Milestone 1

Roy Thompson, Robin Suda, and Hilario Mendez-Vallejo

BNF Grammar

Below is the proposed BNF grammar for our language. Use `StmtList` as the start symbol. A program in our language is considered to be a list of Statements.

```
StmtList    ::= Stmt StmtList | Stmt
Stmt        ::= Assign ";" | Dec ";" | Block | Iter | Cond | Print ";" |
                Expr ";"

Dec         ::= Type IdList
IdList      ::= Id "," IdList | Id
Assign      ::= Id "=" Expr

Block ::= "{" StmtList "}"

Cond        ::= If | If ElseIfList | If Else | If ElseIfList Else
If          ::= "if" "(" Expr ")" Block
ElseIfList  ::= ElseIf ElseIfList | ElseIf
ElseIf      ::= "else if" "(" Expr ")" Block
Else        ::= "else" Block

Iter        ::= ForIter | WhileIter
ForIter     ::= "for" "(" Expr ")" Block
WhileIter   ::= "while" "(" Expr ")" Block

LogOr       ::= LogOr "||" LogAnd | LogAnd
LogAnd      ::= LogAnd "&&" LogEq | LogEq
LogEq       ::= LogEq "==" RelOp | LogEq "!=" RelOp | RelOp
RelOp       ::= RelOp "<" AddOp | RelOp "<=" AddOp | RelOp ">" AddOp |
                RelOp ">=" AddOp | AddOp
AddOp       ::= AddOp "+" MulOp | AddOp "-" MulOp | MulOp
MulOp       ::= MulOp "*" ExpOp | MulOp "/" ExpOp | MulOp "%" ExpOp | ExpOp
ExpOp       ::= Expr "^" ExpOp | AbsOp
AbsOp       ::= "|" AbsOp "|" | UnaryOp
UnaryOp     ::= "++" UnaryOp | "--" UnaryOp | "!" UnaryOp | "-" UnaryOp | PostIncr
PostIncr    ::= PostIncr "++" | PostIncr "--" | Expr
Expr        ::= "(" LogOr ")" | int | bool

Print       ::= "print" "(" Expr ")" ";"
```

Regular Expressions

```
Reserved Symbols = {
    "-",
    "--",
    "!",
    "*",
    "/",
    "%",
    "+",
    "++",
    "<",
    "<=",
    "=",
    "==",
    ">",
    ">=",
    "||",
    "bool",
    "else if",
    "else",
    "for",
    "if",
    "int",
    "print",
    "while",
    "^",
    "|"
}

Type = {
    "int",
    "bool"
}

Id = [a-zA-Z_]+[a-zA-Z0-9]*
int = [1-9][0-9]* | [0]
bool = true | false
```

Precedence and Associativity Table

The precedence and associativity table for the preceding language is described below. Note that the precedence is described in ascending order (1 => lowest)

Operation	Precedence	Associativity	Nonterminal
Logical Or	1 (Low)	Left	LogOr

Operation	Precedence	Associativity	Nonterminal
Logical And	2	Left	`LogAnd`
Equality	3	Left	`LogEq`
Relational	4	Left	`RelOp`
Additive	5	Left	`AddOp`
Multiplicative	6	Left	`MulOp`
Exponentiation	7	Right	`ExpOp`
Absolute Value	8	Left	`AbsOp`
Unary	9	Right	`UnaryOp`
Post Increment	10	Left	`PostIncr`
Parenthetical Expression	11 (High)	Left	`Expr`