Jonathan Jong 40133041
Report Assignment 1 COMP 442

My idea behind this was to first generate the expressions required for every possible case the lexical analyzer could accept. I began with defining the different characters I would have to accept in terms of my language / regulate expression:

Legend

x : all char
l: letter alphabet
n: not zero number
p:plus/add
m:minus/subtract
d: digit (0-9)
0: ‘0’
s: *
? : optional
. : period
_ : underscore

z=cases in the diagram below:

Operators, punctuation and reserved words

==	+	or	(	;	integer	while	localvar
<>	-	and	)	,	float	if	constructor
<	*	not	{	.	void	then	attribute
>	/		}	:	class	else	function
<=	=		[	=>	self	read	public
>=			]	::	isa	write	private
						return	

Once I determined my language, I simply concatenated each case with a regex *OR* ‘|’ character, in which I was able to loop upon each case. I also considered having multiple of the same tokens (a valid case for the lexical analyzer) as here, it is only checking for avoid tokens. This was done by adding a regex * to each case.

Regular expression:

ID: (l(l|d| _)*)
Integer: (n(n | d)*|0)
Float: (n(n | d)*|0)(.(d)*n)|(.0)(e(p|m)?(((n(n | d)*|0)))?))
Inline comment: (//(x)*)
Block comment: ((/s(x)*s/)

This is the final regular expression:
(l(l|d| _)*)(n(n | d)*|0)(n(n|d)*|0)(.(d)*n)|(.0)(e(p|m)?(((n(n | d)*|0)))?))(//(x)*)((/s(x)*s/)

https://cyberzhg.github.io/toolbox/nfa2dfa?regex=KGwgKCBsIHwgKCBkIHwgXyApKikpKnwocykqfCAoKCgobihkKSopfDApKC4oZCkqbil8KC4wKShlKHB8bSk/KCgobihkKSopfDApKSk/KSkqfCAgKG4oZCkqKSo=

Tools:
Using this GitHub link, the following conversion of the REGEX to the DFA was done. This link was helpful in providing the necessary conversion, allowing for the clarity of the language to be displayed with all of its cases.
IntelliJ and the Java language was used for programming, as it was a familiar language to provide ease of coding and testing.
Test files includes all the cases as well as additional cases.

DFA: