https://github.com/roxanazachman01/FLCD

# Documentation:

**Statement:**

**Implement a parser algorithm (final tests)**

**Input:**

1. g1.txt + seq.txt
2. g2.txt + PIF.out (result of Lab 3)

**Output: out1.txt, out2.txt**

Run the program and generate:

- out1.txt (result of parsing if the input was g1.txt);
- out2.txt (result of parsing if the input was g2.txt)

-messages (if conflict exists/if syntax error exists - specify location if possible)

**Deliverables**:

Source code for the parser + in/out files + documentation

Code review

**Grading:**

Program works for g1.txt, max grade = 8

Program works for g1.txt and g2.txt, max grade = 10

**Implementation**:

**The algorithms were implemented in the previous lab. For this lab, just did some further checkings that the parser works with multiple grammars and sequences, including the language grammar.**

**Parsing table algorithm:**

For each nonterminal we get the productions, and for each production we get the first set. For each terminal in the first set, if there already is a value in the parsing table for (nonterm, term), then we have a conflict and we print that to the screen. If not, and if the term is not

epsilon, we add the rhs with the index of the production to the table at (nonterm, term). If that first set contained epsilon, then we need to also check the follow set. For each term in follow, if the term is not epsilon, we check whether there already is a value in the parsing table for (nonterm,term). If yes, we print that we have a conflict, if not, we add to the table. If that term is epsilon, then we add to (nonterm, dollarsign).

**Configuration:**

The configuration class contains the parsing table, list of production indexes, input stack and working stack. We have methods for getting the initial config, pushing, popping, checking if we should push, pop, or if the sequence is accepted. The initial config contains the input sequence in the input stack, and the working stack contains the start symbol. The push operation gets the value from the parsing table corresponding to (top_of_working_stack, top_of_input_stack), pops from the working stack, and if the value in the parse table does not contain epsilon, adds to the working stack the rhs from the table value, then adds the index of the production from the table value to the production indexes list

The pop operation just pops the input and working stack. The shouldPush method checks that the sequence is not accepted, that shouldPop is false and the value in the parsing table is not an error. The method shouldPop checks that the value in the parsing table corresponding to (top_of_working_stack, top_of_input_stack) is pop. In the same way, isAccepted checks that the value in the parsing table is Acc.

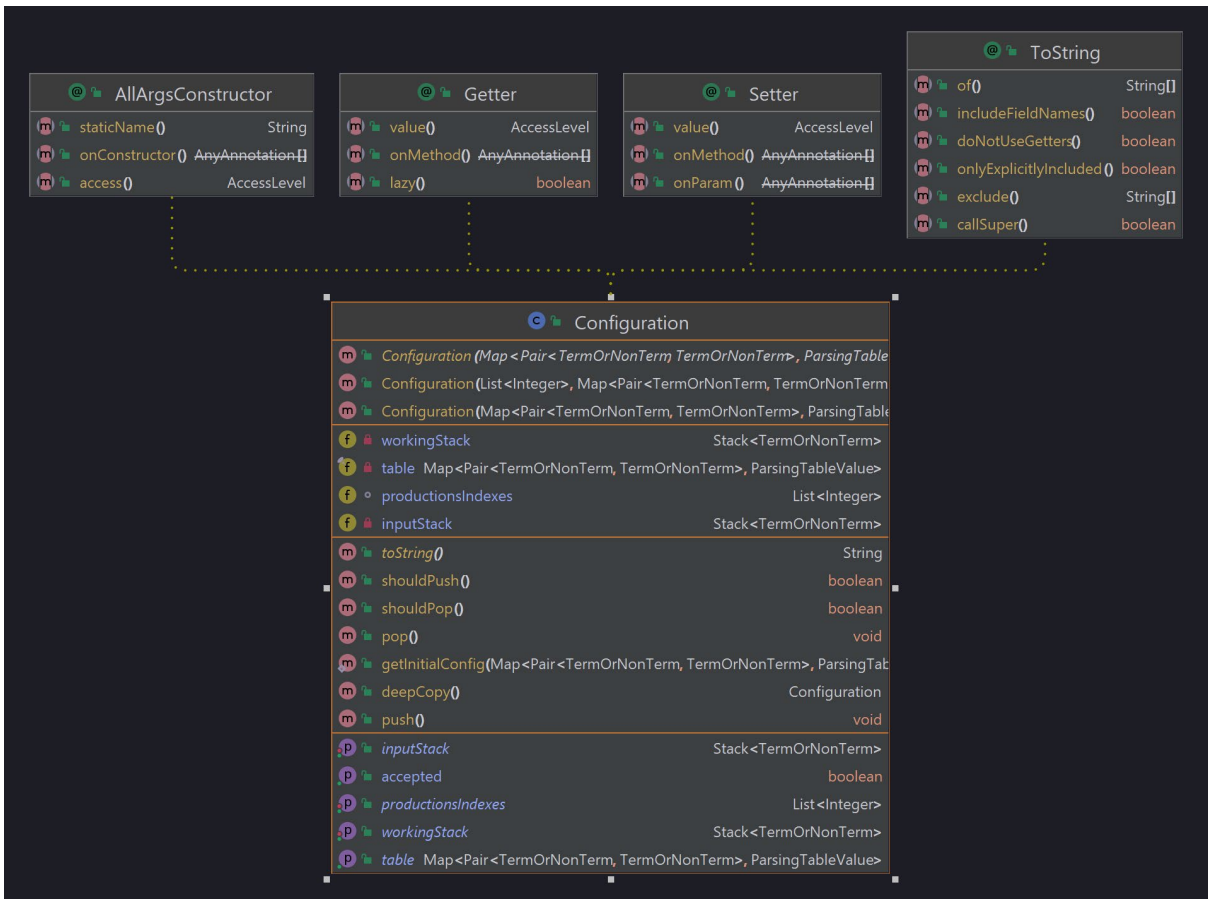
**Parsing sequence algorithm:**

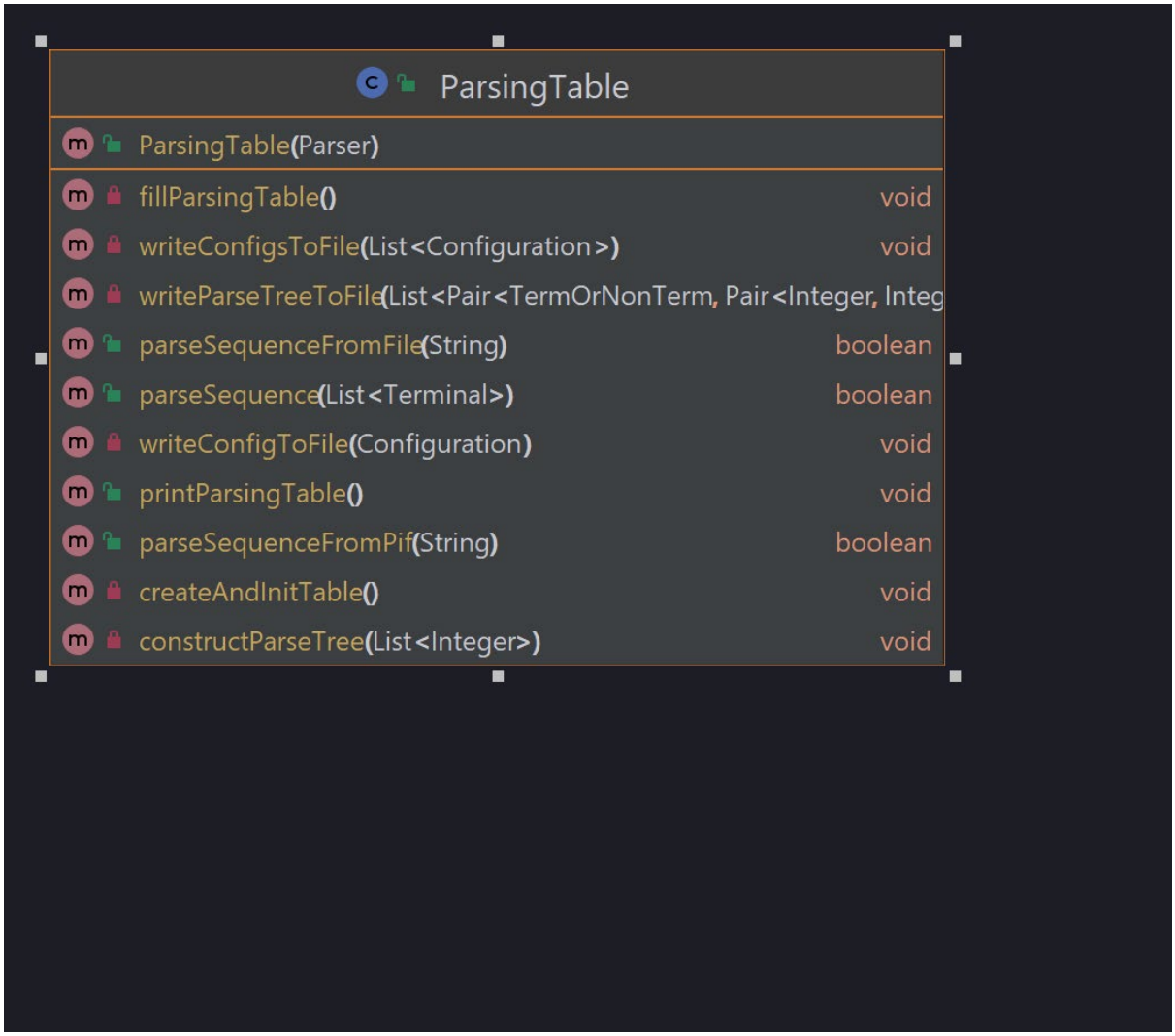
We get the initial config. Set the Boolean flag to true, and while run, print the current configuration to a file. If should push, then push, else, if should pop, then pop, else if is not accepted, set the error flag to true and set the run flag to false, and if it is accepted, just set the run flag to false. Depending on the error flag, we print whether the sequence was accepted or not, and start building the parse tree.

**Building the parse tree – parent sibling representation:**

The data structure is a list of pairs of (TermOrNonTerm, Pair<Integer,Integer>), representing that for each entry in the list, we have the TermOrNonTerm, the index of the parent and the index of the left sibling. We use the value -1 for marking an invalid index, for the case of the root and the leftmost sibling which does not have a left sibling. We first add an entry for the starting symbol with parent -1 and left sibling -1. We also have a queue with indexes, and we first add index 0 corresponding to the root. Then we take each production corresponding to the production indexes found in the parsing sequence step, we get the current parent by popping from the queue, and for each element in the rhs of the production, we add to the table with that current parent, also making sure to put the left sibling as the index of the previous element. If the current element from the rhs is a

nonterminal, we add to the queue, since those elements will be parent. Then we print the parse tree to the file.

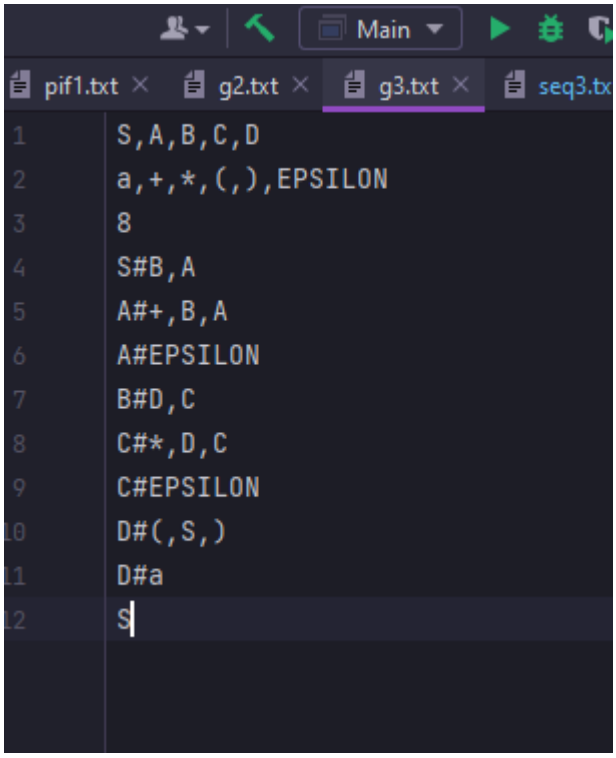


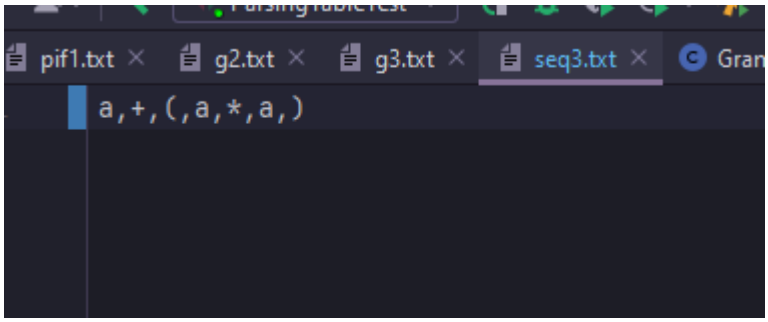


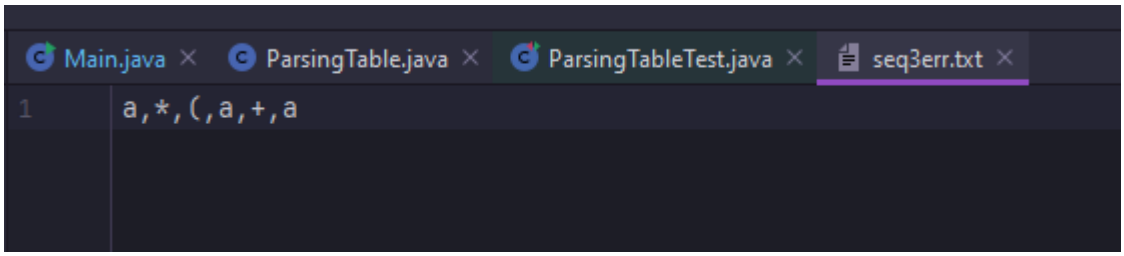

**Testing**: tests the parser accepts or not accepts different sequences from different grammars correctly

Input grammars and sequences:

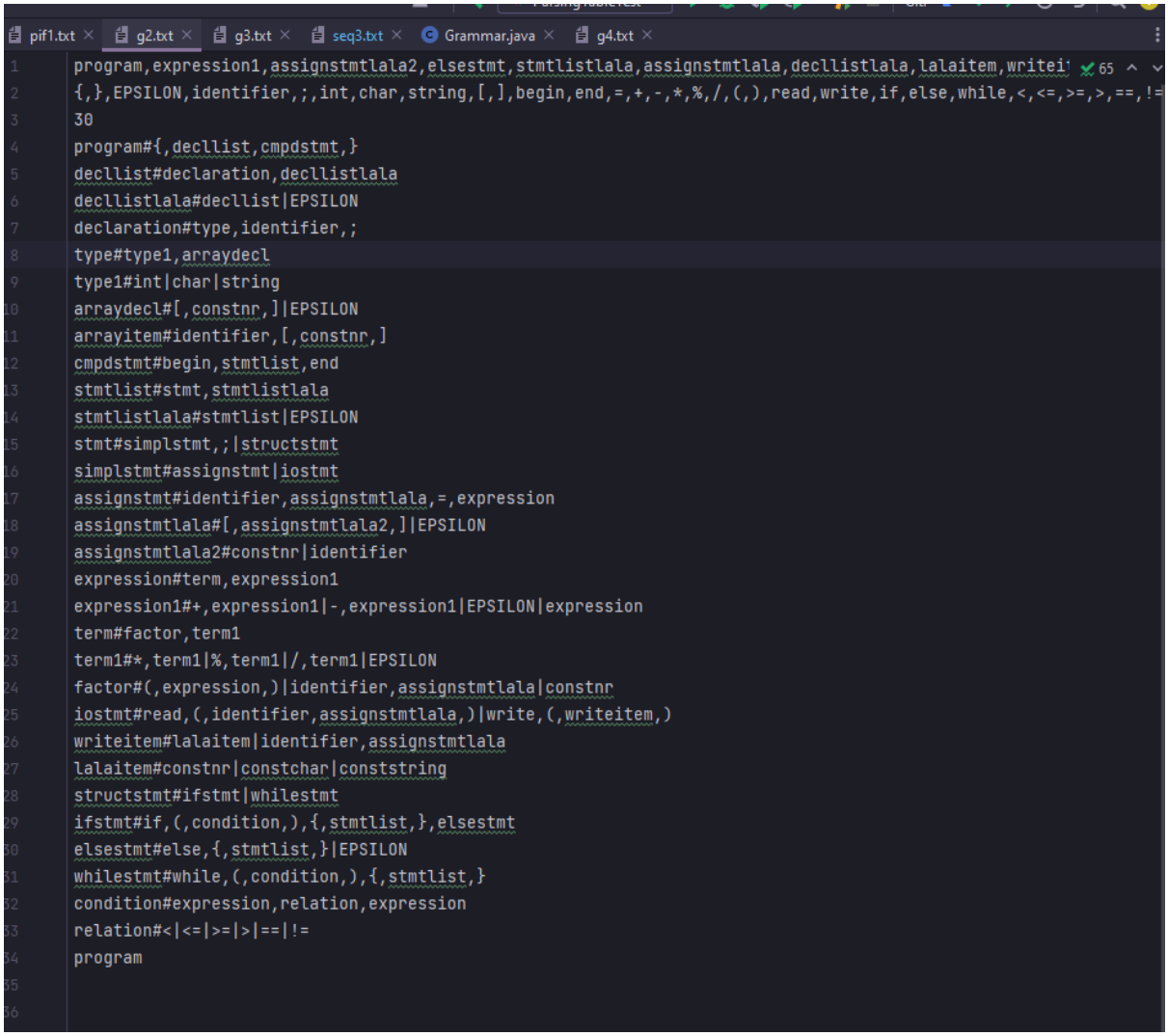
Grammar from seminary:



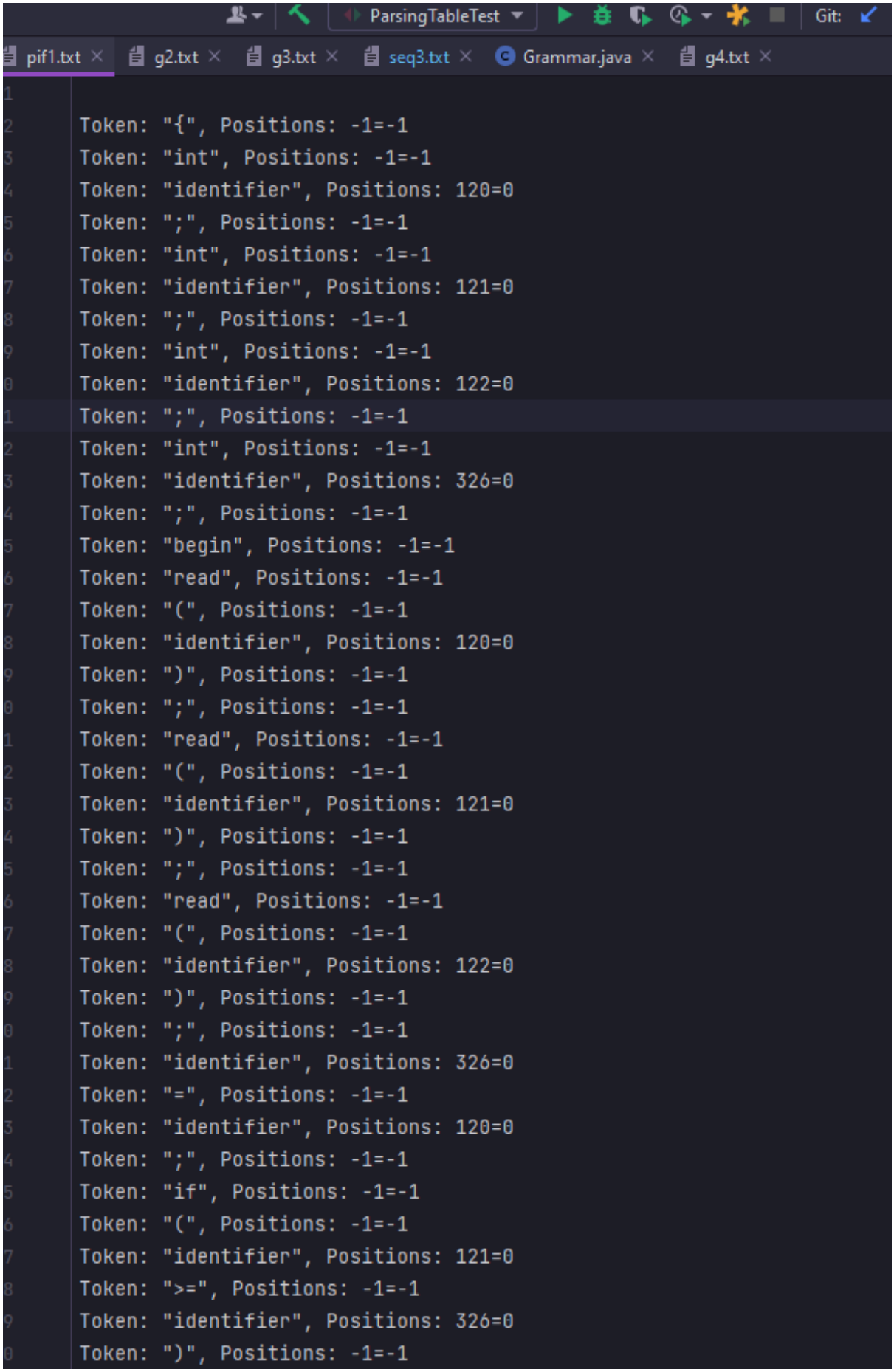




Language grammar:




Pifs corresponding to programs from lab1a (p1.txt,p2.txt,p3.txt, and multiple types of syntax errors)


ParsingTableTest
Git:
pif1.txt
g2.txt
g3.txt
seq3.txt
Grammar.java
g4.txt
Token: "{", Positions: -1=-1
Token: "int", Positions: -1=-1
Token: "identifier", Positions: 120=0
Token: ";", Positions: -1=-1
Token: "int", Positions: -1=-1
Token: "identifier", Positions: 121=0
Token: ";", Positions: -1=-1
Token: "int", Positions: -1=-1
Token: "identifier", Positions: 122=0
Token: ";", Positions: -1=-1
Token: "int", Positions: -1=-1
Token: "identifier", Positions: 326=0
Token: ";", Positions: -1=-1
Token: "begin", Positions: -1=-1
Token: "read", Positions: -1=-1
Token: "(", Positions: -1=-1
Token: "identifier", Positions: 120=0
Token: ")", Positions: -1=-1
Token: ";", Positions: -1=-1
Token: "read", Positions: -1=-1
Token: "(", Positions: -1=-1
Token: "identifier", Positions: 121=0
Token: ")", Positions: -1=-1
Token: ";", Positions: -1=-1
Token: "read", Positions: -1=-1
Token: "(", Positions: -1=-1
Token: "identifier", Positions: 122=0
Token: ")", Positions: -1=-1
Token: ";", Positions: -1=-1
Token: "identifier", Positions: 326=0
Token: "=", Positions: -1=-1
Token: "identifier", Positions: 120=0
Token: ";", Positions: -1=-1
Token: "if", Positions: -1=-1
Token: "(", Positions: -1=-1
Token: "identifier", Positions: 121=0
Token: ">=", Positions: -1=-1
Token: "identifier", Positions: 326=0
Token: ")", Positions: -1=-1

(I will put here the code for the programs, since it would be easier to check)

p1.txt

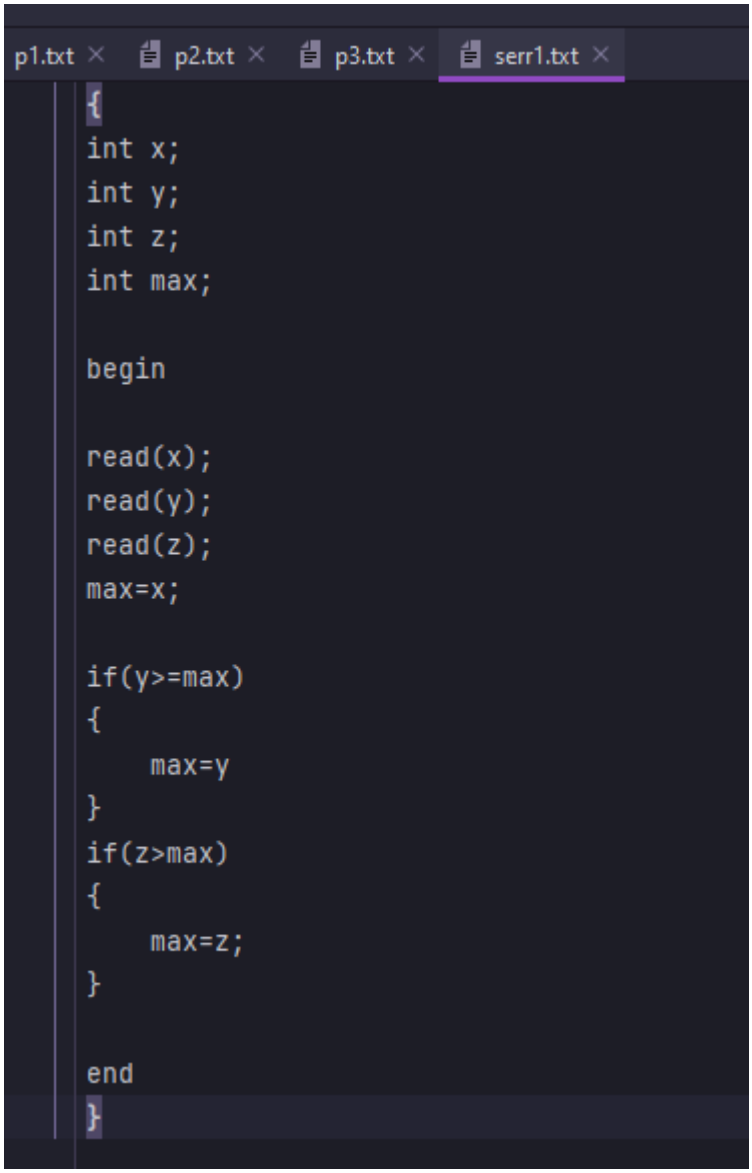
```
{
int x;
int y;
int z;
int max;

begin

read(x);
read(y);
read(z);
max=x;

if(y>=max)
{
    max=y;
}
if(z>max)
{
    max=z;
}

end
}
```

p1.txt p2.txt

```
{
int x;
int y;

begin
read(x);
read(y);
while(x!=y)
{
    if(x>y)
    {
        x=x-y;
    }
    else
    {
        y=y-x;
    }
}
write("Gcd is ");
write(x);
end
}
```

p1.txt p2.txt p3.txt

```
{
int[20] numbers;
int sum;
int index;

begin
sum=0;
index=0;
while(index<20)
{
read(numbers[index]);
index=index+1;
}

index=0;
while(index<20)
{
sum=sum+numbers[index];
index=index+1;
}

write(sum);
end
}
```


p1.txt
p2.txt
p3.txt
serr1.txt
{
int x;
int y;
int z;
int max;
begin
read(x);
read(y);
read(z);
max=x;
if(y>=max)
{
max=y
}
if(z>max)
{
max=z;
}
end
}

p1.txt p2.txt p3.txt serr1.txt serr2.txt

```
{
int x;
int y;
int z;
int max;

begin

read(x);
read(y);
read(z);
max=x;

if(y>=max)
{
    max=y;
}
if(z>max)
{
    max<=z;
}

end
}
```

p1.txt | p2.txt | p3.txt | serr1.txt | serr2.txt | serr3.txt

```
{
int[20] numbers;
int sum;
int index;

begin
sum=0;
index=0;
while(index<20)
{
read(numbers[index]);
index=index+1;
}

index=0;
while(index<20)
{
sum=sum+numbers[index];
index=index+1;
}

write(sum);
end
```

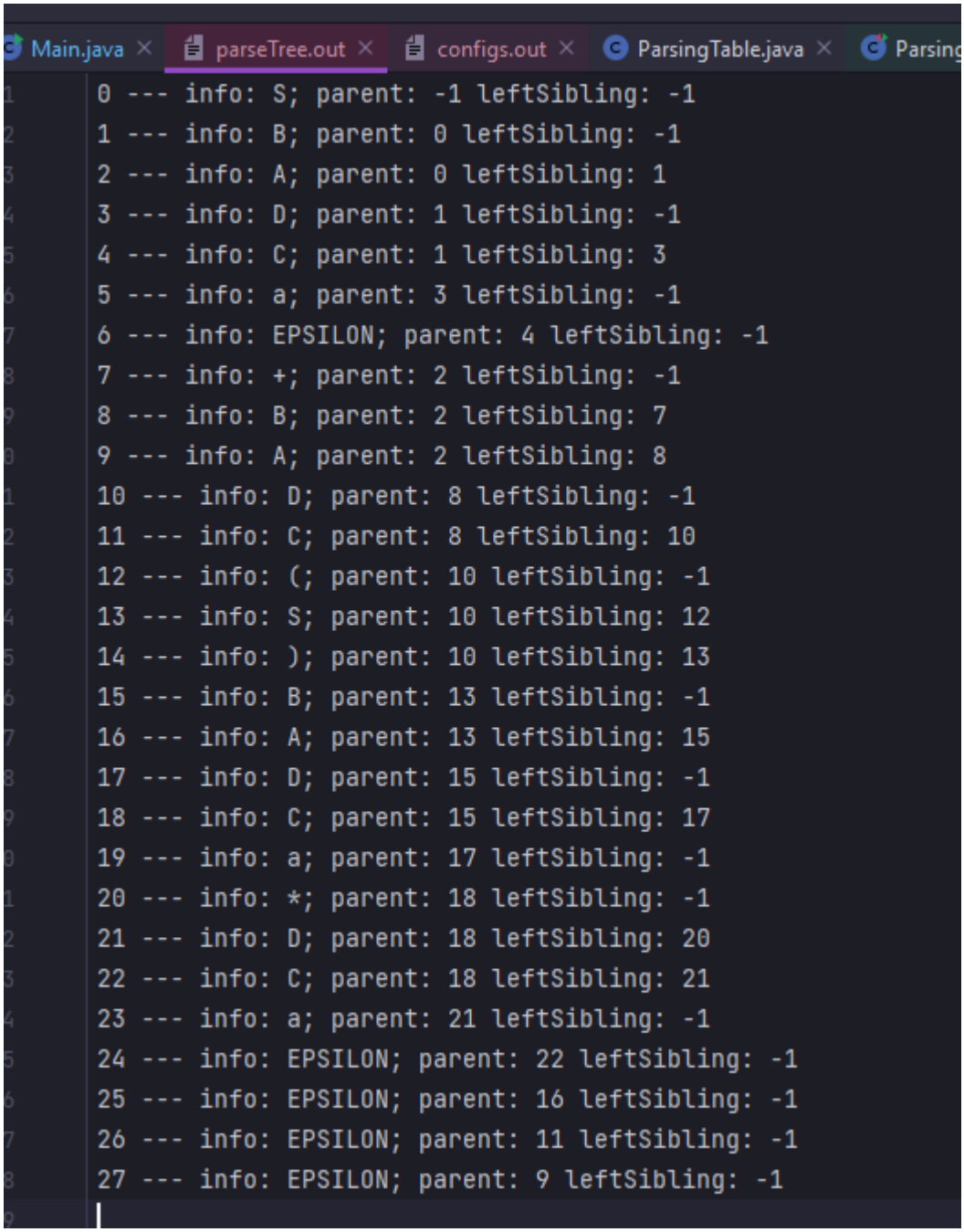
Tests:

```java
no usages  917_Zachman_Roxana
public class ParsingTableTest {

    no usages  917_Zachman_Roxana
    @Test
    public void parseSequenceFromFile() {
        Parser parser = new Parser( grammarPath: "in/g3.txt");
        ParsingTable parsingTable = new ParsingTable(parser);
        assertTrue(parsingTable.parseSequenceFromFile( path: "in/seq3.txt"));

        assertFalse(parsingTable.parseSequenceFromFile( path: "in/seq3err.txt"));
    }

    no usages  917_Zachman_Roxana
    @Test
    public void parseSequenceFromPif() {
        Parser parser = new Parser( grammarPath: "in/g2.txt");
        ParsingTable parsingTable = new ParsingTable(parser);
        assertTrue(parsingTable.parseSequenceFromPif( path: "in/pif1.txt"));
        assertTrue(parsingTable.parseSequenceFromPif( path: "in/pif2.txt"));
        assertTrue(parsingTable.parseSequenceFromPif( path: "in/pif3.txt"));

        assertFalse(parsingTable.parseSequenceFromPif( path: "in/pifserr1.txt"));
        assertFalse(parsingTable.parseSequenceFromPif( path: "in/pifserr2.txt"));
        assertFalse(parsingTable.parseSequenceFromPif( path: "in/pifserr3.txt"));
    }
}
```

For g3 and seq3 the config file and the parse tree file are:

Main.java × parseTree.out × configs.out × ParsingTable.java × ParsingTableTest.java × seq3err.txt ×

```
Configuration(productionsIndexes=[], inputStack=[a, +, (, a, *, a, ), $], workingStack=[S, $])
Configuration(productionsIndexes=[0], inputStack=[a, +, (, a, *, a, ), $], workingStack=[B, A, $])
Configuration(productionsIndexes=[0, 3], inputStack=[a, +, (, a, *, a, ), $], workingStack=[D, C, A, $])
Configuration(productionsIndexes=[0, 3, 7], inputStack=[a, +, (, a, *, a, ), $], workingStack=[a, C, A, $])
Configuration(productionsIndexes=[0, 3, 7], inputStack=[+, (, a, *, a, ), $], workingStack=[C, A, $])
Configuration(productionsIndexes=[0, 3, 7, 5], inputStack=[+, (, a, *, a, ), $], workingStack=[A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1], inputStack=[+, (, a, *, a, ), $], workingStack=[+, B, A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1], inputStack=[(, a, *, a, ), $], workingStack=[B, A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1, 3], inputStack=[(, a, *, a, ), $], workingStack=[D, C, A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1, 3, 6], inputStack=[(, a, *, a, ), $], workingStack=[(, S, ), C, A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1, 3, 6], inputStack=[a, *, a, ), $], workingStack=[S, ), C, A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1, 3, 6, 0], inputStack=[a, *, a, ), $], workingStack=[B, A, ), C, A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1, 3, 6, 0, 3], inputStack=[a, *, a, ), $], workingStack=[D, C, A, ), C, A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1, 3, 6, 0, 3, 7], inputStack=[a, *, a, ), $], workingStack=[a, C, A, ), C, A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1, 3, 6, 0, 3, 7], inputStack=[*, a, ), $], workingStack=[C, A, ), C, A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1, 3, 6, 0, 3, 7, 4], inputStack=[*, a, ), $], workingStack=[*, D, C, A, ), C, A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1, 3, 6, 0, 3, 7, 4], inputStack=[a, ), $], workingStack=[D, C, A, ), C, A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1, 3, 6, 0, 3, 7, 4, 7], inputStack=[a, ), $], workingStack=[a, C, A, ), C, A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1, 3, 6, 0, 3, 7, 4, 7], inputStack=[), $], workingStack=[C, A, ), C, A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1, 3, 6, 0, 3, 7, 4, 7, 5], inputStack=[), $], workingStack=[A, ), C, A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1, 3, 6, 0, 3, 7, 4, 7, 5, 2], inputStack=[), $], workingStack=[), C, A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1, 3, 6, 0, 3, 7, 4, 7, 5, 2], inputStack=[$], workingStack=[C, A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1, 3, 6, 0, 3, 7, 4, 7, 5, 2, 5], inputStack=[$], workingStack=[A, $])
Configuration(productionsIndexes=[0, 3, 7, 5, 1, 3, 6, 0, 3, 7, 4, 7, 5, 2, 5, 2], inputStack=[$], workingStack=[$])
Configuration(productionsIndexes=[0, 3, 7, 5, 1, 3, 6, 0, 3, 7, 4, 7, 5, 2, 5, 2], inputStack=[$], workingStack=[$])
```


Main.java
parseTree.out
configs.out
ParsingTable.java
0 --- info: S; parent: -1 leftSibling: -1
1 --- info: B; parent: 0 leftSibling: -1
2 --- info: A; parent: 0 leftSibling: 1
3 --- info: D; parent: 1 leftSibling: -1
4 --- info: C; parent: 1 leftSibling: 3
5 --- info: a; parent: 3 leftSibling: -1
6 --- info: EPSILON; parent: 4 leftSibling: -1
7 --- info: +; parent: 2 leftSibling: -1
8 --- info: B; parent: 2 leftSibling: 7
9 --- info: A; parent: 2 leftSibling: 8
10 --- info: D; parent: 8 leftSibling: -1
11 --- info: C; parent: 8 leftSibling: 10
12 --- info: (; parent: 10 leftSibling: -1
13 --- info: S; parent: 10 leftSibling: 12
14 --- info: ); parent: 10 leftSibling: 13
15 --- info: B; parent: 13 leftSibling: -1
16 --- info: A; parent: 13 leftSibling: 15
17 --- info: D; parent: 15 leftSibling: -1
18 --- info: C; parent: 15 leftSibling: 17
19 --- info: a; parent: 17 leftSibling: -1
20 --- info: *; parent: 18 leftSibling: -1
21 --- info: D; parent: 18 leftSibling: 20
22 --- info: C; parent: 18 leftSibling: 21
23 --- info: a; parent: 21 leftSibling: -1
24 --- info: EPSILON; parent: 22 leftSibling: -1
25 --- info: EPSILON; parent: 16 leftSibling: -1
26 --- info: EPSILON; parent: 11 leftSibling: -1
27 --- info: EPSILON; parent: 9 leftSibling: -1