Object-oriented Programming

Week 5 | Lecture 3

Member Initialization List

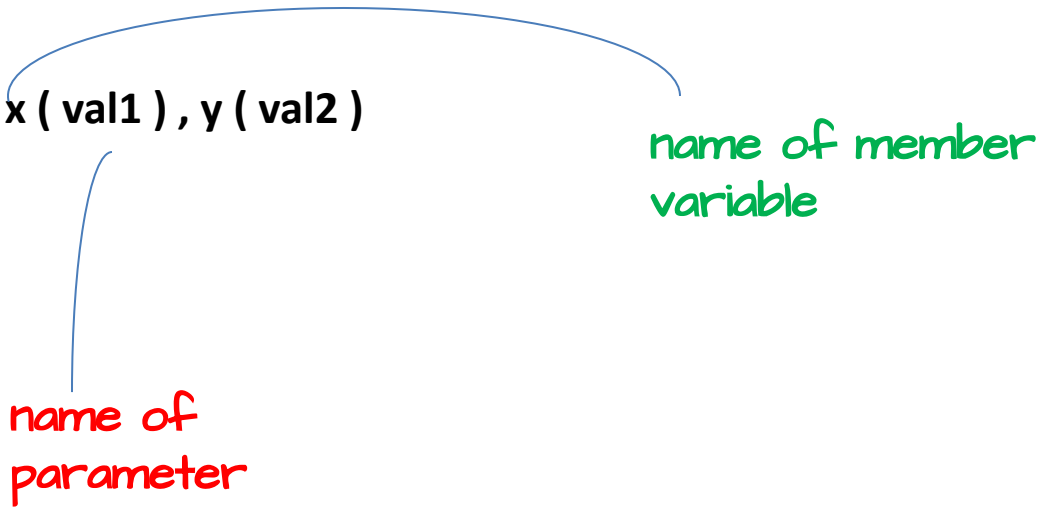
- Constant class members can only be initialized through constructor's member initialization list

Member Initialization List

```
class A
{
        const  int  x;
        const  int  y;

        public:
        A ( int val1 , int val2 ) : x ( val1 ) , y ( val2 )
        {
        }
};


int main()
{
        A  a ( 5 , 10 );
}
```



Storage-class Specifiers

1. Auto
2. Register
3. Extern
4. Static
5. Mutable

A simple function call

```
void staticDemo()
{
      int val = 0;
      ++val;
      cout << “val = "  << val << endl;
}

int main()
{
      staticDemo();   // prints val = 1
      staticDemo();   // prints  val = 1
      staticDemo();   // prints val  = 1
}
```

Static Local Variables

When static local variables in a global function are used, they "*remember*" their values from previous function calls

Static Local Variables

```
void staticDemo()
{
      static int val = 0;
      ++val;
      cout << “val = "  << val << endl;
}

int main()
{
      staticDemo();   // prints val = 1
      staticDemo();   // prints  val = 2
      staticDemo();   // prints val  = 3
}
```

Static Class Variables

- Static member variables are shared between all instances of a class

- Value of a static variable modified through one object will be reflected for all other objects

```
class xyz {
static var_type var_name;
};
var_type xyz::var_name;
```

Example

```
class A
{
        static int val;

        public:
        A(int x)
        {
                val = x;
        }

      void setVal(int y)
      {
        val = y;
      }

        void show( )
        {
                cout << “Static Variable” << x;
        }
}; int A:: val;
```

```
int main()
{
        A ob1(10);
        ob1.show( );   // val becomes 10

        A ob2(20);
        ob2.show( );   // becomes 20

        ob2.setVal(30);
        ob1.show( );   //becomes 30
}
```

Static Member Functions

- Just like static variables, static member functions are shared between all instances of a class
- A non-static (instance) function can call other static functions (and use static variables)
- A static function cannot directly use instance members of the class
- However, they can do it by either making a local object or taking object as an argument

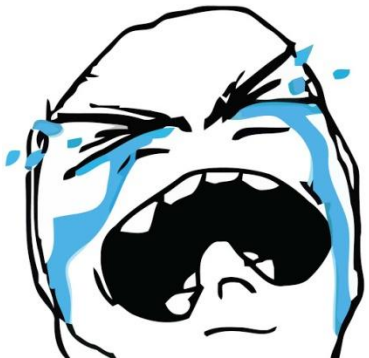
Example

```
class A
{
        public:
        A( )
        {  }

      static void f( )
      {
        f2( ); // will cause error
      }

        void f2( )
        {
        cout << “Instance function”;
        }
};
```

```
int main()
{
        A::f( );
// we use class scope to call static functions
}
```



Example

```
class A
{
        public:
        A( )
        {  }

      static void f( )
      {
        A temp(10);
        temp.f2( );       // OK
      }

        void f2( )
        {
        cout << “Instance function”;
        }
};
```

```
int main()
{
        A::f( );
// we use class scope to call static functions
}
```

Example

```
class A
{
        public:
        A( )
        {  }

      static void f(A myOb)
      {
        myOb.f2( );      // OK
      }

        void f2( )
        {
        cout << “Instance function”;
        }
};
```

```
int main()
{
        A ob(10);
        A::f(ob);
// passing object as an argument
}
```