# Fall'25 CPSC 323.13 Compilers & Languages
# Programming Assignment #1 [100 points]

**Submission deadline:** October 3, 11:59PM, submit it on Canvas.

**Demo Time:** 10/06 – 10/08 (a sign-up timesheet will be sent on Canvas)

*Any HW shall **NOT** be made publicly accessible without the written consent of the instructor.*

**Build a Scanner (Lexer).** Lexical analysis is the process of scanning the stream of characters in a program's source code and breaking it down into tokens. You are given a simple C++ source code file, *input_soucecode.txt*. Your scanner must recognize all valid tokens: IDENTIFIER, INTEGER, KEYWORD, OPERATOR, PUNCTUATION, and STRING. Other tokens may be handled ad-hoc.

***Note:***

- Your scanner will not support octal numbers, e.g., "000" will be considered an invalid.
- During the demo, the input source code may differ slightly from the one provided here.

There are **two** options for building your scanner in a group of 2 persons – you may choose either way. You may also choose to complete either way individually.

Option 1: Your task is to write a scanner from scratch by designing and implementing a DFA that extracts tokens from the given source code file.

**Requirements & Deliverables:**

- **Design:**
  - Define the lexical specification for all tokens using REs.
  - Draw the DFA state diagrams that are implemented in your scanner.
  - Wrap the above two parts into a single file named "*myDesign.doc*".
- **Implementation:**
  - Code your scanner in C++, Python or Java. Your program must include a function named *scanner()* or *lexer()* that reads and returns the next token.
- **User Manual:**

  Briefly describe how to set up and run your program in a file named "*readme*".
- **Output:**
  - Your scanner program should print results in two columns: Token and Lexemes
  - Save the result in a file named "*Output*" (refer to the sample I/O below).
- **Submission:**
  - submit the following files: design file, program of your scanner, readme file, and output file.

**Option 2:** Your task is to use Flex (Fast Lexical Analyzer Generator) to automatically generate a lexer that extracts tokens from a given source code file. Flex handles the conversion from REs to NFA and then to DFA for you. Thus, your main task is to write an input Flex source file (*filename.l* ), where you specify lexical patterns (rules) of tokens using REs. The workflow of Flex is summarized in Figure 1.

**Requirements & Deliverables:**

- **Build a Scanner using Flex:**
  - Write your Flex source file (e.g., *sample.l*), containing the specification of token patterns using REs (and C code)
  - Run Flex to generate *lex.yy.c*
  - Compile *lex.yy.c* using a C compiler (*gcc*)
  - The output executable (`a.out` or `.exe`) is your scanner program
- **User Manual:**

Briefly describe how to set up and run your program in a file named "*readme"*.

- **Output:**
  - Your scanner program should print results in two columns: Token and Lexemes
  - Save the result in a file named "*Output*" (refer to the sample I/O below)
- **Submission:**
  - submit the following files: your flex source file (*filename.l*), program of your scanner, readme file, and output file.

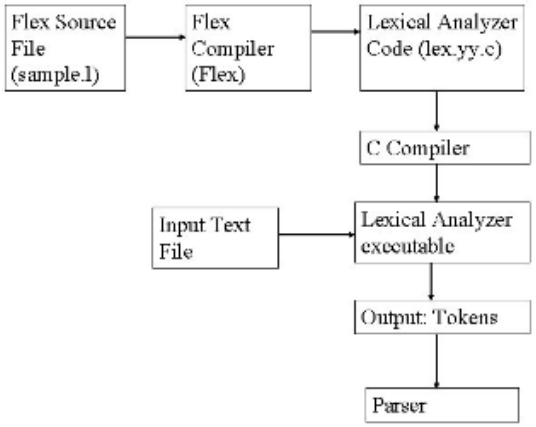


**Figure 1.** Workflow Summary of Flex

## The sample I/O

**Input (source code):**
*while (x > 1)  y = 23.00; //comment*

The scanner breaks the input source code text into tokens while skipping blanks, newlines, and comments.

**Output: (two columns - Token and Lexemes)**

| Token | Lexemes |
|---|---|
| KEYWORD | while |
| PUNCTUATION | (,); |
| IDENTIFIER | x, y |
| OPERATOR | >, = |
| INTEGER | 1 |
| REAL NUMBER | 23.00 |

**Note:** In compiler design, the naming of token types may vary across programming languages and tools being used.

- Most reference and tools (e.g., Flex) categorize symbols such as *(,) and ;* as PUNCTUATION or DELIMITERS. You may follow this convention for our assignments.
- Some programming languages (e.g., Java) call them SEPARATOR.

## Helpful Resources

- **Textbook Appendix B.1**
- **Flex Installation:**
  - **Ubuntu (e.g., CSUF Tuffix), you may use the following commands:**
    - `sudo apt-get install flex`
    - `sudo apt-get update (if new version is needed)`
  - **Windows: http://gnuwin32.sourceforge.net/packages/flex.htm**
    - Download complete package setup and install it
    - Add bin folder (containing flex application) to Path environment variable
- **Flex manual: https://westes.github.io/flex/manual/**
- **Examples: https://www.geeksforgeeks.org/flex-fast-lexical-analyzer-generator/**