$Id: asg2j-jxref-lists.mm,v 1.8 2013-10-15 18:16:58-07 - - $
PWD: /afs/cats.ucsc.edu/courses/cmps012b-wm/Assignments/asg2j-jxref-lists
URL: http://www2.ucsc.edu/courses/cmps012b-wm/:/Assignments/asg2j-jxref-lists/

1. Overview

In this assignment you will implement a singly-linked linear list of words to implement a cross referencing utility. The words will be keys, and each will contain a queue of line numbers. The queue will also be implemented as a linear linked list. Your queue operations must run in $O(1)$ time, but your map (dictionary) operations will run in $O(n)$ time instead of the usually-expected $O(\log_2 n)$ time. Thus, if your entire program scans n words in a file, it will run at speed $O(n^2)$ instead of $O(n \log_2 n)$ time.

2. Program specification

The program is specified in the form of a Unix **man**(1) page.

NAME

jxref — cross referencing and word count utility

SYNOPSIS

jxref [*filename* . . .]

DESCRIPTION

Each file is read in sequence and words are extracted from the file. At the end of each file, a table is printed, sorted in lexicographic order with each word followed by a count of the number of times it occurs and a list of the numbers of the lines where it occurs. Cross reference output is written to **stdout** (**System.out**) and error messages are written to **stderr** (**System.err**).

OPTIONS

None.

OPERANDS

Operands consist of the names of files to be read in sequence. If no filenames are specified, **stdin** is read. If filenames are specified, each file is read in turn. If a filename is specified as a single minus sign (**-**) **stdin** is read at that point. In order to read a file whose name really is a minus sign, it should be specified on the command line as "**./-**".

EXIT STATUS

0 All input files were read successfully.

1 Error(s) occurred and message(s) was printed to **stderr**.

3. Implementation sequence

Follow an implementation sequence by developing your code a little at a time. The **code/** subdirectory contains starter code.

(a) Study the behavior of the program **pxref.perl**, which is a reference implementation of your program written in the Perl programming language. Delete references to that file from your **Makefile** and do not submit the Perl program. You need not understand Perl, but note how short the program is.

(b) A partial main program in Java has been provided for you in `jxref.java`, which just reads in words from files and prints out those words. You need to add in the data structures. In your program you are prohibited from using anything from the package `java.util` except for those packages explicitly listed in the syllabus.

(c) Implement your program in phases, not all at once :

(i) Study the code provided, and add code to it to insert each word into the `listmap`, ensuring that duplicates are not entered. Then add code to print the output in the required manner. Ignore line numbers.

(ii) Add code to create an `intqueue` whenever a new word is found, and increment the count, but do not record line numbers. Adjust the printing to print counts as well as words.

(iii) Complete the program by appending line numbers to each queue.

(d) The main function iterates over each filename given in the argument vector and uses `stdin` when required. For scanning the file, it creates a pattern which describes a word and then matches each word in sequence. It also keeps track of line numbers. These parts are as follows :

(i) `"\\w+([-'.:/]\\w+)*"` is a pattern which matches any word character (`[a-zA-Z_0-9]`), allowing any one of the characters in the set `[-'.:/]` in the middle. Just use it. Regular expressions will be explained in detail in CMPS-104A.

(ii) `wordpat.matcher (line)` creates a pattern matcher for the line.

(iii) `match.find()` checks to see if there is an unscanned word left in the line.

(iv) `match.group()` returns it if there is.

(e) The function `main` is finished and iterates over each of the filenames given on the command line. The function `xref_file` has stub code which just prints out the words as they are found. It also prints out the line numbers. This function must be changed to conform to the program specifications. Leave in the debugging code until your program is finished. Then delete it.

(f) The function `listmap.insert` currently does nothing. Put code in this function to insert a new key in ascending lexicographic order as sorted by `compareTo`. Ignore the class `intqueue` for now. The insertion proceeds in two phases :

(i) Searches down the list using a previous and current pointer, as was done in class to locate the insertion point. Quit the loop either at the end of the list or when the insertion point is found.

(ii) If the word (key) is not already in the list, allocate a new node and insert it in the correct position. If it is in the list, do nothing.

(g) The `Iterator` over the `listmap` is already written for you. If your insertion code work, the foreach loop at the end of `xref_file` will print out the words in lexicographic order. If not, put debug statements in `listmap.insert` to find out why.

(h) The class `intqueue` is partially written for you. It has its `Iterator`, but instead of actually inserting a line number into the queue, it merely increments a count. Modify `listmap.insert` to call `queue.insert` to increment the word count.

(i) This will allow you to modify the printing loop so that it prints a list of words followed by their count. Verify that your program now works like `pxref.perl`, except that only words and counts are printed.

(j) Now work on `intqueue` to finish the `insert` operation. The implementation is a singly linked list of integers with a pointer to the front of the list and the rear of the list.. Both operations must run in $O(1)$ time. Every insertion should increment the count and every deletion should decrement the count.

(k) Now go back to `xref_file` and cause the words, counts, and line numbers all to be printed, as does the Perl program. Use the `Iterator`-style `for`-loop.

(l) If that works, delete the debug code and check your code agains `pxref.perl`. Does your program produce identical output for valid filenames? Almost identical error messages for invalid filenames? Correct use of `stdout` and `stderr`? Correct exit status codes?

(i) Error messages from `pxref.perl` are printed in correct Unix format, e.g.:
```
pxref.perl: foobar: No such file or directory
```

(ii) The Java version prints out the name of the jar file, as obtained from class `auxlib`, and also just uses the format returned by `getMessage()`, e.g.:
```
jxref: foobar (No such file or directory)
```

4. Interfaces

This program uses three interfaces in an attempt to keep the various classes from needing to know anything about their internal structure:

(a) Interface `Iterable<T>` allows collections to be accessed by Java's special iterator syntax.

(i) `Iterator<T> iterator()` — Returns an iterator over a set of elements of type T.

(b) Interface `Iterator<E>` allows an iterator to be used to access elements of the collection in a sequential manner.

(i) `boolean haxNext()` — Returns true if the iteration has more elements.

(ii) `E next()` — Returns the next element in the iteration.

(c) Interface `Map.Entry<K,V>` allows access to elements of a map without knowing the exact structure of the node containing the elements.

(i) `K getKey()` — Returns the key corresponding to this entry.

(ii) `V getValue()` — Returns the value corresponding to this entry.

A `NoSuchElementException` is thrown for those operations that we don't want to bother to implement.

5. Makefile

Graders will compile your code using `make`, so you need a working `Makefile`.

(a) The `Makefile` should have the following targets :

- `all` : the first target, which is also the default. Builds the clases from the sources and puts them all, including inner classes in the jar.
- `ci` : which checks in all source files into the RCS subdirectory.
- `clean` : which deletes all generated files except for the jar `jxref`.
- `spotless` : which depends on clean and also deletes the jar.
- `submit` : which submits all required code. Do not submit the Perl program. Delete it from the list of sources before submitting anything. Also delete the `lis` target.

(b) Make sure your `Makefile` can compile all of the Java code and put all classes into a jar. Submit all Java source files, the `Makefile` and the `README` and verify the submit as you were told to do in Lab 1. The graders will be using `unix.ic.ucsc.edu` exclusively, when doing the grading.

6. What to submit

Submit all necessary Java source files and the `Makefile`. ***Verify your submit.*** If you submit the wrong version or there are missing files, you will lose many points. Read the pair programming description. You may choose a partner or work alone. If you work with a partner, submit a `README` and a `PARTNER` file as specified in that document.