SimpCalc Compiler DFA

kinoko teikoku

Overview

- For this Project, you will create a Scanner and eventually a Parser for a simple calculator programming language (SimpCalc)
- This part specifically will require you to make the DFA diagram for the Scanner

Groupings

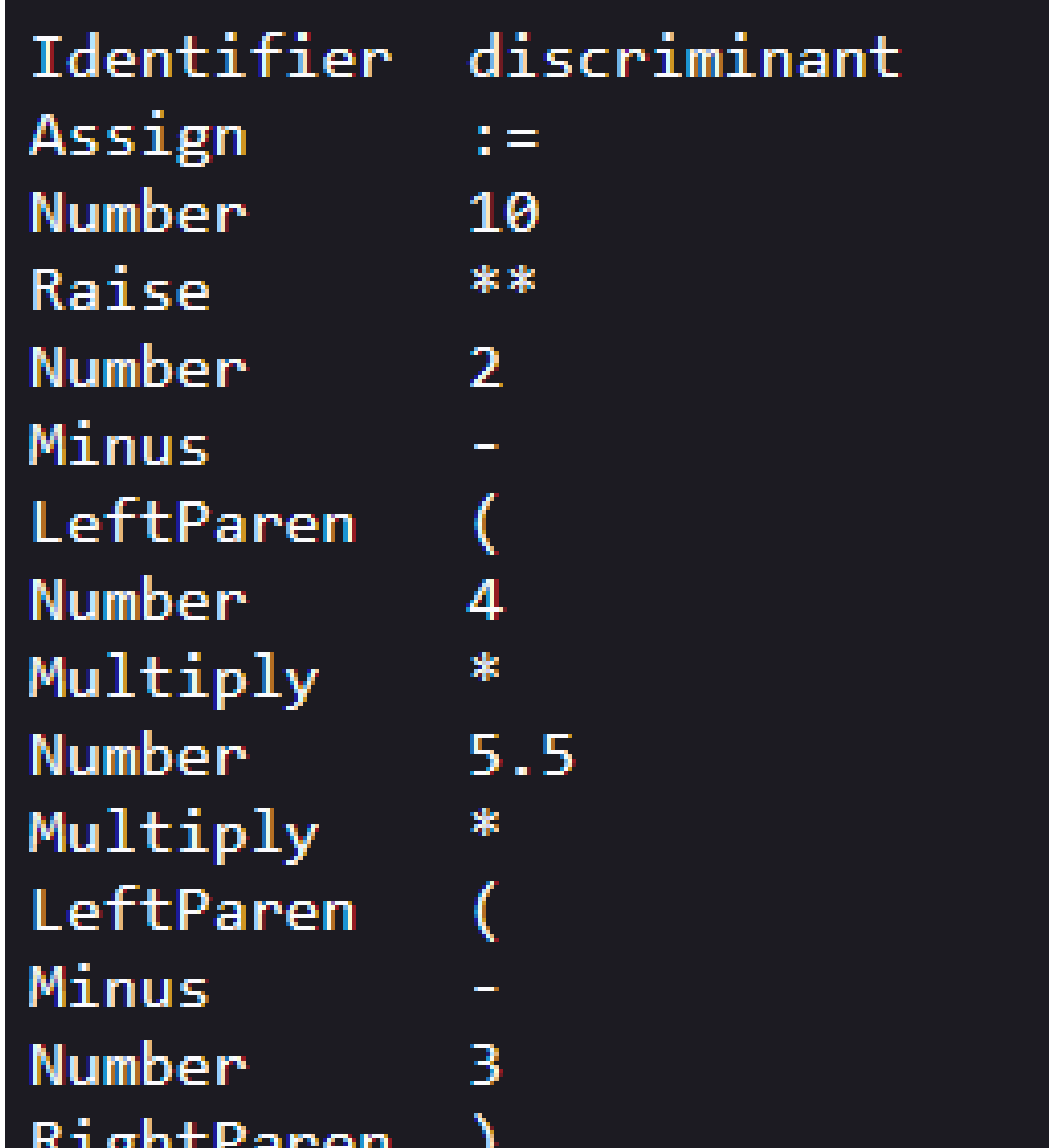
- For this submission, and eventually the Project, you will be in a group with a maximum of 3 members.
- Since I’m handling all sections of this subject, you’re allowed to form groups with members across sections

Overview

- Do note that the Parser that you’ll eventually make will be a simplified version, more details when we get there

```
// this program calculates the roots of the following quadratic equation:
// 5.5x^2 + 10x - 3
discriminant := 10**2 - (4*5.5*(-3));
IF discriminant >= 0:
   root1 := (-10 + SQRT(discriminant))/(2*5.5);
   root2 := (-10 - SQRT(discriminant))/(2*5.5);
   PRINT("roots are",root1,root2);
ELSE // discriminant is negative
   PRINT("no real roots");
ENDIF;
PRINT("end of program");
```

1.	Identifier	
2.	Number	
3.	String	
4.	Assign	:=
5.	Semicolon	;
6.	Colon	:
7.	Comma	,
8.	LeftParen	(
9.	RightParen	)
10.	Plus	+
11.	Minus	-
12.	Multiply	*
13.	Divide	/
14.	Raise	**
15.	LessThan	<
16.	Equal	=
17.	GreaterThan	>
18.	LTEqual	<=
19.	NotEqual	!=
20.	GTEqual	>=
21.	EndOfFile	



```
Identifier  discriminant
Assign      :=
Number      10
Raise       **
Number      2
Minus       -
LeftParen   (
Number      4
Multiply    *
Number      5.5
Multiply    *
LeftParen   (
Minus       -
Number      3
RightParen  )
```

Samples

- In the page for this part of the project there will be a link to a set of text files that you can use as reference
  - Ignore the parser ones for now

Identifier

- Begin with a letter or underscore followed by any number of letters, digits, and underscores
- Letters can be upper or lower case
- (letter | _) (letter | digit | _)*

- note that you can just write transitions as “letter”, “digit”, and “other” – it’s a given that we understand what those are

Keywords

- Keywords are separate tokens, though they are initially recognized as identifiers
- Specifically, if you get an identifier, you must do another step of identifying if it’s a keyword, and what keyword it is
- PRINT, IF, ELSE, ENDIF, SQRT, AND, OR, NOT are the possible keywords (also case sensitive)
- (you don’t have to show this in the DFA)

Numbers

- Whole number digit digit*
- Float (whole num). (whole num)
- Exponent (whole num | float)(e|E)(ε|-|+)(whole num)

- This basically becomes (whole num | float | exponent)

String

- Anything that isn’t a newline between double quotes
- “ (not a newline)* “

Comment

- Comments aren’t tokens
- starts with //
- Anything after that until a newline appears should be ignored

Organization

- Ideally use a program for making diagrams so it’s easier to read
- Figjam, Draw.io, you can hand draw it if you like but please make it legible

Errors

- Errors will occur when a lexeme is built but not resolved to a token
- There will be about 3-4 lexical errors in the DFA

Submission

- The DFA will be submitted via a PDF file, write out the surnames of your group
- Only one member will have to submit
- Don’t forget to include the COA