

CSCI-1200 Computer Science II — Spring 2006

Lecture 12 — String and Character Operations

Test 2 — General Information

- Test 2 will be held **Tuesday, March 7th, 2006 10-11:50am, West Hall Auditorium**. No make-ups will be given except for emergency situations, and even then a written excuse from the Dean of Students office will be required.
- Coverage: Lectures 1-12, Labs 1-7, HW 1-5.
- Closed-book and closed-notes *except for 1 sheet of 8.5x11 inch paper (front & back) that may be handwritten or printed*. Computers, cell-phones, palm pilots, calculators, PDAs, etc. are not permitted and must be turned off.
- All students must bring their Rensselaer photo ID card.
- Practice problems are available on the course website. Solutions will be posted this weekend.
- There will be an optional review session on **Sunday March 5, from 7-9pm in DCC 324**. Bring your questions!

Review of Lecture 11

- First problem solving lecture, focusing on small-scale algorithm design and implementation
- Three steps:
 1. Generating and Evaluating Idea
 2. Mapping the Ideas into Code
 3. Getting the Details Right
- Example problems: perfect numbers, closest in value, remove duplicates, box packing, merge sort, word search, max subsequence sum.

Today's Class — String and Character Operations

Koenig & Moo: Sections 5.6-5.9

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

12.1 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.

12.2 String and Character Manipulation

- Until now, we've been reading strings from the input separated by whitespace. Some of you have experimented with other operations for homework, but they weren't necessary to solve the problems.
- We can also read a whole line of input (including whitespace) with the function `getline`. This function reads characters until a newline character (or the end-of-file) is encountered. Here's the prototype:

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

Returning the `istream` reference may seem 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, for example:

```
std::string name;
while (std::cin >> name) {
    ...
}
```

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

```
std::string s = "My name is Sally Jones";
std::string t = s.substr(11,5); // Starting at location 11, extract the next 5 chars.
std::cout << t << std::endl; // Outputs: Sally
```

- 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.

- The type `char` is a special case of the type `integer`. As such, we can do simple math with values of type `char`. When we do this, the compiler automatically converts the `char` value to be of `integer` type. We can *cast* the value back to type `char` as illustrated below:

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

12.3 Example: Writing a Program Find Palindromes

- A palindrome is a string that reads the same forward and backward.
- We want to write a program to read lines of input and determine if the alphabetic letters on a line form a palindrome.
- To do this, we'll use many of the new functions we learned above.

12.4 Exercise: Finish the Palindrome Code

- 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.

```
#include <algorithm>
#include <cctype>
#include <iostream>
#include <string>

using namespace std;

bool is_palindrome(const string& line);

int main() {
    cout << "This program will read input, one line at a time, and\n"
         << "determine which input lines are palindromes. It will also\n"
         << "output a count of palindromes\n";

    unsigned int count=0;
    string line;
    while (getline(cin, line)) {
        if (is_palindrome(line)) {
            count++ ;
            cout << line << endl;
        }
    }
    cout << "There were " << count << " lines containing palindromes.\n";
}

bool is_palindrome(const string& line) {
    string temp;
    string::const_iterator i;

    // Pull out letters and place them in the temp string. Try to implement
    // this using string iterators and using the += operator on strings.

    // Determine if the letters are the same in the first half and the
    // second half. Return false as soon as a difference is found.

    return true;
}
```

12.5 Problem Solving Approach

Now let's address the text analysis posed at the beginning of the lecture. Here's an outline of how you might approach solving problems like this, which do not involve the design of classes:

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


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

```
}
```

12.8 Putting it All Together

```
int main(int argc, char* argv[]) {
    if (argc != 2) {
        cerr << "Usage: " << argv[0] << " text-file";
        return 1;
    }
    ifstream in_str(argv[1]);
    if (!in_str) {
        cerr << "Couldn't open " << argv[1] << " to read.\n";
        return 1;
    }

    unsigned int character_count = 0;
    unsigned int line_count = 0;
    vector<int> letter_counters(26, 0); // counts for the individual letters
    vector<string> all_words;

    // Handle one line at a time...
    string a_line;
    while (getline(in_str, a_line)) {
        line_count++;
        character_count += count_characters(a_line);
        add_to_letter_counts(a_line, letter_counters);
        vector<string> words_in_line = break_up_line(a_line);
        // Add all words to the back of the all_words vector.
        vector<string>::iterator p;
        for (p = words_in_line.begin(); p != words_in_line.end(); ++ p)
            all_words.push_back(*p);
    }

    // Output char, word and line counters
    cout << "\nHere are the statistics on the input text file:\n"
         << "  char count = " << character_count << "\n"
         << "  word count = " << all_words.size() << "\n"
         << "  line count = " << line_count << "\n";

    // Output the letter counts
    cout << "\nHere are the letter counts:\n";
    for (unsigned int i = 0; i < 26; ++ i) {
        cout << "  " << char('a' + i) << ": " << letter_counters[ i ] << "\n";
    }

    // Output the word occurrences
    count_word_occurrences(all_words);
}
```

12.9 Revisiting Recursion

Here's rewriting `remove_duplicates` with no iteration (no loops):

```
// a helper function for remove duplicates (that's also recursive!)
void remove_element(std::list<int> &lst, int element, std::list<int>::iterator q) {
    if (q == lst.end()) return;
    if (*q == element)
        q = lst.erase(q);
    else
        q++;
    remove_element(lst, element, q);
}

// the main recursion for remove duplicates
void remove_duplicates(std::list<int> &lst, std::list<int>::iterator p) {
    if (p == lst.end()) return;
    list<int>::iterator q = p;
    ++q;
    remove_element(lst, *p, q);
    ++p;
    remove_duplicates(lst, p);
}

// the driver function for the recursive version
void remove_duplicates(std::list<int> &lst) {
    remove_duplicates(lst, lst.begin());
}
```