Alphabet:
 a. Upper (A-Z) and lower case letters (a-z) of the English alphabet
 b. '_', '$', '~', '-', ':'
 c. Decimal digits (0-9);

a.Special symbols, representing:
 - operators: + - * / := = < > <= >= <> not and or
 - separators: [] {} : ; space $ $- -$, newline
 - reserved words:
  decl int char bool dtype real
  new read prnt
  for do
  verif then else
  end
  ret

b.identifiers
 identifier = (letter | undersc) | (letter | undersc){letter}{digit}{undersc}
 letter = "a" | "b" | ... | A" | "B" | ... | "Z"
 digit = "0" | "1" |...| "9"
 undersc = "_"

c.constants
 1.integer
  digitnz = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
  digit   = "0" | digitnz
  nbr = digitnz{digit}
  int = "0" | ["-"] nbr

 2.character
  char = letter | digit

 3.string
  str = """string"""
  string = char{string}

 4. boolean
  bool = "true" | "false"
 5.real = int | float= nbr.digit{digit} | 0.digit{digit}

2.2 Syntax:
The words - predefined tokens are specified between " and ":

program = "~" [decllist] [stmt] "~"

decllist = declaration | declaration "," decllist

declaration = "decl" IDENTIFIER type [":=" expression]

type = type1 | arraydecl

type1 = "bool" | "char" | "int" | "real"

arraydecl = "decl" IDENTIFIER type "[]"

```
dtypedecl = "dtype" IDENTIFIER ":" {decllist} "end"



stmtlist = stmt | stmt ";" stmtlist

stmt = simplstmt | structstmt

simplstmt = assignstmt | iostmt

assignstmt = assign1 | assign2
assign1 = IDENTIFIER ":=" expression
assign2 = IDENTIFIER ":" ((("+"|"-") expression) | ("++"|"--"))

expression = expression ("+" | "-") term | term
term = term ("*" | "/" | "%") factor | factor
factor = "(" expression ")" | IDENTIFIER | REAL

iostmt = ("prnt" | "read") (idlist | expression)

idlist = IDENTIFIER | IDENTIFIER "," idlist

structstmt = cmpdstmt | ifstmt | whilestmt

cmpdstmt = ("begin") stmtlist "end"


ifstmt = "verif" compcond "then" stmt ["else" stmt]
whilestmt = "whilst" compcond "do" stmt
forstmt = "for" (declaration | assignstmt) "," compcond "," assignstmt "do"
retstmt = "ret" [expression]

compcond = condition | condition ("and" | "or") compcond
condition = cond | negcond
negcond = "not" cond
cond = expression RELATION expression
RELATION = "<" | "<=" | "=" | "<>" | ">=" | ">"
b)~

$ verify if a number is prime

decl n int
decl div int := 3

verif n / 2 != 0 then:
 ret false

whilst div * div < n do:
 verif n / div != 0 then:
  ret false

 div :+ 2
end
```

```
ret true

~
```