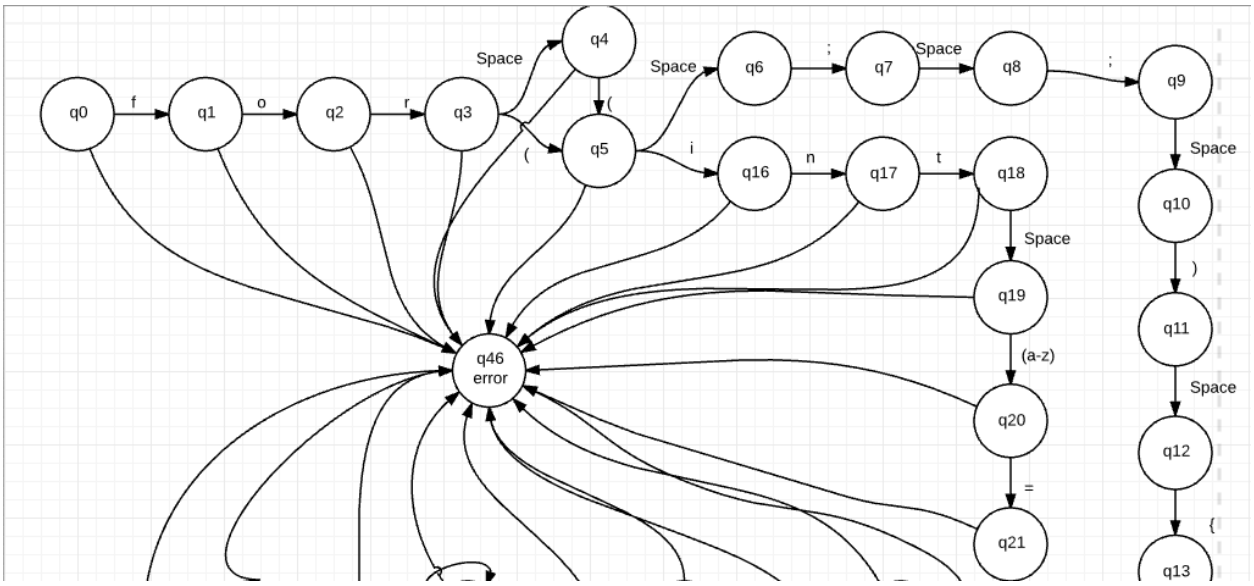
Essay 1 -

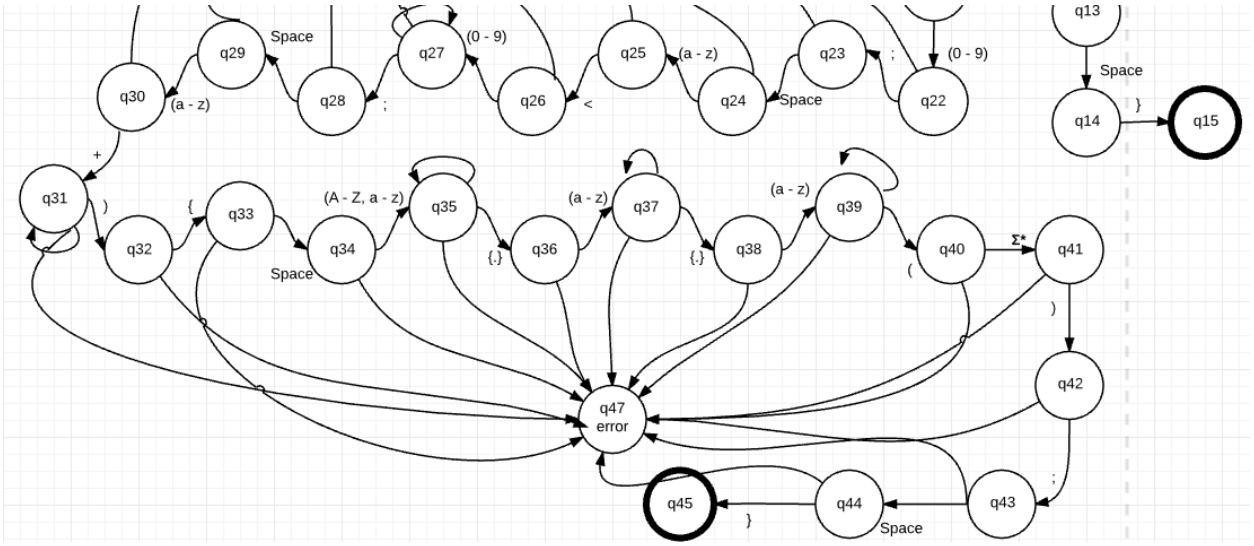
I encountered a few interesting problems while creating this DFA. My initial issue was with the inclusion of the space between 'for' and the '(', which I believe it should allow both formats when initiating the for loop. So to handle this I branched at the case but had them join again at the opening '(', and then I have it split again to account for an empty for loop. Another issue I had was with the formatting of this DFA, and although this is ultimately a small example when compared to the entirety of the language or another loop with extensive functionality. Efficiently formatting the DFA was a challenge, I tried to handle it by having my nodes surround a single error statement, building and branching toward the right and move down once I reached the edge. After traversing down, I moved to the left when my number of nodes became clearly too large to fit, and I used a serpentine pattern to fit the max number of nodes per row. I was forced to snake down and make another area for an error statement, having my nodes surround this additional error statement. Organization, along with creating an efficient traversal strategy to remove the need to input every single character was also difficult, however I was able to handle this to a minimal extent.

Essay 2 -

To implement this DFA in Java, I would use linked lists to form multiple sets of statement validations. Similarly to the DFA, I would format my linked list/ lists so that each node would have singular or multiple pointers. These pointers between nodes would require certain syntax and only following the format will the statement be valid. With multiple pointers, I would be able to account for multiple cases or scenarios of user input, as well as check for and handle errors. I would create sub linked list formats to handle smaller statements that are valid. Trying to put every possibility of valid statements and user input in a single or a minimal amount of linked lists would be confusing, cluttered, and too difficult. I would break down intricate statements into manageable pieces to handle them efficiently, and create a form of organization for ease of implementing new statements, altering neglected statements, or even creating statements left out.


q0
f
q1
o
q2
r
q3
Space
q4
(
q5
Space
q6
;
q7
Space
q8
;
q9
Space
q10
)
q11
Space
q12
{
q13
i
q16
n
q17
t
q18
Space
q19
(a-z)
q20
=
q21
q46
error

as


q30
(a - z)
q29
Space
q28
;
q27
(0 - 9)
q26
<
q25
(a - z)
q24
Space
q23
;
(0 - 9)
q22
q13
Space
q14
}
q15
+
q31
)
q32
{
q33
Space
q34
(A - Z, a - z)
q35
{,}
q36
(a - z)
q37
{,}
q38
(a - z)
q39
(
q40
Σ*
q41
)
q42
;
q43
Space
q44
}
q45
q47
error