

Compiler Construction

Practical Exercise 2: Solutions and Discussion

Michael Engel

2.1 Build a Unix tool using lex

The Unix tool `wc` (word count) outputs the following information about a given text file:

- Number of lines in the file
- Number of words in the file
(words are separated by whitespace or punctuation)
- Number of characters in the file
(including whitespace, punctuation characters, etc.)

Implement a version of `wc` using a lex scanner that outputs these three values for a given input.

- There were a number of details different to real `wc` on Unix that caused some confusion...
 - `wc` only uses whitespace as separators
 - What exactly is whitespace?
 - What about the last line of a file?

2.1 Build a Unix tool using lex

- There were a number of details different to real wc on Unix that caused some confusion...
 - wc only uses whitespace as separators
- We accept solutions that use whitespace only and ones that also use a sensible definition of punctuation
 - This was especially problematic for test case a5
 - This contained an apostrophe (ASCII single quote) in "don't" and dots as well as @ in the address "test.mail@domain.dom"
 - ...so wc gives a different result than a solution also considering punctuation

2.1 Build a Unix tool using lex

- What exactly is whitespace?
 - The `wc` man page on macOS states:
"White space characters are the set of characters for which the `isspace(3)` function returns true."
 - However, `isspace(3)` is described as a function for "wide characters" (16-bit based Unicode character encoding), so that cannot be correct [many macOS manpages are in bad shape and I have complained about this already in 2003... :)]
 - There is, however, a `iswspace(3)` man page which states:
"The `iswspace()` function tests for the white-space characters. For any locale, this includes the following standard characters:
```\t" ``\n" ``\v" ``\f" ``\r" `` "`  
(tab, newline, vertical tab, form feed, carriage return and space)
  - In the "C" locale, `iswspace()` successful test is limited to these characters only (this might be extended for different locales/languages)

# 2.1 Build a Unix tool using lex

- What about the last line of a file?
- The `wc(1)` manpage states:  
"A line is defined as a string of characters delimited by a <newline> character. Characters beyond the final <newline> character will not be included in the line count."
- Some editors do not automatically add a newline character (`\n`, ASCII 0x0a) at the end of the last line (e.g. macOS TextEdit), some others do (e.g. vim)
- To be consistent with `wc`, you only have to count the newline characters, even if this would be inconsistent with your intuition if the last line misses the terminating newline character...

# 2.1 Build a Unix tool using lex

```
%{
#include <stdio.h>
int num_lines = 0, num_chars = 0, num_words = 0;
}%

%%

\n { ++num_lines; ++num_chars; }

[A-Za-z\-\0-9]+ { num_chars += strlen(yytext); ++num_words; }

. { ++num_chars; }

%%

int main() {
 yylex();
 printf("%d %d %d\n",
 num_lines, num_words, num_chars);
}
```

matches newlines (\n) to count the number of lines in a file

tries to identify words as a sequence of upper and lower case characters, digits and hyphens that **does not** contain whitespace, punctuation marks (commas, periods, semicolons etc.) or any other characters.

counts the remaining characters

# 2.1 Build a Unix tool using lex

Alternative using only whitespace as separator

```
%{
#include <stdio.h>
int num_lines = 0, num_chars = 0, num_words = 0;
%}

%%

\n { ++num_lines; ++num_chars; ++num_words; }

[\\t\\v\\r\\]+ { num_chars += strlen(yytext); ++num_words; }

. { ++num_chars; }

%%

int main() {
 yylex();
 printf("%d %d %d\\n",
 num_lines, num_words, num_chars);
}
```

counts the remaining characters

# 2.1 Build a Unix tool using lex

- Compiling and linking the program (by hand)

```
$ lex mywc.lex
$ ls
mywc.lex lex.yy.c
$ cc -o mywc lex.yy.c -ll
```

This links the lex library "libl". On some Linux systems, there is only a GNU flex library "libfl" provided (which should be linked to libl, but sometimes does not seem to be).

For these, using `-lfl` works



# 2.1 Real-world wc strangeness

- The macOS wc man page gives some more historical information:

"Historically, the wc utility was documented to define a word as a ``maximal string of characters delimited by <space>, <tab> or <newline> characters".

The implementation, however, did not handle non-printing characters correctly so that `` ^D^E " counted as 6 spaces, while ``foo^D^Ebar" counted as 8 characters. 4BSD systems after 4.3BSD modified the implementation to be consistent with the documentation. This implementation defines a ``word" in terms of the iswspace(3) function, as required by IEEE Std 1003.2 (``POSIX.2").

- So it's fine to be a bit confused about the wc behavior...

## 2.2 Count the strings

- Extend your wc tool to also count the number of strings (delimited by double quotes) in the file and output the average string length. You may assume that a string never extends beyond the end of a line.
- Of course, one of our test cases (b1) had strings extending beyond the end of the line (oops)
- So there are two options:
  - Believe what we write and don't count these as strings... (so any line with an odd number of double quotes)
  - Don't trust us and count the strings nevertheless :)

## 2.2 Count the strings

```
%{
#include <stdio.h>
int num_lines = 0, num_chars = 0, num_words = 0;
int num_strings = 0, stringlength = 0;
%}

%state STRING

%%

\n { ++num_lines; ++num_chars; }
<INITIAL>[A-Za-z\-\0-9]+ { num_chars += strlen(ytext); ++num_words; }
<INITIAL>\ " { ++num_chars; BEGIN(STRING); }
<STRING>\ " { ++num_chars; ++num_strings; BEGIN(INITIAL); }
<STRING>.\ { ++num_chars; ++stringlength; }
. { ++num_chars; }

%%

// continued on next slide...
```

## 2.2 Count the strings

```
// ...
%%
int main() {
 yylex();
 printf("%d, %d, %d\n", num_lines, num_words, num_chars);

 printf("Number of strings: %d\n", num_strings);

 if (num_strings > 0) {
 printf("Average string length: %d\n", stringlength/num_strings);
 }
}
```

This solution does also work somewhat with strings that extend beyond the end of a line. However, due to matching a newline without a state qualifiers, newline characters are not included in string length calculations.

If you explicitly want to exclude the case of strings extending beyond the end of a line, you would need to switch from the STRING back to the INITIAL state when matching the newline character.

## 2.3 Test cases

- Run your code from question 2.2 against the example test cases provided on the course web site and submit your output in a text file output.txt.
- You have seen that there are several cases of unclear behavior and unclear/inconsistent formulations in the exercise text
  - ...unintended! :)
- Accordingly, it's OK if some of the test cases don't return the exactly correct result for this exercise...