Tema 1 PP

Deadline soft: 3.04, 23:55

Deadline hard*: 30.04, 23:55

Ultima actualizare: 1.04, 22:30

`modificare: fisierul build.xml`

Responsabili: Dinu Alexandru, Weisz Sergiu

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> ::= '+' | '*' | '==' | ‘<’
<assignment> ::= [= <symbol> <expr>]
<prog> ::=      <assignment> |
                [; <prog> <prog>] |
                [if <expr> <prog> <prog>] |
                [for <assignment> <expr> <assignment> <prog>] |
                [assert <expr>] |
                [return <expr>]
```

Observatie: Vazute strict lexical, expresiile si programele primite ca 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

Cerinte

Se cere implementarea unui interpretor pentru limbajul IMP++ in **Java**.
Interpretorul trebuie sa primeasca ca input un program IMP++, prezentat in forma specificata de gramatica de mai sus, sa ii verifice corectitudinea (detalii mai jos), si sa calculeze valoarea returnata de programul IMP++.

Corectitudine

Corectitudinea unui program IMP++ presupune satisfacerea urmatoarelor conditii:

1. Toate variabilele sa fie declarate inainte de folosire(vezi exemplele 8 si 10).
2. Pe toate caile programul sa se termine cu un **return** (vezi exemplul 9 si prioritatea erorilor).

Prioritatea erorilor: Check failed > Missing return > Assert failed.

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

Detalii input/output si implementare

Exemplul 1:

```
[* 2 3]
```

Expresia pentru 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 specific 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] [; [for [= i 0] [< i 10] [= i [+ i 1]] [= x [* x 2]]]
[return x]]]
```

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

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

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 nu a fost declarata deja! Programul nu respecta conditia 1, deci in fisierul de iesire se va scrie **"Check failed"**.

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! Se va scrie in fisier **"Missing return"**.

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 nu a fost declarata inainte sa fie folosita! Programul nu respecta conditia 1.

Exemplul 11:

```
[; [= x 12] [; [assert [< x 10]] [return x]]
```

```
x = 12;

assert (x < 10);

return x;
```

Deoarece asertiunea este falsa, se va scrie in fisier **"Assert failed"**.

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]].

Nu ne asteptam ca instructiunea “if” sa primeasca exemple de forma [= x 2]. Aceasta va primi numai expresii booleene.

Testare

Primul parametru la rularea programului va fi numele fisierului de intrare, iar al doilea parametru va fi fisierul de iesire.
In fisierul de intrare se va afla programul reprezentat ca in structura de mai sus (pe o singura linie) sau pe mai multe linii (separate de whitespace-uri, pentru a imbunati lizibilitatea).

Comportamentul programului:

1) Daca intr-o expresie se foloseste o variabila nedeclarata pana atunci (nu se afla in context), se va scrie in fisierul de iesire “Check failed”
2) Daca intr-un program fara erori de scoping (Check failed) lipseste instructiunea **return**, se va scrie in fisierul de iesire “Missing return”
3) Daca programul citit respecta conditiile de corectitudine de mai sus, acesta va fi evaluat, iar rezultatul evaluarii va fi scris in fisierul de iesire
   Daca o asertiune este falsa, se va scrie in fisierul de iesire “Assert failed"

Pentru rularea checker-ului este nevoie de un fisier “build.xml”. Instructiuni de folosire gasiti aici, si un exemplu puteti gasti aici. Exemplul trebuie particularizat pentru implementarea voastra.

Fisierul **build.xml** trebuie sa contina regulile compile jar si clean.

Regula **compile** va compila codul java dintr-un director (in exemplul atasat, este vorba de directorul src) si fisierele rezultate vor salvate intr-un alt folder.

Regula **jar** va pune toate fisierele obtinute la pasul compile intr-un singur fisier numit **tema.jar**. Aceasta regula trebuie sa depinda de regula compile.

Notare

Fiecare test trecut valoreaza 5p. Punctajul maxim pe aceasta tema este 100p.

Arhiva atasata contine 10 teste publice, checker-ul pe care il vom rula si un exemplu de fisier build.xml.

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

Arhiva va fi incarcata pe cs.curs si trebuie sa contina fisierele sursa, fisierul build.xml cu regulile compile, jar si clean si un fisier README cu informatii relevante despre implementare.

Depunctari: in cazul implementarilor cu un design indescifrabil / departe de cerinta temei, dar care totusi trec testele, depunctarea va fi una "simbolica", unde simbolic inseamna intervalul [0.1, 0.5] puncte.

*Punctajul maxim pe tema se poate obtine si dupa deadline-ul soft, dar incarcarea fiecarei teme cu un punctaj >= 50% inainte de deadline-ul soft conduce la echivalarea punctajului de curs si laborator cu 1 punct (asta inseamna garantarea obtinerii a 0.5 puncte pe curs si 0.5 puncte pe laborator) . Tema valoreaza 1.5p din 10p.