# **Assignment 2**

## Context Free Grammar for Yet Another Language

# Roll No: 10-5714

# Name: Muhammad Taha Zaidi

id is the regular expression for the identifiers

Statements 🡪 assignStmt; Statements | declarationStmt; Statements | ifStmt Statements

| whileLoop Statements | funCall ; Statements | ^

assignStmt 🡪 var 🡨 artihExp | var 🡨 charLiteral

arrElem 🡨 id [id] | id [num]

charLiteral 🡪 ‘letter**’**

declarationStmt 🡪 type id

type 🡪 **integer** | **character** | **void**

ifStmt 🡪 **if** condExp **begin** Statements **end** optionalElse

optionalElse 🡪 **else begin** Statements **end**

whileStmt 🡪 **while** condExp **begin** Statements **end**

condExp 🡪 id relOp id | id relOp arithExp | arithExp relOp id | arithExp relOp arithExp

relOp 🡪 = | =< | => | =/ | > | <

funcDecl 🡪 **function** id ( optionalParam ) optionalReturn **begin** Statements **end** funcDecl | ^

optionalParam 🡪 ^ | parameters

parameters 🡪 type **[]** id **,** parameters | type id **,** parameters | type id | type **[]** id

optionalReturn 🡪 **returns** type | ^

type 🡪 **character | integer | void**

funcCall 🡪 id (funcparam)

funcParam 🡪 ^ | id | num | charLiteral |

arithExp 🡪 arithExp + T | arithExp – T | T

T 🡪 T \* F | T / F | F

F 🡪 id OptSubs | num | (arithExp)

Rewriting the CFC for arithmetic expressions (removing ambiguity and left recursion)

arithExp 🡪 T arithExp‘

arithExp‘ 🡪 + T arithExp‘ | - T arithExp‘ | ^

T 🡪 F T’

T’ 🡪 **\*** F T’ | **/** F T ‘ | ^

F 🡪 id OptSubs | num | **(**arithExp**)**

*Program 🡪 funcDecl Statements*

*(My assumption here is that all the functions are declared first and then comes the program statements for Yet Another language)*