
-
Github
https://github.com/Arnold-21/Compiler

-
Parser

The parser here used is the recursive descent, which is a backtracking based parsing algorithm

The approach is is we expend nonterminals, and put the results in a stack
Rince and Repeat until either we find a terminal which matches the current element in the input, in which case we move forward with the input position
Or we find a terminal which doesnt match the current element, in which case we change states

In the other states we have to option
If we find a nonterminal on top of working stack, we try to find another production for it, if not found we mo ve it back to the inputstack and move forward
if we find a terminal on top of the working stack, we move backwards in the position

There are two special cases one for transforming space (terminal) to the space character when checking t he input
And the epsilon terminal character, in which case we do the advance or back functions without position ch ange of the input

----
-
ParserOutput

The parser output is table with father and sibling relations stored in a vector, to have the tree relationship

It is derived from the working stack of the parser, with depth first seach of the productions
It starts with the starting symbol and searches the productions for values while checking from the working stack the production number of the given non terminal
and checking if the terminals are correct

-
Grammar.H

```
class Grammar{
private:
    void readFromFile(std::ifstream& in);
    void readLine(std::ifstream& in, std::vector<std::string>& vec);
    void readProductions(std::ifstream& in);

    std::vector<std::string> nonTerminals;
    std::vector<std::string> terminals;
    std::map<std::vector<std::string>, std::vector<std::vector<std::string>> > productions;
    std::string startingSymbol;

public:
    Grammar(std::string fileName);
    void printNonTerminals();
    void printTerminals();
    void printProductions();
    void printProductions(std::string symbol);
    void printVector(std::vector<std::string> vec);
    bool checkCFG();

};
```

-
Grammar.cpp

Stores the grammar variables in a vector, and the transformations in a map for easier search

```
Grammar::Grammar(std::string fileName){
    //Read from file
    std::ifstream in(fileName);
    this->readFromFile(in);
    in.close();
}

void Grammar::readFromFile(std::ifstream& in){
    //Read every neccessary info from file
    this->readLine(in, this->nonTerminals);
    this->readLine(in, this->terminals);
    in >> this->startingSymbol;
    this->readProductions(in);
}

void Grammar::readLine(std::ifstream& in, std::vector<std::string>& vec){
    //Getting a line from the file, than reading the words from it
    std::string line;
    std::getline(in, line);
    std::stringstream buffer(line);
    std::string varString;
    while (buffer >> varString){ vec.push_back(varString); }
}

void Grammar::readProductions(std::ifstream& in){
    //Reading the elements line by line than building up the productions
    std::string line;
    while(std::getline(in, line)){
        //Handle empty line
        if (line.size() == 1) continue;

        std::stringstream buffer(line);
        std::string varString;
        std::vector<std::string> leftHandSide;
        std::vector<std::string> rightHandSide;
        bool rightHand = false;

        //Reading the words in the list
        while (buffer >> varString){
            if (varString == "->"){
                rightHand = true;
                continue;
            }
            if (rightHand){
                rightHandSide.push_back(varString);
                continue;
            }
            leftHandSide.push_back(varString);
        }

        //Adding the production to the object
        if (this->productions.find(leftHandSide) == this->productions.end()){
            std::vector<std::vector<std::string>> mapRightHandSide;
            mapRightHandSide.push_back(rightHandSide);
            this->productions.insert({leftHandSide, mapRightHandSide});
        }else{
            this->productions.at(leftHandSide).push_back(rightHandSide);
        }
    }
}

void Grammar::printVector(std::vector<std::string> vec){
    std::cout << "{";
    for (auto i:vec){
        std::cout << i << ",";
    }
    std::cout << "}";
}

void Grammar::printNonTerminals(){ std::cout << std::endl; this->printVector(this->nonTerminals); std::cou
t << std::endl; }

void Grammar::printTerminals(){ std::cout << std::endl; this->printVector(this->terminals); std::cout << std::
endl; }

void Grammar::printProductions(){
    std::cout << std::endl;
    for(const auto& elem : this->productions)
    {
        std::cout << std::endl;
        this->printVector(elem.first);
        std::cout << " -> ";
        for (auto i: elem.second){
            this->printVector(i);
            std::cout << ", ";
        }
    }
    std::cout << std::endl;
}

void Grammar::printProductions(std::string symbol){
    std::cout << std::endl;
    for(const auto& elem : this->productions)
    {
        if (std::find(elem.first.begin(), elem.first.end(), symbol) == elem.first.end()) continue;
        std::cout << std::endl;
        this->printVector(elem.first);
        std::cout << " -> ";
        for (auto i: elem.second){
            this->printVector(i);
            std::cout << ", ";
        }
    }
    std::cout << std::endl;
}

bool Grammar::checkCFG(){
    //Check if every left hand contains only one symbol
    for(const auto& elem : this->productions)
    {
        if (elem.first.size() != 1) return false;
    }
    return true;
}
```

-
g1.txt

```
S A
a b c
S
S -> a A
A -> b A
A -> c
```

-
g1Out.txt

```
0 - S(0) - p: -1 - s: -1
1 - a(-1) - p: 0 - s: -1
2 - A(0) - p: 0 - s: 1
3 - b(-1) - p: 2 - s: -1
4 - A(1) - p: 2 - s: 3
5 - c(-1) - p: 4 - s: -1
```

-
g2.txt

```
optionalsign simpleintexpression simpleboolexpression simplestringexpression program declaration state
ment simpledeclaration arraydeclaration type identifierlist identifier expression intexpression boolexpressi
on charexpression stringexpression intvalue boolvalue charvalue stringvalue positivint simpleidentifierlist
non_zero_digit digit assignstatement ifstatement whilestatement functionstatement functionname expressi
onlist digitlist letter lastofidentifier insideString
+ - * / % && || ! = == < <= > >= { } ( ) [ ] ; space newline " , ’ int bool char string if else while print readInt re
adString array true false set get epsilon a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J
K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9
program
```

```
program -> declaration statement
declaration -> simpledeclaration ; declaration
declaration -> arraydeclaration ; declaration
declaration -> epsilon
simpledeclaration -> type identifierlist
type -> int
type -> bool
type -> char
type -> string
identifierlist -> identifier
identifierlist -> identifier = expression
identifierlist -> identifier , identifierlist
identifierlist -> identifier = expression , identifierlist
expression -> intexpression
expression -> boolexpression
expression -> charexpression
expression -> stringexpression
simpleintexpression -> intvalue
simpleintexpression -> identifier
intexpression -> simpleintexpression
intexpression -> ( simpleintexpression * intexpression )
intexpression -> ( simpleintexpression / intexpression )
intexpression -> ( simpleintexpression % intexpression )
intexpression -> ( simpleintexpression + intexpression )
intexpression -> ( simpleintexpression - intexpression )
intexpression -> simpleintexpression * intexpression
intexpression -> simpleintexpression / intexpression
intexpression -> simpleintexpression % intexpression
intexpression -> simpleintexpression + intexpression
intexpression -> simpleintexpression - intexpression
simpleboolexpression -> boolvalue
simpleboolexpression -> ! identifier
simpleboolexpression -> identifier
boolexpression -> simpleboolexpression
boolexpression -> ( simpleboolexpression && boolexpression )
boolexpression -> ( simpleboolexpression || boolexpression )
boolexpression -> ( simpleboolexpression == boolexpression )
boolexpression -> ( intexpression == intexpression )
boolexpression -> ( intexpression < intexpression )
boolexpression -> ( intexpression <= intexpression )
boolexpression -> ( intexpression > intexpression )
boolexpression -> ( intexpression >= intexpression )
boolexpression -> simpleboolexpression && boolexpression
boolexpression -> simpleboolexpression || boolexpression
boolexpression -> simpleboolexpression == boolexpression
boolexpression -> intexpression == intexpression
boolexpression -> intexpression < intexpression
boolexpression -> intexpression <= intexpression
boolexpression -> intexpression > intexpression
boolexpression -> intexpression >= intexpression
charexpression -> charvalue
charexpression -> identifier
simplestringexpression -> stringvalue
simplestringexpression -> identifier
stringexpression -> simplestringexpression
stringexpression -> ( simplestringexpression + stringexpression )
stringexpression -> simplestringexpression + stringexpression
arraydeclaration -> array [ type ] [ positivint ] simpleidentifierlist
positivint -> non_zero_digit digitlist
digitlist -> digit digitlist
digitlist -> epsilon
simpleidentifierlist -> identifier
simpleidentifierlist -> identifier, simpleidentifierlist
statement -> assignstatement ; statement
statement -> ifstatement statement
statement -> whilestatement statement
statement -> functionstatement ; statement
statement -> epsilon
assignstatement -> identifier = expression
assignstatement -> identifier = functionstatement
ifstatement -> if ( boolexpression ) { statement }
ifstatement -> if ( boolexpression ) { statement } else { statement }
whilestatement -> while ( boolexpression ) { statement }
functionstatement -> functionname ( expressionlist )
functionstatement -> functionname ( )
functionname -> readInt
functionname -> readString
functionname -> get
functionname -> set
functionname -> print
expressionlist -> expression
expressionlist -> expression , expressionlist
letter -> A
letter -> B
letter -> C
letter -> D
letter -> E
letter -> F
letter -> G
letter -> H
letter -> I
letter -> J
letter -> K
letter -> L
letter -> M
letter -> N
letter -> O
letter -> P
letter -> Q
letter -> R
letter -> S
letter -> T
letter -> U
letter -> V
letter -> W
letter -> X
letter -> Y
letter -> Z
letter -> a
letter -> b
```

letter -> c
letter -> d
letter -> e
letter -> f
letter -> g
letter -> h
letter -> i
letter -> j
letter -> k
letter -> l
letter -> m
letter -> n
letter -> o
letter -> p
letter -> q
letter -> r
letter -> s
letter -> t
letter -> u
letter -> v
letter -> w
letter -> x
letter -> y
letter -> z
digit -> 0
digit -> 1
digit -> 2
digit -> 3
digit -> 4
digit -> 5
digit -> 6
digit -> 7
digit -> 8
digit -> 9
non_zero_digit -> 1
non_zero_digit -> 2
non_zero_digit -> 3
non_zero_digit -> 4
non_zero_digit -> 5
non_zero_digit -> 6
non_zero_digit -> 7
non_zero_digit -> 8
non_zero_digit -> 9
identifier -> letter lastofidentifier
lastofidentifier -> letter lastofidentifier
lastofidentifier -> digit lastofidentifier
lastofidentifier -> epsilon
boolvalue -> true
boolvalue -> false
charvalue -> ’ letter ’
charvalue -> ’ digit ’
intvalue -> optionalsign positivint
intvalue -> 0
optionalsign -> +
optionalsign -> -

```
optionalsign -> epsilon
stringvalue -> " insideString "
insideString -> space insideString
insideString -> letter insideString
insideString -> digit insideString
insideString -> epsilon
```

-

g2Out.txt

```
0 - program(0) - p: -1 - s: -1
1 - declaration(0) - p: 0 - s: -1
2 - statement(2) - p: 0 - s: 1
3 - simpledeclaration(0) - p: 1 - s: -1
4 - ;(-1) - p: 1 - s: 3
5 - declaration(0) - p: 1 - s: 4
6 - type(0) - p: 3 - s: -1
7 - identifierlist(1) - p: 3 - s: 6
8 - int(-1) - p: 6 - s: -1
9 - identifier(0) - p: 7 - s: -1
10 - =(-1) - p: 7 - s: 9
11 - expression(0) - p: 7 - s: 10
12 - letter(49) - p: 9 - s: -1
13 - lastofidentifier(2) - p: 9 - s: 12
14 - x(-1) - p: 12 - s: -1
15 - epsilon(-1) - p: 13 - s: -1
16 - intexpression(0) - p: 11 - s: -1
17 - simpleintexpression(0) - p: 16 - s: -1
18 - intvalue(0) - p: 17 - s: -1
19 - optionalsign(2) - p: 18 - s: -1
20 - positivint(0) - p: 18 - s: 19
21 - epsilon(-1) - p: 19 - s: -1
22 - non_zero_digit(0) - p: 20 - s: -1
23 - digitlist(0) - p: 20 - s: 22
24 - 1(-1) - p: 22 - s: -1
25 - digit(2) - p: 23 - s: -1
26 - digitlist(1) - p: 23 - s: 25
27 - 2(-1) - p: 25 - s: -1
28 - epsilon(-1) - p: 26 - s: -1
29 - simpledeclaration(0) - p: 5 - s: -1
30 - ;(-1) - p: 5 - s: 29
31 - declaration(0) - p: 5 - s: 30
32 - type(0) - p: 29 - s: -1
33 - identifierlist(1) - p: 29 - s: 32
34 - int(-1) - p: 32 - s: -1
35 - identifier(0) - p: 33 - s: -1
36 - =(-1) - p: 33 - s: 35
37 - expression(0) - p: 33 - s: 36
38 - letter(34) - p: 35 - s: -1
39 - lastofidentifier(2) - p: 35 - s: 38
40 - i(-1) - p: 38 - s: -1
41 - epsilon(-1) - p: 39 - s: -1
42 - intexpression(0) - p: 37 - s: -1
43 - simpleintexpression(0) - p: 42 - s: -1
```

```
44 - intvalue(0) - p: 43 - s: -1
45 - optionalsign(2) - p: 44 - s: -1
46 - positivint(0) - p: 44 - s: 45
47 - epsilon(-1) - p: 45 - s: -1
48 - non_zero_digit(1) - p: 46 - s: -1
49 - digitlist(1) - p: 46 - s: 48
50 - 2(-1) - p: 48 - s: -1
51 - epsilon(-1) - p: 49 - s: -1
52 - simpledeclaration(0) - p: 31 - s: -1
53 - ;(-1) - p: 31 - s: 52
54 - declaration(2) - p: 31 - s: 53
55 - type(1) - p: 52 - s: -1
56 - identifierlist(1) - p: 52 - s: 55
57 - bool(-1) - p: 55 - s: -1
58 - identifier(0) - p: 56 - s: -1
59 - =(-1) - p: 56 - s: 58
60 - expression(1) - p: 56 - s: 59
61 - letter(41) - p: 58 - s: -1
62 - lastofidentifier(0) - p: 58 - s: 61
63 - p(-1) - p: 61 - s: -1
64 - letter(43) - p: 62 - s: -1
65 - lastofidentifier(0) - p: 62 - s: 64
66 - r(-1) - p: 64 - s: -1
67 - letter(34) - p: 65 - s: -1
68 - lastofidentifier(0) - p: 65 - s: 67
69 - i(-1) - p: 67 - s: -1
70 - letter(38) - p: 68 - s: -1
71 - lastofidentifier(0) - p: 68 - s: 70
72 - m(-1) - p: 70 - s: -1
73 - letter(30) - p: 71 - s: -1
74 - lastofidentifier(2) - p: 71 - s: 73
75 - e(-1) - p: 73 - s: -1
76 - epsilon(-1) - p: 74 - s: -1
77 - boolexpression(0) - p: 60 - s: -1
78 - simpleboolexpression(0) - p: 77 - s: -1
79 - boolvalue(0) - p: 78 - s: -1
80 - true(-1) - p: 79 - s: -1
81 - epsilon(-1) - p: 54 - s: -1
82 - whilestatement(0) - p: 2 - s: -1
83 - statement(3) - p: 2 - s: 82
84 - while(-1) - p: 82 - s: -1
85 - ((-1) - p: 82 - s: 84
86 - boolexpression(9) - p: 82 - s: 85
87 - )(-1) - p: 82 - s: 86
88 - {(-1) - p: 82 - s: 87
89 - statement(1) - p: 82 - s: 88
90 - }(-1) - p: 82 - s: 89
91 - simpleboolexpression(2) - p: 86 - s: -1
92 - &&(-1) - p: 86 - s: 91
93 - boolexpression(14) - p: 86 - s: 92
94 - identifier(0) - p: 91 - s: -1
95 - letter(41) - p: 94 - s: -1
96 - lastofidentifier(0) - p: 94 - s: 95
97 - p(-1) - p: 95 - s: -1
98 - letter(43) - p: 96 - s: -1
```

```
99 - lastofidentifier(0) - p: 96 - s: 98
100 - r(-1) - p: 98 - s: -1
101 - letter(34) - p: 99 - s: -1
102 - lastofidentifier(0) - p: 99 - s: 101
103 - i(-1) - p: 101 - s: -1
104 - letter(38) - p: 102 - s: -1
105 - lastofidentifier(0) - p: 102 - s: 104
106 - m(-1) - p: 104 - s: -1
107 - letter(30) - p: 105 - s: -1
108 - lastofidentifier(2) - p: 105 - s: 107
109 - e(-1) - p: 107 - s: -1
110 - epsilon(-1) - p: 108 - s: -1
111 - intexpression(6) - p: 93 - s: -1
112 - <=(-1) - p: 93 - s: 111
113 - intexpression(0) - p: 93 - s: 112
114 - simpleintexpression(1) - p: 111 - s: -1
115 - *(-1) - p: 111 - s: 114
116 - intexpression(0) - p: 111 - s: 115
117 - identifier(0) - p: 114 - s: -1
118 - letter(34) - p: 117 - s: -1
119 - lastofidentifier(2) - p: 117 - s: 118
120 - i(-1) - p: 118 - s: -1
121 - epsilon(-1) - p: 119 - s: -1
122 - simpleintexpression(1) - p: 116 - s: -1
123 - identifier(0) - p: 122 - s: -1
124 - letter(34) - p: 123 - s: -1
125 - lastofidentifier(2) - p: 123 - s: 124
126 - i(-1) - p: 124 - s: -1
127 - epsilon(-1) - p: 125 - s: -1
128 - simpleintexpression(1) - p: 113 - s: -1
129 - identifier(0) - p: 128 - s: -1
130 - letter(49) - p: 129 - s: -1
131 - lastofidentifier(2) - p: 129 - s: 130
132 - x(-1) - p: 130 - s: -1
133 - epsilon(-1) - p: 131 - s: -1
134 - ifstatement(0) - p: 89 - s: -1
135 - statement(0) - p: 89 - s: 134
136 - if(-1) - p: 134 - s: -1
137 - ((-1) - p: 134 - s: 136
138 - boolexpression(12) - p: 134 - s: 137
139 - )(-1) - p: 134 - s: 138
140 - {(-1) - p: 134 - s: 139
141 - statement(0) - p: 134 - s: 140
142 - }(-1) - p: 134 - s: 141
143 - intexpression(8) - p: 138 - s: -1
144 - ==(-1) - p: 138 - s: 143
145 - intexpression(0) - p: 138 - s: 144
146 - simpleintexpression(1) - p: 143 - s: -1
147 - %(-1) - p: 143 - s: 146
148 - intexpression(0) - p: 143 - s: 147
149 - identifier(0) - p: 146 - s: -1
150 - letter(49) - p: 149 - s: -1
151 - lastofidentifier(2) - p: 149 - s: 150
152 - x(-1) - p: 150 - s: -1
153 - epsilon(-1) - p: 151 - s: -1
```

```
154 - simpleintexpression(1) - p: 148 - s: -1
155 - identifier(0) - p: 154 - s: -1
156 - letter(34) - p: 155 - s: -1
157 - lastofidentifier(2) - p: 155 - s: 156
158 - i(-1) - p: 156 - s: -1
159 - epsilon(-1) - p: 157 - s: -1
160 - simpleintexpression(0) - p: 145 - s: -1
161 - intvalue(1) - p: 160 - s: -1
162 - 0(-1) - p: 161 - s: -1
163 - assignstatement(0) - p: 141 - s: -1
164 - ;(-1) - p: 141 - s: 163
165 - statement(4) - p: 141 - s: 164
166 - identifier(0) - p: 163 - s: -1
167 - =(-1) - p: 163 - s: 166
168 - expression(1) - p: 163 - s: 167
169 - letter(41) - p: 166 - s: -1
170 - lastofidentifier(0) - p: 166 - s: 169
171 - p(-1) - p: 169 - s: -1
172 - letter(43) - p: 170 - s: -1
173 - lastofidentifier(0) - p: 170 - s: 172
174 - r(-1) - p: 172 - s: -1
175 - letter(34) - p: 173 - s: -1
176 - lastofidentifier(0) - p: 173 - s: 175
177 - i(-1) - p: 175 - s: -1
178 - letter(38) - p: 176 - s: -1
179 - lastofidentifier(0) - p: 176 - s: 178
180 - m(-1) - p: 178 - s: -1
181 - letter(30) - p: 179 - s: -1
182 - lastofidentifier(2) - p: 179 - s: 181
183 - e(-1) - p: 181 - s: -1
184 - epsilon(-1) - p: 182 - s: -1
185 - boolexpression(0) - p: 168 - s: -1
186 - simpleboolexpression(0) - p: 185 - s: -1
187 - boolvalue(1) - p: 186 - s: -1
188 - false(-1) - p: 187 - s: -1
189 - epsilon(-1) - p: 165 - s: -1
190 - assignstatement(0) - p: 135 - s: -1
191 - ;(-1) - p: 135 - s: 190
192 - statement(4) - p: 135 - s: 191
193 - identifier(0) - p: 190 - s: -1
194 - =(-1) - p: 190 - s: 193
195 - expression(0) - p: 190 - s: 194
196 - letter(34) - p: 193 - s: -1
197 - lastofidentifier(2) - p: 193 - s: 196
198 - i(-1) - p: 196 - s: -1
199 - epsilon(-1) - p: 197 - s: -1
200 - intexpression(9) - p: 195 - s: -1
201 - simpleintexpression(1) - p: 200 - s: -1
202 - +(-1) - p: 200 - s: 201
203 - intexpression(0) - p: 200 - s: 202
204 - identifier(0) - p: 201 - s: -1
205 - letter(34) - p: 204 - s: -1
206 - lastofidentifier(2) - p: 204 - s: 205
207 - i(-1) - p: 205 - s: -1
208 - epsilon(-1) - p: 206 - s: -1
```

```
209 - simpleintexpression(0) - p: 203 - s: -1
210 - intvalue(0) - p: 209 - s: -1
211 - optionalsign(2) - p: 210 - s: -1
212 - positivint(0) - p: 210 - s: 211
213 - epsilon(-1) - p: 211 - s: -1
214 - non_zero_digit(0) - p: 212 - s: -1
215 - digitlist(1) - p: 212 - s: 214
216 - 1(-1) - p: 214 - s: -1
217 - epsilon(-1) - p: 215 - s: -1
218 - epsilon(-1) - p: 192 - s: -1
219 - functionstatement(0) - p: 83 - s: -1
220 - ;(-1) - p: 83 - s: 219
221 - statement(4) - p: 83 - s: 220
222 - functionname(4) - p: 219 - s: -1
223 - ((-1) - p: 219 - s: 222
224 - expressionlist(0) - p: 219 - s: 223
225 - )(-1) - p: 219 - s: 224
226 - print(-1) - p: 222 - s: -1
227 - expression(0) - p: 224 - s: -1
228 - intexpression(0) - p: 227 - s: -1
229 - simpleintexpression(1) - p: 228 - s: -1
230 - identifier(0) - p: 229 - s: -1
231 - letter(41) - p: 230 - s: -1
232 - lastofidentifier(0) - p: 230 - s: 231
233 - p(-1) - p: 231 - s: -1
234 - letter(43) - p: 232 - s: -1
235 - lastofidentifier(0) - p: 232 - s: 234
236 - r(-1) - p: 234 - s: -1
237 - letter(34) - p: 235 - s: -1
238 - lastofidentifier(0) - p: 235 - s: 237
239 - i(-1) - p: 237 - s: -1
240 - letter(38) - p: 238 - s: -1
241 - lastofidentifier(0) - p: 238 - s: 240
242 - m(-1) - p: 240 - s: -1
243 - letter(30) - p: 241 - s: -1
244 - lastofidentifier(2) - p: 241 - s: 243
245 - e(-1) - p: 243 - s: -1
246 - epsilon(-1) - p: 244 - s: -1
247 - epsilon(-1) - p: 221 - s: -1
```