CS344 - Scanning a Flex

Announcements

- No office hours tomorrow at 9

  L will be around at 2-ish

- HW due Thursday
Flex

-A C driven scanning program.

Scanner
Flex specification $\rightarrow$ FLEX $\rightarrow$ lex.yy.c

lex.yy.c $\rightarrow$ C compiler $\rightarrow$ a.out

Input stream $\rightarrow$ a.out $\rightarrow$ sequence of tokens
To compile:

```
> flex file.lex

> gcc lex.yy.c -lfl

> ./a.out < other-file.txt
```

(if using stdin, might need ctrl-d)
Format for .lex files:

1. definitions
   `% %`

2. rules
   `% %`

3. user code
   (see examples)
Definitions

New definitions to make life easier.

Form: name definition

Ex:

digit  [0-9]
ID   [a-z] [a-z0-9]*

Note: These are regular expressions!
Definitions cont:

- An unindented comment (\*) is copied verbatim to output, up to the next */

- Any indented text or text enclosed in % \% 3 \% is also copied verbatim (with % \% 3 \% removed)

- % top makes sure things are copied to top of output (for example, for # includes)
Rules Section

Format: pattern action

where pattern is unindented, action is on the same line

Any interlaced or %? %? can be used to declare variables, local to the scanning routine.

(other things may cause compile issues)
Allowed Patterns

`x` - match the character x
`
```
```
```
`[^x]` - any char except newline

`[xy]` - matches x, y, or z

`[a-z0-9]` - matches a, b, j, k, l, m, n, o, z
More patterns

`[^A-Z]` - chars other than A-Z (negation)

`[^A-Z\n]` - any char except A-Z or a newline

`[^a-zA-Z\s-\s\[aeiou\]` - any lower case consonant

`[^\s\+\*]`
Patterns (again)

\`r?\`  0 or 1 r's

\`r{2-5}\`  Between 2 \& 5 r's

\`r{2,}\`  2 or more r's

\`r{4}\`  exactly 4 r's

\`name\`  expansion of name definition

\`r$\`  r at end of a line (post webpage)
Precedence:

```
foo | bar*
```

is same as

```
(foo) | (ba(r)*)
```

(since * has higher precedence than concatenation, +
concatenation is higher than or)
C classes

[[:alpha:]] matches anything that satisfies isalpha().

Ex:

[[[:alnum:]]]

[[[:alpha:]] [[:digit:]]]

[[[:alpha:]] [0-9]]

[[a-zA-Z][0-9]]
User code

Optional, just copied directly to the output.

(if empty, leave off last "%%")
Comments

- C style:

  / *   * /

Exceptions

- No comments in the rule section when a regular expression is expected
  (so not beginning of line or after scanner states)

- Not on % option line or definitions
How it works
- Finds longest pattern match possible
- That match (or token) is made available to a global char pointer
  `yytext` w/ length in `yylen`

Then action is performed
- If no match, next char goes to std out.
  (So `%o %` is valid.)
Actions

Ex: \%\% zap me

Ex: \%\% \(t^+\) \(t^+\) $\text{putchar(' ') ;}$ /* ignore */

Stripping large whitespace

Strip excess whitespace
Actions (cont.)

- If action contains a \( \varepsilon \), then action spans until next \( \varepsilon \) (and may go over many lines)

- Action 1 means “same action as the next rule”

- Can be arbitrary C code, including a return. (When run again continues from where it left off.)
Special Actions

- ECHO

- BEGIN followed by name of a start condition places scanner in that condition (more on this later...)

- REJECT tells scanner to go to second best rule

  CAUTION: slow
Ex: Count the # of words

pattern to look for:

character \([a-zA-Z]\)
digit \([0-9]\)

word \((\text{character}\;3\;|\;\text{digit}\;3)\;+\;[^{\text{character}\;3\;|\;\text{digit}\;3}]\)
Ex:

```
% %

a
ab
abc
abcd ECHO . REJECT .

.
```

Scans:  x y z abcd

Output? abcd abc ab a
Conditional Rules

- State based! activated using BEGIN

Define a set of states
- INITIAL is there by default
- Rest defined in %s or %x in first section

Ex:

```
% %
```

```
BEGIN (STRING) 
<STRING> [^"]*  { action; }
```
%s are inclusive start conditions
%ox are exclusive start conditions

After BEGIN, state is active.

If state is inclusive, then rules with no start conditions are still active.

If state is exclusive, then rules with no start conditions are inactive.
Ex:  %s versus %0x
    %s example
    %0%

    <example> foo  action()}
    bar         other_action()}

Vs:
    %0x  example
    %0%

    <example> foo  action()}
    <INITIAL, example> bar  other_action()}

Conditions

- `*` matches all states
- default rule is in all states.

Essentially, pretend:

```
* / in ECHO;
```

is a line of your file.
Ex: Scanner to ignore C comments

```plaintext
% %
% %
int num_line = 1; // local
```
```plaintext
"/\" BEGIN (comment);
```
```plaintext
<comment> [^ \n]
```
```plaintext
<comment> " *" + [^ \n]
```
```plaintext
<comment> \n + line_num;
```
```plaintext
<comment> " *" + "/" BEGIN (INITIAL);
```
```plaintext
```
Can condense

<comment> 3

all rules

3