1. Overview

In this assignment you will write a very simple text formatter in the spirit of nroff(1), which is similar to the more powerful text formatters such as troff(1), which is proprietary, and groff(1), which is GNU-free software or similar programs such as TeX, created by Donald Knuth, and LaTeX. You will also write code for the simple linear abstract data type Queue, implemented as a linked list.

Note: You may not use tio in any program in this course, nor may you use any class from java.util in any final version of your program. From Sun’s library, you may use only java.lang, which, of course, is automatically available to every program, and java.io, which is available by importing it.


This time, we present the manual page in the format of the project itself. See the subdirectory roff/ for the following files: roff.rf is an example of the markup language you are to implement; roff.lis is output from the markup program, and the specification of your program itself; roff.perl is a sample program you can use to see what your program should actually do. If you find any bugs in this program, post a bug report to ucscclass.cmps012b.

3. Implementation Strategy

As usual, you will need an implementation strategy unless you consider this assignment trivial. So implement it a piece at a time, and each time you have a partial implementation completed, submit what you have, lest there be a submit problem on the due date. If you can read Perl, you might find the Perl solution to be of some help.

1. First begin in exactly the same way as you did for the last assignment: start with a trivial main method which reads lines using PerlBox and prints them out unformatted.

2. Then proceed with the formatting routines, and leave the implementation of the dot commands until the end. You will get the most bang for the byte this way, since the formatter will work for simple documents even without any dot commands.

3. Add in an error message handler which can print error messages to System.err and record the fact that this has been done so that upon program exit, a return code of 1 can be returned to the system. Every error message should be of the form:

   roff.jar: filename: linenumber: message

For example, we might see:

   roff.jar: test3.rf: 6: invalid command: .sp foobar

indicating that the current input file is named test3.rf and that there is the specified problem with line 6. PerlBox can give you the filename and line number information.

4. Add code to identify each line as a command, an empty line, or a line with words in it. An empty line may have no characters, or it may consist of characters for all of which Character.isSpaceChar is true. Print an error message for each dot command and ignore empty lines for now. Print out each word in the line, one word per line. A word is any sequence of characters not containing space characters.

5. Now, instead of printing the words, insert each word thus found into the end of a java.util.LinkedList. Then, whenever an empty line or a dot command is found, remove and print each word in the list at that point.

6. Add a print paragraph routine which creates a local StringBuffer, initially empty. Then remove words from the queue repeatedly, and:

   ...
i. If the StringBuffer is empty, append the new word to the end of it.

ii. If it is not empty, figure out how many spaces to append to it. In the case that the last character is a sentence-ender, namely one of the characters period (« . »), bang (« ! »), question (« ? »), colon (« : »), or semi-colon (« ; »), you want two spaces, otherwise one.

iii. Check to see if adding the spaces and the new word to the string buffer would make it exceed the line length.

iv. If so, print the string buffer and clear it. Then append the word, but not the spaces.

v. Otherwise append the spaces and the word and continue.

vi. Eventually, the queue will be empty. At this point of the string buffer is not, then print it.

7. Make sure that in the case of an empty input line, if the queue of words is not empty, print the queue followed by a single blank line. But if the queue is empty, don’t print anything.

8. Now implement the class LinkedQueue. Important note: The grader will use grep util and investigate whether or not you have submitted any code that uses java.util. If so, your score for this assignment will be divided by two.

9. Now start implementing the dot commands. Initially implement .in, .ll, .pl, .po, and .tm, all of which take a numeric argument and record that numeric argument in a variable. If the numeric argument is missing, the default is used. If it is invalid, a message is printed and the default is used. The method Integer.parseInt can be used to convert a String to an integer.

10. Implement the use of these values: You have already handled the line length, so add in the page offset and indentation. Then make sure a new page is generated when the page is full, and print a blank top margin of the specified size. There is no bottom margin, which is why the default page length is 60 rather than 66.

11. There are a few more dot commands to implement. Do this now and you are done!

4. Class LinkedQueue

As part of your implementation strategy, you are required to implement the class LinkedQueue, i.e., the linked list implementation of the abstract data type Queue. You should being with the following outline, as presented in class. You need not bother writing an interface class for it; just use the implementation directly.

The class itself should have the following private fields and inner classes:

1. private Node front, which points at the head of the linked list.

2. private Node rear, which points at the last node in the linked list.

3. private static class Node, which is an inner class with an Object field to contain the data and a Node field to link to the next node in the list.

The class should make available to the client the following public methods and exceptions:

1. public boolean empty(), to indicate whether or not the queue is empty.

2. public void insert( Object ), which accepts an object and inserts it into the rear of the queue.

3. public Object remove(), which removes the item at the front of the queue and returns it, if there is one, and throws NoSuchElementException if the queue is empty.

4. public class NoSuchElementException, which extends RuntimeException, and is thrown if an attempt is made to remove an element from an empty queue.

During the development phase of your program, you will want to have a working main program before you actually have time to develop the queue, you should temporarily make use of java.util.LinkedList until you have time to develop the queue. It has methods addLast, removeFirst, and size, which correspond to the methods you need to write.
5. What to Submit

Submit the files: README, GNUmakefile, Roff.java, LinkedQueue.java, PerlBox.java, and any other Java source files that you may need to write in order to complete the assignment. Typing either gmake or gmake all should build roff.jar.