Computer Science II — CSci 1200
Lecture 14
String and Character Operations

Announcement – Test 2

- Friday, March 23, West Hall Auditorium
- Instructions, coverage, and example problems are posted on-line.

Review of Lecture 13 — Maps, Part 2

- Maps containing more complicated values.
- Example: index mapping words to the text line numbers on which they appear.
- Maps whose keys are class objects.
- Example: maintaining student records.
- Summary discussion of when to use maps.

Today's Class — String and Character Operations

Koenig & Moo, Sections 5.6-5.9; Ford & Topp, Section 1.8

- Motivating problem: input text analysis
- String operations: input a line at a time; substring.
- Character operations: checking character types
- Solving the motivating problem

Motivation

- Problem: analyzing an input text file to find
  - Number of lines
  - Number of words
  - Number of letters
  - Number of occurrences of letters and words
- Challenges:
  - Distinguishing lines
  - Ignoring whitespace characters
  - Avoiding punctuation
  - Mixture of upper and lower case letters
- Assumptions:

- A word is a sequence of uninterrupted letters.
- Whitespace should not be included in the character count, but punctuation should.

String and Character Manipulation

- Reading a line at a time — `getline`. Here's the prototype:

```
istream& getline( istream&, string& );
```

  Returning the `istream` reference seems a bit strange, but it is common practice. It allows the state of the stream to be tested in a conditional. We've seen this already with loops to read integers and strings.

- The `string` class has a `substr` member function that extracts a substring starting at a given location. For example:

```
std::string s = "Hello world";
std::string t = s.substr(6,5);  // Starting at location 6, extract the next 5 char
cout << t << endl;          // Outputs:  world
```

- The header file `<cctype>` provides prototypes for character functions from the C library (hence the 'c' in front of 'ctype'). Here are some examples:
  - `isspace(c)`
  - `isalpha(c)`
  - `isdigit(c)`
  - `ispunct(c)`
  - `isupper(c)`
  - `tolower(c)`
- Each of these functions takes a character and returns true or false.
- Reminder: char is a special case of an integral type. To illustrate,

```
'c' - 'a' == 2                    // this is true
char( 'B' + 4 ) == 'F'            // this is true
cout << 'a' + 10 << endl;         // outputs the integer 107
cout << char('a' + 10) << endl;   // outputs the letter k
```

Exercise: Writing a Program Find Palindromes

Consider the following:

- A palindrome is a string that reads the same forward and backward.
- We will write a program to read lines of input and determine if the alphabetic letters on the line form a palindrome.
- This will illustrate several of the functions described above.
- Much of the program is provided already in a separate handout.

As an exercise, write the details of the `is_palindrome` function. Use the comments in the code as a suggested guide. The solution will be discussed in class and posted on the web.

Problem Solving Approach

We need to address the "Input Text Analysis Problem" posed at the beginning of the lecture. Here's an outline of how I approach solving a problem like this, which does not involve design of classes.

- Outline the flow of the main program, including the major steps of the program.
- Make note of the information that must be kept by the main program. This will dictate (most of) the variables.
- Make a list of the functions that the main program needs.
- Write these functions (and test them). If necessary, repeat the above process for these functions.
- Write the main program and test it.

One Outline

Here's one outline; others are possible:

- Main program:
  1. For each line,
     (a) Increment line counter
     (b) Count characters (*) and add to character count
     (c) Add to letter counters (*)
     (d) Break up into words of small letters only (*)
     (e) Save all words
  2. Sort words (including repetitions) and count occurrences (*)

  Each of the (*) corresponds to a function.

- Variables:
  - Counter: lines, words, letters
  - Vector of 26 individual letter counts
  - Map of strings to represent words

Functions

Here are the prototypes for the four functions:

```
unsigned int count_characters( const string& a_line );

void add_to_letter_counts( const string & a_line,
                           vector<int>& letter_counters );

vector<string> break_up_line( a_line );

void count_word_occurrences( vector<string> & words );
```

We will discuss each in class

Exercise

This exercise will occupy the remainder of the lecture. The first two functions are relatively straight forward. The last is more involved.

1. Write the function `count_characters`.
2. Write the function `add_to_letter_counts`.
3. Write the function `break_up_line`.