**Lexic.txt**

Alphabet:

a. Upper (A-Z) and lower case letters (a-z) of the English alphabet

b. Decimal digits (0-9);

Lexic:

a.Special symbols, representing:

- operators + - \* / // % ?: < <= == >= != ^ += -= \*= !

- separators [ ] {} () : ; $ ## `

- reserved words: true, false, def, int, float, string, read, print, fun, if, else, while, exit, GO, BYE

b.identifiers

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

identifier ::= letter{letter|digit}

letter ::= “a” | “b” | … | “z” | "A" | "B" | . ..| "Z"

digit = "0"|nzdigit

nzdigit = "1"|...|"9"

c.constants

1.integer:

int = "0"|["+"|"-"]nzdigit{digit}

2. string

special\_char ::= “.” | “ ” | “,” | ":" | ";" | “’”

char:='letter'|'digit'|’special\_char’

constchar:=`string`

string:=char{string}

char:=letter|digit|special\_char

3. float

float = int | int “,” digit {digit}

4. boolean

boolean = „true” | „false”

**Syntax.in**

program ::== “GO” {decllist | stmtlist} “BYE”

decllist ::= declaration | declaration decllist

declaration ::= “def” IDENTIFIER ":" type [“=” expression] “;”

type1 ::= "boolean" | "string" | "int" | "float"

arraydecl ::= "[" type1 "]"

type ::= type1|arraydecl

stmtlist ::= stmt | stmt stmtlist

stmt ::= simplstmt | structstmt

simplstmt ::= assignstmt | iostmt

assignstmt ::= IDENTIFIER ("=" | “+=” | “-=” | “\*=”) expression “;”

expression = number\_expression | string\_expression | ternary\_expression | BOOLEAN

number\_expression ::= number\_expression ("+ | “-") term | term

term ::= term ("\*" | “/” | “%” | “//” | “^”) factor | factor

factor ::= "(" number\_expression ")" | IDENTIFIER | INTEGER | FLOAT

string\_expression = STRING | IDENTIFIER | “`”{“$” IDENTIFIER “$” | CHAR} “`”

ternary\_expression = condition “?” expression “:” expression

iostmt ::= "read" “(“ IDENTIFIER ”)” “;”| "print" "(" string\_expression ")" “;”

structstmt ::= ifstmt | whilestmt | exitstmt

body ::= “{” stmtlist “}” | stmt

ifstmt ::= "if" “(” condition “)” body ["else" body]

whilestmt ::= "while" “(” condition “)” body

exitstmt ::== “exit” expression “;”

condition ::= expression RELATION expression

RELATION ::= "<" | "<=" | "==" | ">=" | ">" | “!=”

**tokens.in**

+

-

\*

/

//

%

^

==

<=

>=

<

>

=

!=

+=

-=

\*=

!

{

}

[

]

(

)

;

?

:

`

$

##

true

false

def

int

float

string

read

print

fun

if

else

while

exit

GO

BYE

**Lab1a – updated**

## Program p1 : find the squared hypothenuse, knowing the legs

GO

def x : float;

def y : int = 7;

def z : int = 1;

x = y^2 + z^2;

print(`The squared hypothenuse is $x$`);

BYE

## Program p2: check if number is prime

GO

def number: int;

def isPrime: boolean = true;

print(`Add your number: `);

read(number);

def d : int = 2;

while(d <= n/2) {

if(n % d == 0) isPrime = false;

d += 1;

}

print(isPrime == true ? `$number$ is prime.` : `$number$ is not prime.`);

BYE

## Program p3: arithmetic mean of unknown number of numbers

GO

def numbers : [int] = [];

def input : int = -1;

def sum : int = 0;

def no : int = 0;

print(`Input numebrs, type 0 to stop:`);

while(input != 0) {

print(`New number: `);

read(input);

sum += input;

no += 1;

}

def result : float = sum//no;

print(`Result $result$`);

BYE