Bapl final project

To run an input file:

`lua interpreter.lua input`

Interactive mode:

`lua interpreter -i`

Read TESTS.md for how to run the tests

Tests inputs are in test/inputs

Check the examples directory

Final Project Report: constanta

Language Syntax

In this section, describe the overall syntax of your language.

New Features/Changes

Selene language with some modifications.

Variables names with types:

`"varname"_"type"`

```
_e - empty
_b - boolean
_n - number
_s - string
_f - function
_t - table
```

Example variables:

```
temp_b = false
i_n = 1000
str_s = "tes"
nr_n = 10
```

Example functions:

```
function factorial_n (nr_n) {
    if nr {
        return nr * factorial(nr-1)
    } else {
        return 1
```

```
    }
}

function main_n () {
    input_n = 6
    test_n = factorial(input)
    return test
}
```

The _type annotation is only used on assignement or function parameters. After assign the variable can be used without the annotation.

StackApi This one is inspired by forth language.

Language supports a stack api Any number of stacks can be created (each having a name). 2 stacks are created at initialization: "default" and "temp"

API as language statements:

```
stack operations:
PUSH  element            : push a string or number
FPUSH "filename"         : push the content of the file on top of stack
POP, DEPTH, PRINT, PEEK
```

Stack juggling: notation meaning (initial stack - after operation stack)

```
DROP     : ( n - )
DUP      : ( n - n n )
SWAP     : ( n1 n2 - n2 n1 )
OVER     : ( n1 n2 - n1 n2 n1 )
TUCK     : ( n1 n2 - n2 n1 n2 )
ROT      : ( n1 n2 n3 - n2 n3 n1 )
MINROT   : ( n1 n2 n3 - n3 n1 n2 )
2DROP    : ( n1 n2 - )
2DUP     : ( n1 n2 - n1 n2 n1 n2 )
2SWAP    : ( n1 n2 n3 n4 - n3 n4 n1 n2 )
2OVER    : ( n1 n2 n3 n4 - n1 n2 n3 n4 n1 n2 )
2ROT     : ( n1 n2 n3 n4 n5 n6 - n3 n4 n5 n6 n1 n2 )
2MINROT  : ( n1 n2 n3 n4 n5 n6 - n5 n6 n1 n2 n3 n4 )
```

API to create stacks:

```
SUSE "name"     - change the current stack
SADD "name"     - add stack
SRM "name"      - stack remove
SREP "name"     - stack replace
SCLEAR "name"   - stack clear
SRA             - remove all stacks
```

Arithmetics:

Expects 2 elements on the current stack, pops the 2 elements and push the result.

```
ADD - addition        - ( n1 n2 - sum )
SUB - substraction    - ( n1 n2 - diff )
MUL - multiplication  - ( n1 n2 - prod )
DIV - division        - ( n1 n2 - quot )
MOD - mod             - ( n1 n2 - rem )
```

Can be used as a reverse polish notation calculator.

```
RPNEVAL "rpn_ops"    - reverse polish notation evaluation
Expects a rpn expression (applies all the arithmetic ops using the current stack)
example: RPNEVAL "1 2 + 3 +" will leave the result 6 at the top
```

Simple IO:

```
PRINT  - print the top of current stack
SPRINT - print the content of the current stack
INPUT  - read from stdin and put the value on the top of the current stack
```

Lua evaluation:

```
EVAL - expects a valid lua code on the top of the current stack,
evaluates the top of stack element using lua (load)
Remove the tos and add the result
Used in combination with PUSH and FPUSH.
```

Accessing stack elements from variables:

```
test_s = "$default'tos"
The variable will receive the data from the default stack (tos = top of stack)

test_n = "$default'1"
Varibale will receive the data from default stack index 1
```

In this section, describe the new features or changes that you have added to the programming language. This should include:

Detailed explanation of each feature/change

- Optional type system
- The stack operations described above
- Absence of value (nil) (example: a_e, a_s etc)
- Unless control structure
- Unit tests using bash bats tool (this saved me a lot of times when i created regressions)
- Exp supports bool and strings
- @ can print strings
- Remove semicolons from syntax
- Bash like comparison operators (lt, gt, le, ge, eq, ne)
- Forward function declarations and error check for multiple declarations

Type checks on: assignment,if,function call return, function call params. If you dont use _type when declaring variables no type checks are performed.

Any trade-offs or limitations you are aware of

- Type checks errors could use a better format.
- I didnt add the optional initialization for last function parameter, i dont like that feature.
- Strings dont have escape sequences and concatenation yet
- Not enough error checking everywhere (more time needed)
- Multidimensional array initialization is not included
- I didnt have a clear vision of what this language will be (learning experience)
- Bugs maybe

Future

In this section, discuss the future of your language / DSL, such as deployability (if applicable), features, etc.

What would be needed to get this project ready for production?

- I really like this syntax, to be production ready i need to write a transpiler to a different language like c,java.
- Better error reporting.
- Some kind of standard library

How would you extend this project to do something more? Are there other features you'd like? How would you go about adding them?

- Would like to use this language for scripting purposes.Have some apis for process management.
- Keep the language simple to use, type system is optional.
- I have this idea of using sqlite3 db as files and main data structure. For example this statement:
- CREATEDB "AdressBook FirstName_s LastName_s HomePhone_n WorkPhone_n Adress_s"
- could be used to create an AdressBook table with proper types.
- Then the language will have a syntax to use the AdressBook.

Self assessment

Self assessment of your project: for each criteria described on the final project specs, choose a score (1, 2, 3) and explain your reason for the score in 1-2 sentences.

- Language Completness 2
  - I added booleans, Absence of values, Unless statement aditionally Strings, Type system to a certain degree
- Code Quality and Report 2
  - Code organization is ok from my point of view. Keep it simple and readable was my goal

- Originality & Scope 2
  - I like the current syntax and i have the feeling other people will like it too.

I learned a lot and given the time i had available i am proud of the result.

Have you gone beyond the base requirements? How so?

- Start of a type system.
- The stack api.
- Proper unit tests that check the output of the program (80 input files).
- Organization of the code.
- Strings,bools
- Examples directory (i used some problems from https://projecteuler.net)

References

List any references used in the development of your language besides this courses, including any books, papers, or online resources. - Lua book - Crafting interpreters book - https://www.forth.com/starting-forth/2-stack-manipulation-operators-arithmetic/ - https://projecteuler.net - https://www.complang.tuwien.ac.at/forth/gforth/Docs-html/Data-stack.html - https://mathworld.wolfram.com/ReversePolishNotation.html