

Development of the Predictive Parser for SmallJS

Due Date: May 13th, 10 pm, Monday

Deliverables: The file containing main file, "parser1.java," and all the aiding java files, BNF, EBNF, User's Manual for running your program, Optional Developer's Note, Required Screen Shots Showing the Running of Your Program

The zip file containing all the deliverables should be named as **English_Name_Parser1.zip**. When you use MaC PC, do not include Korean character into the zip file name.

Submit early for more credit.

Remove the statements, *package myPackage*; that may be in the first line of the source code, and test your source code before submitting your workings.

zip file 에 한글 이름 넣지 마시길 바랍니다.

제출하는 source code 의 첫줄에 있을 수 있는 **package myPackage; 와 같은 statement 는 없애고** test 한 후 제출하시길 바랍니다.

Description: Try to make a predictive parser for the test language, which can check the validity of the grammar of sample input programs.

This project is to make a recursive descent parser.

The first thing is to make a BNF from the given example inputs, and convert it to an EBNF, and finally make a recursive descent parser. Making BNF is to prepare the development of LL(1) parser. **Make the recursive descent parser as much as you can**, and record in the user's manual the amount of job you implemented.

이 project 는 recursive descent parser 를 만드는 일입니다. 주어진 예제들을 이용해서 **BNF 를 만드시고, 이것을 EBNF 로 바꾸신 다음에** recursive descent parser 로 만드시길 바랍니다. 이 작업은 LL(1) Parser 만들기 위하여 필요한 BNF 를 만드는 일과 겹칩니다. **할 수 있는 부분까지만** 작업하시고, user's manual 에 어느 정도 implementation 되었는지 기록하시길 바랍니다.

Rules

1. User-defined identifier starts with an alphabet letter followed by alphabet or number.
2. The program is case-sensitive.
3. Comments start with "/*" until the end of current line.
4. The assignment is '=', and the equality symbol is '=='.
5. Program starts with <script_start> and ends with <script_end>.
6. The semicolon is a statement terminator.
7. The parser does not consider the tag names of HTML.

Notice!

When your parser encounters an illegal identifier or incorrect sentence, the parsing process just stops. You can use your scanner which you completed as your first project.

(parser 가 illegal identifier 나 incorrect program 을 만나면, 종료됩니다.)

You do not need to implement calculation. Just check grammar.

(계산 결과는 고려할 필요가 없으며, grammar checking 만 하면 됩니다.)

The following is an example output of the parser. Your output may be slightly different.

```
//sample1.jss is valid
<script_start>
var temperature = 20;
var limit = 40, fan = 0;

// temperature monitoring
while (temperature <= limit) {

    if (temperature == limit) {
        document.writeln("temperature limit");
        temperature = 20;
        fan = 1;
    }
    else
    {
        temperature++;
        fan = 0;
    }
}
<script_end>
```

java parser1 sample1.jss

parsing OK

```
//sample2.jss is invalid
<script_start>
var 9temperature = 20;
var limit = 40, fan = 0;
while (temperature <= limit) {
    if (temperature == limit) {
        document.writeln("temperature limit");
        temperature = 20;
        fan = 1;
    }
    elsa {
        temperature++;
        fan = 0;
    }
}
<script_end>
```

Test illegal syntax.

- missing parenthesis at if and else structure
- missing <script_end>

java parser1 sample2.jss

parsing error

elsa is invalid keyword...

The error message
is up to you.

```
//sample3.jss
<script_start>
var temperature = 20;
var limit = 40, fan = 0;
var initial = 0;
var accumulation;

for (temperature = initial; temperature < limit; temperature++) {
    document.writeln("normal temperature");
    accumulation = accumulation + 5;
}

<script_end>
```

java parser1 sample3.jss

parsing OK

```
//sample4.jss is invalid
<script_start>
var temperature = 20;
var limit = 40, fan = 0;
var initial = 0;
var accumulation;
```

Test the three components of for statement.
Missing a part of for statement...

```
for (temperature = initial; temperature < ; temperature++) {
    document.writeln("normal temperature");
    accumulation = accumulation + 5;
}
<script_end>
```

java parser1 sample4.jss

parsing error
temperature <

```
//sample5.jss
<script_start>
var temperature = 20;
var limit = 40, fan = 0;
var choice = 0;
var accumulation;
```

```
choice = 1;
switch ( choice ) {
    case "1":
        limit = 10;
        fan = 5;
```

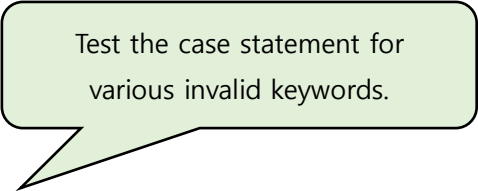
```
        break;
    case "2":
        limit = 20;
        fan = 10;
        break;
    case "3":
        limit = 30;
        fan = 20;
        break;
    default:
        limit = 40;
    }
<script_end>
```

java parser1 sample5.jss

parsing OK

```
//sample6.jss is invalid
<script_start>
var temperature = 20;
var limit = 40, fan = 0;
var choice = 0;
var accumulation;
```

```
choice = 1;
switch ( choice ) {
    sase "1":
        limit = 10;
        fan = 5;
        break;
    case "2":
        limit = 20;
        fan = 10;
        break;
    case "3":
        limit = 30;
        fan = 20;
        break;
    default:
        limit = 40;
    }
<script_end>
```



Test the case statement for
various invalid keywords.

java parser1 sample6.jss

parsing error

```
sase "1":
```

