☑ Lab 1 — Lexical Analysis with flex

Quick Checklist

☐ Build the project with make
☐ Run ./lex sample.csv and compare output with sample.out
☐ Verify tokens match the language specification
☐ Zip your full source directory (after make clean)
☐ Live demo your analyzer to the professor or TA
☐ Write your brief report

📌 Goal

- Learn how to use flex to generate a lexical analyzer.
- Practice writing regular expressions for a realistic dataset.
- Connect regex rules to tokens defined in lexer.h.
- Understand how lexical analysis fits into language tools.
- Reflect on flex and regex in a short written report.

💡 Help & Support

- You may attend either lab section each week (optional, recommended).
- Ask questions in Discord or Canvas forums.
- Office hours and appointments are available.
- Collaboration on setup and testing is encouraged, but your code and report must be your own.

1 Introduction to flex

Flex is a tool for generating scanners. A scanner (tokenizer) recognizes lexical patterns in text. The flex program reads your .l file of regular expressions and actions, and produces lex.yy.c, which implements yylex(). The driver program then calls yylex() to process the input file.

2 Provided Files

You are given these files to start with:

- exp-rules.l (to edit)
- driver.cpp (main program calling yylex)
- lexer.h (token constants)
- makefile (automates build)
- sample.csv, another.csv (input test data)
- sample.out, another.out (expected outputs)

3 Language Specification

Your lexical analyzer must recognize the following tokens and return the constants defined in lexer.h:

Punctuation

Token Constant	Token Value	Example Lexeme
SEPARATOR	20	,

Generic Data Values (can be used in many different data fields)

Token Constant	Token Value	Example Lexeme
DATE	10	2020/04/18 (only concerned with format, not the actual values of year, month, and date)
YES	30	Yes
NO	31	No
UNKNOWN_VALUE	32	Unknown
MISSING	33	Missing

Data Values (used in specific data fields)

Token Constant	Token Value	Example Lexeme
LABORATORY	40	Laboratory-confirmed case
PROBABLE	41	Probable Case
MALE	50	Male
FEMALE	51	Female
OTHER	52	Other
AGE_0X	60	0 - 9 Years
AGE_1X	61	10 - 19 Years
AGE_2X	62	20 - 39 Years
AGE_4X	64	40 - 49 Years
AGE_5X	65	50 - 59 Years
AGE_6X	66	60 - 69 Years
AGE_7X	67	70 - 79 Years
AGE_8X	68	80+ Years
HISPANIC	70	"Hispanic/Latino"
NATIVE_AMERICAN	71	"American Indian / Alaska Native, Non-Hispanic"
ASIAN	72	"Asian, Non-Hispanic"
BLACK	73	"Black, Non-Hispanic"
PACIFIC_ISLANDER	74	"Native Hawaiian / Other Pacific Islander, Non-Hispanic"
WHITE	75	"White, Non-Hispanic"
MULTIPLE_OTHER	76	"Multiple/Other, Non-Hispanic"
EOF_TOKEN	-1	
UNKNOWN_TOKEN	99	

Notes:

- Case: First letter may be upper or lower; remaining must be lowercase.
- Whitespace between lexemes is ignored; whitespace inside a lexeme is significant.
- Newlines increment the line_number variable.
- On lexical error, print error and stop (driver handles this).

4 Implementing exp-rules.l

In your exp-rules.l file:

- Write regex rules that return the correct tokens
- Skip whitespace between tokens
- Increment line_number on newlines
- Return UNKNOWN_TOKEN for unrecognized input

Include the standard file header with your name and NetID.

5 Building & Running

Use the provided makefile:

make
./lex sample.csv

Compare your analyzer output to sample.out:
./lex sample.csv | diff - sample.out

Optional: Stop at first error with:
./lex sample.csv | grep -B 1000 -m 1 ERROR

6 Live Demo

Instead of submitting a screenshot, you will demo your lexical analyzer to the professor or TA. You will run a provided test script and show that your analyzer produces correct output.

7 Written Report

Write a short report (about half a page) reflecting on your Lab 1 experience. Include:

- Two regex rules from your exp-rules.l file. Copy them and explain how they work.
- A paragraph on how flex operates. Look at lex.yy.c and describe what you notice.
- A short opinion: was flex easy or difficult to learn? Did regex save time compared to hand-written scanners?

Your report should be neat, structured, and grammatically correct. Completeness is valued over length.

8 What to Submit

Submit the following:

- A zip of your full source directory (after running make clean)
- Your written report
- Live demo of your analyzer (in lab or office hours)

TL;DR — Build & Submit

Complete all regular expressions in exp-rules.l
make → ./lex sample.csv → compare with sample.out
Demo your analyzer live.
Submit source zip and report.

SUCCESSFUL DEMOS SHOULD LOOK LIKE THIS: