CS 344 - Scanning & Flex

Announcements

- Turn in HW
- HW2 - due next Friday
**Flex**

- A C driven scanning program.

Scanner (flex specification) → Flex → lex.yy.c

lex.yy.c → C compiler → a.out

Input stream → a.out → Sequence of tokens
Format for .lex files:

definitions

% %

rules

% %

user code

(see examples)
Definitions

New definitions to make life easier.

Form: name definition

Ex:

digit [0-9]

ID [a-zA-Z][a-zA-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:
  " " % % % %
  % % % % is also copied verbatim (with % % % % 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 indented or `%%` can be used to declare variables, local to the scanning routine.

(other things may cause compile issues)
Allowed Patterns

`x` - matches the character x

`\` - any char except newline

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

`[a b j k l m n o z]` - 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-z\s]` - any lower case consonant

$r^*$, $r^+$
Patterns (again)

`r?` 0 or 1 r's

`r{2-5}` Between 2 and 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)
Precendence:

foo | bar *
---|---
*is same as* (foo) | (bar) *

(since * has higher precedence than concatenation, and concatenation is higher than or)

((foo) | (bar))

*
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.
Comments
- C style: /* */

Exceptions
- No comments in the rule section when a regular expression is expected (so not beginning of line or after space/whitespace)
- 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 `yylen`

  Then action is performed

- If no match, next char goes to std out.

  (So `%` is valid.)
Actions (empty)

Ex: % %
    "tap me"

I remove this string

Ex: % %
    ... (tabs + put char (')):
    /* ignore */

→ program to strip extra whitespace
Actions (cont.)

- If action contains a $i$, then
  `action` spans until next $i$ (and may go over many lines)

- Action $I$ 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 file

  Caution: slow
Ex: Word count

```
int word_count = 0;
```

kitten special () : REJECT

```
[\[a\] + word_count]
```

(default rule)
Ex:

```
% %

a  
ab  
abc  
abcd  ECHO = REJECT

\n
// does nothing */

Scans:  x y z a b c d

Output?  a b c d a b c a b a
```

```
\n
a  
ab  
abc  
abcd  ECHO = REJECT

// does nothing */

x y z a b c d a b c a b 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: %s STRING

%%

<STRING> `[^`]* \[ action \]
%s are inclusive start conditions
%sx 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 %o x

%s example
%o %

(INITIAL) <example> foo other_action() j
bar <other ex> other_action() j

%o x example
%o %

<example> foo other_action() j

<INITIAL, example> bar other_action() j
var action 2() j
Conditions

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

Essentially, pretend:

```
<*>. /\n```

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

\% \times \text{comment} \quad \text{a count of current input line}

\%\%

\rightarrow \quad \text{int num_line = 1;} \quad \leftarrow \text{local var}

"/\*" BEGIN (comment);

\langle \text{comment}\rangle \{

\langle \text{comment}\rangle \quad [^* \backslash n] \\
\langle \text{comment}\rangle \quad ^* + [^* / \backslash n]

\langle \text{comment}\rangle \quad \backslash n \quad + \text{line_num};

\langle \text{comment}\rangle \quad ^* + ^/ \quad \text{BEGIN (INITIAL)};
Can condense

\[ \langle \text{comment} \rangle \overline{3} \text{ all rules} \]

\[ \downarrow \]

\[ \langle \text{comment} \rangle \text{ rule 1} \]

\[ \langle \text{comment} \rangle \text{ rule 2} \]