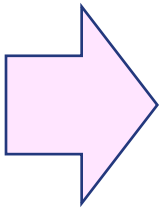
In-Class Member Initializer (Since C++11)

- Non-static Members of a class can be initialized directly where it is declared.
- Simplified form of initializing data members

```
class C {
    int x;
public:
    C() : x(7) {}
};
```



```
class C {
    int x=7; //class member initializer
public:
    C();
};
```

Initializing Class Member Variables

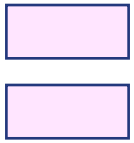
- In-class member initializer can consist of any valid initialization expression, whether that's
  - the traditional equal sign,
  - a pair of parentheses, or
  - the new brace-init:

```
class C {
   double d=0;
   string s("abc");
   char * p {nullptr};
   int y[5] {1,2,3,4};
public:
   C();
};
```

Initializing Class Member Variables

- The compiler conceptually transforms every class member initializer into a corresponding mem-init.

```
class C {
    double d=0;
    string s("abc");
    char * p {nullptr};
    int y[5] {1,2,3,4};
public:
    C();
};
```

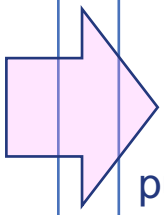


```
class C {
    double d;
    string s;
    char * p;
    int y[5];
public:
    C() : d(0.0), s("abc"), p(nullptr), y{1,2,3,4} { }
};
```

Good Design: 데이터 멤버만 초기화하는 경우 클래스 맴버 초기화를 사용하라! (C.45)

- Using in-class member initializers lets the compiler generate the function for you. The compiler-generated function can be more efficient
- A default constructor should do more than just initialize member variables with constants

```
// BAD: doesn't use member initializers
class X1 {
    string s;
    int i;
public:
    X1() :s{"default"}, i{1} { }
    // ...
};
```



```
class X2 {
    string s = "default";
    int i = 1;
public:
    // use compiler-generated default constructor
    // ...
};
```