Default Arguments

- Sometimes, fun declaration / definition can have some default values. This is called def. arg.

eg Def. arg in set fun.

```
Class student
{
    -
    -
    -
    Public:              // Def. arg. in set fun.
     void set (int x=0, int y=0,
                         int z=0 );
     void get ( );
       :
};

Void student :: set (int x, int y, int z)
{
        rn = x;      // Def arg are provided
        m1 = y;      // above, not here.
        m2 = z;
}
   -
   -
   -
   -
```

• Rule for using Def. arg :-

1) If you do not provide any values to fun, then default arg. will be taken.

2) If you p/v any values, then def. arg will be ignored

Eg
```
student obj1, obj2;
obj1.set(1,11,11); // obj is assigned 1,11,11
//Values r p/v, so def. arg will be ignored.

// obj1.set( ); // obj is assigned 0,0,0
//No values p/v, so def. arg will be taken;

Obj1.set(1); // obj is assigned 1,0,0.
// only 1 value is p/v, so other values will be taken by def. arg.
```

3) def. arg. must be b/w in fun declaration, not in fun def.

Static member fun

- MF can also be static.
- They can only access static var, they can't access non static var.
- A SMF can't have a this ptr.

Const member Fun

- MF wh. r declared as const are called Const. MF.
- Const MF can't change any values You can use these fun to print values, but you can't use these fun to change values.
- For eg set() fun should not be const as it changes/assign values.
- get() fun can be const, as it doesn't change any values.

```
Class stud
{
  =
  Public:
   void set ( - - - );
   void get ( ) const ;
  =
}
```

Dynamic initialization of obj

- It means you are initializing at the time of runtime.
- Static initialization means you r initializing at the compile time.

eg Static init.n...

```
Student obj1;
obj1.set(1,11,11);
// Values r known at compile time
```

eg Dy. init.n

```
int a, b, c;
Stud obj1;
cin >> a >> b >> c;
obj1.set(a, b, c);
// Values r not known at compile time
```

Values r known only at runtime.