JSON Scanner

SIDHANT KUMAR
B00961444

CSCI 2115
Theory of Computer Science

Language Description

Data can be stored and transported in a lightweight format called JSON (JavaScript Object Notation). When sending data from a server to a web page, it is frequently used.

In JSON, values must be of the following data types:

- String
- Number
- Object
- Array
- Boolean
- null

The scanner takes input of JSON (JavaScript Object Notation) type and returns tokens accordingly. The language consists of:

- Token for Punctuation:
    - Braces: '{' , '}' are used to define objects.
    - Brackets: '[' , ']' are used for arrays.
    - Parentheses: '(' , ')' to group symbols.
    - Comma: ',' to separate elements.
    - Colon: ':' to separate keys from its values in key-value pairs.
    - Semicolon: ';' to represent the end of a statement.
- Literals:
    - Strings: text enclosed in double quotes.
    - Numbers: Integer and floating-point numbers.
    - Boolean: Keywords including 'true' , 'false'.
    - Null: represents null values.

DFA Design

The DFA (Deterministic Finite Automaton) for the JSON scanner can be implemented by:

Start state (S0) – this is the initial state.
Whitespace State (S1) – to ignore the whitespace and to skip unnecessary spaces.
Token State (S2) – this handles all the punctuation symbols like '{', '} , ',' etc.
String State (S3) – this state recognizes string when an input of " encountered.

Number State (S4) – this state recognizes number when an input of ‘.’ , ‘e’ , ‘E’ , ‘-‘ , ‘+’ or any number (floating/decimal also).
Boolean State (S5) – this state recognizes Boolean keywords ‘true’ and ‘false’ and null values.
Error State (S6) – this state recognizes any illegal or unexpected character.

Current State	Input Character	Next State	Output Token
S0	Whitespace or unnecessary space	S1	None
S0	‘{’ , ’}’ , ‘[‘ , ‘]’ , ‘(‘ , ‘)’ , ‘,’ , ‘:’ , ‘;’	S2	LPAREN etc. respectively.
S0	‘ “ ’	S3	None
S0	0-9 or ‘.’ , ‘e’ , ‘E’ , ‘-‘ , ‘+’	S4	None
S0	Letters	S5	None
S0	Any other character	S6	Error
S1	Whitespace or unnecessary space	S1	None
S1	‘{’ , ’}’ , ‘[‘ , ‘]’ , ‘(‘ , ‘)’ , ‘,’ , ‘:’ , ‘;’	S2	LPAREN etc. respectively.
S1	‘ “ ’	S3	None
S1	0-9 or ‘.’ , ‘e’ , ‘E’ , ‘-‘ , ‘+’	S4	None
S1	Letters	S5	None
S1	Any other character	S6	Error
S2	‘{’ , ’}’ , ‘[‘ , ‘]’ , ‘(‘ , ‘)’ , ‘,’ , ‘:’ , ‘;’	S2	LPAREN etc. respectively.
S2	‘ “ ’	S3	None
S2	0-9 or ‘.’ , ‘e’ , ‘E’ , ‘-‘ , ‘+’	S4	None
S2	Letters	S5	None
S2	Any other character	S6	Error

S3	‘ “ ’	S0	STRING
S4	0-9 or ‘.’ , ‘e’ , ‘E’ , ‘-‘ , ‘+’	S4	None
S4	Any other character	S6	Error
S4	End of Input	S0	Number
S5	true	S0	True
S5	false	S0	False
S5	null	S0	NULL
S5	Any other character	S6	Error
S6	Any character	S6	Error

Hence the above DFA is implemented to take any JSON type input and output the tokens accordingly to the DFA mentioned above.

Code Explanation

There are some key classes in the code, to understand the gist of the written code:
1. Token- which represents each token and also prints them.

2. TokenType - which has different token types.

3. JSONScanner - which recognizes different token into predetermined types like STRING, BOOL etc.

4. LexerError - which raises an exception when an illegal or unexpected value is encountered.

The JSON scanner code follows the above mentioned DFA to process the code into tokens. The scanner first starts at initial state (S0) and when it encounters any whitespace it transitions to (S1), which also helps to skip unnecessary spaces. Then if it encounters any symbols state such as ‘{‘ , ‘}’ etc. it will transition to (S2). Now, if the scanner encounters ‘ “ ‘ then it transitions to (S3) recognizing STRING. Similarly, if the scanner encounters any number or any from ‘.’ , ‘e’ , ‘E’ , ‘-‘ , ‘+’ , it transitions to (S4). And again, if the scanner encounters an alphabetic character, then it transitions to (S5) where it recognizes if the token is ‘true’ or ‘false’ or ‘null’. Lastly, if the scanner detects any illegal or unexpected character then it transitions to (S6). Thus, returning the tokens respectively.

The challenge that I faced was to remove unnecessary spaces and whitespaces between the elements. As my scanner was only scanning one element from input.json before and I was confused as why my scanner was not scanning other elements.