input records
        }

        for( i = 0 ; i < n ; i++ )
        {
             p[i] . out();  //  print records
        }
  }
```

DMA (Dynamic Memory Allocation)
Memory using

1. Static or Compile time Allocation [Array]
2. Dynamic or Run. time Allocation
[Pointer]

DYNAMIC OR RUN - TIME ALLOCATION

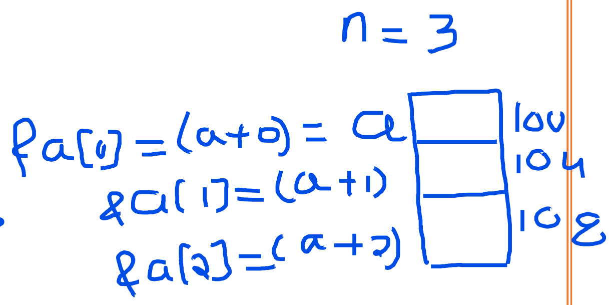
c	c++	
1. malloc()		
2. calloc()		
3. realloc()	new	// CREATE MEMORY
4. free()	delete	// DESTROY MEMORY
function	keyword	

DYNAMIC MEMORY ALLOCATION (DMA)

new :- **syntax**

new datatype

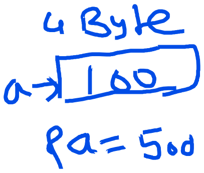
e.g. 1. FOR INTEGER DATA



```
int *a;

a = new int;
```

2. FOR ONE DIM. ARRAY // VECTOR



```
int *a;
a = new int [ n ];
```

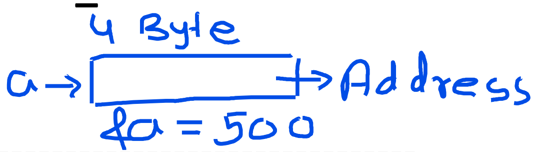
&a[i] = (a+i)

----

delete :- syntax

delete pointer_name

e.g. delete a; →



DYNAMIC INITIALIZATION THRUGHT CONSTRUCTOR

DYNAMICALLY INPUT AND PRINT N NOS

```
#include<iostream>
```

```
using namespace std;

  class vector // one dim.
  {
        private : int  *a , n;

        public  : vector()
                {
                }
                vector( int  x )  // DYNAMIC  CONSTRUCTOR
                {
                   n = x ;
                   a = new  int [n]; // DYNAMIC INITIALIZATION
                }
                void   get()
                {
                     cout<<" ENTER NO " << endl;
                         for( int i = 0 ; i < n; i++)
                         {
                                 cin >> a[i];
                         }
                   }
                   void   out()
                   {
                         cout<<" NO = " << endl;

                         for( int i= 0 ; i < n ; i++)
                         {
```

```
                                        cout << a[i] << endl;
                                }
                        }
                    ~vector()
                        {
                                delete  ~~a;~~ [] a;
                        }
};
int    main()
{
        vector  p(3);

        p . get();  // 10 , 20 , 30

        p . out(); //  10 , 20 , 30
}
```