# SER 502

## STARK Programming Language

**Language Name**: Stark (Simple, Clear and Complete)
**Programming paradigm**: Imperative
**File extension** : .ark
**Tools/language used** : Antlr 4, Java

The source code in Stark will be compiled and executed in two steps :

1. Compile the source code to intermediate code
2. Interpret the intermediate code to produce the final output

The compilation will happen in a series of steps. The source code will be given to lexical analyser to generate the token, which will then be used by the syntax analyzer to give a parse tree. The parse tree would be given semantics to generate an intermediate code. The intermediate code is then interpreted by the runtime environment to generate an output.

Stark language supports two data types- int and bool and mathematical, relational and logical operations (+,-,*,/,%,>,<,>=,<=,==,&&,||, ! etc). The language also supports function declaration and function calls (currently the function definition is limited to the end of the program structure). Also if-else decision construct and while loop are supported by this language.

We have used the following Data Structures for the interpreter:

1) Stack.
2) Hash Map.

**Grammar for Stark (.g4):**

```
grammar stark;

// Program contains list of statements and function calls
program :  statementList functionDefn;

// StatementList could be a simple statement and or a list of
statements
statementList : statement  statementList |
                Statement;
```

```
// Statement could be any of the options listed
statement : declarationStmt ';' |
            initializationStmt ';'|
            assignmentStmt ';'|
            ifStatement |
            whileStatement |
            displayStatement ';' |
            functionCall ';';

initializationStmt: 'int' IDENTIFIER '=' expression|
                     //int a = 10 or int a = b+c
                    'bool' IDENTIFIER '=' boolExpression|
                     //bool x= 3>2
                    'int' IDENTIFIER '=' functionCall|
                     // int z = add(x,y)
                    'bool' IDENTIFIER '=' functionCall;

declarationStmt : 'int' IDENTIFIER|          // int x
                  'bool' IDENTIFIER;         // bool y

assignmentStmt : IDENTIFIER '=' boolExpression|  // x = true
                 IDENTIFIER '=' expression ;     // x = y or
                                                    X = y + z

// if statement with or without else and nested if statements
ifStatement : 'if' '(' boolExpression ')' '{' statementList '}'
|                                              //if(a>b) {c=a;}
              'if' '(' boolExpression ')' '{' statementList '}'
'else' '{' statementList '}' ;  // else {c=b;}

// while statement that check a condition defined by a boolean
expression and computes a list of statements
whileStatement : 'while' '(' boolExpression ')' '{'
statementList '}' ;  //while(a>5){a=a+1;}

// Display statement to display the value of a expression(which
could be identifier or a number) or a function call
displayStatement : 'display' expression|
                   'display' functionCall;     //display a
```

```
// Relational expression is to compare two numeric expression
relationalExpression : expression '==' expression |
                       expression '!=' expression|
                       expression '<' expression |
                       expression '<=' expression|
                       expression '>' expression |
                       expression '>=' expression|
                       expression '==' boolvalues|
                       expression '!=' boolvalues;

// Logical expression to compare two boolean values or
expression
logicalExpression : relationalExpression'&&'relationalExpression
                  |relationalExpression '||' relationalExpression
                  |relationalExpression '&&' logicalExpression |
                   relationalExpression '||' logicalExpression |
                   '!'relationalExpression |
                   '!'logicalExpression;

// Common rule for statements that would return true or false
boolExpression : relationalExpression |
                 logicalExpression |
                 boolvalues;

// Rules to handle normal numeric operation and precedence rules
expression : term '+' expression|
             term '-' expression|
             term;

term :  factor '*' term|
        factor '/' term|
        factor '%' term|
        factor;

factor : '(' expression ')' | IDENTIFIER | NUMBER ;



// Function definition rule
```

```
functionDefn : 'func' functionName '(' parameters ')' '{'
statementList  returnStatement '}'|'func' functionName '('
parameters ')' '{' returnStatement '}'|  ;

// Function name rule
functionName : IDENTIFIER;

// Function parameter rule
parameters : declarationStmt ',' parameters |
             declarationStmt |  ;

// Function return statement rule
returnStatement : 'return' expression ';'|
                  'return' boolExpression';'|  ;

// Function call rule
functionCall : IDENTIFIER '=' functionName '(' arguments
')'|functionName '(' arguments ')' ;

// Function argument rule
arguments : IDENTIFIER ',' arguments | NUMBER ',' arguments |
NUMBER | IDENTIFIER |  ;

// Terminals
boolvalues: 'true' | 'false' ;
IDENTIFIER : [a-zA-Z][a-zA-Z0-9]* ;
NUMBER :[0-9]+;
WS : [ \t\r\n]+ -> skip ;
MULTILINECOMMENT : '/*'.*?'*/' -> skip;
SINGLELINECOMMENT : '//' ~[\r\n]* -> skip;
```

**Sample Program:**

```
bool a=true;
```

```
bool b=false;
bool c=3>2;
bool z= add(x,y);
int x;
bool y;
if(a>b){
c=a;
}
else{
c=b;
}
while(a>=10){
display a;
a=a-1;
function(2,3);
}
func function(int x,int y){
return x*y;
}
```