Parsare de expresii (Updated 2)

Tema 3

Responsabili: Ștefan Rușeți și Rareș Tăerel

1 Expresii matematice

Se considera un fisier cu expresii aritmetice, formate din

- Operatori binari: **+, -, *, /** ;
- Operanzi: identificatori formati din **maxim 5 litere mici** (exemple: a, val, pret);
- Tipuri de date: **int, double, string;**

Se definește funcția priority(operator), reprezentând prioritatea unui operator în cadrul rezolvării unei expresii.

- priority(*) == priority(/)
- priority(+) == priority(-)
- priority(*) > priority(+)
- priority(/) > priority(-)

În cazul priorității egale, se va evalua de la stânga spre dreapta. În cadrul expresiei pot exista paranteze, acestea modificând modul de evaluare a expresiei și implicit prioritățile.

exemplu:

E = a + b + c * d – e

2 Cerințe specifice

Pentru evaluarea unei expresii se vor construi un arbore de parsare, conform algoritmului descris la [1]. **Pentru evaluare se recomandă utilizarea design pattern-ul visitor. În cazul utilizării unui alt design pattern este necesră argumentarea în readme. Pentru construirea nodurilor se va utiliza design pattern-ul factory, împreună cu singleton.**

În cadrul temei se vor utiliza următoarele convenții:

- Comanda atribuire: “**tip nume_variabile = valoare;**“

- Evaluare expresie: “**eval expresie;**”

Exemplu:

arbore.in	arbore.out

int a = 2;
int b = 4;
int c = 3;
int d = 7;
int e = 1;
int z = 0;
eval a + (b + c) * d - e;
string f = "abc";
eval f + (b + c) * d - e;
double g = 0.14;
eval a + (b + c) * g - e;
string h = "abcd";
eval f * h;
eval f + (a / z);
	50
	abc4
	1.98
	12
	abcNaN

(!) Pentru cazul unei expresii cu un tip de date double, rezultatele se vor afișa rotunjit la două zecimale.

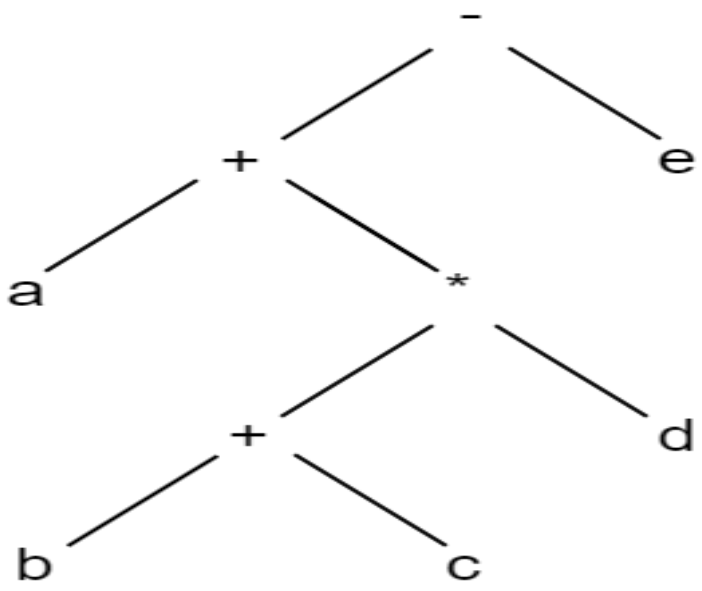

Figura 1

3 Rezultatul operațiilor

+	**Int**	**Double**	**String**
Int	Int	Double	String
Double	Double	Double	String
String	String	String	String

-	**Int**	**Double**	**String**
Int	Int	Double	Int
Double	Double	Double	Double
String	String	Double	Int

*	**Int**	**Double**	**String**
Int	Int	Double	String
Double	Double	Double	Double
String	String	Double	Int

/	**Int**	**Double**	**String**
Int	Int	Double	Int
Double	Double	Double	Double
String	String	Double	Int

4 Definiția operațiilor

Operații numerice (între Int și Double)

Operațiile între valori numerice se vor desfășura la fel ca în Java. Tipul întors de operație va fi cel mai general dintre cele două (conform tabelelor din 3). Împărțirea între 2 întregi va fi împărțire întreagă, iar dacă cel puțin unul dintre operanzi este Double, împărțirea se va face normal.

În cazul împărțirii la 0, rezultatul va fi NaN. Acesta poate fi ori Double, ori Int, în funcție de tipul împărțirii. Orice operație aritmetică cu NaN va avea rezultatul NaN, păstrând tipul pe care operația l-ar întoarce în mod normal.

Operații pe String-uri

- Adunarea între un String și orice alt tip de date va fi făcută ca în Java (concatenare)
- (String a) – (Int b): se vor sterge ultimile *b* caractere din șirul *a*.

  Excepții:
  - b > len(a) => șirul vid
  - b < 0 => a + (-b) * "#" (ex. "abc" - (-2) = "abc##")
  - b = NaN => a
- (String a) * (Int b) sau (Int b) * (String a): șirul a va fi multiplicat de *b* ori.

  Echivalent cu a + a + ... + a (de *b* ori). Dacă b este <= 0 sau NaN, rezultatul va fi șirul vid
- (String a) / (Int b): se vor păstra doar primele *len(a)* / *b* caractere din *a*. Dacă *b* este <= 0 sau NaN, rezultatul va fi *a*.

Orice alta combinație (în afară de adunare) vor converti șirul la Int. Convertirea presupune înlocuirea șirului cu lungimea acestuia.

5 Testare

Tema va fi testată pe **VMChecker** (va apărea în curând), astfel încât va trebui să aveți și un **makefile** cu regulile build, clean și run.

Teste vor fi disponibile spre sfârșitul săptămânii viitoare.

6 Constrângeri

- Pentru realizarea temei trebuie folosită o versiune de Java nu mai recentă de Java 7, aceasta fiind versiunea care rulează pe VMChecker.
- Limita de timp pentru fiecare test în parte va fi publicată mai târziu, odată cu postarea temei pe VMChecker

7 Punctaj

Punctajul pe tema va consta din:

- 40% testare automată
- 40% utilizare design patterns
- 20% JavaDoc
- Readme (în cazaul în care considerați că este necesar; ex.: cazul în care nu se utilizează visitor pattern)

8 Resurse

[1] - https://en.wikipedia.org/wiki/Shunting-yard_algorithm

[2] - Definirea comentariilor în cod

9 Update

- A fost actualizată figura 1
- Se recomandă utilizarea visitor pattern dar se poate utiliza și alt design pettern atât timp cât se justifică în readme
- Caracterele din cadrul unei expresii vor fi separate prin spații
- Readme (în cazaul în care considerați că este necesar; ex.: cazul în care nu se utilizează visitor pattern)
- String + NaN = concatenare (A fost trecut un nou exemplu în secțiunea 2)