SER 502 Language Grammar Documentation

1. Brief Overview of the Language:
   This language is a minimalistic, single-letter syntax language designed for efficient and compact code expression. Each command or keyword is represented by a single uppercase letter, reducing verbosity, increasing speed and making the language intuitive for those familiar with foundational programming constructs.

2. Design Principles:
   - Single-letter keywords for common programming constructs
   - Statically typed
   - Support for complex data structures and control flow
   - Clear and unambiguous grammar

3. Tokens:

3.1 Keywords (Single Letters)

- T: Integer type declaration
- B: Boolean type declaration
- S: String type declaration
- P: Print statement
- I: If statement
- E: Else statement
- W: While loop
- F: For loop
- M: Method/Function declaration
- R: Return statement
- A: Array declaration
- N: Input operation
- K: Stack declaration
- Q: Queue declaration
- C: Constant declaration
- U: String to uppercase
- L: String to lowercase
- J: String join

3.2 Literals

- Integer literals: [0-9]+
- String literals: "[^"]*"
- Boolean literals: 0|1

3.3 Identifiers

- Pattern: [a-zA-Z][a-zA-Z0-9]*
- Cannot be a single-letter keyword

3.4 Operators

- Arithmetic: +, -, *, /
- Relational: <, >, =
- Logical: & (AND), | (OR), ! (NOT)
- Ternary: ?, :

3.5 Delimiters

- Parentheses: (,)
- Braces: {, }
- Brackets: [,]
- Semicolon: ;
- Dot: .
- Comma: ,

4. Grammar (EBNF):

```
program = { statement } ;

statement = declaration_stmt
        | if_stmt
        | while_stmt
        | for_stmt
        | print_stmt
        | function_stmt
        | return_stmt
        | array_stmt
        | stack_stmt
        | queue_stmt
        | const_stmt
        | input_stmt
        | expression_stmt ;

declaration_stmt = type identifier [ "=" expression ] ";" ;

type = "T" | "B" | "S" ;

if_stmt = "I" "(" expression ")" block [ "E" block ] ;

while_stmt = "W" "(" expression ")" block ;
```

```
for_stmt = "F" "(" statement ";" expression ";" expression ")" block ;

print_stmt = "P" expression ";" ;

function_stmt = "M" identifier "(" [ parameter_list ] ")" block ;

return_stmt = "R" expression ";" ;

array_stmt = "A" type identifier "=" "[" [ expression_list ] "]" ";" ;

stack_stmt = "K" type identifier ";" ;

queue_stmt = "Q" type identifier ";" ;

const_stmt = "C" type identifier "=" expression ";" ;

input_stmt = "N" [ string_literal ] ";" ;

block = "{" { statement } "}" ;

expression_stmt = expression ";" ;

expression = ternary_expr ;

ternary_expr = logical_expr [ "?" expression ":" expression ] ;

logical_expr = relational_expr { ("&" | "|") relational_expr } ;

relational_expr = arithmetic_expr { ("<" | ">" | "=") arithmetic_expr } ;

arithmetic_expr = term { ("+" | "-") term } ;

term = factor { ("*" | "/") factor } ;

factor = integer_literal
      | string_literal
      | boolean_literal
      | identifier
      | array_access
```

| function_call
| method_call
| "(" expression ")"
| "!" factor ;

array_access = identifier "[" expression "]" ;

function_call = identifier "(" [expression_list] ")" ;

method_call = identifier "." identifier "(" [expression] ")" ;

parameter_list = parameter { "," parameter } ;

parameter = type identifier ;

expression_list = expression { "," expression } ;

5. Type System:
5.1 Basic Types

- Integer (T): Whole numbers
- Boolean (B): Truth values (0 or 1)
- String (S): Text enclosed in double quotes
- Array: Collection of values of the same type

5.2 Data Structures

- Stack: Last-In-First-Out structure
- Queue: First-In-First-Out structure

6. Operator Precedence (highest to lowest)

1. Parentheses ()
2. Array access []
3. Method/Function calls
4. Unary operators !
5. Multiplicative *, /
6. Additive +, -
7. Relational <, >, =
8. Logical AND &
9. Logical OR |
10. Ternary ?:

7. Comments
    - Single-line: # comment
    - Multi-line: #* comment *#