**Minilanguage Specification**

**(I have updated examples, they are down)**

Alphabet:

a. [A-Za-z]

b. [0-9]

c. Underscore ('\_')

Lexic:

1. 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);

}