Job No: 05

Job Name: **Write a program to count the number of lexeme.**

Theory:

A lexeme is a sequence of alphanumeric characters in a token. In the context of computer programming, lexemes are part of the input stream from which tokens are identified. A lexeme is one of the building blocks of language.

For example,

int a = 10;

Here, Total number of lexemes = 5

Code:

```
let countKeywords = 0;
let countIdentifierConstant = 0;
let countSeparators = 0;

let keywords = ["auto","double","int","struct","break","else","long",
"switch","case","enum","register","typedef","char",
"extern","return","union","const","float","short",
"unsigned","continue","for","signed","void","default",
"goto","sizeof","voltile","do","if","static","while"
];

let expression = "if(float(b) + double(c) == 9text--) { char(ABC+++)}";
let separators =
["==","=",",","++","+","--","-","*","/","%","{","}","(",")","<",">","[",
"]"];

function hasSeparators(separators) {
    for(let newline of expression){
        for(let separator of separators){
            if(expression.includes(separator)){
                countSeparators++;
                expression = expression.replace(`${separator}`,' ');
```

```
            }
        }
    }
    return expression;
}

function detectKeyword(keywords,newExpression) {
    let resultKeywords = [];
    for(let expression of newExpression){
        for(let keyword of keywords){
            if(keyword==expression){
                countKeywords++;
                resultKeywords.push(keyword);
            }
        }
    }
    return resultKeywords;
}

function detectIdentifierConstant(keywordList,newExpression) {
    newExpression.filter(keyword => !keywordList.includes(keyword) )
                 .forEach(element=> countIdentifierConstant++);
}

function detectKeywordIndentifier(){
    console.log(`The given expression:\n ${expression}\n`)
    let newExpression = hasSeparators(separators).split(" ")
                        .filter(e => e != "");
    let keywordList = detectKeyword(keywords,newExpression);
    detectIdentifierConstant(keywordList,newExpression);
};

detectKeywordIndentifier();
let countLexeme = countIdentifierConstant + countKeywords
                  + countSeparators;
console.log(`The total lexeme: ${countLexeme}`)
```

Input/Output:

```
The given expression:
 if(float(b) + double(c) == 9text--) { char(ABC+++)}

The total lexeme: 23
```

Job No: 06

Job Name: **Write a program to convert expression from infix to postfix**

Theory: When the operator is written in between the operands, then it is known as infix notation. For example, (p + q) * (r + s) .

The postfix expression is an expression in which the operator is written after the operands. For example, the postfix expression of infix notation (2+3) can be written as 23+.

Code:

```
function precedence (c) {
    if (c == '^') {
        return 3;
    }
    else if (c == '/' || c == '*') {
        return 2;
    }
    else if (c == '+' || c == '-') {
        return 1;
    }
    else {
        return 0;
    }
}

function isOperand (c) {
    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c
<= '9')) {
        return 1;
    }
    return 0;
}

function infixToPostfix (infix) {
    let stack = [];
    let postFix = "";

    for (let i = 0; i < infix.length; i++) {
        if (isOperand(infix[i])) {
```

```
            postFix += infix[i];
        }
        else if (infix[i] == '(') {
            stack.push('(');
        }
        else if (infix[i] == ')') {
            while (stack[stack.length - 1] != '(') {
                postFix += stack[stack.length - 1];
                stack.pop();
            }
            stack.pop();
        }
        else {
            while (stack.length != 0 && precedence(infix[i])
                   <= precedence(stack[stack.length - 1])) {
                postFix += stack[stack.length - 1];
                stack.pop();
            }
            stack.push(infix[i]);
        }
    }
    while (stack.length != 0) {
        postFix += stack[stack.length - 1];
        stack.pop();
    }
    return postFix;
}

let infix = "(A+B)*(c/d)+2";
console.log(`The given infix expression is: ${infix}`);
console.log(`The given postfix expression is: ${infixToPostfix(infix)}`);
```

Input/Output:

```
The given infix expression is: (A+B)*(c/d)+2
The given postfix expression is: AB+cd/*2+
```

Job No: 05

Job Name: **Write a program to count the number of lexeme.**

Theory:

A lexeme is a sequence of alphanumeric characters in a token. In the context of computer programming, lexemes are part of the input stream from which tokens are identified. A lexeme is one of the building blocks of language.

For example,

int a = 10;

Here, Total number of lexemes = 5

Code:

```
let countKeywords = 0;
let countIdentifierConstant = 0;
let countSeparators = 0;

let keywords = ["auto","double","int","struct","break","else","long",
"switch","case","enum","register","typedef","char",
"extern","return","union","const","float","short",
"unsigned","continue","for","signed","void","default",
"goto","sizeof","voltile","do","if","static","while"
];

let expression = "if(i%2==0) { i++; }";
let separators =
["==","=",",","++","+","--","-","*","/","%","{","}","(",")","<",">","[",
"]"];

function hasSeparators(separators) {
    for(let newline of expression){
        for(let separator of separators){
            if(expression.includes(separator)){
                countSeparators++;
                expression = expression.replace(`${separator}`,' ');
```

```
            }
        }
    }
    return expression;
}

function detectKeyword(keywords,newExpression) {
    let resultKeywords = [];
    for(let expression of newExpression){
        for(let keyword of keywords){
            if(keyword==expression){
                countKeywords++;
                resultKeywords.push(keyword);
            }
        }
    }
    return resultKeywords;
}

function detectIdentifierConstant(keywordList,newExpression) {
    newExpression.filter(keyword => !keywordList.includes(keyword) )
                 .forEach(element=> countIdentifierConstant++);
}

function detectKeywordIndentifier(){
    console.log(`The given expression:\n ${expression}\n`)
    let newExpression = hasSeparators(separators).split(" ")
                        .filter(e => e != "");
    let keywordList = detectKeyword(keywords,newExpression);
    detectIdentifierConstant(keywordList,newExpression);
};

detectKeywordIndentifier();
let countLexeme = countIdentifierConstant + countKeywords
                  + countSeparators;
console.log(`The total lexeme: ${countLexeme}`)
```

Input/Output:

```
The given expression:
 if(i%2==0) { i++; }

The total lexeme: 13
```

Job No: 06

Job Name: **Write a program to convert expression from infix to postfix**

Theory: When the operator is written in between the operands, then it is known as infix notation. For example, (p + q) * (r + s) .

The postfix expression is an expression in which the operator is written after the operands. For example, the postfix expression of infix notation (2+3) can be written as 23+.

Code:

```
function precedence (c) {
    if (c == '^') {
        return 3;
    }
    else if (c == '/' || c == '*') {
        return 2;
    }
    else if (c == '+' || c == '-') {
        return 1;
    }
    else {
        return 0;
    }
}

function isOperand (c) {
    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c
<= '9')) {
        return 1;
    }
    return 0;
}

function infixToPostfix (infix) {
    let stack = [];
    let postFix = "";

    for (let i = 0; i < infix.length; i++) {
        if (isOperand(infix[i])) {
```

```
            postFix += infix[i];
        }
        else if (infix[i] == '(') {
            stack.push('(');
        }
        else if (infix[i] == ')') {
            while (stack[stack.length - 1] != '(') {
                postFix += stack[stack.length - 1];
                stack.pop();
            }
            stack.pop();
        }
        else {
            while (stack.length != 0 && precedence(infix[i])
                   <= precedence(stack[stack.length - 1])) {
                postFix += stack[stack.length - 1];
                stack.pop();
            }
            stack.push(infix[i]);
        }
    }
    while (stack.length != 0) {
        postFix += stack[stack.length - 1];
        stack.pop();
    }
    return postFix;
}

let infix = "(X-Y)*(P/5)+Z";
console.log(`The given infix expression is: ${infix}`);
console.log(`The given postfix expression is: ${infixToPostfix(infix)}`);
```

Input/Output:

```
The given infix expression is: (X-Y)*(P/5)+Z
The given postfix expression is: XY-P5/*Z+
```