# LOO project

This project aims to implement an Abstract Syntax Tree of a micro language (LOO language).

## LOO language

### *Example*

let

a := 42

in

let

b := if a = 0 then 42 else 51

res := a + b

in

if (a + b = 93) = 0 then

res := -1

else

res := 1

end

res

end

When this code is executed, the following value is displayed : 1.

### *Grammar*

<program> ::= <statement>

<statement> ::= <integer>

| <variable>

| <variable> := <statement>

| if <statement> then <statement> else <statement>

| let <declaration>+ in <statement>+ end

<declaration> ::= <variable> := <statement>

<op> ::= + | - | \* | /

| < | =

### *Types*

Only two types exist in LOO language : Integer for statements that have an int value, and Void for instructions that cannot have any value.

Rules are defined as follow :

* Program

<program> ::= <statement>

Statement must be an int value.

* Litteral

<statement> ::= <integer>

Result is an int value.

* Variable

<statement> ::= <variable>

As variables are only Integer types, this statement is an int.

Variable must have been defined before.

* Assignment

Value assigned must be an int. Assignment is an instruction of type Void.

* Binary operations

<statement> ::= <statement> <op> <statement>

Both left and right operands must be int. Result is an int.

* Instruction and conditional statements

<statement> ::= if <statement> then <statement> else <statement>

Two operations are hidden behind this syntax. If both operations (from then and else) are statements, then resultat is Void.

If both of them are an int, then result is an int too.

Other cases lead to errors.

* Scope

<statement> ::= let <declaration>+ in <statement>+ end

No restriction on statements (between in and end).

Type is the one that is related to the last statement.

* Declarations

<declaration> ::= <variable> := <statement>

A variable must not have been defined in current scope, statement must be an int.

## Goal of LOO project

No parser will be developed in this project.

The user does not enter any LOO code : those LOO codes are defined in my code, and user chooses one of these for test purposes.

## Environment

### *Definition*

An environment will be defined in this project to register symbols.

This structure is the same as an associative array with operations dedicated to manage scopes.

It is generic : not dependent on key and value types.

### *Example*

class LOO\_ENVIRONMENT[KEY->HASHABLE, VALUE]

feature

-- Enter a new scope : previous variables are temporary hidden.

push is ...

-- Previous scope is left.

pop is ...

insert (k: KEY, v: VALUE)

-- k must be defined.

set (k:KEY, v: VALUE)

has (k: KEY) : BOOLEAN

-- k must exist in this environment.

get (k: KEY) : VALUE

end

### *Role*

Role of environment can be understood thanks to the following example :

let

a := 1

in

let

a := 2

in

a

end

a

end

Result is 1 and not 2.