Semester project: Compiler Construction

This project is designed to help you understand the different components of a compiler by using a simple grammar. You will be selecting tools for lexing and parsing or implement them manually, and then manually implementing an Abstract Syntax Tree (AST) for expressions, a symbol table, and generating three-address code. Your work should be done in Java or C++, or Python

Part 1: Lexer Generator Selection

**Option A :** Choose one lexer generator tool that works with Java, C/C++ or Python. You will use this tool to tokenize the input based on the defined lexical structures below.

Some options for Java include:

- JFlex: A lexer/scanner generator for Java.

- ANTLR: Can also be used to generate lexers in addition to parsers.

Options for C++ include:

- Flex: A fast lexical analyser generator compatible with Bison parsers.

- RE/flex: The regex-centric, fast lexer for C++ and C.

Options for Python include:

- ANTLR

- PLY: an implementation of lex and yacc parsing tools for Python.

**Option B:** Write Your Own Scanner with AI assistance

You may choose to write your own lexer (scanner) from scratch using your preferred programming language (Java, C/C++, or Python).

As part of this option, you are encouraged to freely and extensively use AI tools such as ChatGPT, GitHub Copilot, Claude, or others.

This part of the project is also an experiment in integrating AI into software development workflows, so there are no restrictions on how you use AI. You may use it for any part of the development process.

The only requirement is to document AI Usage that outlines your experience working with AI tools.

Your report should include:

Which AI tool(s) you used (e.g., ChatGPT, Copilot, Claude)

How you used them (e.g., types of prompts or questions, code suggestions followed)

Notable benefits, limitations, or surprises you encountered

Reflection on how AI affected your development process, thinking, or efficiency

=============Tokens ===========

1) keywords

     double int long char bool fun

if then else true false orelse andalso

2) operators and other

assignment operator (right associative) = +=  -=  \*=  /=

arithmetic binary operator +   -   \*   / (\* double division \*) // (int division)

comparison > < ==   !=

unary operator - !

3) integer literal

DecimalIntegerLiteral defined here

https://docs.oracle.com/javase/specs/jls/se17/html/jls-3.html#jls-DecimalIntegerLiteral

Examples of valid DecimalIntegerLiteral:

0

1

52

5\_2

5\_2L

5\_2l

5\_\_2

5\_21

Examples of invalid DecimalIntegerLiteral:

00  (\* leading zero \*)

01 (\* leading zero \*)

0\_52 (\* leading zero \*)

\_52 (\* starts with underscore \*)

52\_ (\* end with underscore \*)

4) double literal

one or more digits.one or more digits without leading zero

or

.one or more digits

valid double: 0.2, 1.2, .2, 1.0, 0.0

invalid double: 00.2, 1.

5) char literal: single char inside " "

6) identifier

  a letter followed by sequence of digits or letter

  cannot be a keyword in 1)

 letter: a-z, or A-Z

 digit: 0-9

7) separators, one of

;  ,  ( ) { }

8)  use "white space separates tokens" rule

Separators and operators also separate other tokens

"longest token rule":  x+=2; has 4 tokens x += 2 ;

int2x+3;   (\* int2x + 3 ; \*)

2int,    (\* illegal token \*)

9) comment is not a token

(\* this is a comment \*)

Examples of possible input:

(\* comments \*) s

fun sq (int x)=x\*x; (\* fun sq ( int x ) = x \* x ;

if i > j then 1 else 2 ;

bool x = false;   (\*bool x = false ; \*)

int x=-2; (\* int x = -2 ; \*)

int x=2l; (\* int x = 21 ; \*)

long x12=2L; (\* long x12 = 2L ; \*)

{

double i,j; (\* double i , j ; \*)

i+=j; (\* i += j ;\*)

}

char c = "a";

i-j=3; (\* i - j = 3 ; \*)

fun isWeekend x = (x = Sa orelse x = Su);

The lexer should read a file which may contain many lines, output the tokens (including the invalid ones) on screen and file.

Submit

(1) Lexer

Option A (Lexer Generator):

The lexer grammar/specification file (e.g., .flex, .l, or ANTLR lexer definition)

The generated lexer code

Option B (Own Lexer):

Your complete lexer implementation code in your chosen

(2) Test Cases

A set of test inputs and their expected outputs (token streams)

Please include a variety of cases to demonstrate correct tokenization, edge cases

(3) Video Demonstration

Every team member must appear and speak in the video. The video should include:

A short explanation (2–3 sentences) of why you chose your method/tool

An overview of your development process

A brief walkthrough of your lexer structure

A demo of your lexer running on your test cases

For Option B only: Highlight how AI tools helped or influenced your development

Clearly state the specific contributions of each team member

(4) AI Usage Report for Option B only

A brief written report that includes:

Which AI tools you used (e.g., ChatGPT, GitHub Copilot, Claude)

How you used them (e.g., sample prompts, advice or code you used)

Any benefits, limitations, or surprises you encountered

A short reflection on how AI impacted your development process