A Scheme Word Count Program
A tutorial introduction to literate programming in Scheme

Raja Sooriamurthi

The Unix wc program has been used a didactic example to illustrate literate programming [2, 3]. In continuing that tradition we present below a Scheme version of the wc program written using noweb.

The calling convention of the program is: (wc 'l 'c 'w "file-1" "file-2" ... ) . Where c, w, and l are counterparts of the Unix wc options specifying whether we want a count of the number of characters (bytes), words (white space separated character sequences), or lines (newline characters). If no options are explicitly specified then it is assumed to be (c 1 w). Files are identified by strings following the options. If no files are specified then input is read from the (current-input-port).

Output is returned as a list of the various counts. For instance

\[
\begin{align*}
\text{>} & (\text{wc }"/\text{etc}/\text{motd}" ) \\
& (\text{"/etc/motd" (chars 92) (words 13) (lines 4))} \\
\text{>} & (\text{wc }'l 'c 'w "/\text{etc}/\text{motd}" "/\text{etc}/\text{passwd}" ) \\
& (\text{"/etc/motd" (chars 92) (words 13) (lines 4))} \\
& (\text{"/etc/passwd" (chars 1629) (words 59) (lines 28))} \\
& (\text{total (chars 1721) (words 72) (lines 32))} \\
\text{>} & (\text{wc}) \\
\text{>} & (\text{wc}) \\
\text{Literate} \\
\text{Programming} \\
\text{can be} \\
\text{fun} \\
(\text{<stdio> (chars 33) (words 5) (lines 5))}
\end{align*}
\]

1 The Idea

The wc is a simple Mealy machine as given below:

![Mealy machine diagram]

Figure 1: A Mealy machine that represents the behaviour of the wc program.

The arc transitions occur on a newline (\n), a space or tab (\s, \t) or any other character (\o). Associated with each arc transition we have an associated action. All of these actions increment a counter.
and \( w \) are counters for the number of lines and words in the input. There is another counter for characters \( c \) whose increment is associated with all the arcs and hence has not been explicitly shown.

## 2 The Program

Given the specifications above we have the outline of the program as:

\[
\langle ^* 2a \rangle \equiv \\
\langle Some help routines 4c \rangle \\
\langle Scheme wc 2b \rangle \\
\langle Process an input 3b \rangle
\]

Root chunk (not used in this document).

The main \textit{wc} program consists of two phases (1) processing the input option argument to determine the format of the output \(^1\) and (2) processing the individual files. (In this code we make use of Chez Scheme’s multiple values \([1]\).)

\[
\langle Scheme wc 2b \rangle \equiv \\
\text{(define wc} \\
\text{(lambda args} \\
\text{(call-with-values} \\
\text{(Split input args 4b) \\
\text{(Process all of the input files 2c))})
\]

Defines:
\text{wc, never used.}
This code is used in chunk 2a.

To process the input we have to check if no files have been specified and if so we receive input from the \text{(current-input-port)} or else we process the specified files.

\[
\langle Process all of the input files 2c \rangle \equiv \\
\text{(lambda (options all-files)} \\
\text{(if (null? all-files)} \\
\text{;; then process the (current-input-port) \\
\text{(call-with-values} \\
\text{(lambda () (process-input (current-input-port))) \\
\text{(lambda (chars words lines) \\
\text{‘(<stdio> (chars ,chars) (words ,words) (lines ,lines))) \\
\text{;; else process the list of the input files \\
\text{;; keeping track of a running total \\
\text{(let loop \\
\text{([files all-files] \\
\text[[total_chars 0] \\
\text[[total_words 0] \\
\text[[total_lines 0]) \\
\text(if (null? files) \\
\text(if (null? (cdr all-files)) \\
\text;; there was only 1 file \\
\text’() \\
\text;; else return a summary \\
\text((list ‘(Total (chars ,total_chars) (words ,total_words) \\
\text(lines ,total_lines)) \\
\text(Process a single file 3a)))))
\]

Defines:
\text{total_chars, used in chunk 3a.}
\text{total_lines, used in chunk 3a.}

\(^1\)This program isn’t complete: it only processes the full set of options and not a particular subset.
To process a single file we first open it, count the number of characters, words and lines, and then update the running total.

\[\text{Process a single file } 3a\] =
(call-with-input-file (first files)
  (lambda (port)
    (call-with-values
      (lambda () (process-input port))
      (lambda (chars words lines)
        (cons ‘,(first files) (chars ,chars) (words ,words) (lines ,lines))
        (loop (rest files)
          (+ total_chars chars)
          (+ total_words words)
          (+ total_lines lines)))))
)

Uses chars 3b, first 4c, lines 3b, process-input 3b, rest 4c, total_chars 3c, total_lines 2c, total_words 2c, and words 3b.

This code is used in chunk 2c.

To process an input we just traverse the aforementioned two state Mealy machine.

\[\text{Process an input } 3b\] =
(define process-input
  (lambda (port)
    (let loop
      ([chars 0]
       [words 0]
       [lines 0]
       [state 's0]
       [in (read-char port)])
      (if (eof-object? in)
        (values chars words lines)
        (case state
          (State 0 3c) (State 1 4a))))))

Defines:
chars, used in chunks 2-4.
in, used in chunks 3c and 4a.
lines, used in chunks 2-4.
process-input, used in chunks 2c and 3a.
state, never used.
words, used in chunks 2-4.
This code is used in chunk 2a.

The first state of the FSA is defined by:

\[\text{State 0 } 3c\] =
[(s0)
  (case in
    [#\newline]
    (loop (add1 chars) words (add1 lines) ’s0 (read-char port))]
  [#\space #\tab]
  (loop (add1 chars) words lines ’s0 (read-char port))]
  [else (loop (add1 chars) words lines ’s1 (read-char port))])
)

Uses chars 3b, in 3b, lines 3b, and words 3b.
This code is used in chunk 3b.
The second state of the FSA is:

\[
\text{State 1 4a) = (s1)}
\]

\[
\text{(case in)}
\]

\[
\text{[(newline)}
\]

\[
\text{(loop (add1 chars) (add1 words) (add1 lines) 's0 (read-char port))]}
\]

\[
\text{[(\#\space \#\tab)}
\]

\[
\text{(loop (add1 chars) (add1 words) lines 's0 (read-char port))]}
\]

\[
\text{[else (loop (add1 chars) words lines 's1 (read-char port))])]
\]

Uses \text{chars}, \text{in}, \text{lines}, and \text{words}. This code is used in chunk 3b.

To determine what are the options to be used while producing the final output we split the input \text{args} into a list of options and a list of the file names.

\[
\text{Split input args 4b) = (lambda ()}
\]

\[
\text{(let loop)}
\]

\[
\text{([args args]}
\]

\[
\text{[options '()])}
\]

\[
\text{(cond}
\]

\[
\text{[(null? args) (values options args)]}
\]

\[
\text{[(string? (first args))}
\]

\[
\text{;; the options have been processed}
\]

\[
\text{(values options args)]}
\]

\[
\text{[else (loop (rest args) (cons (first args) options))])])}
\]

Uses \text{first} and \text{rest}. This code is used in chunk 2b.

Finally some simple help routines.

\[
\text{Some help routines 4c) = (define first car)}
\]

\[
\text{(define rest cdr)}
\]

Defines: \text{first}, used in chunks 3a and 4b. \text{rest}, used in chunks 3a and 4b. This code is used in chunk 3a.

Chunks:

\[
\text{Process a single file 3a)}
\]

\[
\text{Process all of the input files 2c)}
\]

\[
\text{Process an input 3b)}
\]

\[
\text{Scheme wc 2b)}
\]

\[
\text{Some help routines 4c)}
\]

\[
\text{Split input args 4b)}
\]

\[
\text{State 0 3c)}
\]

\[
\text{State 1 4a)}
\]

References

