We built an original programming language from scratch. We have also designed a compiler that can do syntactic and semantic analysis on a code written in our language.

Our language includes:

- **type declarations**:
    - predefined types (int , float, char, string, bool)
    - array types
    - user defined data types
    - specific syntax to allow initialization and use of variables of user defined types
    - specific syntax for accessing fields and methods
    - variable declarations/definition, constant definitions, function definitions
    - control statements (if, for, while, etc.), assignment statements
    - 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
    - function calls which can have as parameters: expressions, other function calls, identifiers, constants

  The language also includes a predefined function *Eval(arg)* (arg can be an arithmetic expression, variable or number) and a predefined function *TypeOf*(*arg*)
  The code should be structured in 4 sections: a section for global variables, a section for functions, a section for user defined data types and a special function representing the entry point of the program

- Specific **symbol tables** for every input source program, which include:

    - information regarding variable or constant identifiers (type, value)
    - information regarding function identifiers (the returned type, the type and and name of each formal parameter)

  The symbol tables are printed in two files:
  symbol_table.txt and **symbol_table_functions.txt** (for functions)

- **Semantic analysis** that checks the following:

    - any variable that appears in a program has been previously defined and any function that is called has been defined
    - a variable should not be declared more than once
    - all the operands in the right side of an expression must have the same type (the language should not support casting)
    - 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)
    - the parameters of a function call have the types from the function definition

  Detailed error messages are provided if these conditions do not hold (e.g. which variable is not defined or it is defined twice and the program line);
  A program in doesn't execute if there are any semantic or syntactic errors!

- **Eval** and **TypeOf** predefined functions:
    - TypeOf(x + f(y)) causes a semantic error if TypeOf(x) != TypeOf(f(y))
    - In order to implement Eval, abstract syntax trees (AST) are used