

The Flex Scanner Generator

September 17, 2017

Brian A. Malloy



Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



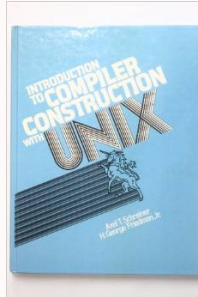
Slide **1** of **21**

Go Back

Full Screen

Quit

1. Text Books



Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



Slide 2 of 21

Go Back

Full Screen

Quit

2. Overview

- Historically, parsing was broken into phases; scanning and parsing were the first 2 phases.
- The scanner provided a sequence of tokens to the parser by reading characters from the input stream and building the tokens.
- flex is a **scanner generator**
- flex reads a spec describing the tokens, and then generates a scanner.



Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



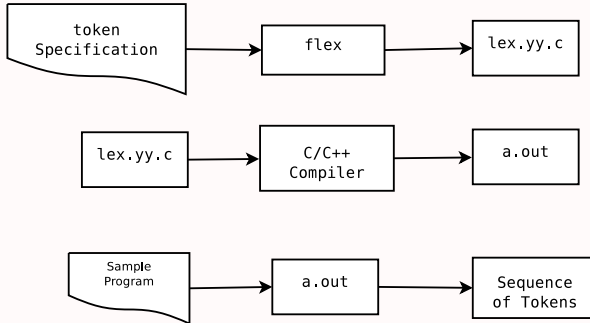
Slide 3 of 21

Go Back

Full Screen

Quit

2.1. Scanner Generation



Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



Slide 4 of 21

Go Back

Full Screen

Quit

2.2. How it works

- flex generates `yylex`, a C fn that implements a DFA, based on the flex specification
- `yylex` reads from stdin and returns a number (token) associated with the matched pattern.
- To match more than one pattern, call `yylex` repeatedly (`yylex` returns 0 at eof):

```
int main() {  
    int token = yylex();  
    while ( token ) {  
        std::cout << "token: " << token << std::endl;  
        token = yylex();  
    }  
}
```



[Text Books](#)

[Overview](#)

[Flex Sections](#)

[Regular Expressions](#)

[Ambiguity](#)

[Start States](#)

[Debugging Flex](#)



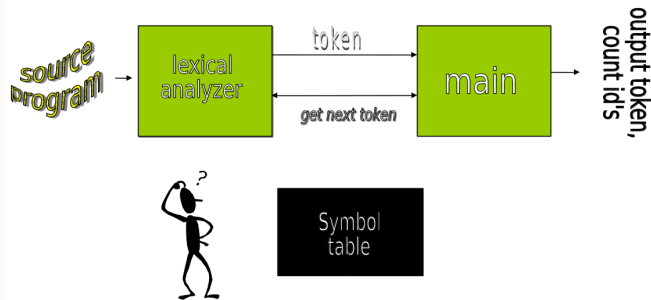
Slide 5 of 21

[Go Back](#)

[Full Screen](#)

[Quit](#)

2.3. Works with main or parser



Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



Slide 6 of 21

Go Back

Full Screen

Quit



Slide 7 of 21

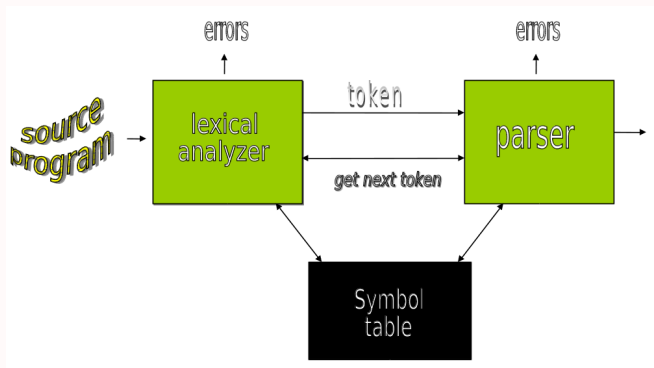
Go Back

Full Screen

Quit

2.4. Works with main or parser

- flex recognizes regular expressions.
- Need **bison** to recognize language constructs



2.5. Lexical Analysis (scanner)

- Usually first phase in compilation
- Also used in editors, query language, testing, ...
- Approaches to building a scanner:
 - Write it by hand,
 - use a tool,
 - Incorporate into parser.



Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



Slide 8 of 21

Go Back

Full Screen

Quit

2.6. Tasks in Lexical Analysis

- Find extraneous chars,
- store names in symbol table,
- strip white space,
- recognize tokens.



Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



Slide 9 of 21

Go Back

Full Screen

Quit

2.7. yywrap()

- called on eof by yylex;
 - if yywrap returns 1, then flex terminates;
 - Otherwise, flex makes another pass.

```
int yywrap() {  
    std::cout << "terminating flex" << std::endl;  
    return 1;  
}
```



Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



Slide **10** of **21**

Go Back

Full Screen

Quit



Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



Slide 11 of 21

Go Back

Full Screen

Quit

2.8. Makefile to use flex in debug mode

```
CCC = g++
LEX = flex
CXXFLAGS=-g -W -Wall -std=c++11 -Weffc++ -Wextra -O0

LEXFLAGS = -Wno-unused
FLEXDEBUG = -d
OBJS = main.o lex.yy.o

run: $(OBJS)
    $(CCC) $(CFLAGS) -o run $(OBJS)
main.o: main.cpp
    $(CCC) $(CFLAGS) -c main.cpp
lex.yy.c: scan.l
    $(LEX) $(FLEXDEBUG) -i scan.l
lex.yy.o: lex.yy.c
    $(CCC) $(CFLAGS) $(LEXFLAGS) -c lex.yy.c
```

2.9. Reading from a file

```
#include <iostream>
#include <fstream>

void main(int argc, char * argv[]) {
    if (argc != 2) {
        cout << "usage: " << argv[0] << "<filename>\n";
    }
    FILE * infile; // Must use C-style I/O
    infile = fopen(argv[1], "r");
    if (!infile) {
        cout << "Could not open: " << argv[1] << endl;
    }
    yyin = infile;
    yylex();
}
```



Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



Slide **12** of **21**

Go Back

Full Screen

Quit

3. Flex Sections

`%{`

C/C++ code

`%}`

Scanner declarations

`%%`

Token definitions and semantic actions

`%%`

C/C++ subroutines

(need prototype in C/C++ code section)



Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



Slide **13** of **21**

Go Back

Full Screen

Quit



4. Regular Expressions

Flex characters have special meanings:

1. `.` matches any single char except newline
2. `[]` character class, matches any char w/in brackets; if first char is `^` it matches any char except those in bracket.
3. `^` matches the beginning of a line as first char in regular expr.
4. `$` matches the end of line as last char
5. `\` escapes metacharacters
6. `*` matches 0 or more
7. `+` matches 1 or more
8. `?` matches 0 or 1 occurrence
9. `|` is alternation

Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



Slide **14** of **21**

Go Back

Full Screen

Quit



10. `()` group

11. `{}` if numbers, specifies how many (`A{1, 3}` matches 1 to 3 consecutive A's), and (`A{2}` matches 2 consecutive A's)

12. `(?s:pattern)` apply option `s` while interpreting pattern. Options include:

- `i` \Rightarrow case insensitive;
- `-i` \Rightarrow case sensitive

13. `(?# comment)` \Rightarrow comments in specs

Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



Slide **15** of **21**

Go Back

Full Screen

Quit

Some Pattern Examples:

```
%{
#include <iostream>
%}

letter      [a-zA-Z]

%%

ab          { cout << "Matching ab" << endl;      }
(?i:c)      { cout << "upper or lower case c" << endl; }
(?-i:d)     { cout << "only lower case d" << endl;  }
(?-i:E)     { cout << "only upper case E" << endl;  }
{letter}*   { cout << "Matching 'letter'" << endl;  }
(?:#: this is a comment)
.           { /* matches everything except \n */ }
```



Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



Slide **16** of **21**

Go Back

Full Screen

Quit



4.1. RE Examples

<code>a+b+</code>	1 or more a's, followed by 1 or more b's
<code>a b</code>	either an a or a b
<code>x</code>	the character x
<code>[abc]</code>	a, b, or c
<code>[0-9]+</code>	an integer
<code>[-+]?[0-9]+</code>	integer with opt sign (- must come 1st)
<code>[\t\n]</code>	whitespace
<code>[mM]</code>	use this rather than (?i:m)
<code>{word}</code>	whatever <i>word</i> is defined as
<code>^r</code>	an r, only at begin of line
<code>r\$</code>	an r, only at end of line
<code>r{3}</code>	exactly 3 r's
<code>r{1,3}</code>	1 to 3 r's
<code>r{2,}</code>	2 or more r's

Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



Slide 17 of 21

Go Back

Full Screen

Quit

5. Ambiguity

- If multiple patterns match a given input:
 - Match longest string,
 - In case of tie, match first pattern in specification.

```
%%  
[0-9]+ { std::cout << "matched 9" << std::endl; }  
9      { std::cout << "no way!" << std::endl; }
```



Slide 18 of 21

Go Back

Full Screen

Quit



6. Start States

- Permits control of what gets matched when
- `\x` defines the **start state**
- When scanner is in a **state**, it can only match the patterns specified in that state.
- Can define as many start states as needed
- The macro **BEGIN** switches states
- **BEGIN(INITIAL)** or **BEGIN(0)** return to start state

Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



Slide **19** of **21**

Go Back

Full Screen

Quit

6.1. C Comments

`%x COMMENT`

`%%`

```
"/*"      { BEGIN(COMMENT); ++comments; }  
<COMMENT>"*/" { BEGIN(0); do_newline(); }  
<COMMENT>\n { do_newline(); }  
<COMMENT>. { ; }
```



Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



Slide 20 of 21

Go Back

Full Screen

Quit



7. Debugging Flex

- The `-d` flag tells flex to go into debug mode:
`flex -d scan.l`
- Flex will then print the rules that are matched:

for `a+b+` on line 12, and `\n` on line 14:

```
aaab
--accepting rule at line 12 ("aaab")
match: aaab
--accepting rule at line 14 ("
")
```

Text Books

Overview

Flex Sections

Regular Expressions

Ambiguity

Start States

Debugging Flex



Slide 21 of 21

Go Back

Full Screen

Quit