Lab 5 - Parser Algorithm

Korodi Andreea-Cristina
Georgina Loba

Git link: https://github.com/GeorgianaLoba/FLCD---but-teamed

Assignment for a team of 2 students!

Statement: Implement a parser algorithm

Input: g1.txt seq.tx

g2.txt, PIF.out (result of lab 3)

Output: out1.txt, out2.txt

One of the following parsing methods will be chosen (assigned by teaching staff):

1.a. recursive descendent

The representation of the parsing tree (output) will be (decided by the team):

2.c. table (using father and sibling relation) (max grade = 10)

PART 1: Deliverables

1. Class grammar (required operations: read a grammar from file, print set of nonterminals, set of terminals, set of productions, production for a given nonterminal)
2. Input file: g1.txt (grammar from seminar); g2.txt (grammar of the minilanguage; syntax rules from Lab1)
3. Functions corresponding to parsing strategy (see table below)

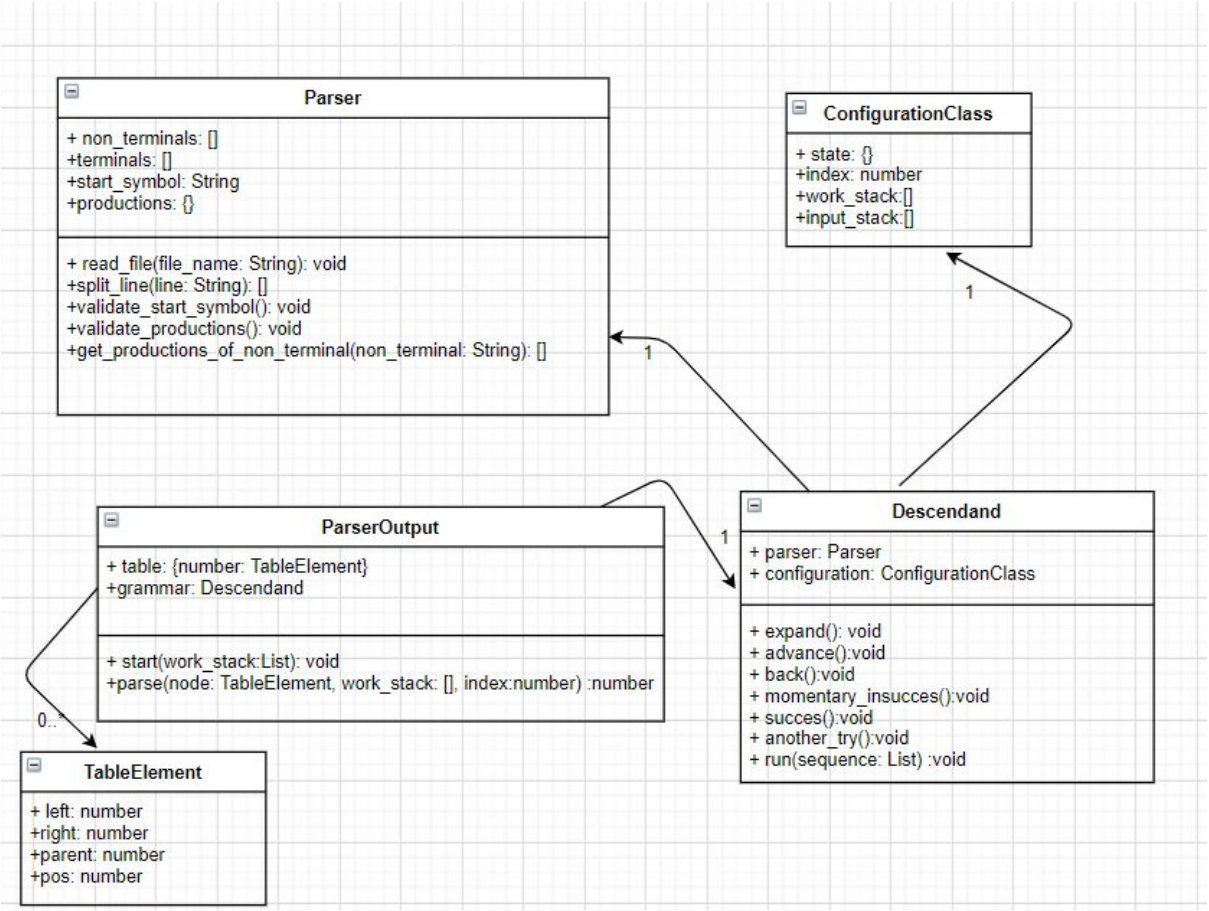
PART 2: Deliverables

1. Algorithm corresponding to parsing tables (if needed) and parsing strategy
2. Class ParserOutput - DS and operations corresponding to choice 2.a/2.b/2.c (required operations: transform parsing tree into representation; print DS to screen and to file)

PART 3: Deliverables

1. Source code
2. 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)
3. Code review

Class diagram:



1. **Proposed Solution**

   Parsing - Parser Class

- Read line by line the given file;
- Functions to validate productions and starting symbol;
- Function for getting productions of a given non-terminal;

  Descendand - Descendant Class

- Functions expand, advance, back, momentary_insuccess, success, another_try checks the conditions and circumstances presented in the lecture notes;
- The run function will parse the sequence list, character by character and call the required functions given certain conditions;

Parsing Output - ParserOutput Class

- In the beginning, we set the root in the table dictionary. The dictionary has form key: Integer and val: TableElement which represent our tree nodes;
- Recursively calling the parse function on non-terminals present in the sequence;

2. **File Structures**

File.in:

File.in has the following structure:

First line: non_terminals
Second line: terminals
Third line: start_symbol
Starting from the fourth line we have the productions.

S Aa
a b
S
S - a Aa
A - a Aa | b Aa | a | b

Grammar.in:

First line: the non-terminals
Second line: terminals
Third line: start symbol
Starting from the fourth line we have productions

----
program type statement_list statement simple_declaration assigned_declaration assignment function_arguments if_statement while_statement condition expression operand output_statement input_statement array relation

+ - * / % = == != is and or true false < > <= >= { } ; ( ) , let func returns or and print while return if else then integer boolean array scan id cst

program

program - let func id function_arguments returns type { statement_list } ;
type - integer
statement_list - statement | statement statement_list
statement - simple_declaration | assigned_declaration | assignment | if_statement | while_statement | output_statement | input_statement
simple_declaration - let type id ;
assigned_declaration - let id = expression ;
assignment - id = expression ;
function_arguments - (type id) | (type id , function_arguments)
if_statement - if condition then { statement_list } | if condition then { statement_list } else { statement_list }
while_statement - while condition then { statement_list }
condition - (expression relation expression)
expression - cst | id | id operand cst | cst operand id | id operand id | (expression) operand (expression) | id operand (expression) | (expression) operand id
operand - + | * | / | %
output_statement - print id ; | print cst ;
input_statement - scan type id ;
array - id [integer] | id [id]
relation - = | < | > | <= | >= | is | !=

---

our.in

```
let func check_max(integer a) returns integer {
   print 2;
   print a;
   let integer d;
   scan integer d;
   d=8;
   if (d > 3) then { print a; } else { print d; }
   while ( d < 30) then {
      print d;
      d=d+10;
   }
};
```

PIF.out

On each line we have first the terminal which will be a part of the sequence then where it is found on the symbol table. We split after the “on” word and take the first word from there and then add it to a list which will be our sequence.

---

let on symbol table: (-1;-1)
func on symbol table: (-1;-1)
id on symbol table: (9;0)
(on symbol table: (-1;-1)
integer on symbol table: (-1;-1)
id on symbol table: (9;1)
) on symbol table: (-1;-1)
returns on symbol table: (-1;-1)
integer on symbol table: (-1;-1)
{ on symbol table: (-1;-1)
print on symbol table: (-1;-1)
cst on symbol table: (2;0)
; on symbol table: (-1;-1)
print on symbol table: (-1;-1)
id on symbol table: (9;1)
; on symbol table: (-1;-1)
let on symbol table: (-1;-1)
integer on symbol table: (-1;-1)
id on symbol table: (2;1)
; on symbol table: (-1;-1)
scan on symbol table: (-1;-1)
integer on symbol table: (-1;-1)
id on symbol table: (2;1)
; on symbol table: (-1;-1)
id on symbol table: (2;1)
= on symbol table: (-1;-1)
cst on symbol table: (8;0)
; on symbol table: (-1;-1)
if on symbol table: (-1;-1)
(on symbol table: (-1;-1)
id on symbol table: (2;1)
> on symbol table: (-1;-1)
cst on symbol table: (3;0)
) on symbol table: (-1;-1)
then on symbol table: (-1;-1)
{ on symbol table: (-1;-1)
print on symbol table: (-1;-1)
id on symbol table: (9;1)
; on symbol table: (-1;-1)
} on symbol table: (-1;-1)
else on symbol table: (-1;-1)
{ on symbol table: (-1;-1)
print on symbol table: (-1;-1)
id on symbol table: (2;1)
; on symbol table: (-1;-1)
} on symbol table: (-1;-1)
while on symbol table: (-1;-1)
(on symbol table: (-1;-1)

id on symbol table: (2;1)
< on symbol table: (-1;-1)
cst on symbol table: (3;1)
) on symbol table: (-1;-1)
then on symbol table: (-1;-1)
{ on symbol table: (-1;-1)
print on symbol table: (-1;-1)
id on symbol table: (2;1)
; on symbol table: (-1;-1)
id on symbol table: (2;1)
= on symbol table: (-1;-1)
id on symbol table: (2;1)
+ on symbol table: (-1;-1)
cst on symbol table: (1;0)
; on symbol table: (-1;-1)
} on symbol table: (-1;-1)
} on symbol table: (-1;-1)
; on symbol table: (-1;-1)
