CS1321 Homework 4*

Jeffrey D. Oldham

2000 Feb 15

Due Tuesday, 2000 Feb 29, at the beginning of class.

Contents

1 Revisions

2000Feb28: Revised lstring.h, naming the `lstring` object's linked list `seq` because the input `operator>>` assumed its name was `seq`.

If you have given the linked list a different name, e.g., `foo`, just change the appropriate line of `operator>>`. For example, change

```
s.seq = temp;
```

to

```
s.foo = temp;
```

2000Feb16: Revised seq.h to make `length()` a `const` member function. Be sure to save this revised copy in your local directory.

2 Reading

Read chapters 2 and 3 of the textbook.

function prototype	example use	explanation
`lstring()`	`lstring v;`	creates a string with no characters
`lstring(const char c)`	`lstring w('a');`	creates a string having exactly one character
`lstring(const char [])`	`lstring x("cheerio");`	creates a string having the characters specified in the given C-style string
`bool empty(void) const`	`bool b = v.empty();`	returns true if and only if the string has no characters
`length_pos`		the `lstring` type for measuring lengths and positions of characters within a string
`length_pos length(void)`	`length_pos l = v.length();`	returns the number of characters in the string
`void append(const lstring & a)`	`x.append(v);`	modifies the string by adding the characters in `v` to `x`'s end
`void insertBefore(const lstring & s, const length_pos p)`	`v.insertBefore(v,2);`	modifies the string by inserting the string `s` before the character at position `p`. Unspecified behavior if `p` is greater than the string's length.
`bool find(const lstring & s) const`	`bool b = v.find("hello");`	returns a boolean indicating whether the string `s` is a substring of the string object. A substring is any consecutive sequence of characters within the string. The shortest substring is the empty string and should always be found in any string. The longest substring contained within a string is the string itself.
`char getElementAt(const length_pos p) const`	`char c = v.getElementAt(0);`	returns the character at the specified position. Unspecified behavior if `p` is greater than or equal to the string's length.
`void replaceElementAt(const length_pos p, const char c)`	`v.replaceElementAt(0,'d');`	changes the character at position `p` to the specified character. Unspecified behavior if `p` is greater than or equal to the string's length.
`operator +=(const lstring & s)`	`v += x;`	append the argument to the string

Table 1: `lstring` Member Functions and Types

3 Problem Statement

You and possibly a CS1321 classmate are to implement a string class using linked lists.

The Standard Template Library has a wonderful string class, which greatly eases use of strings in C++. In this homework, we implement a class called `lstring` that provides part of the functionality of the string class.

Our strategy for implementing `lstring` is to store each character of the string[1] as an element of a `Seq` linked list. For example, the string "cheerio" is stored as a list of length seven with the second list element being 'h'. While there are many ways to implement classes, we recommend (and, in fact, require) implementing `lstring` using the C++ `class` notation presented in the shopping cart example.

`lstring` objects should support the operations listed in Table 1. Also, the friend members in Table 2 should be implemented. For example uses, see the demo program.

3.1 Lengths and Positions

The length of a string is its number of characters. In particular, an empty string has length 0.

A character's position within a string is the number of characters before it in the string. The very first character has no characters before it so it is in position 0. When inserting characters into a string, the specified position is just before the character of the same number. For example, inserting at position 0 specifies inserting before character at

[1] Unless otherwise specified, the word "string" will refer to an `lstring` object.

function prototype	example use	explanation
`lstring operator+(const lstring & s1, const lstring & s2)`	`lstring v = x+y;`	form a new string by appending the latter string to the former
`bool operator==(const lstring & s1, const lstring & s2)`	`if (v == x)`	return a boolean indicating the strings have exactly the same characters
`bool operator!=(const lstring & s1, const lstring & s2)`	`if (v != x)`	return a boolean indicating the strings differ in at least one character. For example, one may be longer than the other.
`bool operator<(const lstring & s1, const lstring & s2)`	`if (v < x)`	return a boolean indicating whether the first parameter is less than the second parameter with respect to alphabetic ordering.
`bool operator>(const lstring & s1, const lstring & s2)`	`bool b = v > x;`	return a boolean indicating whether the first parameter is greater than the second parameter with respect to alphabetic ordering.
`bool operator<=(const lstring & s1, const lstring & s2)`	`if (v <= x)`	return a boolean indicating whether the first parameter is less than or equal to the second parameter with respect to alphabetic ordering.
`bool operator>=(const lstring & s1, const lstring & s2)`	`if (v >= x)`	return a boolean indicating whether the first parameter is greater than or equal to the second parameter with respect to alphabetic ordering.
`istream& operator>>(istream & istr, lstring & s)`	`cin >> x;`	read a string from an istream
`ostream& operator<<(ostream & ostr, lstring & s)`	`cout << x;`	print a string to an ostream

Table 2: `lstring` Friend Functions

position 0, i.e., the very first character. It is legal to insert at any position up to and including the string's length, i.e., at the end of the string.

3.2 Alphabetic Ordering

`lstring`s are ordered according to alphabetic ordering where spaces and capitalization are important. For example, "hello world" is less than "helloworld" because the space character is less than 'w'. "Hello" is less than "hello" because 'H' is less than 'h'. If two strings have exactly the same characters but one is shorter than the other, the shorter string is less. For example, "hell" is less than "hello". When comparing individual `char`s, use the default ordering, which is *usually*, but not always, ASCII ordering. Thus, one can just use `<` to determine which of two characters is less.

4 Some More C++ Class Notation

C++ is a very verbose language so we rarely have time to introduce all the notation during lecture. Read the textbook chapter 2 for more explanations.

4.1 `const`

The `const` keyword is an indication that a variable's value cannot change. It is not strictly necessary, but effective use of this keyword can yield a measurable reduction in running time. Function parameters are frequently declared `const` *`type`* `&` *`variable`*. `const` means the parameter's value may not be changed. Passing by reference (`&`) is a hack that permits writing faster code since passing by reference is frequently faster than copying an entire function argument. See also the textbook pp. 66–67.

Another use of `const` is in a declaration of a member function. For example, `bool empty(void) const;` indicates that the `empty` member function will not change any values in the object for which it is called. Thus, this member function can be used for a `const` object. When defining a `const` member function, write `const` between the function prototype and the opening bracket {. See also the textbook p. 33.

4.2 `typedef` Within a Class

As we saw in Homeworks 2 and 3, `typedef`s permit creating synonyms for types. This is also legal inside a class definition. For example, we may write

```
typedef unsigned int length_pos
```

inside `lstring`'s definition. Inside any code implementing the class, we can freely refer to `length_pos`. If the `typedef` is public, users of the code can refer to it as `lstring::length_pos`.

5 What Files Do I Need?

There are three essential files and one recommended file.

- lstring.h contains the skeleton of the `class lstring` definition. Write your function and friend declarations here.
- seq.h declares the `Seq` linked list class. This is the same code we used for Homework 2.
- seq.cc implements the `Seq` linked list class. This is the same code we used for Homework 2.
- demo-lstring.cc uses the `lstring` class. We recommend using this code to check your implementation.

There are two styles for implementing C++ classes. The textbook declares member functions in the `class` declaration and implements them separately, sometimes in a separate file. The other style is presented in the shopping cart example. Use whichever style makes you more comfortable.

To use the code, write a .cc file containing a main function definition. Be sure to `#include "lstring.h"` and put the file in the same directory as lstring.h. Then compile the .cc file. For example,

```
g++ -Wall -pedantic demo-lstring.cc -o demo-lstring
```

6 Rules

6.1 Programming Rules

As for previous homeworks, working with other people during your planning phase is encouraged. For this homework, you are permitted to write the code, i.e., program, with one other person in CS1321. To learn the material, both of you should be actively involved in the programming. Failure to do so will almost certainly hurt your comprehension of the material in the rest of the course.

6.2 Submission Rules

Each one- or two-person team of programmers should submit only your completed lstring.h. You need not send any other files. Please send only text documents, do *not* send Microsoft Word documents, PDF documents, HTML documents, etc. **Please include both your names and email addresses at the top of your program.**

We will test your code using our own main() function. Please be sure it compiles without warning when using `g++ -Wall -pedantic`.

See the submission details for information how to email the programs. If a team of two are in different sections, submit exactly once to one of the two permissible email addresses.