DigiPen
INSTITUTE OF TECHNOLOGY

CS170#04.2

Member Initialization List

Vadim Surov

Outline

- How?
- Why?
- Order Of Initialization
- Initialization List And Default Member Initializer

How?

- C++ provides an alternative method of initializing members in a constructor using **member initialization list**

```
class Foo {
      int x, y, z;
  public:
      Foo(int X=0, int Y=0, int Z=0)
        : x(X), y(Y), z(Z)
       { }
}
```

How?

- Or

```
Foo(int x=0, int y=0, int z=0)
    : x(x), y(y), z(z)
{

}
```

Why?

- Suppose your class has a const or a reference data member:

```
class Foo {
  int x;
  const int y;
  int& z;
public:
  Foo(int x = 0, int y = 0, int z = 0);
}
```

Why?

- If constructor is like this:

```
Foo::Foo(int x, int y, int z) {
  this->x = x;
  this->y = y;
  this->z = z;
}
```

- g++ outputs these errors:

```
In constructor `Foo::Foo(int, int, int)':
error: uninitialized member `Foo::y' with `const' type `const
int'
error: uninitialized reference member `Foo::z'
error: assignment of read-only data-member `Foo::y'
```

Why?

- The problem is that both const variables and references must be initialized
  - The constructor as written performs assignment, not initialization
- Using an initialization list solves this problem:

```
Foo(int x, int y, int z)
    : x(x), y(y), z(z)
{

}
```

Remember

- So remember: if your class has const or reference data members
  1. you must initialize them with an initialization list. There is no other way.
  2. The compiler will not generate a default constructor and a default assignment operator for you.

Order Of Initialization

- The members of a class are always initialized in the order that they are declared in the class
- Reordering the initialization list does not matter (and might cause the compiler to give a warning)

Order Of Initialization

```
class D {
   int a;
   int b;
   public:
   D() : b(3), a(b) {
      cout << a << " " << b;
   }
} d;
```

Output: 0 3

Initialization List And Default Member Initializer

- If a non-static data member has an default member initializer and also appears in a member initializer list, then member initializer list is executed and the default member initializer is ignored:

```
class S {
   int n = 42; // C++11 default member
                          initializer
   public:
   S() : n(7) {} // will set n to 7, not 42
};
```