Day 25 (Mon 03/28)

- exercise 24 review
- day 25 recap questions
- exercise 25

Announcements/reminders

- HW5: due Wednesday 4/6 by 11pm

Exercise 24

Part 3

```
// map to store the occurrence count for each word
map<string, int> counters;

// read each word in the input, and update the occurrence counts
string word;
while (cin >> word) {
  counters[word]++;
}
```

WHY DOES THIS WORK?

In a map, each key/value pair is represented by a std::pair object

For a std::map<std::string, int> collection, the pair type is std::pair<std::string, int>.

When using the subscript operator to refer to a key not currently in the map, a new pair object is created for that key.

The corresponding value is created using the value type's default constructor.

Primitive types (int, char, double, etc.) do have a default constructor! The behavior is to produce the value 0. So, when the code does

counters[word]++;

if word does not yet exist as a key, a pair is created with the int value set to 0, which is then incremented to 1.

Part 4

```
// create a map of occurrence counts to vectors of words
// with that occurrence count

map<int, vector<string> > words_by_freq;

for (map<string, int>::const_iterator i = counters.cbegin();
       i != counters.cend();
       i++) {
  words_by_freq[i->second].push_back(i->first);
}
```

Again, this works because when a new pair is added to the words_by_freq map, its vector is initialized using the default constructor, so the vector is initially empty.

Part 5

```
for (map<int, vector<string>>::const_iterator i = words_by_freq.cbegin();
        i != words_by_freq.cend();
        i++) {
  cout << "Frequency: " << i->first << endl;
  for (vector<string>::const_iterator j = i->second.cbegin();
          j != i->second.cend();
        j++) {
    cout << *j << endl;
  }
}
```

In the body of the outer loop, i->first is the occurrence count, and i->second is the vector of strings representing input words with that occurrence count.

Part 7

The std::sort function is in the <algorithm> header:

#include <algorithm>

Using std::sort to sort the vec2 vector:

std::sort(vec2.begin(), vec2.end());

Part 8

When I tried it:

```
$ ./sort
Enter the count: 10000000
Your sort time = 13301(ms)
STL's sort time = 2105(ms)
```

Merge sort is asymptotically optimal, but has relatively high per-element overhead due to the copying of data between the vector being sorted and the temporary vector (or array) used to hold the merged elements.

Day 25 recap questions

1. How do you read and write files in C++?
2. What is a stringstream in C++?
3. How do you extract the contents of a stringstream?
4. What does a constructor do?
5. What does a destructor do?

1. How do you read and write files in C++?

Read a file: std::ifstream

```
std::ifstream in("input.txt");
if (!in.open()) { /* error, couldn't open */ }

// ...use in to read input, works just like any istream (such as std::cin)...
```

Write a file: std::ofstream

```
std::ofstream out("output.txt");
if (!out.open()) { /* error, couldn't open */ }

// ...use in to write output, works just like any ostream (such as std::cout)...
```

2. What is a stringstream in C++?

Allows you to read formatted input from a string, or write formatted output to a string. Use with #include <sstream>

E.g.:

```
std::string s = "foo bar 123", tok1, tok2;
int n;

std::stringstream in(s);
in >> tok1 >> tok2 >> n;
assert(tok1 == "foo");
assert(tok2 == "bar");
assert(n == 123);

std::stringstream out;
out << "Hello, n=" << n;
std::string s2 = out.str();
assert(s2 == "Hello, n=123");
```

3. How do you extract the contents of a stringstream?

Use the .str() member function. (See previous slide.) It returns the string data in the stringstream as a std::string value.

4. What does a constructor do?

Initializes the fields of a newly-constructor object.

"Object" = "instance of a class or struct type"

Every object is initialized by a call to a constructor when its lifetime begins.

The call to the constructor happens before the object is used by the program.

5. What does a destructor do?

A destructor "cleans up" an object whose lifetime is ending.

The primary purpose of a destructor is de-allocating resources associated with the object.

Examples of resources requiring cleanup:

- dynamically-allocated memory, cleaned up by freeing
- open file(s), cleaned up by closing

The compiler will automatically invoke a destructor for any object declared as a local variable, when the function returns.

"Resource Allocation Is Initialization" (a.k.a. "RAII"): this is the principle that dynamic resources should managed by an object. As long as the object's destructor is called (which happens automatically for local variables), the programmer doesn't need to write special code to clean up resources.