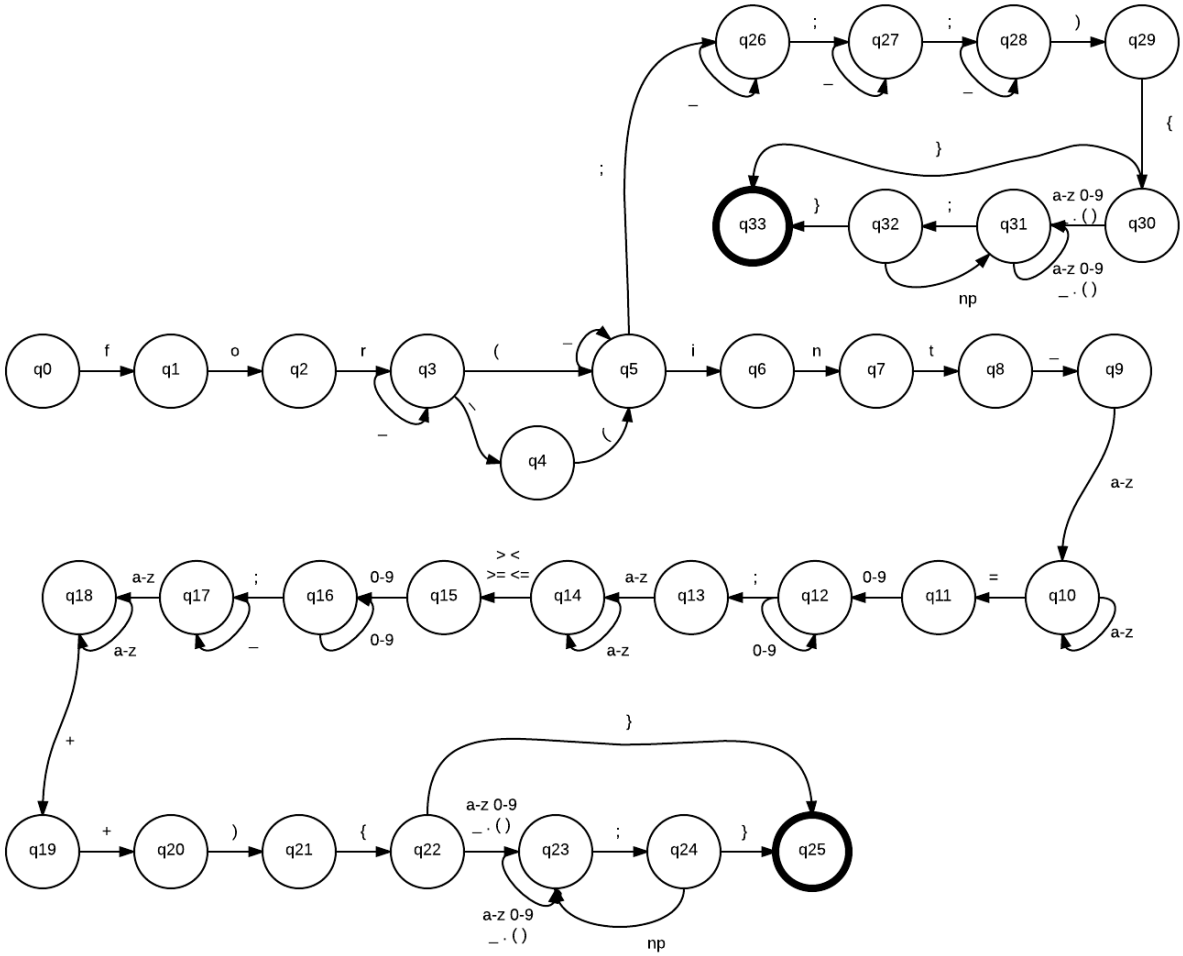
q0
f
q1
o
q2
r
q3
(
_
q5
i
q6
n
q7
t
q8
_
q9
\
_
q4
(
a-z
q10
=
q11
0-9
q12
;
q13
a-z
q14
> <
>= <=
q15
0-9
q16
;
q17
a-z
q18
a-z
_
0-9
a-z
0-9
a-z
_
q26
;
q27
;
q28
)
q29
{
}
q30
a-z 0-9
_ . ( )
q31
;
q32
}
q33
np
a-z 0-9
_ . ( )
;
+
q19
+
q20
)
q21
{
q22
a-z 0-9
_ . ( )
q23
;
q24
}
q25
}
a-z 0-9
_ . ( )
np

Steven O'Meara

Essay 1

I encountered a lot of problems while making the DFA for what seems like a simple for loop. One of the main problems I ran into is dealing with the amount of spaces that a user can put. There are times when the programmer can input as many spaces as he wants before or after a word or line of code. To deal with this I have put np marks to describe a new paragraph as well as recurring space loops on the correct q's. Another big problem that I ran into was dealing with the case where there is no integer declaration. I set it up so that it was possible to never call an initial integer and then refer to a non-existent one later on. To remedy this situation the q's in the top left came in order. They make a specific path so that if the user does not declare an integer they can only continue the next two parts with two semicolons. This makes it so that this problem is avoided, forcing the user into a position where they cannot break the use case in this way. From these problems that I ran into while doing this assignment came the realization of how difficult even a simple for loop is to check if it is completely correct. For the majority of this project, I just had to check that the for loop was created right but on top of this you would also have to check if the statements that come in are punctually correct. If that was also included in this DFA it would easy become extremely huge and hard to follow. Checking that code is of the correct grammar and part of the alphabet is a task that is not easy to implement.

Essay 2

There are many different Java classes that can be used to help implement a DFA. Personally, the one that comes to mind when I think of a DFA is a tree diagram. The issue that arises is that you cannot use a binary search tree as it only allows one option. The only way to tackle this with a tree to define a custom node class that allows multiple child nodes so that there is always a correction option to be chosen. The children would be contained in a custom array list of the <node> class. On top of this it would include a name for the current node so it knows what to look for currently and then compare the next part of the string to the names of the nodes that are underneath the current node. As a string buffer or something of the sort read the string in, it could compare the next part to see if it matches up with any syntax option. Other than creating a custom node class I don't see another simple way of accomplishing this task other than a very convoluted if else checker to see if the next part adds up. Though the if method would technically work, it would be very tedious to set and check for every single case. The easiest option is creating a custom tree structure with a specific node class.