Tema 1 PP

Deadline: 23.03.2016

Responsabili: Vlad-Andrei Ursu, Mihai-Dan Masala

Se da un limbaj de programare, denumit in continuare ***IMP*** definit de urmatoarea gramatica:

```
<expr> ::= [<op> <expr> <expr>] | <symbol> | <value>
<symbol> ::= [a-zA-Z]+
<value> ::= [0-9]+
<op> ::= '+' | '*' | '==' | '<'
<prog> ::= [= <symbol> <expr>] |
                  [; <prog> <prog>] |
                  [if <expr> <prog> <prog>]
                  [while <expr> <prog>]
                  [return <expr>]
```

Observatie: Vazute strict lexical, expresiile si programele primite drept input sunt liste (demarcate prin caracterele **'['** si **']'**), in care elementele sunt separate prin **whitespace**. In cazul expresiilor, primul element reprezinta operatorul, iar restul elementelor – operanzii. Acestia pot fi la randul lor alte liste.

Obiectivele temei

Se cere implementarea unui **interpretor** pentru limbajul IMP in **Java**. Interpretorul primeste ca input, un program IMP, prezentat in forma specificata de gramatica de mai sus.

Cerinte:

1) verificarea corectitudinii inputului (detalii mai jos);
2) calcularea valorii returnate de programul IMP.

Corectitudinea unui program IMP presupune satisfacerea urmatoarelor conditii:

1. Toate variabilele sa fie in **scope** (vezi *exemplele 8 si 10*).
2. Programul se termina cu un program de tip **[return <expr>]** (vezi *exemplul 9*).

Observatie: Verificarea corectitudinii nu presupune examinarea sintaxei. Toate programele folosite pentru testarea temei sunt **sintactic corecte** (respecta gramatica descrisa mai sus).

Detalii input/output si interpretare

Exemplul 1:

[* 2 3]

Expresia din *exemplul 1* *este:* **(2*3).**

Exemplul 2:

[* [+ 1 2] [+ 4 3]]

Expresia din *exemplul 2* este: **(1+2)*(4+3).**

In cazul programelor, primul element reprezinta instructiunea. Restul elementelor reprezinta componentele care sunt specifice fiecarui instructiuni. Acestea pot fi expresii si/sau alte programe.

Exemplul 3:

[; [= x 2] [= y 3]]

Programul din *exemplul 3* este echivalent cu urmatoarea secventa de instructiuni:

```
x=2;
y=3;
```

Exemplul 4:

[; [= x 2] [= x [+ x 1]]

Programul din *exemplul 4* este echivalent cu urmatoarea secventa de instructiuni:

```
x=2;
x=x+1;
```

Exemplul 5:

[; [= x 2] [return x]]

Programul din *exemplul 5* este echivalent cu urmatoarea secventa de instructiuni:

```
x=2;

return x;
```

Valoarea returnata de programul din *exemplul 5* este **2**.

Exemplul 6:

[; [= x 10] [; [if [< x 3] [= x [+ x 2]] [= x [* x 2]]] [return x]]]

Programul din *exemplul 6* este echivalent cu urmatoarea secventa de instructiuni:

```
x=10;
if (x < 3)
  x = x + 2;
else x = x * 2;

return x;
```

Valoarea returnata de programul din *exemplul 6* este **20**.

Exemplul 7:

[; [= x 1] [; [= i 0] [; [while [< i 10] [; [= x [* x 2]] [= i [+ i 1]]]] [return x]]]]

Programul din *exemplul 7* este echivalent cu urmatoarea secventa de instructiuni:

```
x=1; i=0;
while (i < 10)
{
        x = x * 2;
        i = i + 1;
}
return x;
```

Valoare returnata de programul din *exemplul 7* este **1024**.

Exemplul 8:

[; [= y [+ 1 x]] [return y]]

Programul din *exemplul 8* este echivalent cu urmatoarea secventa de instructiuni:

```
y = 1 + x;

return y;
```

Variabila x este **out of scope**! Programul **nu** respecta **conditia 1**.

Exemplul 9:

[; [= y 3] [= x 2]]

Programul din *exemplul 9* este echivalent cu urmatoarea secventa de instructiuni:

```
y = 3;

x = 2;
```

Programul **nu** respecta **conditia 2**.

Exemplul 10:

[; [= x [+ x 1]] [return x]]

Programul din *exemplul 10* este echivalent cu urmatoarea secventa de instructiuni:

```
x = x + 1;

return x;
```

Variabila x este **out of scope**! Programul **nu** respecta **conditia 1**.

Observatie! Testele **nu** vor include exemple de forma: **[if [<x 2] [return 1] [return 2]]** (nu va trebui sa acoperiti cazurile in care exista un return pe mai mult de o ramura de executie). De asemenea, nu vor fi exemple de forma **[= x [= x 2]]**.

Detalii implementare

Tema este insotita de un *schelet minimal,* care contine urmatoarele clase: **Main** si **Context**.

***)** Obiectele de tip **Context** sunt o modalitate de a retine legaturile dintre variabile si valorile lor. In clasa Context exista metoda **“void add(String v, Integer i)”**, metoda care trebuie implementata. In urma apelarii acestei metode, variabila desemnata de stringul **v** este legata la valoarea **i**.

****)** Clasa **Main** contine patru metode statice:

1. *public Integer evalExpression(String expression, Context context)*
2. *public Boolean checkCorrectness(String program)*
3. *public Integer evalProgram(String program)*
4. *public String[] spliList(String s)*

Metoda 1 primeste ca input **o expresie**, codificata sub forma de String si un obiect de tip Context si returneaza valoarea expresiei in contextul dat.

Metoda 2 primeste ca input **un program**, codificat sub forma de String si returneaza **True** daca programul este *corect* (vezi conditiile mentionate mai sus), respectiv **False**, daca cel putin una din conditii nu este satisfacuta.

Metoda 3 primeste ca input **un program corect** (ce satisface conditiile de corectitudine mentionate in enunt) si returneaza valoarea rezultata in urma rularii programului.

Metoda 4 primeste ca input **una sau mai multe liste** separate prin spatiu, codificate ca un String, si returneaza, sub forma de vector de String-uri listele componente. Aceasta metoda este deja implementata(vezi schelet).

IMPORTANT! Testarea temei se bazeaza pe apelarea celor 3 metode specificate. Neimplementarea acestora duce la obtinerea punctajului de 0 puncte. Felul in care aceste metode sunt implementate este decizia voastra, atat timp cat asigura functionalitatea precizata.

HINT! Recomandam definirea unor interfete pentru *expresii* si *programe* si folosirea design pattern-urile **Visitor** si **Decorator** ([1] & [2]). Patternul **Decorator** poate fi folosit

pentru parsarea expresiilor si evaluarea lor. Patternul **Visitor** poate fi folosit pentru verificare codului.

[1] https://en.wikipedia.org/wiki/Visitor_pattern

[2] https://en.wikipedia.org/wiki/Decorator_pattern

Detalii notare

Pentru verificarea temei exista un **set de teste publice** si un **set de teste private**.

Testarea temelor se va face folosind module **JUnit**. Modulul cu setul de teste publice este deja inclus in scheletul minimal.

Impartire punctaj:

1) testele publice valoreaza **50%** din punctajul total al temei(**0.0625 p/test; 8 teste**)
2) testele private valoreaza **50%** din punctajul total al temei**(0.0625 p/test; 8 teste)**

IMPORTANT! Folosirea design pattern-urilor **Visitor & Decorator nu** este imperativa, punctajul fiind acordat pe baza rezultatelor testelor. Totusi, implementarile **“muncitoresti”**, vor fi depunctate simbolic.

Tema se va uploada pe **cs.curs**, in sectiunea corespunzatoare **temei 1.**