CS344 – Scanning & flex

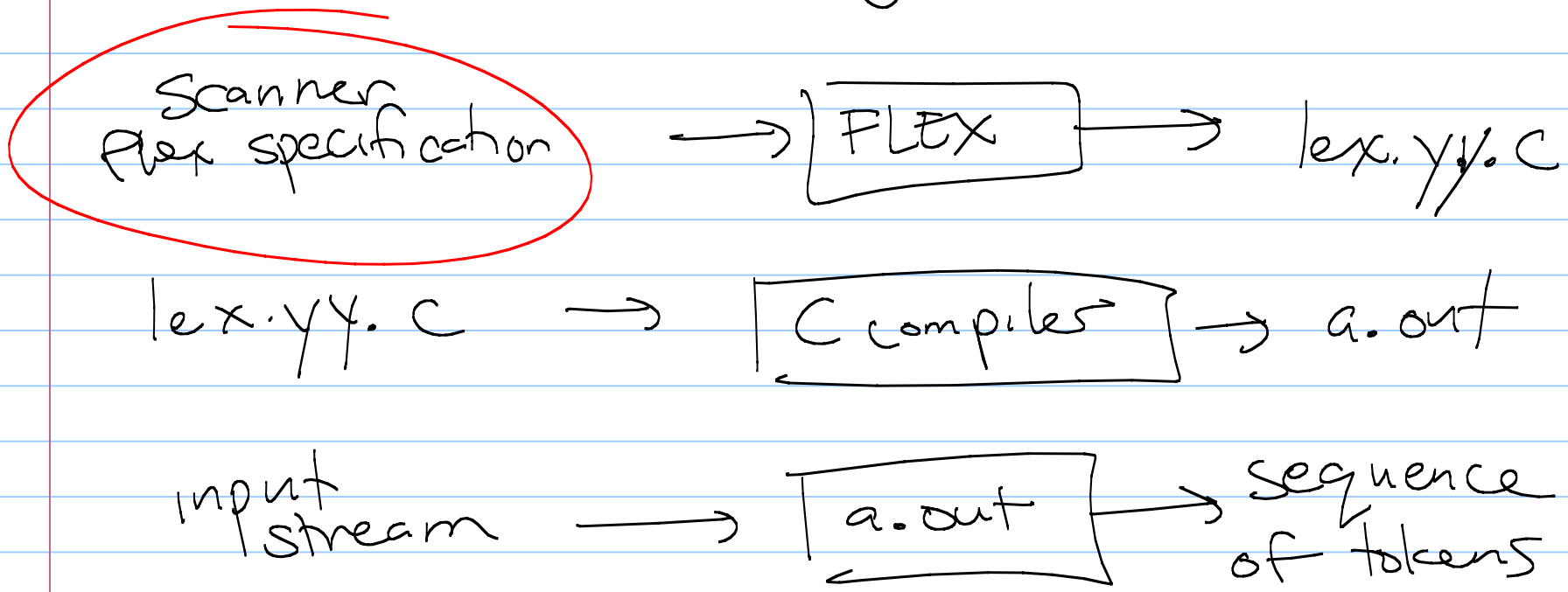
Note Title 1/27/2012

Announcements

- Turn in HW
- HW2 – due next Friday

Flex

- A C driven scanning program.



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-z] [a-zO-9]*

Note: These are regular expressions!

Definitions cont:

- An unidented comment (/*) is copied verbatim to output, up to the next */
- Any indented text or text enclosed in: %{ ... }% is also ~~commented~~ 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 intended 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

`'.'` – any char except newline

`'[x y z]'` – matches x, y, or z

`'[ab j-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]{-}[aeiou]` - any lower case Consenant

r^*, r^+

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 (variables: digit)

`'r$'` r at end of a line

:

(post webpage)

Precendence:

foo | bar*

is same as (foo) | (ba(r)*)

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

((foo) | (bar))*

C classes →

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

Ex:

[[:alnum:]]

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

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

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

User code

Optional, it 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 scanner states)
- Not on % option line of 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 yyleng
  Then action is performed
- If no match, next char goes to std out.
  (So
  %% %%
  is valid.)

Actions (empty)

Ex:

```
%%
"zap me"
```

] remove this string

Ex:

```
%%
[ \t]+$    putchar(' '); /* ignore $*/
```

/* strip tabs & spaces */

→ program to strip extra whitespace

pattern action

Actions (cont)

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

- Action | 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: Word count

```
int   word_count = 0;
%%
kitten      special(); ++word_count REJECT;
[^ \t\n]+   ++word_count;
```

(default rule)

Ex:

```
%%

a     |
ab    |
abc   |
abcd  ECHO; REJECT;
.|\n  /* does nothing */
```

↑ anything but \n

Scans: xyzabcd

Output? abcdabcaba

```
a     |
ab    |
abc   |
abcd  ECHO; REJECT;
```

xyzabcdabc
abaabcd

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
%x 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 %x

```
%s example
%%
<INITIAL>example>foo    action();
<INITIAL>bar            other_action();
<otherex>               action2();
```

```
%x example
%%
<example>foo    action();
<INITIAL,example>bar    other_action();
var                     action2();
```

Conditions

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

  Essentially, pretend:

  <*>./\n ECHO;

  is a line of your file.

Ex: Scanner to ignore C comments + count of current input line

```
%x comment

%%

        int num_line = 1;   <- local var

"/*"    BEGIN(comment);

<comment>{
<comment>[^*\n]*
<comment>"*"+[^*/\n]

<comment>\n     ++line_num;
<comment>"*"+"/"    BEGIN(INITIAL);
```

Can condense

<comment> {

all rules

}

<comment> rule1
<comment> rule2
.
.