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

-
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

```
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
```

-
g2.txt

program declaration statement simpledeclaration arraydeclaration type identifierlist identifier expression in
texpression boolexpression charexpression stringexpression intvalue boolvalue charvalue stringvalue pos
itivint simpleidentifierlist non_zero_digit digit assignstatement ifstatement whilestatement functionstateme
nt functionname expressionlist 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
intexpression -> intvalue
intexpression -> identifier
intexpression -> intexpression + intexpression
intexpression -> intexpression - intexpression
intexpression -> intexpression * intexpression
intexpression -> intexpression / intexpression
intexpression -> intexpression % intexpression
intexpression -> (intexpression + intexpression)
intexpression -> (intexpression - intexpression)
intexpression -> (intexpression * intexpression)
intexpression -> (intexpression / intexpression)
intexpression -> (intexpression % intexpression)
boolexpression -> boolvalue
boolexpression -> ! identifier
boolexpression -> identifier
boolexpression -> boolexpression && boolexpression
boolexpression -> boolexpression || boolexpression
boolexpression -> boolexpression == boolexpression
boolexpression -> boolexpression < boolexpression
boolexpression -> boolexpression <= boolexpression
boolexpression -> boolexpression > boolexpression
boolexpression -> boolexpression >= boolexpression
boolexpression -> (boolexpression && boolexpression)
boolexpression -> (boolexpression || boolexpression)
boolexpression -> (boolexpression == boolexpression)
boolexpression -> (boolexpression < boolexpression)
boolexpression -> (boolexpression <= boolexpression)
boolexpression -> (boolexpression > boolexpression)
boolexpression -> (boolexpression >= boolexpression)

```
charexpression -> charvalue
charexpression -> identifier
stringexpression -> stringvalue
stringexpression -> identifier
stringexpression -> stringexpression + stringexpression
stringexpression -> ( stringexpression + stringexpression )
arraydeclaration -> array [ type ] [ positivint ] simpleidentifierlist
positivint -> non_zero_digit digitlist
digitlist -> epsilon
digitlist -> digit digitlist
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
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 -> epsilon
lastofidentifier -> letter lastofidentifier
lastofidentifier -> digit lastofidentifier
boolvalue -> true
boolvalue -> false
charvalue -> ’ letter ’
charvalue -> ’ digit ’
intvalue -> optionalsign positivint
intvalue -> 0
optionalsign -> epsilon
optionalsign -> +
optionalsign -> -
stringvalue -> " insideString "
insideString -> epsilon
insideString -> space insideString
insideString -> letter insideString
insideString -> digit insideString
```