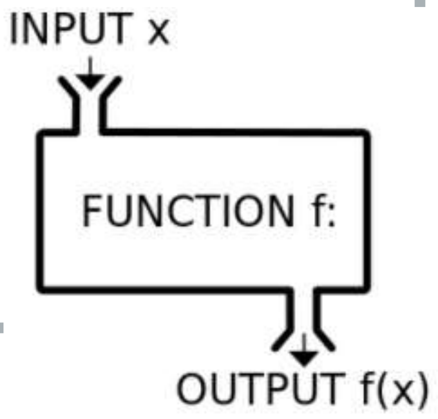
Lesson 7: Functors & Iterators

Callable objects

Predicates

Custom iterators



7.1 Function Objects

- Functor
  - Class that overloads operator()
  - Callable
  - Tracks state between calls

7.2 Predicates

- Predicate
  - Function that returns boolean
  - Class that overloads operator() that returns boolean

7.3 Predicates Should be Pure

- Pure function
  - A function that always returns the same value for a given input
- Non-pure predicates
  - Can cause algorithms to fail when copies of function objects are made
- Guidelines

7.4 Passing Member Function to Algorithm

- How to pass member function to algorithm?
  - One solution: Pass a functor
  - Limited because functor can only have one overload operator() for given parameters

7.5 mem_fun and mem_ref

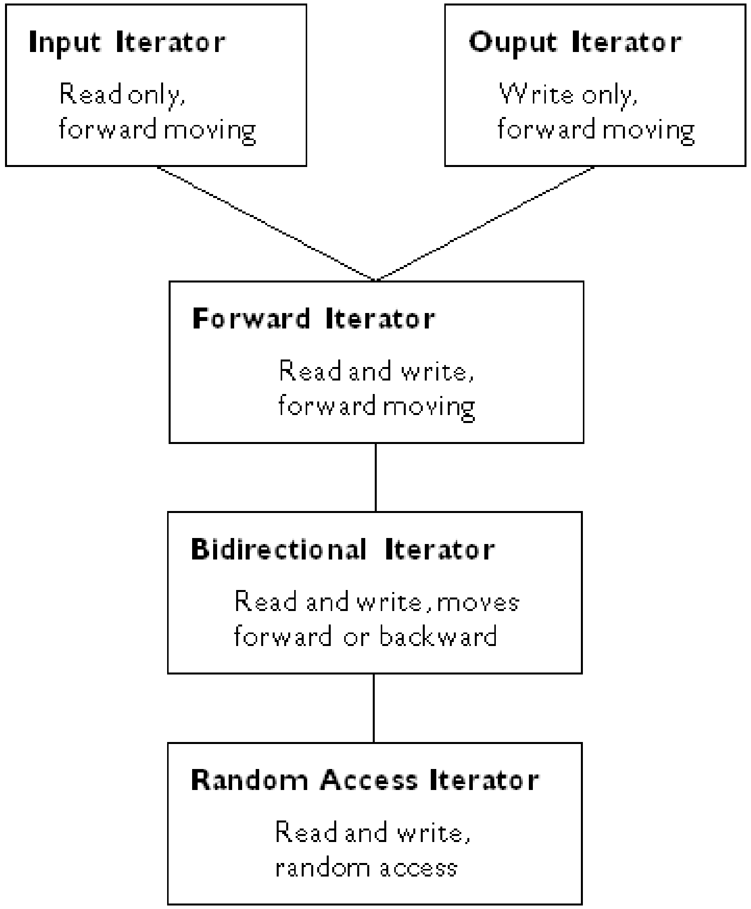
- Helper function templates that generate functors at compile time
- Allow more different operator() for each member function

7.6 Negating a Predicate

- not1 & not2
  - Helper function templates that reverse logic of predicates
- Benefit
  - Avoid writing two versions of predicates

7.7 Iterator Review

- Categories
  - Input
  - Output
  - Forward
  - Bidirectional
  - Random Access


Input Iterator
Read only, forward moving
Ouput Iterator
Write only, forward moving
Forward Iterator
Read and write, forward moving
Bidirectional Iterator
Read and write, moves forward or backward
Random Access Iterator
Read and write, random access

7.8 typename Keyword

- Qualified & unqualified names
- Dependent & non-dependent names
- You must explicitly specify when…
  - qualified, dependent name is a type
- Rules for using typename

7.9 iterator_traits

- Standard class template
  - Defines useful typedefs for working with iterators
  - Algorithms use these typedefs
- Overview of typedefs

7.10 distance() Algorithm

- Custom implementation of this standard algorithm
  - Optimized for faster execution with Random Access iterators
  - For other iterator types runs slower but still works

7.11 iterator Base Class

- Standard class template for defining new iterator types
- Sample custom iterator

7.12 Custom Class with Iterator

- Implement custom class that supports a custom iterator
  - Array<T>
  - Similar to how std::vector<T> is implemented