|  |
| --- |
| CS 440 |
| Programming Languages and Translators |
|  |
| MCL: Grammar report |
| 02/26/2017 |

|  |
| --- |
| Cai Chen  zdthm007@gmail.com  Yuyang Luo  yluo41@hawk.iit.edu |

Note: Blue brackets( [] ) means one or zero, and blue braces( () ) means one or more.

Program → Startlist

Startlist → Startlist Declarationlist | Declarationlist

Declarationlist → Declarations | FunDeclaration | Deflists

Declarations → **{** Declaration **}**

Declaration → Type Identifier [ [ int[:int] ] **]** { , Identifier [ [ int[:int] ] ] } ;

Type → int | bool | real | matrix

FunDeclaration → Type Identifier (Params ){ Declarations Statements BuildinFun }

Params → Param | ɛ

Param → Param ; Declaration | Declaration

BuildinFun → \_Identifier\_ (Identifier)

Statements → { Statement }

Statement → Expression; | ; | Block | Assignment | IfStatement | ForStatement | ReturnStatement | BreakStatement

Block → { Statements }

Assignment → Identifier [ [ Expression ] ] = Expression ;

IfStatement → if ( Expression ) Statement [ else Statement ]

ForStatement → for ( Assignment; Expression; Identifier ( ++|-- ) ) Statement

ReturnStatement → return; | return Expression;

BreakStatement → break;

Deflists → { Deflist }

Deflist → DefVar | DefFun

DefVar → def Identifier [ Literal ]

DefFun → def Identifier (DefParam) { { Assignment | Expression | ; } }

DefParam → Identifier [ [ int[:int] ] **]** { , Identifier [ [ int[:int] ] ] }

Expression → !Expression | Condition

Condition → Conjunction { || Conjunction }

Conjunction → Equality { && Equality }

Equality → Relation [ EquOp Relation ]

EquOp → == | !=

Relation → Addition [ RelOp Addition ]

RelOp → < | <= | > | >=

Addition → MObject { MMulOp MObject }

MMulOp → .\* | ./

MObject → Term { AddOp Term }

AddOp → + | -

Term → Factor { MulOp Factor }

MulOp → \* | / | %

Factor → Element [ ExpOp Element ]

ExpOp → ^

Element → [ UnaryOp ] Primary

UnaryOp → - | ! | ++ | --

Primary → Identifier [ [ Expression ] ] | Literal | ( Expression ) | Type ( Expression )

Identifier → Letter { Letter | Digit }

Letter → a | b | … | z | A | B | … | Z

Digit → 0 | 1 | … | 9

Literal → int | bool | real | matrix | null

int → Digit { Digit }

bool → true | False

real → int . int

matrix → Identifier[int:int]

Conclusion:

This grammar is unambiguous since I don’t want to cause any trouble made by incorrect order of operations. For example, in the expression 5\*3-5, if minus operation works first and multiple operation executes late, the result will be -10. So I use unambiguous grammar to give multiple operation a high precedence than minus operation.