Lab 1b

Language Specification:

1. Language Definition
   1.1 Alphabet:

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

   b. Underline character '_';

   c. Decimal digits (0-9);

   Lexic:

   a. Special symbols, representing:
      - Operators:
        Arithmetic: +, -, X*, /
        Assignment: ←
        Equalitiy testing: = , !=
        Increment and decrement: +, -
        Order relations: <, <=, >, >=
      - Separators: {}, space
      - Reserved words:
        array, input, justFor, while, do, onlyIf, soElse, prettyPrint, int, float

   b. Identifiers
      -a sequence of letters and digits, such that the first character is a letter; the rule is:
      identifier = letter{alphanumeric}
      alphanumeric = letter | digit
      letter = "A" | "B" |…| "Z" | "a" | "b" |… | "z"
      digit = "0" | "1" | … | "9"
      zero_digit="0"
      non_zero_digit= "1" | … | "9"
      sign = ["+" | "-"]
      comma = ","

   c. Constants
      1. Integer:

         Integer = zero_digit | sign non_zero_digit {sign}

      2. Character:
         Character = 'letter'|'digit'
      3. Float:
         Float = sign digit {digit} [comma digit {digit}]

   Syntax:

The words – predefined tokens are specified between “ and “:

a. Syntactical rules:

- Program = decllist cmpdstmt

- Cmpdstmt = stmtlist
- Stmtlist= stmt | stmt “\n” stmtlist
- Stmt = simplestmt | structstmt

- Decllist = declaration | declaration “/n” decllist
- Declaration = primarytypes identifier
- Arraydeclaration = primarytypes “[]” identifider

- Type = primarytypes | arraytype
- Primarytypes = “int” | “float” | “char”
- Arraytype = primarytypes “[“ nr “]”

- Simplstmt = assignstmt | inputstmt
- Structstmt = ifstmt | forstmt

- Assignstmt = identifier “←” expression

- Arrayaccess = identifier “[“ nr | identifier“]”

- Inputstmt = “input” “(“ identifier “)”
- Outputstmt = “prettyPrint” “(“ identifier “)”

- Ifstmt = “onlyIf” “(“ condition “)” “{“ stmt “}” [“soElse””{“ stmt “}”]
- Forstmt = “justFor” “(“ assignment; condition; increment “)” “{“ stmt “}”
- Condition = expression RELATION expression
- Expression = expression sign term |term
- Term = term md factor | factor
- md = “X*” | “/”
- Factor = expression | identifier
- Sign = “+” | “-“
- RELATION = “<” | “<=” | “=” | “!=” | “>=” | “>” | “&” | “||”

Codification table:

Token type	Code
Id	0
Const	1
[	2
]	3
{	4
}	5
<	6
>	7
<=	8
>=	9
+	10
-	11
=	16
!=	17
&	18
||	19
X*	20
/	21
%	22
Input	23
prettyString	24
Int	25
Float	26
Char	27
onlyIf	28
soElse	29
justFor	30
Void	31
←	32
;	33
While	34
Do	35
(	36
)	37

Lab 1a

```
//p1
// compute the max of 5 numbers

Int n1
Int n2
Int n3
Int n4
Int n5
Int result
Int[5] array

array[5] <- []

n1 <- input("Number 1:")
array[0] <- n1
n2 <- input("Number 2:")
array[1] <- n2
n3 <- input("Number 3:")
array[2] <- n3
n4 <- input("Number 4:")
array[3] <- n4
n5 <- input("Number 5:")
array[4] <- n5

result <- 0

justFor (i <- 0 while i < 5 do i+)
```

```
{
        onlyIf (array[i] > result )
                {result <- array[i]}
}

prettyPrint(result)
```

```
//p2
// solve 2 * x^2 + x + 4 = ?

Float x
Float x_squared
Float solution

x <- input("Enter value for x:")

solution <- 0

x_sqared <- x X* x

solution <- 2 X* x_sqared + x + 4

prettyPrint(solution)

//p3
//lexical error
```

```
Float x

Float x_squared

Float solution


x <- input("Enter value for x:)   //missing “


solution <- 0


x_sqared <- x * x   // the operator should be X* , not just *


solution <- 2 * x_sqared + 3x + 4


prettyPrint(solution)
```