Pseudo Assembler interpreter

Linguaggi e Traduttori

1 Introduction

The **Pseudo Assembler interpreter** is an interpreter that takes as input a program similar to the assembler program language and executes it.

It is a very simplified language, where are present only basic types (i.e. *integer* and *double*) and arrays of one dimension. Also the instruction set is reduced to the evaluation of **expressions** (i.e. *arithmetic*, *comparison* and *boolean*), to **jump** and **assignment**.

There is not present error checking (both lexical and semantic), because input program is supposed to be correct.

2 The language

The language allows an instruction per line, each instruction begins with the **instruction name**, followed by one ore more **arguments**.

```
NAME args
```

A program begins with variables declarations and it is followed by a code section. The only possible I/O instruction is to print the value of a variable.

Comments are possible with a C syntax (`/* <comment> */`).

2.1 Declaration instructions

```
INT <varname>
DOUBLE <varname>
```

Two types of variable are present, namely, **integer** and **double**. The declaration of a variable can be done with the instructions: `INT` and **DOUBLE**, followed by the name of the variable to declare.

Example:

```
INT a
DOUBLE b
```

`a` is declared as an integer variable, while `b` is declared as a double variable.

The declaration of arrays, such as their usage, follow the C syntax, with the exception that only mono-dimensional arrays can be declared and used.

Example:

```
INT a[10]
DOUBLE b[2]
```

`a` is an array of 10 `integer` elements, while `b` is an array of 2 `double` elements.

2.2 Expression instruction

```
EVAL <expression>
```

Compute the result of an expression. The result of the last `EVAL` function is stored by the interpreter. The value can then be used by the `ASS` assignment or `GOTO`, `GOTOT`, `GOTOF` jump instructions.

¡expression¿ is a list of operators and operands separated by spaces. Expressions are written with a **Reverse Polish notation** syntax, where operands are written before the operators. Operators can be **arithmetic** (i.e., $+$, $-$, $*$, $/$), **comparison** (i.e., <, >, >=, <=, ==) and **boolen** (i.e., &, |, !).

Example:

```
EVAL 3 2 + 5 * 6 -
EVAL 3 2 + 2 3 - > & 0
```

The two equations of the example are the following:

```
(3 + 2) * 5 - 6
((3 + 2) > (2 - 3)) & 0
```

For the *boolean* operators, a `FALSE` operand is the number 0, while a `TRUE` operand is an integer number different from 0. Inside `EVAL` expressions can be used both variables and vectors with a C-like syntax. Vectors can be indexed using an integer number or a variable (i.e., `a[3]`, `a[b]`), expressions **can not** to index a vector.

Example:

```
EVAL b 2 +
EVAL a[3] 2 +
```

2.3 Jump instructions

```
GOTO <label>
GOTOT <label>
GOTOF <label>
```

- `<label>` is an identifier.
- `GOTO` jump instruction does a jump to the position of label `<label>`.
- `<GOTOT>` jump instruction does a jump to the position of label `<label>` if the last `EVAL` instruction has a `TRUE` result (i.e., a number that is not 0).
- `<GOTOF>` jump instruction does a jump to the position of label `<label>` if the last `EVAL` instruction has a `FALSE` result (i.e., the 0 number).

  Example:

  ```
  EVAL 3 2 >
  GOTOT L1
      EVAL 2 3 +
  L1: EVAL 4 5 +
  ```

  The result of `EVAL 3 2 >` is 1 (i.e. `TRUE`), so `GOTOT L1` jumps to `L1:` and the instruction `EVAL 4 5 +` is executed.

2.4 Assignment instruction

```
ASS <var_name>
```

The assignment instruction `ASS` assigns the result of the previous `EVAL` instruction to a variable name (i.e. `<var_name>`. `<var_name>` can be a variable or a mono-dimensional vector.

2.5 Print instruction

```
PRINT <var_name>
```

The print instruction `PRINT` prints the value of a variable or a vector.

3 Usage

To run the interpreter type:

```
java -jar interpreter.jar <filename> (<debug_level>)?
```

where `<filename>` is an input file written in the `Pseudo Assigns` syntax.

For instance with the command:

```
java -jar interpreter.jar bubble.asm
```

the file `bubble.asm` is executed without any debugging information.

If the input file `bubble.asm` is:

```
DOUBLE x[5]
INT i
INT j
DOUBLE swap
INT pos
EVAL -2.0
ASS x[0]
EVAL -3.0
ASS x[1]
EVAL 3.0
ASS x[2]
EVAL 5.0
ASS x[3]
EVAL 2.5
ASS x[4]
EVAL 5
ASS pos
L1: EVAL pos 0 >  /* while (line 18) */
GOTOF L2
EVAL 0
ASS i
L3: EVAL i pos 1 -  <  /* while (line 20) */
GOTOF L4
EVAL i 1 +
ASS j
EVAL x[i] x[j] >  /* if (line 22) */
GOTOF L5
EVAL x[j]
ASS swap
EVAL x[i]
ASS x[j]
EVAL swap
ASS x[i]
L5: EVAL i 1 +
ASS i
GOTO L3
L4: EVAL pos 1 -
ASS pos
GOTO L1
L2: EVAL 0
ASS i
L6: EVAL i 5 <  /* while (line 35) */
GOTOF L7
PRINT x[i]
EVAL i 1 +
ASS i
GOTO L6
L7: END
```

the obtained output will be:

```
-3.0
-2.0
2.5
3.0
5.0
```

3.1 Debug Level 1

```
java -jar interpreter.jar bubble.asm 1
```

Using the `<debug_level>` debugging option equal to 1, the output of the interpreter is:

```
0 Istruz EVAL [-2.0]
1 Istruz ASS [x[0]]
2 Istruz EVAL [-3.0]
3 Istruz ASS [x[1]]
4 Istruz EVAL [3.0]
5 Istruz ASS [x[2]]
6 Istruz EVAL [5.0]
7 Istruz ASS [x[3]]
8 Istruz EVAL [2.5]
9 Istruz ASS [x[4]]
```

```
10 Istruz EVAL [5]
11 Istruz ASS [pos]
12 Istruz EVAL [pos, 0, >]
13 Istruz GOTOF [2]
14 Istruz EVAL [0]
15 Istruz ASS [i]
16 Istruz EVAL [i, pos, 1, -, <]
17 Istruz GOTOF [4]
18 Istruz EVAL [i, 1, +]
19 Istruz ASS [j]
20 Istruz EVAL [x[i], x[j], >]
21 Istruz GOTOF [5]
22 Istruz EVAL [x[j]]
23 Istruz ASS [swap]
24 Istruz EVAL [x[i]]
25 Istruz ASS [x[j]]
26 Istruz EVAL [swap]
27 Istruz ASS [x[i]]
28 Istruz EVAL [i, 1, +]
29 Istruz ASS [i]
30 Istruz GOTO [3]
31 Istruz EVAL [pos, 1, -]
```

```
32 Istruz ASS [pos]
33 Istruz GOTO [1]
34 Istruz EVAL [0]
35 Istruz ASS [i]
36 Istruz EVAL [i, 5, <]
37 Istruz GOTOF [7]
38 Istruz PRINT [x[i]]
39 Istruz EVAL [i, 1, +]
40 Istruz ASS [i]
41 Istruz GOTO [6]
42 Istruz END null

LABEL TABLE:
{3=16, 2=34, 1=12, 7=42, 6=36, 5=28, 4=31}

PROGRAM EXECUTION:
-3.0
-2.0
2.5
3.0
5.0
```

where before the execution of the program are reported both the *list of instructions* of the program and the *label table*. The *list of instructions* is the list of all the instructions of the program with their line numbers. *label table* is a hash that maps the label name with the number of line to jump in the case of the jump is performed.

3.2 Debug Level 2

```
java -jar interpreter.jar bubble.asm 2
```

```
0 Istruz EVAL [-2.0]
1 Istruz ASS [x[0]]
2 Istruz EVAL [-3.0]
3 Istruz ASS [x[1]]
4 Istruz EVAL [3.0]
5 Istruz ASS [x[2]]
6 Istruz EVAL [5.0]
7 Istruz ASS [x[3]]
8 Istruz EVAL [2.5]
9 Istruz ASS [x[4]]
10 Istruz EVAL [5]
11 Istruz ASS [pos]
12 Istruz EVAL [pos, 0, >]
13 Istruz GOTOF [2]
14 Istruz EVAL [0]
15 Istruz ASS [i]
16 Istruz EVAL [i, pos, 1, -, <]
17 Istruz GOTOF [4]
18 Istruz EVAL [i, 1, +]
19 Istruz ASS [j]
20 Istruz EVAL [x[i], x[j], >]
21 Istruz GOTOF [5]
22 Istruz EVAL [x[j]]
23 Istruz ASS [swap]
24 Istruz EVAL [x[i]]
25 Istruz ASS [x[j]]
26 Istruz EVAL [swap]
27 Istruz ASS [x[i]]
28 Istruz EVAL [i, 1, +]
29 Istruz ASS [i]
30 Istruz GOTO [3]
```

```
31 Istruz EVAL [pos, 1, -]
32 Istruz ASS [pos]
33 Istruz GOTO [1]
34 Istruz EVAL [0]
35 Istruz ASS [i]
36 Istruz EVAL [i, 5, <]
37 Istruz GOTOF [7]
38 Istruz PRINT [x[i]]
39 Istruz EVAL [i, 1, +]
40 Istruz ASS [i]
41 Istruz GOTO [6]
42 Istruz END null

LABEL TABLE:
{3=16, 2=34, 1=12, 7=42,
 6=36, 5=28, 4=31}

PROGRAM EXECUTION:
1 EVAL [-2.0]  RES: -2.0
2 ASS [x[0]] -2.0
3 EVAL [-3.0]  RES: -3.0
4 ASS [x[1]] -3.0
5 EVAL [3.0]  RES: 3.0
6 ASS [x[2]] 3.0
7 EVAL [5.0]  RES: 5.0
8 ASS [x[3]] 5.0
9 EVAL [2.5]  RES: 2.5
10 ASS [x[4]] 2.5
11 EVAL [5]  RES: 5
12 ASS [pos] 5
13 EVAL [pos, 0, >]  RES: 1
```

```
14 GOTOF [2]  LABEL: 2 -> LINE: 34
15 EVAL [0]  RES: 0
16 ASS [i] 0
17 EVAL [i, pos, 1, -, <]  RES: 1
18 GOTOF [4]  LABEL: 4 -> LINE: 31
19 EVAL [i, 1, +]  RES: 1
20 ASS [j] 1
21 EVAL [x[i], x[j], >]  RES: 1
22 GOTOF [5]  LABEL: 5 -> LINE: 28
23 EVAL [x[j]]  RES: -3.0
24 ASS [swap] -3.0
25 EVAL [x[i]]  RES: -2.0
26 ASS [x[j]] -2.0
27 EVAL [swap]  RES: -3.0
28 ASS [x[i]] -3.0
29 EVAL [i, 1, +]  RES: 1
30 ASS [i] 1
31 GOTO [3]  LABEL: 3 -> LINE: 16
17 EVAL [i, pos, 1, -, <]  RES: 1
18 GOTOF [4]  LABEL: 4 -> LINE: 31
19 EVAL [i, 1, +]  RES: 2
20 ASS [j] 2
21 EVAL [x[i], x[j], >]  RES: 0
22 GOTOF [5]  LABEL: 5 -> LINE: 28
29 EVAL [i, 1, +]  RES: 2
30 ASS [i] 2
31 GOTO [3]  LABEL: 3 -> LINE: 16
17 EVAL [i, pos, 1, -, <]  RES: 1
18 GOTOF [4]  LABEL: 4 -> LINE: 31
19 EVAL [i, 1, +]  RES: 3
20 ASS [j] 3
```

```
21 EVAL [x[i], x[j], >]  RES: 0
22 GOTOF [5]  LABEL: 5 -> LINE: 28
29 EVAL [i, 1, +]  RES: 3
30 ASS [i] 3
31 GOTO [3]  LABEL: 3 -> LINE: 16
17 EVAL [i, pos, 1, -, <]  RES: 1
18 GOTOF [4]  LABEL: 4 -> LINE: 31
19 EVAL [i, 1, +]  RES: 4
20 ASS [j] 4
21 EVAL [x[i], x[j], >]  RES: 1
22 GOTOF [5]  LABEL: 5 -> LINE: 28
23 EVAL [x[j]]  RES: 2.5
24 ASS [swap] 2.5
25 EVAL [x[i]]  RES: 5.0
26 ASS [x[j]] 5.0
27 EVAL [swap]  RES: 2.5
28 ASS [x[i]] 2.5
29 EVAL [i, 1, +]  RES: 4
30 ASS [i] 4
31 GOTO [3]  LABEL: 3 -> LINE: 16
17 EVAL [i, pos, 1, -, <]  RES: 0
18 GOTOF [4]  LABEL: 4 -> LINE: 31
32 EVAL [pos, 1, -]  RES: 4
33 ASS [pos] 4
34 GOTO [1]  LABEL: 1 -> LINE: 12
13 EVAL [pos, 0, >]  RES: 1
14 GOTOF [2]  LABEL: 2 -> LINE: 34
15 EVAL [0]  RES: 0
16 ASS [i] 0
17 EVAL [i, pos, 1, -, <]  RES: 1
18 GOTOF [4]  LABEL: 4 -> LINE: 31
19 EVAL [i, 1, +]  RES: 1
20 ASS [j] 1
21 EVAL [x[i], x[j], >]  RES: 0
22 GOTOF [5]  LABEL: 5 -> LINE: 28
29 EVAL [i, 1, +]  RES: 1
30 ASS [i] 1
31 GOTO [3]  LABEL: 3 -> LINE: 16
17 EVAL [i, pos, 1, -, <]  RES: 1
18 GOTOF [4]  LABEL: 4 -> LINE: 31
19 EVAL [i, 1, +]  RES: 2
20 ASS [j] 2
21 EVAL [x[i], x[j], >]  RES: 0
22 GOTOF [5]  LABEL: 5 -> LINE: 28
29 EVAL [i, 1, +]  RES: 2
30 ASS [i] 2
31 GOTO [3]  LABEL: 3 -> LINE: 16
17 EVAL [i, pos, 1, -, <]  RES: 1
18 GOTOF [4]  LABEL: 4 -> LINE: 31
19 EVAL [i, 1, +]  RES: 3
20 ASS [j] 3
21 EVAL [x[i], x[j], >]  RES: 1
22 GOTOF [5]  LABEL: 5 -> LINE: 28
23 EVAL [x[j]]  RES: 2.5
24 ASS [swap] 2.5
25 EVAL [x[i]]  RES: 3.0
26 ASS [x[j]] 3.0
27 EVAL [swap]  RES: 2.5
28 ASS [x[i]] 2.5
29 EVAL [i, 1, +]  RES: 3
30 ASS [i] 3
31 GOTO [3]  LABEL: 3 -> LINE: 16
17 EVAL [i, pos, 1, -, <]  RES: 0
18 GOTOF [4]  LABEL: 4 -> LINE: 31
32 EVAL [pos, 1, -]  RES: 3
33 ASS [pos] 3
34 GOTO [1]  LABEL: 1 -> LINE: 12
13 EVAL [pos, 0, >]  RES: 1
14 GOTOF [2]  LABEL: 2 -> LINE: 34
15 EVAL [0]  RES: 0
16 ASS [i] 0
17 EVAL [i, pos, 1, -, <]  RES: 1
18 GOTOF [4]  LABEL: 4 -> LINE: 31
19 EVAL [i, 1, +]  RES: 1
20 ASS [j] 1
21 EVAL [x[i], x[j], >]  RES: 0
22 GOTOF [5]  LABEL: 5 -> LINE: 28
29 EVAL [i, 1, +]  RES: 1
30 ASS [i] 1
31 GOTO [3]  LABEL: 3 -> LINE: 16
17 EVAL [i, pos, 1, -, <]  RES: 1
18 GOTOF [4]  LABEL: 4 -> LINE: 31
19 EVAL [i, 1, +]  RES: 2
20 ASS [j] 2
21 EVAL [x[i], x[j], >]  RES: 0
22 GOTOF [5]  LABEL: 5 -> LINE: 28
29 EVAL [i, 1, +]  RES: 2
30 ASS [i] 2
31 GOTO [3]  LABEL: 3 -> LINE: 16
17 EVAL [i, pos, 1, -, <]  RES: 0
18 GOTOF [4]  LABEL: 4 -> LINE: 31
32 EVAL [pos, 1, -]  RES: 2
33 ASS [pos] 2
34 GOTO [1]  LABEL: 1 -> LINE: 12
13 EVAL [pos, 0, >]  RES: 1
14 GOTOF [2]  LABEL: 2 -> LINE: 34
15 EVAL [0]  RES: 0
16 ASS [i] 0
17 EVAL [i, pos, 1, -, <]  RES: 1
18 GOTOF [4]  LABEL: 4 -> LINE: 31
19 EVAL [i, 1, +]  RES: 1
20 ASS [j] 1
21 EVAL [x[i], x[j], >]  RES: 0
22 GOTOF [5]  LABEL: 5 -> LINE: 28
29 EVAL [i, 1, +]  RES: 1
30 ASS [i] 1
31 GOTO [3]  LABEL: 3 -> LINE: 16
17 EVAL [i, pos, 1, -, <]  RES: 0
18 GOTOF [4]  LABEL: 4 -> LINE: 31
32 EVAL [pos, 1, -]  RES: 1
33 ASS [pos] 1
34 GOTO [1]  LABEL: 1 -> LINE: 12
13 EVAL [pos, 0, >]  RES: 1
14 GOTOF [2]  LABEL: 2 -> LINE: 34
15 EVAL [0]  RES: 0
16 ASS [i] 0
17 EVAL [i, pos, 1, -, <]  RES: 0
18 GOTOF [4]  LABEL: 4 -> LINE: 31
32 EVAL [pos, 1, -]  RES: 0
33 ASS [pos] 0
34 GOTO [1]  LABEL: 1 -> LINE: 12
13 EVAL [pos, 0, >]  RES: 0
14 GOTOF [2]  LABEL: 2 -> LINE: 34
35 EVAL [0]  RES: 0
36 ASS [i] 0
37 EVAL [i, 5, <]  RES: 1
38 GOTOF [7]  LABEL: 7 -> LINE: 42
39 PRINT [x[i]]
-3.0
40 EVAL [i, 1, +]  RES: 1
41 ASS [i] 1
42 GOTO [6]  LABEL: 6 -> LINE: 36
37 EVAL [i, 5, <]  RES: 1
38 GOTOF [7]  LABEL: 7 -> LINE: 42
39 PRINT [x[i]]
-2.0
40 EVAL [i, 1, +]  RES: 2
41 ASS [i] 2
42 GOTO [6]  LABEL: 6 -> LINE: 36
37 EVAL [i, 5, <]  RES: 1
38 GOTOF [7]  LABEL: 7 -> LINE: 42
39 PRINT [x[i]]
2.5
40 EVAL [i, 1, +]  RES: 3
41 ASS [i] 3
42 GOTO [6]  LABEL: 6 -> LINE: 36
37 EVAL [i, 5, <]  RES: 1
38 GOTOF [7]  LABEL: 7 -> LINE: 42
39 PRINT [x[i]]
3.0
40 EVAL [i, 1, +]  RES: 4
41 ASS [i] 4
42 GOTO [6]  LABEL: 6 -> LINE: 36
37 EVAL [i, 5, <]  RES: 1
38 GOTOF [7]  LABEL: 7 -> LINE: 42
39 PRINT [x[i]]
5.0
40 EVAL [i, 1, +]  RES: 5
41 ASS [i] 5
42 GOTO [6]  LABEL: 6 -> LINE: 36
37 EVAL [i, 5, <]  RES: 0
38 GOTOF [7]  LABEL: 7 -> LINE: 42
```

The output is similar to the one reported for debug level 1, but when the program is executed all the executed instructions with their results are reported.