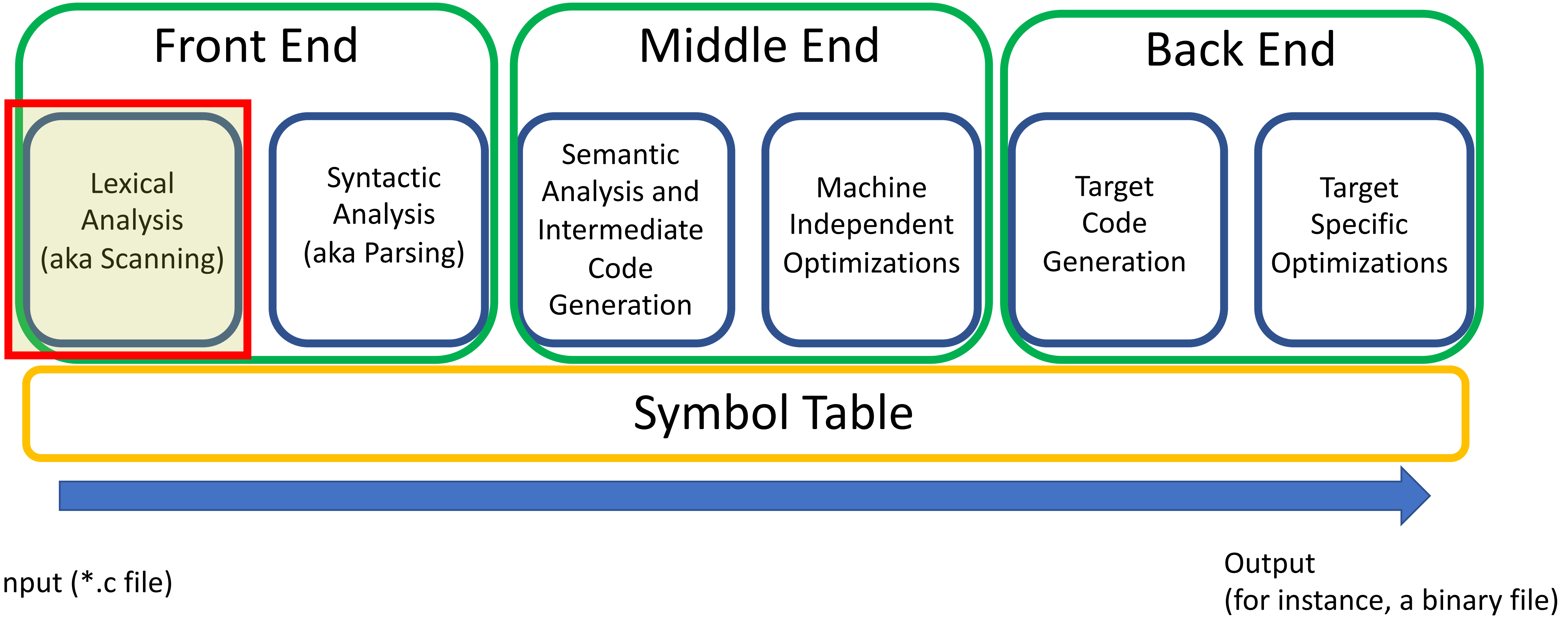
Group Project Part 1: Lexical Analysis

CS 3323 - Spring 2024

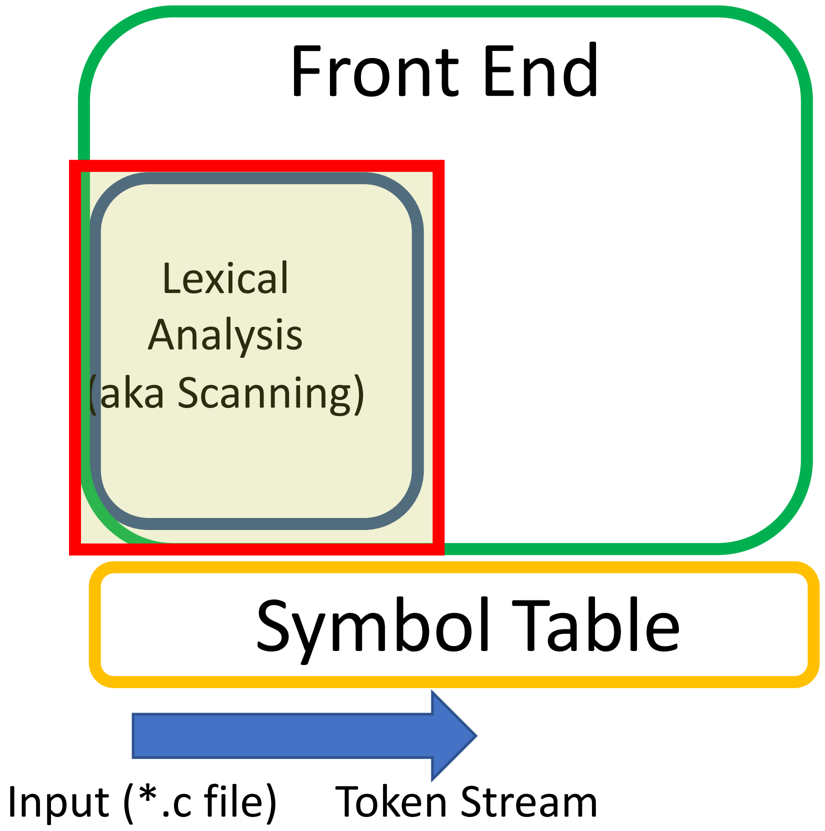
Reminder Individual Assignments

- SSH Login → Due 02/09/2024 11:59 pm
- NIX Crash Course → Due 02/09/2024 11:59 pm

Compilation Overview


Front End
Lexical Analysis (aka Scanning)
Syntactic Analysis (aka Parsing)
Middle End
Semantic Analysis and Intermediate Code Generation
Machine Independent Optimizations
Back End
Target Code Generation
Target Specific Optimizations
Symbol Table
Input (*.c file)
Output (for instance, a binary file)

Lexical Analysis Overview



Lexical Analysis
Goal: Verify that syntax of code is valid and produce a stream of token
Input: Program in Source Language (ex: c file)
Output: A stream of tokens

Procedure:
1. Decompose Input into a stream of small strings (Lexemes) that match the available tokens in the language (typically implemented as Finite State Machines).

State: Symbol table keeps track of variable names and values.

Lexical Analysis (Scanning)

- ❑ Decompose the input file (stream of characters) into stream of strings
- ❑ Ignore white space (tab, return, etc)
- ❑ Assign **token (a category)** to each string: **keywords (K)**, **operators (OP)**, **literals (L)**, identifiers (T_ID)
- ❑ Job performed by the scanner
- ❑ Automatic tools available: Scanner Generators (e.g. Lex, **Flex**, Jflex, etc) that take regular expressions (a DFA) and produce compilable code

Input:

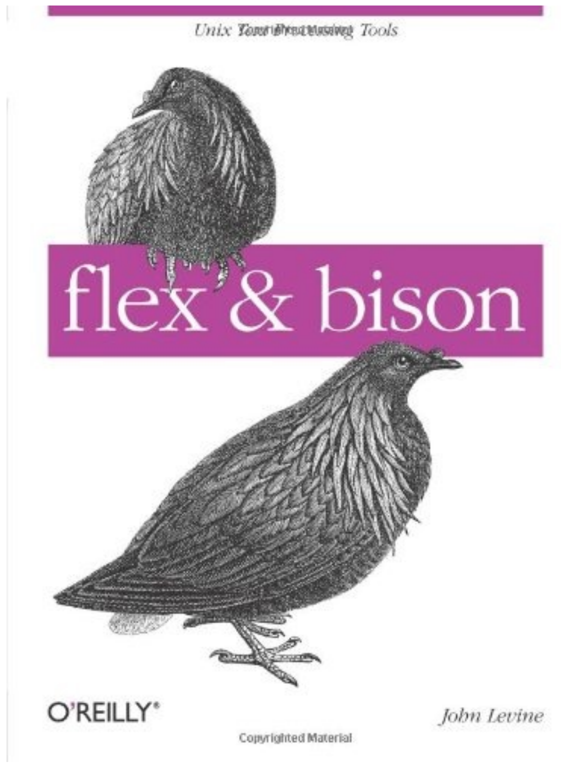
```
if (A > B):
  A = A * 2 + 3.5;
```

Output: **Recognized strings and tokens**

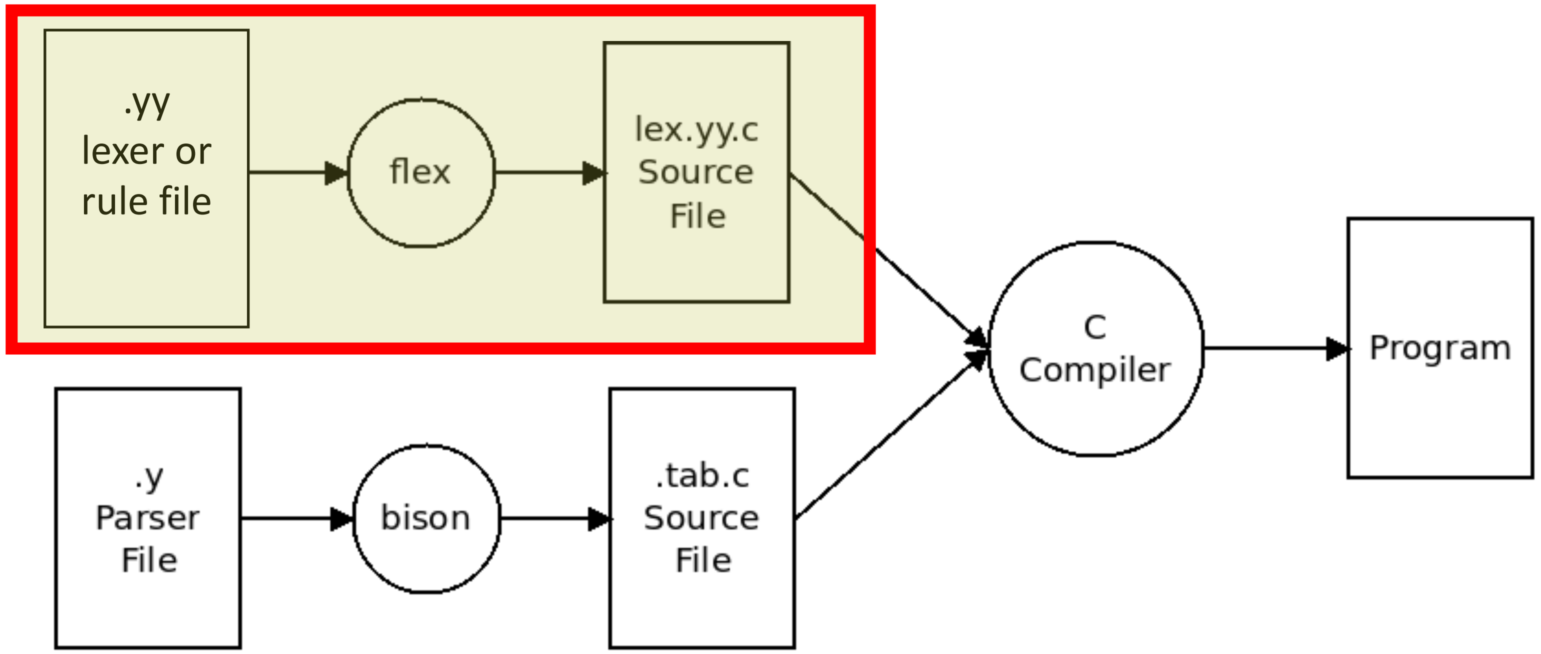
String / Lexeme	Token
if	K_IF
(	LEFT_PAR
A	T_ID
>	OP_GT
B	T_ID
)	RIGHT_PAR
:	COLON
A	T_ID
=	OP_ASSIGN
A	T_ID
*	OP_MUL
2	L_INTEGER
+	OP_ADD
3.5	L_FLOAT
;	SEMICOLON

Lexical Analysis (Scanning)

Compiler-Compiler Tools.



Using Flex and Bison to build a compiler frontend.



Picture modified from https://ianfinlayson.net/class/cpsc401/notes/08-bison

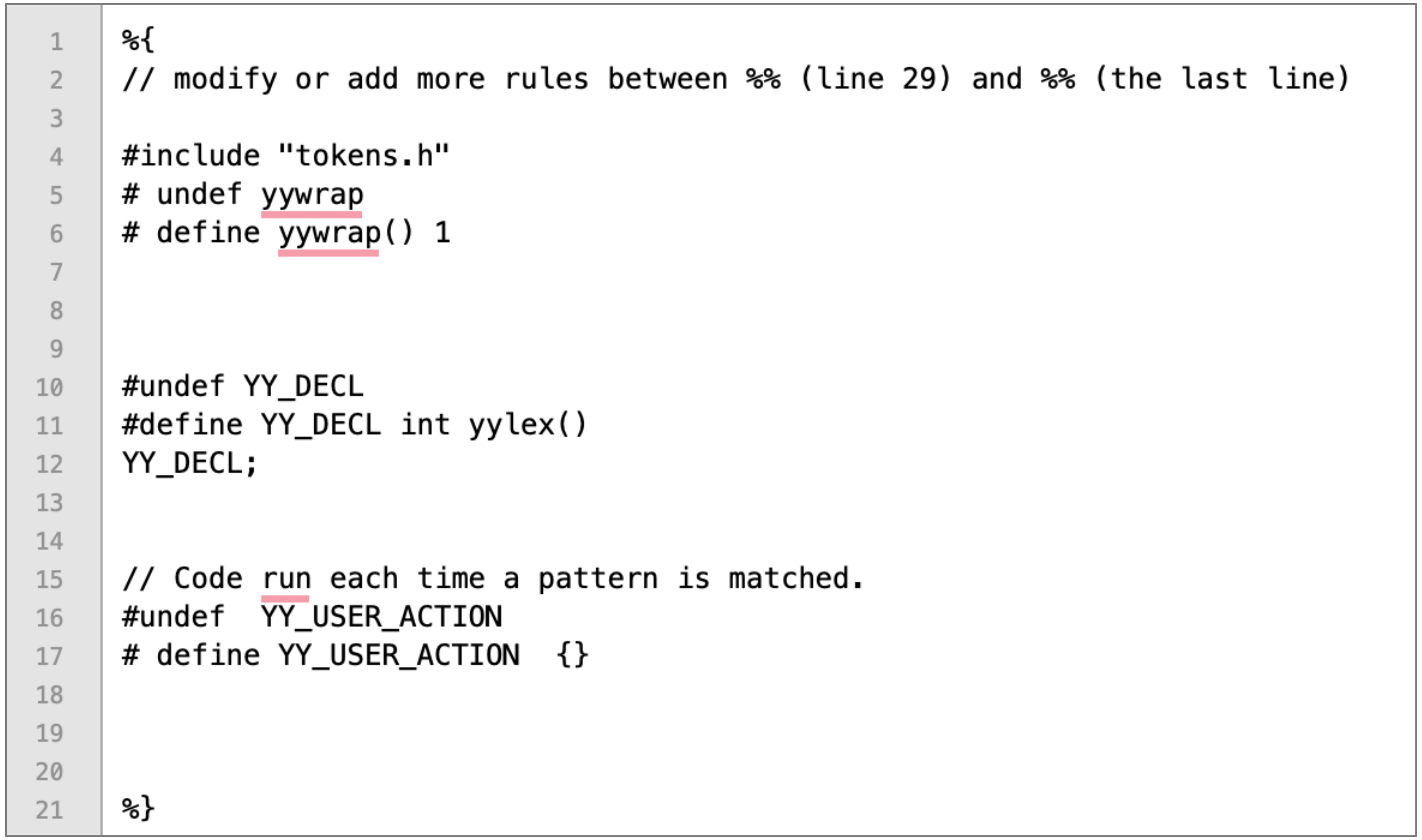
flex, an automatic scanner generators that take regular expressions (a DFA) and produce compilable code

Group Project Part 1

- **We will:**
  - build a mini scanner for a C like programming language (let’s name it “abc” PL)
    - use flex as automatic scanner generator
    - modify the **lexer or rule file** to meet some requirements

- **TODO:**
  1. Download all files under **Files/Group Project/Part1** from Canvas
  2. Compile and run the original (unmodified) code first (in local or remote machine)
  3. Read the instructions described in the PDF file
  4. Modify the rule file (pl-scanner.yy) according to the requirements (you have **2 weeks** to work in a group)
  5. Anytime you modify the rule file, always compile and run it again
  6. Submit the modified pl-scanner.yy file **by 02/23/2024 11:59 pm** (one submission per group), please ensure no compilation error (see #5)

pl-scanner.yy



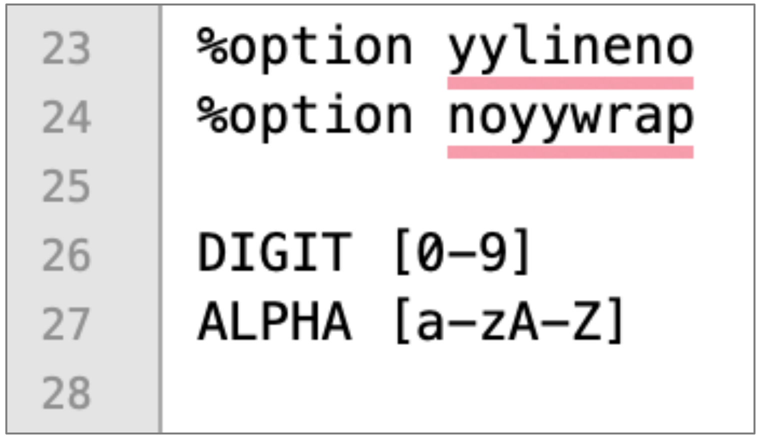
```
 1  %{
 2  // modify or add more rules between %% (line 29) and %% (the last line)
 3
 4  #include "tokens.h"
 5  # undef yywrap
 6  # define yywrap() 1
 7
 8
 9
10  #undef YY_DECL
11  #define YY_DECL int yylex()
12  YY_DECL;
13
14
15  // Code run each time a pattern is matched.
16  #undef  YY_USER_ACTION
17  # define YY_USER_ACTION  {}
18
19
20
21  %}
```

%{ … %} block (Lines 1-21) is used to insert C code into the generated lexer.

It's typically used for including header files, defining macros, and other C preprocessor directives.

Do not modify!

pl-scanner.yy



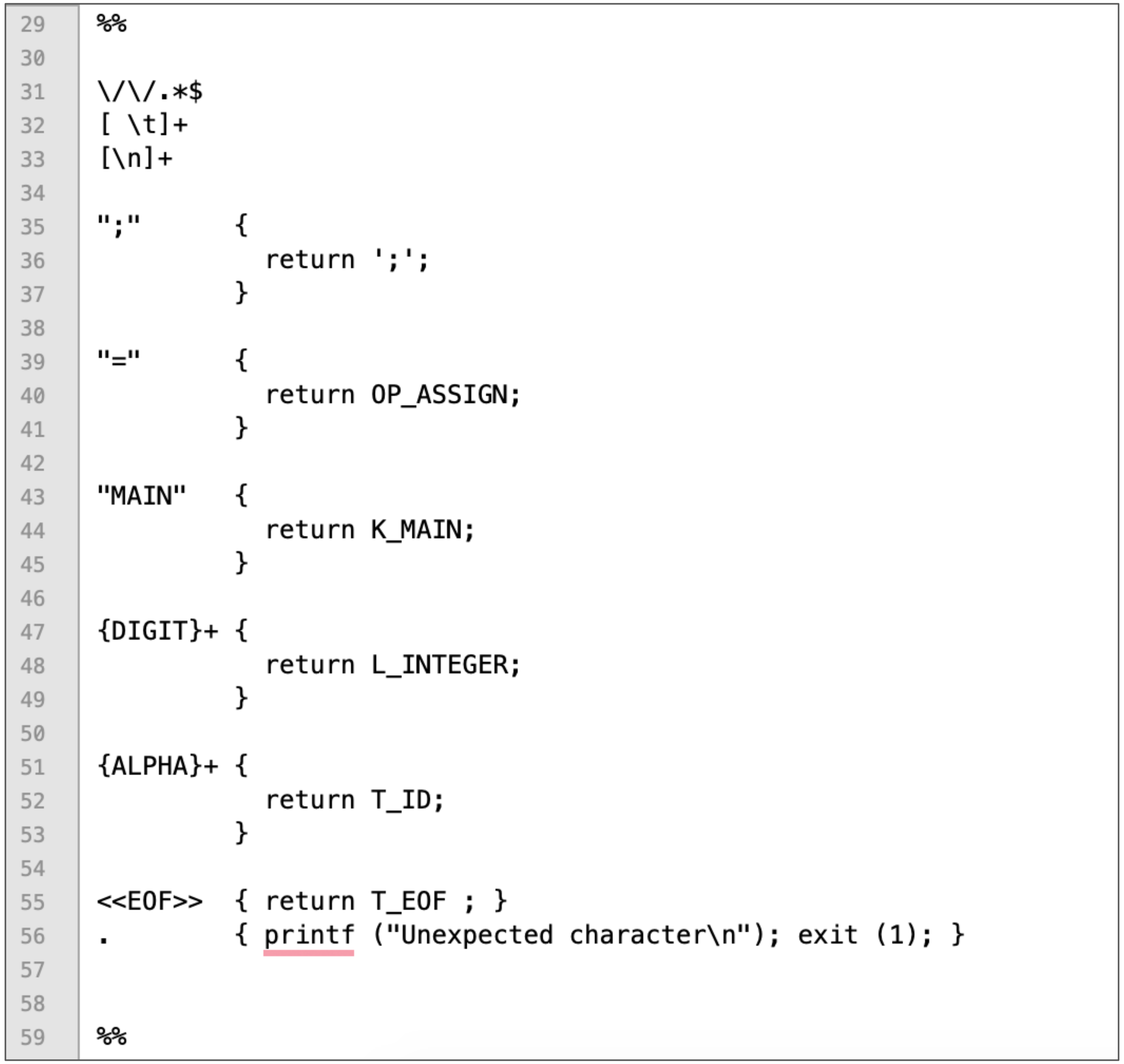
```
%option yylineno
%option noyywrap

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

```

- %option yylineno tells Flex to maintain a line number count.
- %option noyywrap disables the default behavior of Flex to wrap input with yywrap().

defines the macro ‘DIGIT’ to represent any digit from '0' to '9'.

defines the macro ALPHA to represent any uppercase or lowercase letter from 'a' to 'z' or 'A' to 'Z'.

pl-scanner.yy



```
29  %%
30
31  \/\/.*$
32  [ \t]+
33  [\n]+
34
35  ";"      {
36             return ';';
37           }
38
39  "="      {
40             return OP_ASSIGN;
41           }
42
43  "MAIN"   {
44             return K_MAIN;
45           }
46
47  {DIGIT}+ {
48             return L_INTEGER;
49           }
50
51  {ALPHA}+ {
52             return T_ID;
53           }
54
55  <<EOF>>  { return T_EOF ; }
56  .        { printf ("Unexpected character\n"); exit (1); }
57
58
59  %%
```

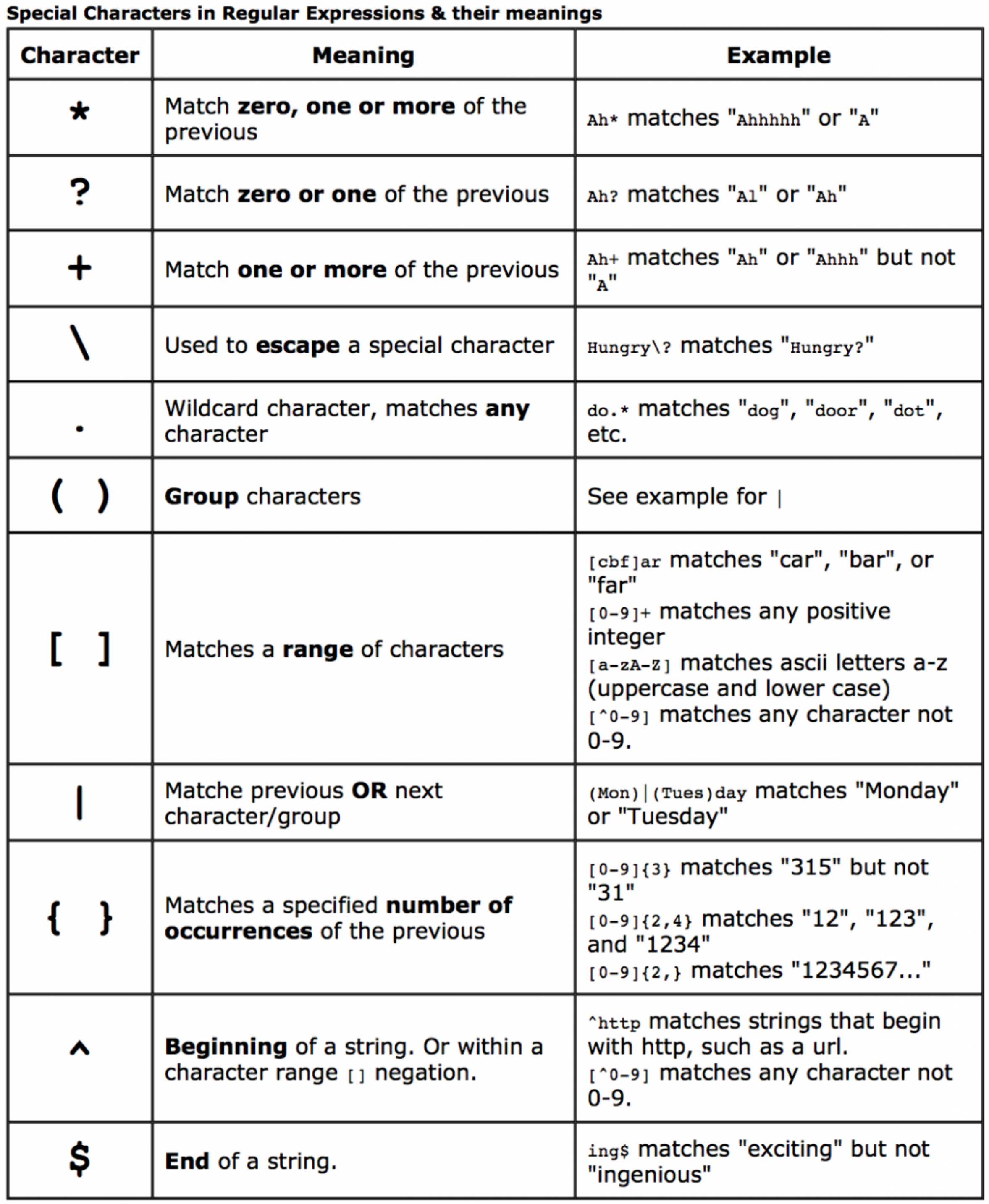
The section between %% (lines 29-59) **defines the lexical rules**

TODO: Modify (add or update) the lexical rules section according to the project instruction

Some tips for the project:

- Regular Expression (Regex) Cheat Sheet
- TA office hours, we are here to help ☺

https://computersciencewiki.org/index.php/Regular_expressions



Special Characters in Regular Expressions & their meanings

Character	Meaning	Example
*****	Match **zero, one or more** of the previous	`Ah*` matches "`Ahhhhh`" or "`A`"
?	Match **zero or one** of the previous	`Ah?` matches "`Al`" or "`Ah`"
+	Match **one or more** of the previous	`Ah+` matches "`Ah`" or "`Ahhh`" but not "`A`"
\	Used to **escape** a special character	`Hungry\?` matches "`Hungry?`"
.	Wildcard character, matches **any** character	`do.*` matches "`dog`", "`door`", "`dot`", etc.
( )	**Group** characters	See example for `|`
[ ]	Matches a **range** of characters	`[cbf]ar` matches "car", "bar", or "far" `[0-9]+` matches any positive integer `[a-zA-Z]` matches ascii letters a-z (uppercase and lower case) `[^0-9]` matches any character not 0-9.
|	Matche previous **OR** next character/group	`(Mon)|(Tues)day` matches "Monday" or "Tuesday"
{ }	Matches a specified **number of occurrences** of the previous	`[0-9]{3}` matches "315" but not "31" `[0-9]{2,4}` matches "12", "123", and "1234" `[0-9]{2,}` matches "1234567..."
^	**Beginning** of a string. Or within a character range `[]` negation.	`^http` matches strings that begin with http, such as a url. `[^0-9]` matches any character not 0-9.
$	**End** of a string.	`ing$` matches "exciting" but not "ingenious"

Regex Examples

Try it at https://regex101.com

Input string	Regex	Match
aaabbbccc	ab	ab
aaabbbccc	a*b*	aaabbb
aaabbbccc	a*b*c*	aaabbbccc
aaabbb	a*b*c+	
aaabbbcc	a*b*c+	aaabbbcc
CS 3323 is cool	[0-9]	3 3 2 3
CS 3323 is cool	[0-9]{4}	3323
CS 3323 is cool	[0-9]{2}	332 323
CS 3323 is cool	[0-9]*	3323
CS 3323 is cool	[a-z]+	is cool
CS3323iscool	[a-zA-Z0-9_]+	CS3323iscool
CS3323iscool	\w+	CS3323iscool

Regex Examples

Regex	Description	Possible matched strings
ab	match the exact sequence ‘ab’	ab
a*b*	match strings that consist of zero or more 'a's followed by zero or more 'b's	a, b, ab, abb, aab, aaaaaabb, …
a*b*c*	match strings starting with zero or more ‘a’s, followed by zero or more ‘b’s and end with zero or more ‘c’s	a, b, c, ab, ac, bc, abc, aaabbb, bcccc, …
a*b*c+	match strings starting with zero or more ‘a’, followed by zero or more ‘b’ and end with zero or more ‘c’	ac, bc, abc, aaaaabbbbc, abccccccc, …
[0-9]	match any digit from '0' to '9'.	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
[0-9]{4}	match four occurrences of digits from '0' to '9'	1234, 5678, 1456, 7910, …
[0-9]{2}	match two occurrences of digits from '0' to '9'	23, 12, 45, 89, …
[0-9]*	match zero or more occurrences of digits from '0' to '9'	, 1, 5, 6, 89, 12345567
[0-9]+	match one or more occurrences of digits from '0' to '9'	1, 2, 3, 12, 123, 5678, …
[a-z]+	match zero or more occurrences of lowercase letters from 'a' to 'z'.	a, aaaaa, abcded, fgh, z
[a-zA-Z0-9_]+	match strings that contain one or more alphanumeric characters (letters or digits) or underscores	C, A, B, ABC, CS3323iscool, _, A_B, ..
\w	match any single character that is a letter (either lowercase or uppercase), a digit, or an underscore	A, b, C, _, …

Compile and Run Project Part 1

❖ **Option 1 use a gpel machine (remote machine)**

1. Download and copy all Part 1 files to a gpel*x* machine, *x* = [8-13]
2. cd to the directory where the files are located
3. To compile, type and run: `make`
4. To run, type: `./pl-scanner.exe < test1.in`

❖ **Option 2 use your local machine**

- ✓ Make sure Flex is installed on your machine (if you use Windows, install it on the **Windows Subsystem for Linux (WSL)**)
- ✓ Open terminal and type: `which flex` or `flex –version`
- ✓ No flex installed?
    - Linux/WSL, open terminal and type:
        - sudo apt-get update
        - sudo apt -y install flex
    - macOS
        - You can use **MacPorts** **or** **Homebrew** to install flex.
        - Need to install MacPorts? https://www.macports.org/install.php
        - **Or** Homebrew? https://brew.sh
        - After installing MacPorts or Homebrew, open the terminal, type, and run:
            - `sudo port install flex`

            or
            - `brew install flex`
- ✓ Download all Part 1 files and follow Steps 2-4 described in Option 1

TA Office Hours

Ega at DEH 115

or on Zoom:
https://oklahoma.zoom.us/j/94320587521?pwd=L2RCVVRCVDFDSW9FSUR1dUlhaUpoZz09
Meeting ID: 943 2058 7521
Passcode: 39931089

- Monday (10:00 - 11:00 am)
- Tuesday (12:00 noon - 1:00 pm)
- Wednesday (9:00 - 10:00 am)
- Thursday (9:00 - 10:00 am)

James on Zoom

https://oklahoma.zoom.us/j/4808825257?pwd=dDAvRTlwSVh2S0FtUkVtTWVzMlBuUT09

Meeting ID: 480 882 5257
Passcode: 7$*%?4CC

- Tuesday (2:00-3:00 pm)
- Wednesday (1:00-2:00 pm)
- Thursday (3:00- 4:00 pm)
- Friday (2:00-3:00 pm)