CSE 232, Lab Exercise 4

Partner

Choose a partner in the lab to work with on this exercise. Two people should work at one computer. Occasionally switch who is typing.

More Command Line

I/O redirection

Please read the page on I/O redirection in the Lab04 directory labeled `IO-redirection.pdf`

★ `cd` to your x2go Desktop. Show your TA the sorted filenames there (using a combination of `ls`, `sort` and redirection).

Lab Assignment

John Napier was a Scottish mathematician who lived in the late 16th and early 17th centuries. He is known for a number of mathematical inventions, one of which is termed *location arithmetic*.

You job is to write functions that can convert back and forth between location and decimal representations, as well as some support functions for the process.

Overview

Location arithmetic is a way to represent numbers as binary values, using a notation that is not positional, but representational. You can see a detailed description in http://en.wikipedia.org/wiki/Location_arithmetic but here are the basics.

The representation

Napier used letters to represent powers of two. In using letters, the position of the letter is not important, allowing for multiple representations of the same number. For example, in location arithmetic:

a=1, b=2, c=4, d=8, e=16, f=32 … and thus: acf → 1 + 4 + 32 ← caf or 37

For convenience, the letters are typically sorted but only to make reading easier. Napier allowed for redundant occurrences of letters, though he acknowledged that there is a normal form that had no repeats. He described this as the *extended* (repeated) vs. *abbreviated* (no repeats). To create the abbreviated form, any pair of letters can be *reduced* to a single occurrence of the next "higher letter". For example:

(extended) (2b's to c) … (abbreviated)

abbccd → acccd → acdd → ace → 1 + 4 + 16 → 21

Addition

Addition is particularly easy. Take all the letters from the two values, put them into a single string, sort, reduce, convert!

mix reduce convert

abc + bcd → abbccd → acccd → acdd → ace → 1 + 4 + 16 → 21

Your Task

Write the following four functions and main program to do location arithmetic:

1. `long loc_to_dec(string loc)` : convert location arithmetic string to an integer
2. `string abreviate(string loc)` : take a location string and reduce it to its abbreviated form. We want you to experiment with string manipulation so **you cannot convert it to integer** first. You must do the abbreviation directly.

★ Show your TA a working abbreviate function.

3. `string dec_to_loc(long dec)` : convert an integer to an *abbreviated* location string
4. `long add_loc (string loc1, string loc2)` : take two location strings, add them, provided the integer result. For this function, *do it the way described above!!*, that is:
    a. mix the strings
    b. sort them
    c. reduce the string (using your `abbreviate` function)
    d. convert the result (using your `loc_to_dec` function)
5. Write a main function that shows off your work:
    a. prompt for two elements: a location string and a long
    b. prints on a single line 4, space separated elements
        i. loc__to_dec on the input location string
        ii. abbreviate on the input location string
        iii. dec_to_long on the input long
        iv. add_loc using the input location string twice (adding it to itself).

★ Show your TA your working program.

Testing

You can check yourself on Mimir, Lab04:location strings. There is an empty file lab04/lab04.cpp for you to begin with.

Sorting

We haven't done sorting yet, but C++ makes it pretty easy. There's an example in the directory, but here is the gist. Sorting sorts the individual elements of a collection in sorted order from two points in the collection. Elements in strings are chars, so sorting from the beginning to the end of the string, character by character, is shown below. Note that *sorting changes the string in place!*

```
#include<algorithm>
using std::sort;
…
string my_str = "adbche";
//sort string, char by char, in place, from begin() to end(). Note the parentheses!
sort(my_str.begin(), my_str.end());
cout << my_str; // prints abcdeh
```

Hints

1. You could define a constant for each letter of the alphabet, but what a pain. Better to note the following. The smallest character of a location string is `'a'`, which represents 2^0 power (i.e. 1). The difference between any letter and `'a'` is the power of 2 the letter represents. For example:
   a. `'a'-'a'` = 0 → which in location arithmetic is 2^0 →1
   b. `'c'-'a'` = 2 → which in location arithmetic is 2^2 → 4
   c. `'h'-'a'` = 7 → which in location arithmetic is 2^7 → 128
2. The function `dec_to_loc` is really nothing more than creating a "long string" and using `abbreviate` to clean it up. The `abbreviate` function is the key.
3. The function `add_loc` is nothing more than a concatenation of the two strings, a call to `abbreviate`, followed by a call to `loc_to_dec`.
4. Your functions will make use of:
   a. `substr` method
      i. substring takes two parameters: a position and a length. Length defaults to the end of the string (or if the value is beyond the length of the string, it defaults to the end.)
   b. `static_cast<char>`
      i. if you do addition/subtraction on a character, you need to cast it to a character for the "printable" version.
   c. `push_back` method.
      i. you can push a character onto the end of the string using this method.
   d. indexing via the `[]` operator