

CMPILER

Lexical Rules and Context Free Grammar

In partial fulfillment of the requirements

for CMPILER, Term 2, AY 2015-2016

Submitted by

Dionio, William Paolo Jose

Palabay, Hannah Patricia

Portales, Naomi

Uy, Justin Kerrbie

CMPILER S19

1. **Lexical Rules**

|  |  |  |  |
| --- | --- | --- | --- |
| **Token Type** | **Regular Expression** | **English Description of RegEx** | **Description** |
| ***OPERATORS*** | | | |
| ADD | + |  | Addition |
| SUB | - |  | Subtraction |
| MUL | \* |  | Multiplication |
| DIV | / |  | Division |
| MOD | % |  | Modulo |
| ASSIGN | = |  | Assignment operator |
| EQUAL | == |  | Equality operator |
| INCR | ++ |  | Increment Operator |
| DECR | -- |  | Decrement Operator |
| OR | || |  | Logical OR Operator |
| AND | && |  | Logical AND operator |
| NOT | ! |  | Logical NOT operator |
| Not Equal operator | != |  | Not Equal to |
| LT | < |  | Less than |
| GT | > |  | Greater than |
| LE | <= |  | Less than or Equal to |
| GE | >= |  | Greater than or Equal to |
| ADD\_ASSIGN | += |  | Add AND assignment operator |
| SUB\_ASSIGN | -= |  | Subtract AND assignment operator |
| MUL\_ASSIGN | \*= |  | Multiply AND assignment operator |
| DIV\_ASSIGN | /= |  | Divide AND assignment operator |
| MOD\_ASSIGN | %= |  | Modulo AND assignment operator |
| ***SYMBOLS*** | | | |
| LPAREN | ( |  | Open parenthesis |
| RPAREN | ) |  | Close parenthesis |
| LBRACK | [ |  | Open square bracket |
| RBRACK | ] |  | Close square bracket |
| LBRACE | { |  | Open brace |
| RBRACE | } |  | Close brace |
| SEMI | ; |  | Semicolon |
| DOT | . |  | Single dot |
| COMMA | , |  | Separates entities |
| ***DATATYPES*** | | | |
| INT | int |  | Integer datatype |
| STRING | String |  | String datatype |
| FLOAT | float |  | Float Datatype |
| CHAR | char |  | Char Datatype |
| VOID | void |  | Void Datatype |
| BOOLEAN | boolean |  | Boolean Datatype |
| ***Literals*** | | | |
| IntegerLiteral | PositiveIntegerLiteral|NegativeSign Digits |  | Positive or Negative Integer Literal |
| Float Literal | NegativeSign Digits ‘.’ Digits | Digits ‘.’ Digits |  | Positive or Negative Float Literal |
| NegativeIntegerLiteral | NegativeSign Digits |  | Negative Integer |
| PositiveIntegerLiteral | ‘+’? Digits |  | Positive Integer |
| StringLiteral | ‘“‘ StringCharacters ‘“‘ |  | String Literal encased in "" |
| NullLiteral | ‘null’ |  | NULL |
| booleanLiteral | true | false |  | Boolean Literals |
| **Number Related Fragments** | | | |
| Digits | Digit+ | Accepts series of digits | Series of Numbers |
| Digit | [0-9] | Accepts a number between 0-9 | A Number |
| NegativeSign | ‘-’ |  | Negative Sign |
| **Letter Related Fragments** | | | |
| Letters | Letter+ | Accepts a series of letters | Series of Letters |
| Space | ‘ ‘ |  | Space |
| Letter | [A-Za-z] | Accepts any uppercase or lowercase letter | A single Letter |
| StringCharacters | [A-Za-z0-9.!?\_+\-,@#$%^&\*();\\\/|<>”’~ ]\* | Accepts any series of letters, numbers and symbols | Series of letters, numbers and symbols |
|  |  |  |  |
| ***CONDITIONAL STATEMENTS*** | | | |
| If statement | trooper |  | If statement |
| Else statement | storm |  | Else statement |
| Else if statement | stormtrooper |  | Else if statement |
| Switch statement | kamino |  | Switch statement |
| Case statement | clone |  | Case statement |
| Break statement | fett |  | Break statement |
| Default statement | kaminoans |  | Default statement |
| Return statement | jedi |  | Return statement |
| True statement | TRUE |  | True statement |
| False statement | FALSE |  | False statement |
| ***LOOPS*** | | | |
| For statement | force |  | For Loop |
| Do statement | iloveyou |  | Do Loop |
| While statement | iknow |  | While Loop |
| ***OTHERS*** | | | |
| VariableFuncName | Letters Digits? | Function & Variable names are made up of letters and can have digits | Variable/Function Name |
| comment | ‘meesa ‘ .\*? ‘ yousa’ | Reads meesa, starting the comment block. Ends the comment block when yousa is read. | Comment Block |
| WS | [ \t\r\n\u000C]+ -> skip |  | White Space |
| **PRINTER AND SCANNERS** | | | |
| PRINTER | ‘forcePrint’ | Calls print function | Print Function |
| SCANNER | ‘forceScan’ | Calls scan function | Scan Function |
| SCANNERINT | ‘\’%d\’’ |  | Formatted Scanner for INT |
| SCANNERFLOAT | ‘\’%f\’’ |  | Formatted Scanner for FLOAT |
| SCANNERBOOLEAN | ‘\’%b\’’ |  | Formatted Scanner for BOOLEAN |
| SCANNERCHAR | ‘\’%c\’’ |  | Formatted Scanner for CHAR |
| SCANNERSTRING | ‘\’%s\’’ |  | Formatted Scanner for STRING |
|  |  |  |  |

1. **Context Free Grammar**

Legend:

HighlightedWords - lexical tokens

**BoldFacedWords** - variables

**datatype**  -> int | float | string | char | boolean

**accessModifier** -> public | private | protected

**operator**  -> + | - | \* | / | %

**specialOperator** -> += | -= | \*= | /= | %=

**special2Operator** -> ++ | --

**logicalOperator**  -> && | || | == | != | > | < | >= | <= | !

**value**  -> intliteral | floatliteral | charliteral | stringliteral | boolliteral

**specialValue**  -> value | variableName | expression | functionCallNoTerminator

1) Variable Declaration / Initialization

**declaration** ->datatype multiDec SEMI

**singleDec** -> VariableFuncName | VariableFuncName ASSIGN specialValue

**multiDec** -> singleDec COMMA multiDec | singleDec

2) Assignment Statement

**assignment** -> VariableFuncName ASSIGN specialValue SEMI | VariableFuncName specialOperator specialValue Semi

3) Conditional Statement

**conditionalStatement** -> ifBlock | switchBlock

**ifBlock** -> ifCondition elseIfBlock?

**ElseIfBlock** -> elseIf ElseIfBlock | elseCondition

**ifCondition** -> IF LPAREN conditionalExpression RPAREN LBRACE codeBlock? RBRACE

**elseIf** -> ELSEIF LPAREN conditionalExpression RPAREN LBRACE codeBlock? RBRACE

**elseCondition** -> ELSE LBRACE codeBlock? RBRACE

**switchBlock** -> SWITCH LPAREN conditionalExpression RPAREN LBRACE switchCase RBRACE

**switchCase** -> caseBlock switchCase | caseBlock | defaultBlock

**caseBlock** -> caseCondition caseBlock | caseCondition

**caseCondtion** -> CASE value ‘:’ LBRACE codeBlock? RBRACE | CASE value ‘:’ LBRACE codeBlock? RBRACE BREAK SEMI

**defaultBlock** ->DEFAULT ‘:’ LBRACE codeBlock? RBRACE | DEFAULT ‘:’ LBRACE codeBlock? } RBRACE BREAK SEMI

**conditionalExpression** -> specialValue conditionalExpression2?

**conditionalExpression2** -> logicalOperator specialValue

4) Loop / Iterative Statement

**loopStatement** -> whileBlock | doWhileBlock | forBlock

**whileBlock** -> WHILE LPAREN conditionalExpression RPAREN LBRACE codeBlock? RBRACE

**doWhileBlock** -> DO LBRACE codeBlock? RBRACE WHILE LPAREN conditionalExpression RPAREN SEMI

**forBlock** -> FOR LPAREN decValue SEMI conditionalExpression SEMI step RPAREN LBRACE codeBlock? RBRACE

**decValue** -> INT VariableFuncName ASSIGN initValue | VariableFuncName ASSIGN initValue

**initValue** -> IntegerLiteral |VariableFuncName| NegativeIntegerLiteral

**step** -> VariableFuncName special2Operator | VariableFuncName specialOperator (IntegerLiteral|NegativeIntegerLiteral)

5) Expressions

**expression** -> expr

**expr** -> value | VariableFuncName | functionCallNoTerminator

**expr** -> expr operator expr

**expr** -> LPAREN expr RPAREN

**expr** -> VariableFuncName special2Operator SEMI?

6) Function Declaration / Definition

**functionDeclaration** -> datatype fVariableFuncName LPAREN declarationParameter RPAREN LBRACE codeBlock? returnStatement RBRACE functionDeclaration? | VOID VariableFunctionName LPAREN declarationParameter RPAREN LBRACE codeBlock? RBRACE functionDeclaration?

**declarationParameter** -> singleDeclarationParameter multiDeclartationParameter | singleDeclarationParameter

**multiDeclarationParameter** -> COMMA declarationParameter

**singleDeclarationParameter** -> returntype VariableFuncName

**returnStatement** -> RETURN specialValue SEMI

7) Function Call

**functionCall**  -> VariableFunctionName LPAREN callParameter? RPAREN SEMI

**functionCallNoTerminator** ->VariableFunctionName LPAREN callParameter? RPAREN

**callParameter** -> specialValue COMMA callParameter callParameter?| specialValue

8) Arrays

**array**  -> datatype VariableFuncName arrayAssignment SEMI

**arrayAssignment** -> LBRACK IntegerLiteral RBRACK | LBRACK RBRACK ASSIGN LBRACE list RBRACE

**list**  -> integerList | charList | stringList | floatList | boolList

**integerList** -> (NegativeIntegerLiteral | integerliteral ) COMMAintlist | intliteral

**charList ->** charliteralCOMMAcharlist | charliteral

**stringList**  -> stringliteral COMMA stringlist | stringliteral

**floatList**  -> floatliteral COMMA floatlist | floatliteral

**boolList**  -> boolliteral COMMA boollist | boolliteral

9) Code Block

**CodeBlock** -> declaration CodeBlock? | assignment CodeBlock? | conditionalStatement CodeBlock? | loopStatement CodeBlock? | functionCall CodeBlock? | array CodeBlock? | expression CodeBlock? | comment codeBlock? | printer codeBlock? | scanner codeBlock?

10) Main

**main** -> INT ‘jarjarbinks’ LPAREN RPARE LBRACE codeBlock? returnMain RBRACE

**returnMain** -> RETURN IntegerLiteral SEMI

11) Printer

**printer** -> PRINTER LPAREN StringLiteral RPAREN SEMI |PRINTER LPAREN specialValue RPAREN SEMI

11) Scanner

**scanner** -> SCANNER LPAREN SCANNERINT COMMA VariableFuncName RPAREN SEMI

**scanner** -> SCANNER LPAREN SCANNERFLOAT COMMA VariableFuncName RPAREN SEMI

**scanner** -> SCANNER LPAREN SCANNERBOOLEAN COMMA VariableFuncName RPAREN SEMI

**scanner** -> SCANNER LPAREN SCANNERCHAR COMMA VariableFuncName RPAREN SEMI

**scanner** -> SCANNER LPAREN SCANNERSTRING COMMA VariableFuncName RPAREN SEMI