```
$Id: asg3j-jxref-bstree.mm,v 1.8 2014-01-16 17:48:14-08 - - $
PWD: /afs/cats.ucsc.edu/courses/cmps012b-wm/Assignments/asg3j-jxref-bstree
URL: http://www2.ucsc.edu/courses/cmps012b-wm/:/Assignments/asg3j-jxref-bstree/
```

1. Overview

In this assignment, you will implement a cross reference utility that will scan files for words and for each word found, print a list of those words in lexicographic order along with a list of line numbers indicating where the words were found.

This program will make use of a binary search tree data structure, which means that each search and insertion of a word will run in $O(\log_2 n)$ time. No balancing will be performed, as that is beyond the scope of this course. As a value in each tree node, there will be a queue of line numbers, with the linked implementation of the queue where each operation runs in $O(1)$ time.

2. Program Specification

Once again we present the specification in the form of a Unix `man`(1) page.

NAME

jxref — cross reference utility

SYNOPSIS

`jxref` [**`-df`**] [*filename* . . .]

DESCRIPTION

Each file is read in sequence and a printout of the words found in the file is generated at the end of each file, one word per line, followed by a list of line numbers where the word occurred. All normal output is sent to **`stdout`**. Error messages are printed to **`stderr`**.

OPTIONS

All options, if any, must appear as the first word following the name of the command, unlike the standard **`getopt`**(3c) used for C programs.

`-d` Instead of producing normal output, the tree is dumped in debug format, showing each key along with its level within the tree.

`-f` Upper case letters are folded into lower case before insertion into the binary search tree.

OPERANDS

Each operand is a filename, which is processed in sequence, each file causing to be created a new tree. If any filename is specified as a minus sign (**`-`**), **`stdin`** is read at that point. If no filenames are specified, **`stdin`** is read. As an output filename, the minus sign is used as the name of **`stdin`**.

EXIT STATUS

0 No errors occurred.

1 Errors occurred, either in scanning options or opening files.

3. Implementation Sequence

With every non-trivial program (and this is one of them), you should develop the program a small piece at a time, adding a small cluster of working code to a working program. You thus have a working program at all times, albeit only a partially complete one. Here is one approach to completion of your program.

(1) Print out the Postscript or PDF version of this file and also print out all of the source code provided. You have been given some starter code.

(2) Study `pxref.perl`, which is an implementation of your project written in the Perl language. Your program should exactly duplicate its functionality, except for some slight differences in the error messages. Note also that the Perl program does not handle options.

(3) Study how it handls input file names when they are given, how it handles the input file name given as a minus sign (-), and what happens where there are no file names given. What happens when filenames are incorrect? What output does it produce to `stdout` and to `stderr`? What exit status codes does it return?

(4) The file `auxlib.java` provides some useful functions for printing error messages, and handling the exit status.

(5) As provided, `jxref.java` iterates over the files given on the command line, and for each file, extracts words from each line.

(6) The function `String.split` is given the regular expression string `"\\W+"`, which represents the regex `\W+`, which matches one or more characters outside the set `[a-zA-Z_0-9]`. I.e., it eliminates all characters outside that set and returns an array of words.

(7) The function `String.matches` is given the argument string `"^\\d*$"`, which matches the beginning of a word, zero or more digits, up to the end of a word. I.e., If the word is empty or consists only of digits, it is skipped.

(8) Otherwise it is printed. This print statement is a debug statement which you must remove before submitting your program.

(9) Write an options analysis function similar to `getopt`(3c), except that yours is not as complicated. If `args[0]` starts with the character minus (-), then it is an option string. Scan it for the flags `-d` and `-f` and print an error message for anything else. Note that the flags may appear in any order, but not separately. Thus `-d`, `-f`, `-df`, or `-fd` are all acceptable, but not `-d -f`.

(10) Pass a lower case flag parameter into `xref_file` and convert words to lower case (`String.toLowerCase`) if it is true.

(11) Begin debugging your program by implementing `debug_dump_rec`, which performs an inorder traversal of the tree, printing all key and value pairs, indented to show the level of each node.

(12) For the debug dump, the tree should be printed in the following format:

```
    2 aaaaa queue@9cab16
  1 bar queue@1a46e30
    2 eeeee queue@3e25a5
0 foo queue@19821f
    2 hello queue@addbf1
  1 qux queue@42e816
    2 world queue@9304b1
```

(13) First print an integer which indicates the depth of a node in the tree, with the root being at depth 0, then print the key followed by the value. Note that in this case we are depending on the `toString` function of each, and since class `queue` does not have one, its identity is printed as the default. Note that each line is also indented with two spaces per level down in the tree. The root is not indented, its children are indented 2 spaces, its grandchildren 4 spaces, etc.

(14) Implement the `treemap.put` function. Do a binary search on the tree for a node whose key (`compareTo` returns 0) is equal to the node. If found, replace the old value by the new one, and return the old value. If not found, create a new node as a leaf node, and insert the key and value pair, and return `null`.

(15) ***Do not do any balancing.*** Balancing a tree is beyond the scope of this course.

(16) Implement `treemap.get` using a binary search. Return the value if found and `null` if not found.

(17) As you work through this project, you must eventually replace all occurrences of throwing an `UnsupportedOperationException` with working code.

(18) Go back to `jxref.java` and implement the `-d` option. If `-d` is specified, continue to dump in the debug format. For normal output, call `treemap.do_visit` to print out the tree.

(19) Complete `do_visit_rec` to apply the `visit_fn` to each node in the tree using an inorder traversal. This function in `treemap` has no print statements.

(20) Finish the implementation of `queue` by replacing the `UnsupportedOperationException` throws with working code for the linked list implementation of the queue.

(21) For each word found in `xref_file`, get the queue from the tree using the word as a key. If found, insert the new integer line number into that queue. If not found, create a new queue, insert the line number into it, and then put the word and the queue into the tree.

(22) Finish `printer` in the main class so that it produces the same output as the perl program. To do this, the tree will apply the visitor to each queue, using an iterator.

(23) Test your program thoroughly. Delete the reference to `pxref.perl` from the `Makefile`, since `pxref.perl` should not be submitted. Delete the `lis` target as well.

(24) Make sure that all inner classes and only those are listed in the `Makefile`. Make sure that all inner class names have a `\$$` in place of the dollar sign. Are all inner classes being placed in the jar ?

(25) Do `gmake spotless` and `gmake` work ? Finish the `submit` target.

4. Submit

Submit the `Makefile` and all necessary Java source files. ***Verify the submit !***