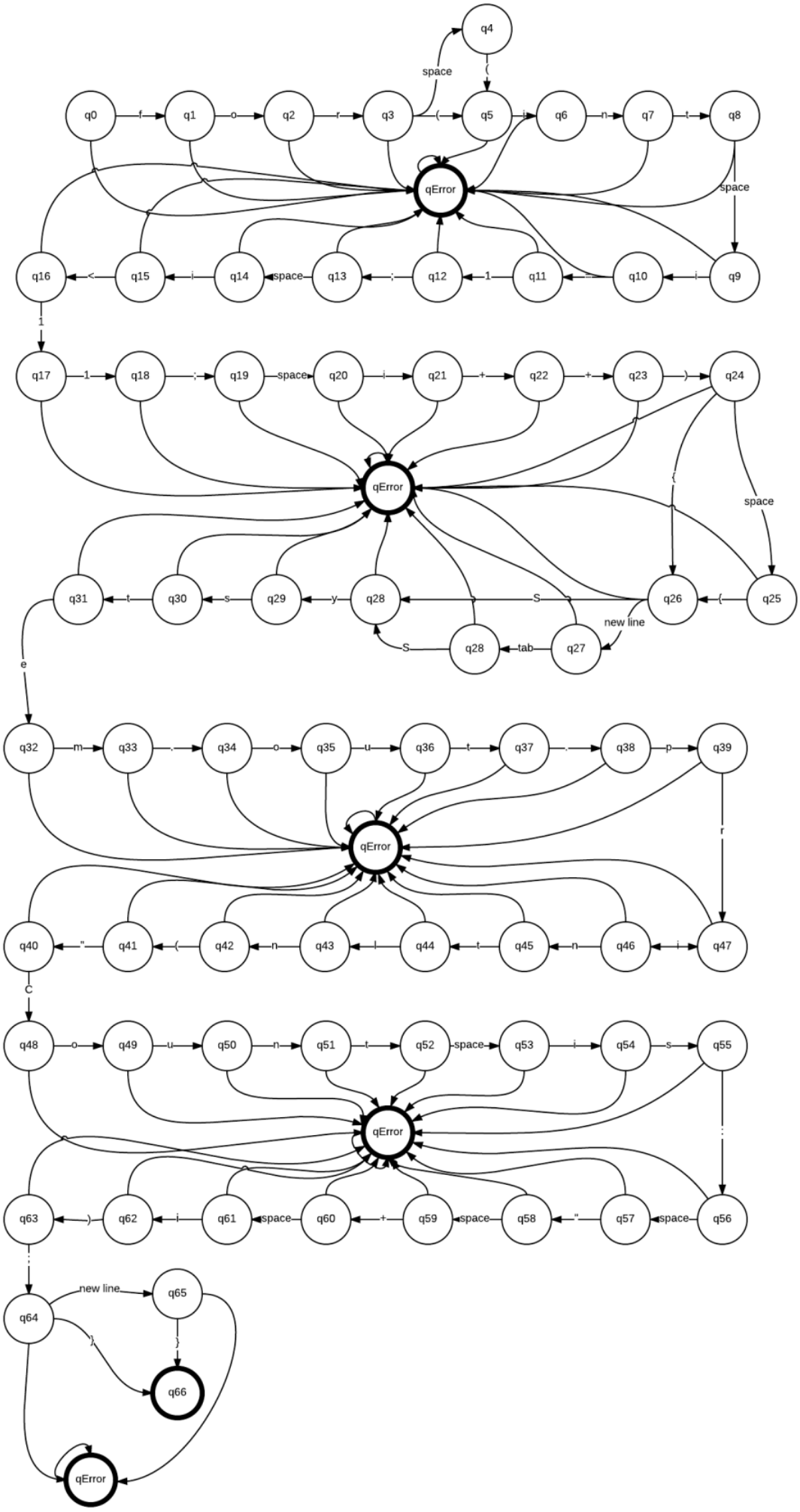
q0
f
q1
o
q2
r
q3
(
q5
i
q6
n
q7
t
q8
space
q4
)
qError
space
q9
i
q10
=
q11
1
q12
;
q13
space
q14
i
q15
<
q16
1
q17
1
q18
;
q19
space
q20
i
q21
+
q22
+
q23
)
q24
{
space
q25
(
q26
S
q28
y
q29
s
q30
t
q31
new line
tab
q27
S
q28
e
q32
m
q33
.
q34
o
q35
u
q36
t
q37
.
q38
p
q39
r
q40
"
q41
(
q42
n
q43
l
q44
t
q45
n
q46
i
q47
C
q48
o
q49
u
q50
n
q51
t
q52
space
q53
i
q54
s
q55
:
q56
space
q57
"
q58
space
q59
+
q60
space
q61
i
q62
)
q63
:
q64
new line
q65
}
q66
}
qError

Essay 1: The largest problem I encountered was with creating the branching paths for the two for loops. Based on the assumption that the second for loop is simply a one line implementation of the first, I had to account for the potential new lines. To accomplish this I introduced branching paths to allow for the input of either of the two for loops. The other problem I had was figuring out the best way to lay out the entire diagram. I decided that having an error node for every node would be repetitive and overly clutter the chart. So I decided to have it such that there was one error node for every two lines of regular nodes. I also left out labeling the arrows to the error nodes because listing all characters aside from the one expected character would also introduce an obscene amount of clutter. I also made sure that the error nodes have the ability to loop back on themselves. Because once you hit an error node no matter what is entered after you can never leave the error node. Upon completing this assignment I realized just how complicated DFA loops and checking for proper syntax really can be. I had figured creating a DFA for a simple for loop would take no more than 20 minutes. In actuality it took me close to an hour and a half.

Essay 2: I am fairly certain that implementing this in Java would be fairly straightforward. You could implement it with a huge nested if statement but that seems both impractical and extremely difficult to code it all. It would also be very hard to scale. A much better solution and the one I would choose would be to use; a for loop. Going character by character I would compare the input of the user to the expected input. Going character by character I would check to see if the user input exactly matches the expected input. If the first character matches one of the allowed first character inputs it will move on to checking the second character and will continue until it reaches the end of user input. When the program checks a character and it does not match it will return a false and jump out of the loop. Implementing the branching possibilities to account for multiple possible for loops would not be all the hard either. I imagine the best way to do this would be to allow for the input of multiple expected inputs. Then when the loop checks for matching input it will compare against all of the expected inputs. If any of the characters don't match, that expected input option will be eliminated. As long as one of the expected inputs evaluate to true, it will all be good. But if they all result in false returns, then the string being evaluated will return false.