p1 – compute gcd of 2 numbers

```
a INTEGER;
b INTEGER;
value INTEGER;
READ a;
READ b;

? ( a < b )
{ value = a; }
: { value = b; }

REPEAT ( value > 0 )
{
            ? ( a % value == 0 AND b % value == 0 )
            { RETURN value; }
            : { value = value – 1; }
}
```

---

p2 – verify if a number is prime

```
a INTEGER;
flag BOOLEAN;
value INTEGER;

READ a;

flag = FALSE;
value = 2;

REPEAT ( value <= a / 2 )
{
            ? ( a % value == 0 )
            {
                        flag = TRUE;
            }
            : { value = value  + 1; }
}

? ( flag == FALSE)
{ RETURN TRUE; }
RETURN FALSE;
```

---

p3 – sum of n numbers

```
a ARRAY<INTEGER>;
s INTEGER;
i INTEGER;
i = 0;

READ a;

s = 0;

EACH ( n INTEGER IN a )
{
            s = s + n;
}

RETURN s;
```

---

p1err – compute gcd of 2 numbers

```
a INTEGER;
b INTEGER;
value INTEGER;

? ( a < b )
{ val = a; }  # ERROR 1
: { value = b; }

REPEAT ( value > 0 )
{
            ? ( a % value == 0 AND b % value == 0 )
            { return value; } # ERROR 2
            : { value = value – 1; }
}
```

LEXIC
Alphabet

```
upper and lower case letters of the English alphabet
decimal digits

LEXIC
special symbols, representing:

operators + - * / % < <= = == != >= > AND OR

separators [ ] { } ( ) ; space

reserved words: ? : INTEGER REPEAT RETURN ARRAY EACH IN NULL BOOLEAN READ PRINT

identifiers (sequence of letters and digits,
such that the first character is a letter; the rule is):

            <identifier> ::= <letter> | <identifier><letter> | <identifier><digit>
            <letter> ::= A | B | … | Z | a | b | … | z
            <digit> ::= 0 | <non-zero-digit>
            <non-zero-digits> ::= 1 | 2 | .. | 9

constants
            integer
                        <integer-constant> ::= 0 | <non-zero-number> | <sign><non-zero-number>
                        <non-zero-number> ::=  <non-zero-digit> | <non-zero-number><digit>
                        <sign> ::= + | -
            boolean
                        <boolean-constant> ::= TRUE | FALSE

SYNTAX

<program> ::= <declaration-list> ; <compound-statement>
<declaration-list> ::= <declaration> | <declaration> ; <declaration-list>
<declaration> ::= <identifier> <type>
<type> ::= <type1> | <array-declaration>
<type1> ::= INTEGER | BOOLEAN
<array-declaration> ::= <identifier> ARRAY< <type1> >

<compound-statement> ::= <statement> | <statement> ; <compound-statement>
<statement> ::= <simple-statement> | <special-statement>
<simple-statement> ::= <assign-statement> | <input-output-statement>
<assign-statement> ::= <identifier> = <expression>

<expression> ::= <expression> - <term> | <expression> + <term>| <term>
<term> ::=  <term> * <factor> |       <term> / <factor> |
                        <term> % <factor> | <factor>
<factor> ::= ( <expression> ) | <identifier> | CONSTANT

<input-output-statement> ::= READ <identifier> | PRINT <identifier> |
                                                PRINT CONSTANTS
<special-statement> ::= <if-statement> | <while-statement> |
                                    <for-each-statement>
<if-statement> ::= ? (<condition>) { <compound-statement> } |
            ? (<condition>) { <compound-statement> } : { <compound-statement> }
<while-statement> REPEAT (<condition>) { <compound-statement> }
<for-each-statement> ::= EACH ( <declaration> IN <identifier> ) { <compound-statement> }
<condition> ::= <expression> <relation> <expression>
<relation> ::= < | <= | == | != | >= | > | AND | OR


TOKENS
+
-
*
/
%
<
<=
=
==
!=
>=
>
AND
OR
[
]
{
}
(
)
;

?
:
INTEGER
BOOLEAN
REPEAT
RETURN
ARRAY
```

EACH
IN
NULL
READ
PRINT