Andre Ibrahim 40132881

SECTION 1: lexical specifications

For this assignment i will be using https://cyberzhg.github.io/toolbox/min_dfa/ to build my DFA from the regex and because of that we need to build our regex very carefully so that we end up with a readable dfa that can help me implement the lexer.
There are a few simplification that we can make, by setting aliases to a set of characters
We can use Letter as is instead of defining every single letter in our DFA which would make it massive. We can also do the same for non-zero, however for digit we have to be careful since if we had aliases for both digit and non-zero then we might create some non-deterministic transitions since there is overlap for the characters. To solve that issue we can just look at the digit to be a non-zero or 0 instead of giving it its own alias. Finally since the regular expression language uses + we need to give that an alias and we will do the same for the - as well. Lower case letters or symbols are as is in the string read. Also to simplify the diagram only a subset of operators/punctuations/reserved keywords will be put in our final regex to reduce the size of it.
One thing that our regex can't include is the imbricated comments since that requires memory (stack) and that is not supported by a regular language.

Aliases:
letter : L
Nonzero: N
+: P
-: M
*: S
A: alphabet

Now that we have our aliases all we have to do is simply define all of our cases where the string would be valid and or them so we can have one regex that we can turn into a dfa.

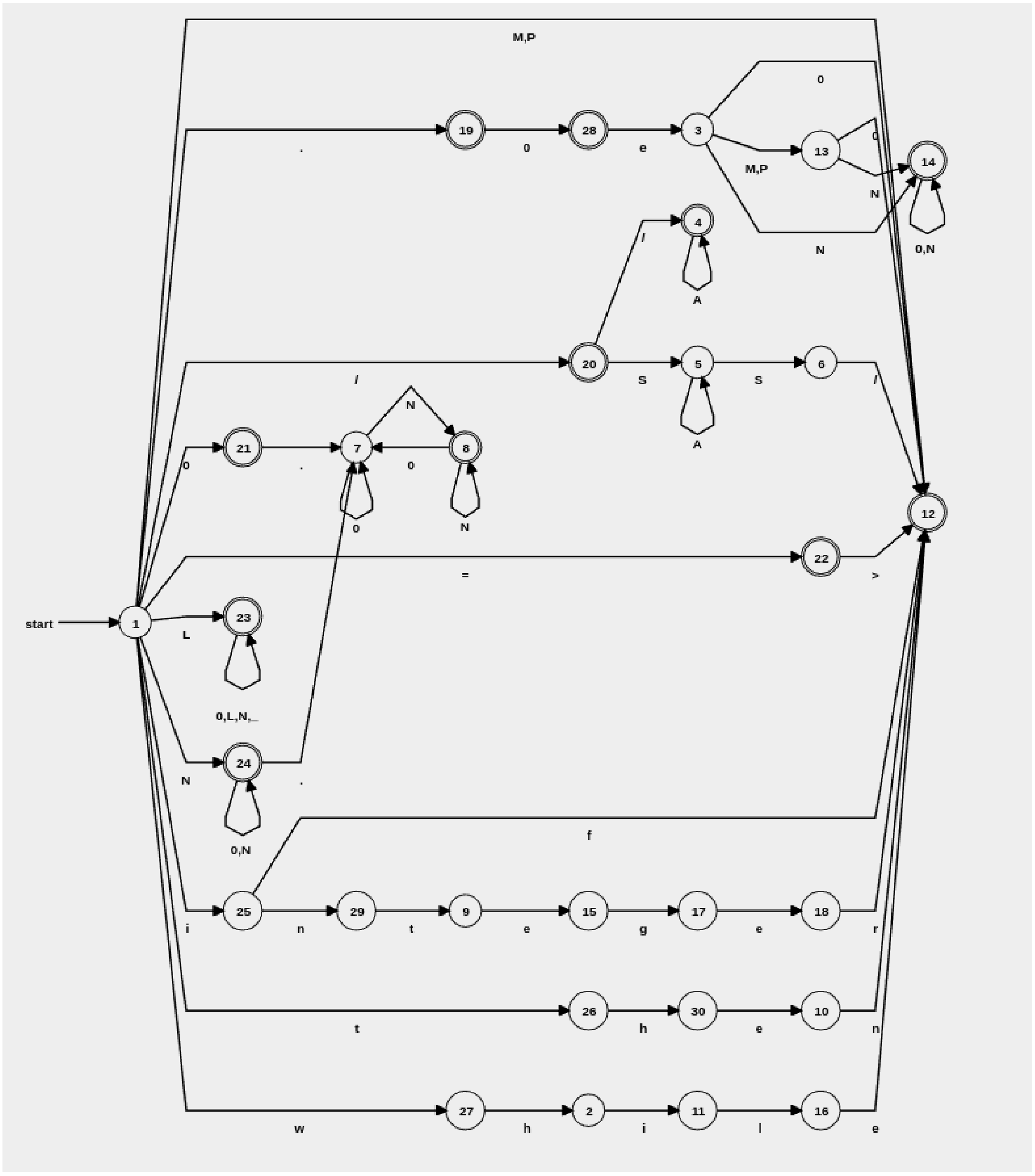
Intermediate regular expression:
identifier : (L(L|(0|N|_))*)
Integer: ((N(0|N)*)|0)
Float: (((N(0|N)*)|0)(.(0|N)*N)|(.0)(e(P|M)?(((N(0|N)*)|0)))?)
Inline comment: (//(A)*)
Block comment: (/S(A)*S/)

And the subset of operators/punctuations/reserved keyword are:
Integer ,while, if , then, ., +, -, =, =>

Final regular expression:
(L(L|(0|N|_))*)|((N(0|N)*)|0)|(((N(0|N)*)|0)(.(0|N)*N)|(.0)(e(P|M)?(((N(0|N)*)|0)))?)|(//(A)*)|(/S(A)*S/
)|integer|while|if|then|.|P|M|=|=>|/

SECTION 2: Finite state automaton

DFA from the regex above:



SECTION 3: Design

The implementation of this lexer is for alternative 1. As we can see from the state diagram above my lexer decides on which state to enter based on the next character. We make use of memory to detect invalid types since that can't be part of a regular language. For convenience we also make use of memory to check if a string is a reserved keyword.

The program has 6 main components, a Lexer, Token,TokenType,a string_helper, a token_helper and finally a lexerdriver.

TokenType is an enum that defines all the possible tokens in the system this include invalid tokens.

Token is a data structure that holds the TokenType, lexeme and position of a token.

Lexer is the class that is responsible for getting the next token through a nextToken function. We can create an instance of that class by passing it the content of the file to be analyzed.

string_helper and token_helper contain helper functions to handle strings and tokens respectively.
Lexerdriver takes all the src files given and creates a .outlextokens and a .outlexerros file for each file in the examples folder.

SECTION 4: use of tools

Regular expression to DFA: https://cyberzhg.github.io/

This tool allowed me to write a regular expression and it converted into a minimized DFA the other tool I tried was JFLAP however it only converts a regular expression to a NFA and it requires manual assistance which is not ideal since it would be prone to user error.

Programing language: Typescript through ts-node

I am using typescript since I am the most familiar with it lately which would make the implementation of this complex project so much easier for me.

Linter: eslint

I am using a linter for this project which allows me to keep my code clean. eslint is the standard linter for typescript/javascript projects.

Testing framework: jest

I am using a testing framework to allow me to test the cases that aren’t present in the given files. I also have a few edge cases I wanted to check.