Indian Institute of Technology Gandhinagar

Parser for ***C***--

Compilers Assignment 4

- Group 6
- Reuben Devanesan, 19110059
- Kamal Vaishnav, 20110089
- Preetam Chhimpa, 20110145
- Rishab Jain, 20110164
- GitHub Repository: https://github.com/IITGN-CS327-2024/our-own-compiler-com-piler-t6

Contents

Introduction

CMinusMinus (C--) is a programming language designed to offer a simplified syntax and semantics aimed at educational purposes and ease of understanding for beginners. It uses the ***.cmm*** file extension and is case-sensitive, distinguishing between uppercase and lowercase letters in variable names, keywords, and other identifiers. The language is structured to start program execution from the first line of the script, similar to Python, making it straightforward for new programmers to grasp the flow of control in programs.

In Assignment 2, we built a lexer for C-- and as part of Assignment 4 we have created a Context Free Grammar (CFG) for our language. **Context Free Grammar (CFG)** is a set of recursive rules used to generate patterns of strings. CFG is defined by a four tuple (V,T,P,S) where:

- V - It is the collection of variables or nonterminal symbols.
- T - It is a set of terminals.
- P - It is the production rules that consist of both terminals and nonterminals.
- S - It is the Starting symbol.

Chomsky Normal Form (CNF) is a specific form of a context-free grammar where all production rules are of the form

- A -> BC, or
- A -> a, or
- S -> ε,

where S, A, B, and C are non-terminal symbols, and *a* is a terminal symbol. This form simplifies parsing algorithms like CYK by standardizing the grammar's structure. Converting a CFG to CNF involves rewriting production rules and introducing new non-terminal symbols as needed.

The **CYK (Cocke-Younger-Kasami) algorithm** is a parsing algorithm used for context-free grammars (CFGs). It works by filling a table with non-terminal symbols that could generate substrings of the input string. By efficiently checking all possible combinations of substrings, it determines if the input string is derivable from the given grammar. The algorithm is efficient, with a time complexity of $O(n^3)$, where n is the length of the input string. However, it requires the grammar to be in Chomsky Normal Form (CNF) for optimal performance.

Context Free Grammar

Below is the CFG for our C--, you can also view it **here**.

```
Program -> Statements
Statements -> Statement Statements | ε
Statement -> VariableDecl | ImmutableVariableDecl | FunctionDecl | Loop | Conditional | ExceptionHandling | Comment | Assignment | UnaryOperation | PrintFunction | InputFunction | ArrayDecl | ListDecl | TupleDecl | DictionaryDecl | Block

VariableDecl -> VariableName LPAREN DataType RPAREN EQUALTO Expression ENDOST
ImmutableVariableDecl -> "immute" VariableName LPAREN DataType RPAREN EQUALTO Literal ENDOST
DataType -> "integer" | "character" | "floatpoint" | "textwave" | "flag"

Expression -> SimpleExpression | LogicalExpr | AccessExpression | DictionaryAccess
SimpleExpression -> Term | SimpleExpression BinaryOperator Term
Term -> Factor | LPAREN Expression RPAREN
Factor -> VariableName | Number | FloatNumber | "\"" AnyText "\"" | "True" | "False"
BinaryOperator -> "+" | "-" | "*" | "/" | "%" | "^"
LogicalExpr -> Expression LogicalOperator Expression
LogicalOperator -> "&&" | "$$" | "!"

AccessExpression -> VariableName AccessOperator
AccessOperator -> LSQUARE Number RSQUARE | LSQUARE Number COLON Number RSQUARE
DictionaryAccess -> VariableName LSQUARE Key RSQUARE

Number -> Digit Number | Digit
FloatNumber -> Number DOT Number
Literal -> Number | FloatNumber | "\"" AnyText "\"" | "True" | "False"
Digit -> "0" | "1" | "2" | ... | "9"

FunctionDecl -> "fn" FunctionName LPAREN Arguments RPAREN Block
FunctionName -> VariableName
Arguments -> Argument ArgumentsRest | ε
ArgumentsRest -> COMMA Argument ArgumentsRest | ε
Argument -> VariableName LPAREN DataType RPAREN

Conditional -> GivenClause ElseGivenClause OtherwiseClause
GivenClause -> "Given" LPAREN Expression RPAREN Block
ElseGivenClause -> "ElseGiven" LPAREN Expression RPAREN Block | ε
```

```
OtherwiseClause -> "Otherwise" Block | ε

Loop -> WhileLoop | IterateLoop | RangeLoop
WhileLoop -> "while" LPAREN Condition RPAREN Block
IterateLoop -> "iterate" LPAREN VariableName LPAREN DataType RPAREN EQUALTO Expression ENDOST "through" ListName ENDOST RPAREN Block
RangeLoop -> "iterate" LPAREN VariableName LPAREN DataType RPAREN EQUALTO Expression ENDOST "through" "range" LPAREN Expression "..." Expression COMMA Expression RPAREN ENDOST RPAREN Block
LoopStatements -> LoopStatement LoopStatements | ε
LoopStatement -> Statement | LoopInterrupt | LoopResume
LoopInterrupt -> "interrupt" ENDOST
LoopResume -> "resume" ENDOST

ExceptionHandling -> StriveBlock CaptureBlock
StriveBlock -> "strive" Block
CaptureBlock -> "capture" Block | "capture" Block

UnaryOperation -> VariableName "++" ENDOST | VariableName "--" ENDOST
Assignment -> VariableName EQUALTO Expression ENDOST | VariableName AssignmentOperator Expression ENDOST
AssignmentOperator -> "+=" | "-=" | "*=" | "/=" | "%="

ArrayDecl -> VariableName LPAREN DataType COMMA Number RPAREN EQUALTO LCURL Elements RCURL
ListDecl -> VariableName EQUALTO LSQUARE Elements RSQUARE
TupleDecl -> VariableName EQUALTO LPAREN Elements RPAREN
DictionaryDecl -> VariableName EQUALTO LCURL KeyValues RCURL
Elements -> Element Elements | ε
Element -> Expression
KeyValues -> KeyValue KeyValues | ε
KeyValue -> Expression COLON Expression

ArrayMethods -> ArrayDim | ArrayExchange | ArrayGetIdx | ArraySort | ArrayTail | ArrayHead
ListMethods -> ListDim | ListExchange | ListGetIdx | ListSort | ListTail | ListHead | ListAdd | ListRemove
TupleMethods -> TupleDim | TupleExchange | TupleGetIdx | TupleSort | TupleTail | TupleHead
DictionaryMethods -> DictionarySet
```

```
ArrayDim -> VariableName DOT "dim" LPAREN RPAREN
ArrayExchange -> VariableName DOT "exchange" LPAREN Number COMMA Element RPAREN
ArrayGetIdx -> VariableName DOT "getidx" LPAREN Element RPAREN
ArraySort -> VariableName DOT "sort" LPAREN SortOrder RPAREN
ArrayTail -> VariableName DOT "tail" LPAREN RPAREN
ArrayHead -> VariableName DOT "head" LPAREN RPAREN

ListAdd -> VariableName DOT "add" LPAREN Element RPAREN
ListRemove -> VariableName DOT "remove" LPAREN Number RPAREN

DictionarySet -> VariableName DOT "set" LPAREN Key COMMA Value RPAREN

SortOrder -> "Asc" | "Dsc"
Key -> Expression
Value -> Expression

VariableName -> Identifier
Identifier -> Letter | "_" | Identifier IdentifierDigits
IdentifierDigits -> Identifier Digit | Identifier Letter | ε
Letter -> "a" | "b" | "c" | ... | "z" | "A" | "B" | ... | "Z"

Comment -> SingleLineComment | MultiLineComment
SingleLineComment -> "#" AnyText "\n"
MultiLineComment -> "\"\"\"" AnyText "\"\"\""
AnyText -> AnyChar AnyText | ε
AnyChar -> Letter | Digit | AnySymbol
AnySymbol -> " " | "!" | "@" | "#" | ... (and so on for all printable characters)

PrintFunction -> "pendown" LPAREN FunctionArgs RPAREN
FunctionArg -> Expression
FunctionArgs -> FunctionArg | FunctionArg COMMA FunctionArgs

Block -> LCURL Statements RCURL
```

CFGs in Chomsky Normal Form

We have converted our CFG into CNF form to make sure that the input can be parsed using the CYK algorithm. Below are some portions of the CFG in CNF form, you can also view it **here**.

Variable Declaration:

```
S -> VD_Content  X6
VD_Content -> X1 Expression
X1 -> X2 X7
X2 -> X3 X8
X3 -> X4 DataType
X4 -> X5 X9
X5 -> Identifier
X6 -> SEMICOLON
X7 -> EQUALS
X8 -> RPAREN
X9 -> LPAREN
DataType -> integer | character | floatpoint | doublepoint | textwave | flag
Expression -> integer_literal | char_literal | float_literal | True | False
```

Conditionals:

```
S → GivenBlock Y1 | GivenBlock OtherwiseBlock
Y1 → ElseGivenBlock OtherwiseBlock | ElseGivenBlock Y1
GivenBlock → GivenCondition Block
ElseGivenBlock → ElseGivenCondition Block
OtherwiseBlock → Otherwise Block
GivenCondition → Y3 Y2
ElseGivenCondition →  Y4 Y2
Y2 - > Y5 COND
COND -> Y6 Y7
Y6 -> Y8 Y9 | True | False
Y9 -> Operators Y10
Y3 -> GIVEN
```

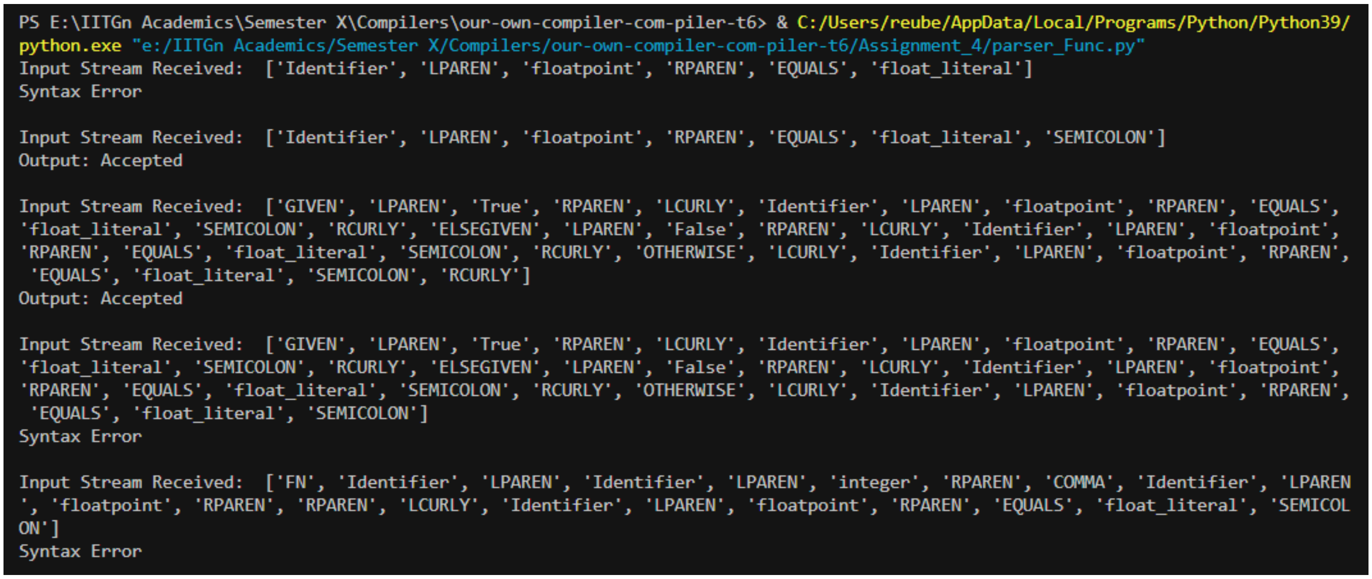
```
Y4 -> ELSEGIVEN
Y5 -> LPAREN
Y7 -> RPAREN
Y8 -> Identifier
Y10 -> integer_literal | float_literal
Operators -> == | >= | !=
BLOCK -> Z1 Z2
Z1 -> Z3 Z4
Z4 -> VD_CONTENT X6
Z2 -> RCURLY
Z3 -> LCURLY
```

Functions:

```
S -> FuncBlock Block
FuncBlock -> W1 DEFN
DEFN -> W2 ARGS
ARGS -> W3 W4
W4 -> W5 W6
W5 -> W7 W8 | W7 W5
W7 -> W9 W10
W10 -> W11 W12
W12 -> DataType W14
W1 -> FN
W2 -> Identifier
W3 -> LPAREN
W6 -> RPAREN
W8 -> COMMA
W9 -> Identifier
W11 -> LPAREN
W14 -> RPAREN
```

TestCases & Outputs

You can also view the codes **here**.



```
PS E:\IITGn Academics\Semester X\Compilers\our-own-compiler-com-piler-t6> & C:/Users/reube/AppData/Local/Programs/Python/Python39/
python.exe "e:/IITGn Academics/Semester X/Compilers/our-own-compiler-com-piler-t6/Assignment_4/parser_Func.py"
Input Stream Received:  ['Identifier', 'LPAREN', 'floatpoint', 'RPAREN', 'EQUALS', 'float_literal']
Syntax Error

Input Stream Received:  ['Identifier', 'LPAREN', 'floatpoint', 'RPAREN', 'EQUALS', 'float_literal', 'SEMICOLON']
Output: Accepted

Input Stream Received:  ['GIVEN', 'LPAREN', 'True', 'RPAREN', 'LCURLY', 'Identifier', 'LPAREN', 'floatpoint', 'RPAREN', 'EQUALS',
'float_literal', 'SEMICOLON', 'RCURLY', 'ELSEGIVEN', 'LPAREN', 'False', 'RPAREN', 'LCURLY', 'Identifier', 'LPAREN', 'floatpoint',
'RPAREN', 'EQUALS', 'float_literal', 'SEMICOLON', 'RCURLY', 'OTHERWISE', 'LCURLY', 'Identifier', 'LPAREN', 'floatpoint', 'RPAREN',
 'EQUALS', 'float_literal', 'SEMICOLON', 'RCURLY']
Output: Accepted

Input Stream Received:  ['GIVEN', 'LPAREN', 'True', 'RPAREN', 'LCURLY', 'Identifier', 'LPAREN', 'floatpoint', 'RPAREN', 'EQUALS',
'float_literal', 'SEMICOLON', 'RCURLY', 'ELSEGIVEN', 'LPAREN', 'False', 'RPAREN', 'LCURLY', 'Identifier', 'LPAREN', 'floatpoint',
'RPAREN', 'EQUALS', 'float_literal', 'SEMICOLON', 'RCURLY', 'OTHERWISE', 'LCURLY', 'Identifier', 'LPAREN', 'floatpoint', 'RPAREN',
 'EQUALS', 'float_literal', 'SEMICOLON']
Syntax Error

Input Stream Received:  ['FN', 'Identifier', 'LPAREN', 'Identifier', 'LPAREN', 'integer', 'RPAREN', 'COMMA', 'Identifier', 'LPAREN
', 'floatpoint', 'RPAREN', 'RPAREN', 'LCURLY', 'Identifier', 'LPAREN', 'floatpoint', 'RPAREN', 'EQUALS', 'float_literal', 'SEMICOL
ON']
Syntax Error
```

Input Token Stream 1: ["Identifier", "LPAREN", "floatpoint", "RPAREN", "EQUALS", "float_literal"]
Output: Syntax Error
Reason: SEMICOLON missing
Example: myFloat (floatpoint) = 10.0

Input Token Stream 2: ["Identifier", "LPAREN", "floatpoint", "RPAREN", "EQUALS", "float_literal", “SEMICOLON”]
Output: Accepted
Example: myFloat (floatpoint) = 10.0

Input Token Stream 3:
["GIVEN", "LPAREN", "True", "RPAREN", "LCURLY",
"Identifier", "LPAREN", "floatpoint", "RPAREN", "EQUALS", "float_literal", "SEMICOLON", "RCURLY",
"ELSEGIVEN", "LPAREN", "False", "RPAREN", "LCURLY",
"Identifier", "LPAREN", "floatpoint", "RPAREN", "EQUALS", "float_literal", "SEMICOLON", "RCURLY",
"OTHERWISE", "LCURLY",
"Identifier", "LPAREN", "floatpoint", "RPAREN", "EQUALS", "float_literal", "SEMICOLON", "RCURLY"]
Output: Accepted

Example:

```
Given (True) {
        myFloat1 (floatpoint) = 10.0;
}
ElseGiven (True) {
        myFloat2 (floatpoint) = 10.0;
}
Otherwise {
        myFloat3 (floatpoint) = 10.0;
}
```

Input Token Stream 4:

["GIVEN", "LPAREN", "True", "RPAREN", "LCURLY",
"Identifier", "LPAREN", "floatpoint", "RPAREN", "EQUALS", "float_literal", "SEMICOLON", "RCURLY",
"ELSEGIVEN", "LPAREN", "False", "RPAREN", "LCURLY",
"Identifier", "LPAREN", "floatpoint", "RPAREN", "EQUALS", "float_literal", "SEMICOLON", "RCURLY",
"OTHERWISE", "LCURLY",
"Identifier", "LPAREN", "floatpoint", "RPAREN", "EQUALS", "float_literal", "SEMICOLON"]

Output: Syntax Error

Reason: Right Curly } missing

Example:

```
Given (True) {
        myFloat1 (floatpoint) = 10.0;
}
ElseGiven (True) {
        myFloat2 (floatpoint) = 10.0;
}
Otherwise {
        myFloat3 (floatpoint) = 10.0;
```

Input Token Stream 5:

```
[
   "FN", "Identifier", "LPAREN",
   "Identifier", "LPAREN", "integer", "RPAREN", "COMMA",
   "Identifier", "LPAREN", "floatpoint", "RPAREN", "RPAREN",
   "LCURLY",
   "Identifier", "LPAREN", "floatpoint", "RPAREN", "EQUALS", "float_literal", "SEMICOLON",
 ]
```

Output: Syntax Error

Reason: Right Curly } missing

Example:

```
fn summation( a (integer) , b (floatpoint)){
          myFloat (floatpoint) = 10.0;
```