Design an original programming language using YACC/BISON. **The code associated with the grammar rules must be written in C++**

1) (3.5pt) Your language should should include

- type declarations **(0.75pt)**:
    - o predefined types (int , float, char, string, bool),
    - o array types
    - o user defined data types (similar to classes in object oriented languages, but with your own syntax):
        - ▪ provide specific syntax to allow initialization and use of variables of user defined types
        - ▪ provide specific syntax for accessing fields and methods
- variable declarations/definition, constant definitions, function definitions **0.25pt;**
- control statements (if, for, while, etc.), assignment statements **0.25;**
  assignment statements should be of the form: *left_value = expression* (where left_value can be an identifier, an element of an array, or anything else specific to your language)
- arithmetic and boolean expressions **1.5pt**
  The values of boolean expressions are *true* and *false*.
- function calls which can have as parameters: expressions, other function calls, identifiers, constants,etc. **0.75pt**

➔ Your language should include a predefined function *Eval(arg)* (*arg* can be an arithmetic or boolean expression, a float, bool or int variable/ literal) and a predefined function *TypeOf*(*arg*)

➔ Your programs should be structured in 4 sections as follows: 1) a section for user defined data types 2) a section for global variables , 3) a section for global function definitions, 4) a special function representing the entry point of the program

2) (1.5pt) create a symbol table for every input source program in your language, which should include:

a) information regarding variable or constant identifiers (type, name, value, the function or the class where it is defined) **0.75 pt**
b) information regarding function identifiers (name, the returned type, the type of each formal parameter, the class in which the identifier is defined) - **0.75 pt**

The symbol table should be printable in a separate text file

3) (2pt) provide semantic analysis and check that:

a) any variable that appears in a program has been previously defined and any function that is called has been defined **0.25**
b) a variable should not be declared more than once; **0.25**
c) all the operands in the right side of an expression must have the same type (the language should not support casting) **0.5**
d) the left side of an assignment has the same type as the right side (the left side can be an element of an array, an identifier etc) **0.5**
e) the parameters of a function call have the types from the function definition **0.5pt**

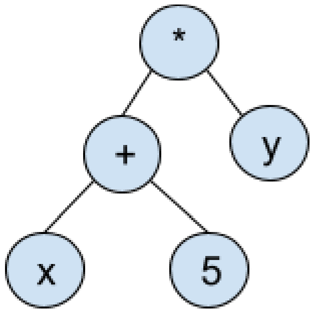
Detailed error messages should be provided if these conditions do not hold (e.g. which variable is not defined or it is defined twice and the program line);

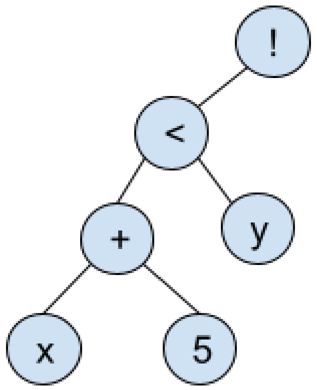
4) (3pt) Evaluation of arithmetic expressions and boolean expressions, type evaluation

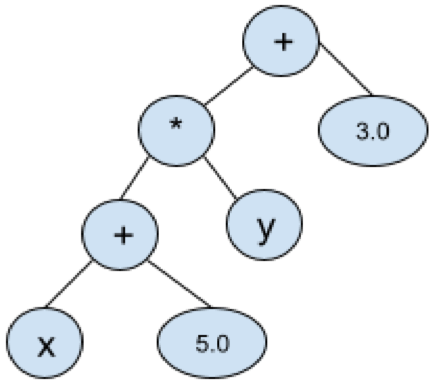
a) **(2.5)** The evaluation of arithmetic expressions and boolean expressions using Abstract Syntax Trees (AST):
Abstract Syntax Trees for expressions are intermediary representations built during the parsing of an expression. The AST will be evaluated to produce the value of the expression. The AST should store, as well, the type of the expression. An AST for an expression is a tree such that:

- if the expression is a number (float, int), a boolean value, an identifier, an element of a vector, a function call, a class member etc (anything that can appear as an operand in an expression), the tree has only one node.
- if the expression has the form *expr1 op expr2* , the AST has the root labeled with the operator *op*, the left subtree is the tree corresponding to *expr1* and the right subtree is the tree for *expr2*.

- if the the expression has the form *op expr1* (*op* is a unary operand) the root is labeled with *op* and has only a left subtree corresponding to *expr1*







For ASTs:

- write a class representing an AST
- write a member function / functions that evluate the AST. You need to deal with different return values (float, int, boolean)!
    - if the root has no children and is labeled with:
        - a float, an , a boolean value: return the value
        - an id: return the value of the identifier
        - an element of the vector: return the corresponding value
        - anything else: return 0
    - else (*ast* is a tree with the root labeled with an operator):
        - evaluate the subtrees
        - combine the results according to the operation in the root

For every call of the form *Eval(expr)* in a programme in your language, the AST for the expression will be evaluated and the actual value of *expr* will be printed.
Also, for every assignment instruction *left_value = exp*r (left_value is an identifier or element of an array with int type), the AST will be evaluated in order to compute the *left_value*

b) **(0.5)** Implement a function TypeOf that will return the type of an expression
Remark: TypeOf(x + f(y)) should cause a semantic error if TypeOf(x) != TypeOf(f(y)) (see 3(c) above)

Remark: Besides the homework presentation, students should be able to:

- answer specific questions regarding grammars and parsing algorithms or yacc/bison details related to the second part (the answers will also be graded).
- modify the grammar/the code

Deadline: week 13 (week 14 for students who missed one laboratory on 30.11 or 1.12)