// front.js - a lexical analyzer system for simple arithmetic expressions

// Global declarations

let charClass;

let lexeme = ''; // Stores the current lexeme

let nextChar = ''; // Stores the current character read from input

let lexLen = 0; // Length of the current lexeme

let token = 0; // The current token

let nextToken = 0; // The next token to be processed

// Token codes

const INT\_LIT = 10; // Integer literal token

const IDENT = 11; // Identifier token

const ASSIGN\_OP = 20; // Assignment operator token

const ADD\_OP = 21; // Addition operator token

const SUB\_OP = 22; // Subtraction operator token

const MULT\_OP = 23; // Multiplication operator token

const DIV\_OP = 24; // Division operator token

const LEFT\_PAREN = 25; // Left parenthesis token

const RIGHT\_PAREN = 26; // Right parenthesis token

// Character classes

const LETTER = 0; // Represents alphabetic characters

const DIGIT = 1; // Represents numeric characters

const UNKNOWN = 99; // Represents an unknown character

// Function declarations

function addChar() {

if (lexLen <= 98) { // Check if the lexeme is within the maximum length

lexeme += nextChar; // Add the character to the lexeme

lexLen++; // Increase the lexeme length

}

else {

console.log("Error - lexeme is too long");

}

}

function getChar(input, index) {

if (index < input.length) {

nextChar = input.charAt(index); // Get the next character from the input string

if (/ [a - zA - Z] / .test(nextChar)) { // If the character is a letter

charClass = LETTER;

}

else if (/ \d / .test(nextChar)) { // If the character is a digit

charClass = DIGIT;

}

else {

charClass = UNKNOWN; // If it's neither a letter nor a digit

}

return index + 1; // Return the next index to continue processing

}

else {

charClass = EOF; // End of input

return index; // Return the same index as no more characters are available

}

}

function getNonBlank(input, index) {

while (index < input.length && / \s / .test(input.charAt(index))) {

index = getChar(input, index); // Skip whitespace characters

}

return index;

}

function lookup(ch) {

switch (ch) {

case '(':

addChar();

nextToken = LEFT\_PAREN;

break;

case ')':

addChar();

nextToken = RIGHT\_PAREN;

break;

case '+':

addChar();

nextToken = ADD\_OP;

break;

case '-':

addChar();

nextToken = SUB\_OP;

break;

case '\*':

addChar();

nextToken = MULT\_OP;

break;

case '/':

addChar();

nextToken = DIV\_OP;

break;

default:

addChar();

nextToken = EOF;

break;

}

return nextToken;

}

function lex(input) {

lexLen = 0;

let index = 0;

getNonBlank(input, index); // Skip over any blank spaces

index = getChar(input, index); // Get the first character

switch (charClass) {

case LETTER:

addChar(); // Add the letter to the lexeme

index = getChar(input, index); // Get the next character

while (/ [a - zA - Z0 - 9] / .test(nextChar)) { // While it's a letter or a digit

addChar();

index = getChar(input, index);

}

nextToken = IDENT;

break;

case DIGIT:

addChar(); // Add the digit to the lexeme

index = getChar(input, index); // Get the next character

while (/ \d / .test(nextChar)) { // While it's a digit

addChar();

index = getChar(input, index);

}

nextToken = INT\_LIT;

break;

case UNKNOWN:

lookup(nextChar); // If it's an unknown character, lookup the token

index = getChar(input, index); // Get the next character

break;

case EOF:

nextToken = EOF; // If we reached the end of input

lexeme = 'EOF'; // Set lexeme to "EOF"

break;

}

console.log(`Next token is : ${ nextToken }, Next lexeme is : ${ lexeme }`);

return index; // Return the current index for further processing

}

// Example usage

let input = "a = 8+ 4 \* (3 - 9)";

let index = 0;

while (index < input.length) {

index = lex(input); // Process the next token

}