Day 36 (Mon 4/25)

- exercise 35 review
- day 36 recap questions
- exercise 36

Announcements/reminders

- Final project due Friday 4/29 by 11pm
  - But, you may submit by 11pm Monday 5/2 without penalty
  - Note that the final projects contributions survey must be submitted no later than 11pm on Monday 5/2

Final exam: Tuesday May 10th, 2pm, Remsen Hall 1

# Exercise 35

Checking whether file was opened successfully in readFile function:

```
std::ifstream fin(filename);
if (!fin.is_open()) {
  throw std::ios_base::failure("could not open file");
}
```

Exercise 35

Handling the exception in main:

```
std::vector<int> numbers;
try {
  numbers = readFile(argv[1]);
  // ...code to print out the values in the vector...
  return 0;
} catch (std::ios_base::failure &ex) {
  std::cerr << "Error: " << ex.what() << "\n";
  return 1;
}
```

Exercise 35

Seeing what happens when readFile reads non-integer data:

```
$ ./exceptionExercise nonIntData.txt
terminate called after throwing an instance of 'std::invalid_argument'
  what():  File contains non-integer data!
```

Handling std::invalid_argument in main:

```
  try {
    // code calling readFile
  } catch (std::ios_base::failure &ex) {
    std::cerr << "Error: " << ex.what() << "\n";
    return 1;
  } catch (std::invalid_argument &ex) {          // <-- add another catch block
    std::cerr << "Error: " << ex.what() << "\n";
  }
```

Day 36 recap questions

1. Why use iterators?
2. What are the bare minimum operators that need to be overloaded by an iterator?
3. When won’t a simple pointer correctly iterate through a collection?
4. Given a container class, how/where should its iterator class be specified?
5. In addition to defining the iterator class, what else should the container do to support iterators?
6. What might go wrong if we don’t also define a const_iterator for a container

# 1. Why use iterators?

They provide a uniform way to traverse the values in a collection, independent of what underlying data structure the collection uses to store the values.

Using iterators is especially important when implementing generic functions, where the function might be used with different kinds of collections.

Generic function to compute the sum of the values of any collection:

```
template<typename Coll>
typename Coll::value_type generic_sum(const Coll &coll) {
  using EltType = typename Coll::value_type;
  EltType sum = EltType();
  for (typename Coll::const_iterator i = coll.cbegin();
       i != coll.cend();
       i++) {
    sum += *i;
  }
  return sum;
}
```

Example of using this function with different kinds of collections and element types:

```
int main() {
  std::vector<int> icoll { 1, 2, 3, 4, 5 };
  std::list<double> dcoll { 6.0, 7.0, 8.0, 9.0, 10.0 };
  std::set<std::string> scoll { "foo", "bar", "baz" };
  std::cout << "sum of integers: " << generic_sum(icoll) << "\n";
  std::cout << "sum of doubles: " << generic_sum(dcoll) << "\n";
  std::cout << "\"sum\" of strings: " << generic_sum(scoll) << "\n";
  return 0;
}
```

Output:

```
sum of integers: 15
sum of doubles: 40
"sum" of strings: barbazfoo
```

2. What are the bare minimum operators that need to be overloaded by an iterator?

++it

Minimally: inequality (operator !=), dereference (operator*), pre-increment (operator++), copy constructor, assignment operator

Nice to have: equality (operator==), arrow (operator->), post-increment

( it++

3. When won’t a simple pointer correctly iterate through a collection?

Using a pointer as an iterator ONLY works if the elements are contiguous in memory, i.e., are stored in an array.

Any kind of data structure where the elements are NOT stored contiguously (e.g., in the nodes of a linked list) can not use plain pointers as its iterator type.

4. Given a container class, how/where should its iterator class be specified?

The iterator type(s) of a container class should be member types.

E.g.:

```
class MyCollection {
public:
  class iterator {
    // definitions of iterator operations
  };
}
```

In this case, MyCollection's iterator type is MyCollection::iterator

It would also be an excellent idea to define a const_iterator type.

5. In addition to defining the iterator class, what else should the container do to support iterators?

The collection should have begin() and end() functions which return iterators positioned at (respectively) the first element, and just past the last element:

```
class MyCollection {
public:
  // ...

  iterator begin();
  iterator end();

  // ...
};
```

To support a const_iterator type, cbegin() and cend() should be defined. Note that these should be const member functions.

6. What might go wrong if we don’t also define a const_iterator for a container?

A container class without a const_iterator type would not be very useful when passed to a function by const reference. I.e.:

```
void foo(const MyCollection &coll) {
  // This won't work because MyCollection::begin() and MyCollection::end()
  // are non-const member functions. (This is because the iterators they
  // create could be used to modify the elements of the collection.)

  for (MyCollection::iterator i = coll.begin(); i != coll.end(); i++) {
    // ...
  }
}
```

```
class Board {
public:
    class const_iterator {
    publ.c:

        Piece*   operator*() {
            return (*board)( pos);
        }

private:
    Board * board;
    Position pos;
```