LAMBDA INTERPRETER

This is a brief manual that show the main feautres added to this interpreter.

Author: Iago Fernandez Picos

Year : 2020

Subject: DLP

NOTE: All the changes, except the 3.4 has been implemented in lmabda-2

3.1 Direct Recursive Functions

Added a new term type:

```
TmRapp  of string * term * term
```

So now it will acept things like:

```
letrec sum = lambda n. lambda m. if iszero n then m else succ (sum (pred
n) m) in  sum 1 34
```

And the result will be

```
35
```

3.2 Add Context vor variables

This features makes able the interpreter to assing a value to a variable. For that there is a Hashtbl string, term that will hold all the assignations make during the execution of the interpreter.

For that you'll only need to do `id = Lx.x`. The interpreter when evaluates the term id it will return the value.

```
>> id = Lx.x
(lambda x. x)
>> id true
true
>>
```

To achieve that, now the interpreter will have the type instruction:

```
type instruction =
    TmAssigment of string * term
  | TmEvaluation of term
;;
```

Also a new functino has been created:

```
let execute_ctx ctx inst = match inst with
  TmEvaluation tm ->
    print_endline( string_of_term(eval ctx tm) );
     ctx
  | TmAssigment(key, value) ->
    print_endline(string_of_term (eval ctx value));
    Hashtbl.add ctx key value ; ctx
;;
```

When recieves an input of somenthing like `string = value` It will evaluate the value and add it to the context and return the context. If you create another variable with the same name, it will create a new one with the same same, but the program always retrieve the most recent by default.

If recieves just a term it will evaluate the term and return the context without changes.

3.3 Add term tuple

Added a new type of term: `TmTuple of term * term`.

The interpreter accepts things like `1,2`. There are some examples:

```
>> t = 1,2
Tuple:{1, 2}
>> pred t
Tuple:{0, 1}
>> iszero (pred t)
Tuple:{true, false}
```

3.4 Add type string to lambda-3

Only accpets strings in lowercase and has to be surrounded by simple quotes.

Some examples:

```
>> (Lx:String.x) ('hello world')
'(hello world)' : String
>> iszero ('hello')
type error: argument of iszero is not a number
```

```
>> 'hello'
'hello' : String
```