Definition of the programming language:

The proposed functional programming language is based on three representable values: Integers **Eint**, Boolean **Ebool**, and **Closure**. Closure is a functional abstraction used in order to deal with functions as first-class objects, and is made of the function name, the function parameters, and the environment of declaration time.

The environment represents the binding at runtime of identifier of variables and values of variables, so is made of a list of couples (ide, value) where the value is polymorphic, so can assume every representable value.

The expressions accepted by the interpreter are definition of a new variable in the environment, simple binary operations between integers, if then else branch statements, definition of functions, and calls of function. Here is a list of basic operation present in the language:

- **Let(id,e1,e2)** is used to add to the current environment the couple (id, e1) and evaluate the expression e2 inside the new environment.

- **If(e1,e2,e3)** represent an if then else statement where e1 is the guard, e2 is the branch taken in case the guard is true and e3 is the branch taken in the else case.

- **Binop(op,e1,e2)** represent a binary operation between two integer values, the values are represented by e1 and e2 and op can assume one of these values: Sum, Minus, Times, Equal, Less, Greater.

- **Fun(ide,exp)** represent a function with function name **ide** and the function body **exp.**

- **Call(fun, fun_arg)** represent a function call made of the function **fun** and the argument **fun_arg.**

The interpreter was extended with a **TEE for mobile code**, where mobile code represent code that comes from the network and which execution can lead to security problems such leakage or corruption of protected data. The trusted environment function is to prevent the execution of dangerous mobile code that expose code declared as protected for certain operations, ensuring at the same time a correct evaluation for mobile code expressions that no leads to any security policy violation.

- The set of accepted expression is extended with the expression **LocalCode(myCode, Flags)** which identifies the local code **myCode** that we want to protect against inappropriate execution of mobile code. **Flags** is a list of flags where each one is a specification of protection of local code against a specific method.

- In order to simulate interaction with the network the set of expression is extended with some proxy network-based expressions **(Read(exp), Send(exp), Write(exp))** that evaluates if and only if none of the protected codes exploited in the expression **exp** has a protection flag against the specific network-based method.

- The expression **Execute(exp)** evaluates the expression **exp** inside the TEE and represent the expression used to evaluate mobile code with respect to our security policy. When mobile code is executed all the **LocalCode** and **Network-based methods** included inside **exp** are taken into account in order to check if there is a security policy violation.

Examples about the functioning of programming language:

```
let env0 = [];;

let envT = [];;

let e1 = Call(Fun("y", Binop(Sum, Den "y", Eint 1)), Eint 0);;
eval env0 envT e1;;
#yelds 1

let e2 = Call(Fun("y", Binop (Sum, Eint 1, Den "y")), Eint 1);;
eval env0 envT e2;;
#yelds 2

let e3 = Call(Let("x", Eint 2, Fun("y", Binop(Sum, Den "y", Den "x"))), Eint 1);;
eval env0 envT e3;;
#yelds 3

let e4 = Call(Fun("x", If (Binop(Equal ,Den "x", Eint 1), Binop(Sum, Den "x",Eint 2),
Binop(Sum,Den "x", Eint 3))),Eint 1);;
eval env0 envT e4;;
#yelds 4
```

Examples about protected data and mobile code evaluation:

- **myAge** is an unprotected data
- **myRealAge** is a LocalData protected from mobile code execution of a write operation.

let **myAge** = Eint 32;;
let **myRealAge** = LocalData(23,[NoWrite]);;

These expressions are both all evaluated without troubles: the Execute method is not called at all, so we are executing code locally.

eval env Tenv (let x = **myAge** in **send**(x))
eval env Tenv (let x = **myRealAge** in **write**(x))

These two are also both evaluated because the Execute is called with respect to security policies of used LocalData: myAge and myRealAge are both not protected against send method.

eval env Tenv (**Execute**(let x = **myAge** in **send**(x)))
eval env Tenv (**Execute**(let x = **myRealAge** in **send**(x)))

These two expressions fail cause of invalid execution of mobile code: myRealAge is protected against write operation so the execution is stopped by the TEE.

eval env Tenv (**Execute**(let x = **myRealAge** in **write**(x)))

eval env Tenv (**Execute**(**Execute**(let x = **myRealAge** in
write(Call(Fun("y",Prim("+",Den"y",Eint 10)),(x)))))