

## Delegating Ctors

Chapter 1 Safe Features



### See Also

- “Forwarding References” (§2.1, p. 377) provides perfect forwarding of arguments from one [ctor](#) to another.
- “Variadic Templates” (§2.1, p. 873) describes how to implement constructors that forward an arbitrary list of arguments to other constructors.

## Delegating Ctors

## Chapter 1 Safe Features

### Predelegation work is awkward and error prone

Delegating constructors provide a convenient way for a constructor to first call another constructor and then execute additional **statements** after that first constructor has completed. These additional **statements** are placed in the body of the **delegating constructor**. If we wish to implement a constructor that executes additional **statements** before delegating to another constructor, we have no convenient place for those **statements**. Our only option is to insert **expressions** in the **argument** list of the member initializer. Such awkward constructions are error-prone, particularly if correct functioning requires them to be evaluated in a particular order:

```
class PointsInsideItself {
    std::vector<int> d_contents;
    std::vector<int>::const_iterator d_contentsIter;

    PointsInsideItself(int index, std::vector<int> contents)
        : d_contents(std::move(contents)),
          d_contentsIter(contents.begin() + index)
    {}

public:
    PointsInsideItself(M&& source) // delegating constructor
        : PointsInsideItself(source.d_contentsIter - source.d_contents.begin()
                              std::move(source.d_contents))
    {
    }
};
```

In the example delegating constructor above, we are trying to calculate the index, `source.d_contentsIter - source.d_contents.begin()`, before entering the target constructor. However, if the second parameter of the target constructor happens to be initialized before the first parameter, then the initialization of the second **parameter** moves from `source`, which invalidates `source.d_contentsIter` and thus implies that the evaluation of `source.d_contentsIter - source.d_contents.begin()` has undefined behavior.

To make this example correct, we could change the target constructor’s **signature** so that the second **parameter** is an **rvalue reference**, thus ensuring that the initialization of the target constructor’s **parameters** has no **side effects**. Operations that have no **side effects** can be evaluated in any order without affecting the correctness of the program.

### See Also

- “Forwarding References” (§2.1, p. 377) provides **perfect forwarding** of arguments from one constructor to another.
- “Variadic Templates” (§2.1, p. 873) describes how to implement constructors that forward an arbitrary list of arguments to other constructors.