

601.220 Intermediate Programming

Spring 2023, Day 26 (March 29th)

Today's agenda

- Review exercise 25
- Day 26 recap questions
- Exercise 26

Reminders/Announcements

→ HW5 is due **Friday, March 31st** by 11 pm

Exercise 25 review

abbreviate function loop:

```
bool last_was_vowel = false;
for (size_t i = 0; i < word.size(); i++) {
    bool cur_is_vowel = is_vowel(word[i]);
    if (cur_is_vowel) {
        if (!last_was_vowel) { result += " "; }
    } else {
        result += word[i];
    }
    last_was_vowel = cur_is_vowel;
}
```

Exercise 25 review

main function, opening input and output files:

```
ifstream in(argv[1]);  
if (!in.is_open()) {  
    cerr << "Couldn't open input file " << argv[1] << "\n";  
    return 1;  
}  
  
ofstream out(argv[2]);  
if (!out.is_open()) {  
    cerr << "Couldn't open output file " << argv[2] << "\n";  
    return 1;  
}
```

Exercise 25 review

main function main loop:

```
string line;
while (getline(in, line)) {
    stringstream line_in(line);
    string word;
    while (line_in >> word) {
        out << abbreviate(word) << " ";
    }
    out << "\n";
}
```

`std::getline` is useful for programs which process input one line at a time.

Exercise 25 review

classify program body of loop, variables:

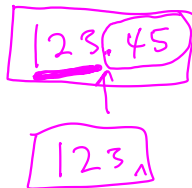
```
double fpval;  
int ival;  
string extra;  
bool is_ival = false, is_fpval = false;
```

Goal of loop body is tok classify one token.

Exercise 25 review

Check whether token is an integer value:

```
stringstream as_i(token);  
if (as_i >> ival) {  
    if (!(as_i >> extra)) {  
        sum_i += ival;  
        is_ival = true;  
    }  
}
```



Idea is that when extracting an integer, there should not be any input “left over”.

Exercise 25 review

Determine whether token is a floating point value:

```
if (!is_ival) {  
    stringstream as_fp(token);  
    if (as_fp >> fpval) {  
        sum_fp += fpval;  
        is_fpval = true;  
    }  
}
```

Exercise 25 review

Handle other tokens:

```
if (!is_ival && !is_fpval) {  
    ntok++;  
    ntok_c += token.size();  
}
```

Exercise 25 review

letter_freq program, initialize vector of Buckets:

```
vector<Bucket> buckets;  
for (int i = 0; i < 26; i++) {  
    Bucket b;  
    b.letter = 'a' + i;  
    b.count = 0;  
    buckets.push_back(b);  
}
```

Exercise 25 review

letter_freq program, open input file, read characters, classify them:

```
ifstream in(argv[1]);  
if (!in.is_open()) {  
    cerr << "Couldn't open input file " << argv[1] << "\n";  
    return 1;  
}  
  
char c;  
while (in.get(c)) {  
    c = tolower(c);  
    if (isalpha(c)) {  
        buckets[c - 'a'].count++;  
    }  
}
```

Exercise 25 review

letter_freq program, bucket comparison function:

```
// we want Buckets with a higher count to compare as "less"  
// (so that the overall ordering is from most frequent  
// to least frequent)
```

```
→ bool compare_buckets(const Bucket &left, const Bucket &right) {  
    if (left.count > right.count) { return true; }  
    if (left.count < right.count) { return false; }  
    return left.letter < right.letter;  
}
```

Sorting the vector of Buckets:

```
sort(buckets.begin(), buckets.end(), compare_buckets);  
                                     comparator
```

Exercise 25 review

letter_freq program, printing letter frequencies:

```
for (vector<Bucket>::const_iterator i = buckets.cbegin();  
     i != buckets.cend();  
     ++i) {  
    if (i->count > 0) {  
        cout << i->letter << ": " << i->count << "\n";  
    }  
}
```

Day 26 recap questions

- ❶ What is a C++ reference?
- ❷ When should you use C++ references?
- ❸ What is the difference between a pointer and a reference?
- ❹ How do you dynamically allocate memory in C++?
- ❺ How do you free memory in C++?

1. What is a C++ reference?

A reference is an alias (alternate name) for a variable or object.

In its lifetime, a reference can only refer to **one** variable or object.

Most common use: true reference parameters. E.g.:

```
void swap(int &a, int &b) {  
    // ... a and b are aliases for the argument variables ...  
}  
  
// ...  
  
int x = 2, y = 3;  
swap(x, y); // the swap function can modify x and y
```


Const references

Another important use of references: `const` reference parameters. Very useful for passing a large object or collection to a function, since it avoids copying. E.g.:

```
// all elements from argument vector must be copied  
// into a_vec, could be very slow  
void myfunc1(vector<int> a_vec) {  
    // ...  
}
```

```
// a_vec is an alias for the argument vector,  
// no copying required  
void myfunc2(const vector<int> &a_vec) {  
    // ...  
}
```

2. When should you use C++ references?

Allowing a function to have an alias to an argument variable, so it can modify the argument variable.

Accepting a const reference to an object where copying would be slow.

Occasionally: capture a reference to a collection element so you can modify it. E.g.:

```
vector<int> myvec;  
  
// ...  
int &element = myvec[i];  
  
element *= 2; // this modifies myvec[i]
```

3. What is the difference between a pointer and a reference?

Reference:

- Does not require explicit address-of (&) to create, or explicit dereference (*) to access the variable or object the reference refers to.
- Cannot be reassigned. (It can only refer to one variable or object during its lifetime.)

Pointer:

- Requires explicit address-of (&) to create, and explicit dereference (*) to access the variable the pointer points to.
- Can be reassigned. An assignment to a pointer variable changes what the pointer variable points to.

4. How do you dynamically allocate memory in C++?

new or new[]

Dynamically create one variable:

```
int *p = new int;  
*p = 42;
```

Dynamically create an array:

```
int *p = new int[10];  
for (int i = 0; i < 10; i++) {  
    p[i] = i;  
}
```

You should avoid using malloc in a C++ program.

5. How do you free memory in C++?

delete or delete[]

Example:

```
int *p = new int;  
*p = 42;  
delete p;
```

Example:

```
int *p = new int[10];  
for (int i = 0; i < 10; i++) { p[i] = i; }  
delete[] p;
```

Exercise 26

- Given probability distribution for rolling weighted N -sided die, compute *cumulative distribution function* representing probability of rolling “ i or less”
- “Naive” and “fast” functions to get an iterator positioned at last element in sorted vector less than or equal to v
 - Naive: use sequential search
 - Fast: use binary search
- Talk to us if you have questions!

Notes

Notes

Notes

Notes

Notes