Minilanguage Specification

(I have updated examples, they are down)

Alphabet:

a. [A-Za-z]

b. [0-9]

c. Underscore ('_')

Lexic:

a. Special symbols, representing:

Operators:

- + - * ** / % (Arithmetic operators: *addition, subtraction, multiplication, power, division, mod*)
- < <= > >= == != (Relational operators: *smaller, smaller or equal, greater, greater or equal, equality, inequality*)
- && || ! (Logical operators: *and, or, not*)
- = (Assignment operator)
- [ ] (Index operator)

Separators:

- { } ( ) , ; <space> <newline> <indent>

Reserved words:

- read, write, if, else, for, while, break, int, string, char, list, return

b. Identifiers

- IDENTIFIER = letter {letter | digit | underscore}
- letter = "a" | "b" | ... | "z" | "A" | "B" | ... | "Z"
- digit = "0" | non_zero_digit
- non_zero_digit = "1" | ... | "9"
- underscore = "_"

c. Constants

- integer = "0" | ["+" | "-"] non_zero_digit{digit}
- character = ‘letter’ | ‘digit’ | ‘underscore’
- string = "{character}"
- CONSTANT = integer | character | string

Tokens:

(	&&	char
)	||	list
[	!	return
]	=	
{	,	
}	;	
+	<space>	
-	<newline>	
*	<indent>	
**	read	
/	write	
%	if	
<	else	
<=	for	
>	while	
>=	break	
==	int	
!=	string	

Syntax:

- program = “START” compound_statement

- compound_statement = “{“ statement_list “}”
- statement_list = statement | statement “;” statement_list
- statement = simple_statement | struct_statement
- simple_statement = assign_statement | io_statement | declaration
- struct_statement = compound_statement | if_statement | while_statement | for_statement
- assign_statement = (IDENTIFIER | indexed_identifier) “=“ expression “;”
- io_statement = read_statement | write_statement
- read_statement = “read” “(“ (IDENTIFIER | indexed_identifier) {“,” (IDENTIFIER| indexed_identifier)} “)” “;”
- write_statement = “write” “(“ id {“,” id} “)” “;”
- if_statement = “if” “(“ condition_statement “) compound_statement “else” compound_statement
- for_statement = “for” “(“ “int” assign_statement “;” condition “;” assign_statement “)” compound_statement
- while_statement = “while” “(“ condition_statement “)” compound_statement
- condition_statement = cond | cond LOGICAL cond

- expression = [expression("+"|"-")] term
- term = term("*" | "/") factor | factor
- factor = "(" expression ")" | id

- id = IDENTIFIER | CONSTANT | indexed_identifier
- declaration = type “ “ IDENTIFIER {“,” IDENTIFIER} “;”
- type = simple_type | array_declaration
- simple_type = “int” | “string” | “char”
- array_declaration = “list” “<” simple_type “>”
- condition = [“!”] expression RELATION expression
- indexed_identifier = IDENTIFIER "[" integer "]"

- RELATION ::= "<" | "<=" | "==" | "!=" | ">=" | ">"
- LOGICAL :== "&&" | "||"

Examples (Problems updated)

P1. Max of three numbers

```
START {
	int a, b, c, max;
	read(a,b,c);

	if(a>b && a>c){
		max=a;
	}
	else{
		if(b>c && b>a){
			max=b;
		}
		else{
			max=c;
		}
	}

	write(max);
}
```

P2. Check if an input is a prime number.

```
START {
	int a, i, is_prime;

	is_prime=0;
	read(a);

	for(i=2;i<a;i=i+1){
		if(a%i==0){
			is_prime=1;
			break;
		}
	}

	if(is_prime==1){
```

```
            write("a is prime");
        }else{
            write("a is not prime");
        }
}
```

P3. Compute the sum of n numbers

```
START {
        int n, m, sum, current_number;

        sum=0;
        read(n, m);

        for(i=0; i<n; i=i+1){
            read(current_number);
            sum=sum+current_number;
        }

        write(sum);
}
```

P1err. Max of three numbers

```
START {
        Int 2a; <-lexical error
        int a, b, c, max;
        read(a,b,c);

        if(a>b && a>>c){ <- lexical error
            max=a;
        }
        else{
            if(b>c && b>a){
                max=b;
            }
            else{
                max=c;
            }
        }
        write(max);
}
```