Calculadora d'expressions aritmètiques

Enunciat de la Pràctica de PRO2
Tardor 2016

2 de novembre de 2016

1 Introducció

Volem desenvolupar una calculadora d'expressions aritmètiques formades amb una sintaxi a l'estil del llenguatge de programació funcional Lisp. Una *expressió avaluable* és:

1. una constant entera o una variable, que anomenem *àtom* (o *expressió atòmica*);
2. la llista `()`, que anomenem *llista buida*;
3. una llista no buida `(l1 ... ln)` on cada `l1`, ..., `ln` és una expressió avaluable o bé el primer element `l1` és un un nom d'operador i la resta són expressions avaluables.

Noteu que la definició prèvia és recursiva, ja que les expressions dins d'una llista poden ser a la vegada atòmiques o llistes. Algunes llistes correspondran a operacions aritmètiques o booleanes i faran servir la notació prefix, com ara `(+ 1 2)` o bé `(and b1 b2)`.

Els tipus de dades que tractarem són els enters, els booleans (representats pels enters 0 i 1) i les llistes d'enters.

Definim una *expressió entera* (o simplement *enter*) com:

- Una constant entera (seqüència de dígits amb o sense signe `-` inicial)
- Una variable que conté un valor enter
- Una expressió `(op` $\texttt{arg}_1 \ldots \texttt{arg}_n$), on `op` és un operador n-ari (primitiu o definit) que retorna un valor enter quan s'aplica als arguments $\texttt{arg}_1 \ldots \texttt{arg}_n$. Remarquem que aquests arguments $\texttt{arg}_i$ no sempre seran expressions enteres (veurem algun contraexemple més endavant)

Considerarem que una *expressió booleana* és un cas particular d'expressió entera en el qual el valor de l'expressió només pot ser 0 o 1, que representen respectivament les constants booleanes *false* i *true*. Noteu que no farem servir mai els strings `false` i `true` per representar aquestes constants.

Una *llista d'enters* es defineix com:

- Una llista buida o una llista `(`$\mathtt{e}_1 \ldots \mathtt{e}_n$`)` on cada $\mathtt{e}_i$ és una expressió entera
- Una expressió `(op` $\mathtt{arg}_1 \ldots \mathtt{arg}_m$`)`, on `op` és un operador m-ari (primitiu o definit) que retorna una llista d'enters quan s'aplica als arguments $\mathtt{arg}_1, \ldots, \mathtt{arg}_m$
- Una variable que conté un valor llista d'enters

La nostra calculadora només obtindrà resultats d'avaluació correctes quan s'apliqui a expressions avaluables que siguin expressions enteres, booleanes o llistes d'enters. La definició general d'expressió avaluable que hem donat a l'inici d'aquesta secció també inclouria tipus de dades com llistes de llistes o llistes amb alguns elements enters i altres elements llistes. En aquests darrers casos, el resultat d'avaluar l'expressió serà un valor especial `indefinit`, que considerarem que no és de cap dels tipus de dades permesos. Altres possibles errors a l'hora d'avaluar una expresssió, per exemple quan una expressió contingui una variable no inicialitzada, també donaran lloc al mateix resultat especial `indefinit`.

A més de les expressions avaluables, també haurem de processar un tipus especial d'instruccions que corresponen a *definicions* (de variables o d'operacions) amb una sintaxi molt concreta que especificarem més endavant.

A continuació s'exposen amb més detall les operacions de la calculadora.

2 Operacions de la calculadora

Hi ha dues maneres d'interactuar amb la calculadora. La primera consisteix a escriure una expressió perquè la calculadora l'avaluï i escrigui el resultat de la seva avaluació. La segona consisteix a introduir definicions de variables o d'operacions.

2.1 Operacions primitives

A continuació es mostren les operacions primitives que ja estan predefinides i que es poden fer servir per formar expressions:

1. `(+ x y)`. Si `x` i `y` són expressions enteres (i, per tant, s'avaluen a enters), retorna la seva suma. En cas contrari, el seu valor està indefinit.

2. `(- x)`. Retorna `x` canviat de signe si `x` és un enter. En cas contrari, el seu valor està indefinit.

3. `(cons x y)`. Si `x` és una expressió entera i `y` és una llista d'enters `(y1 y2 ... yN)`, retorna la llista d'enters `(x y1 y2 ... yN)`. En cas contrari, el seu valor està indefinit.

4. `(head x)`. Retorna el primer element de `x` si `x` és una llista no buida d'enters. En cas contrari, el seu valor està indefinit.

5. `(tail x)`. Retorna la llista d'enters obtinguda eliminant el primer element de `x` si `x` és una llista no buida d'enters. En cas contrari, el seu valor està indefinit.

6. `(= x y)`. Retorna `1` si `x` és igual a `y` i `0` si són diferents. Si `x` i `y` no són del mateix tipus, el valor del procediment està indefinit.

7. `(< x y)`. Retorna `1` si $x < y$ i `0` en cas contrari. Si `x` i `y` no són del mateix tipus, el valor del procediment està indefinit.

8. `(not x)`. Retorna `1` si `x` és `0` i `0` si `x` és `1`. Si `x` no és un booleà, el valor del procediment està indefinit.

9. `(and x y)`. Retorna `1` si `x` i `y` són certs i `0` en cas contrari. Si `x` o `y` no són booleans, el valor del procediment està indefinit.

10. `(or x y)`. Retorna `1` si `x` o `y` són certs i `0` en cas contrari. Si `x` o `y` no són booleans, el valor del procediment està indefinit.

11. `(if x y z)`. Retorna `y` si `x` és `1` i `z` si `x` és `0`. Si `x` no és un booleà, el valor del procediment està indefinit. Noteu que, a diferència de la resta d'operacions primitives, el resultat retornat pot ser booleà, enter o llista d'enters, depenent del cas.

Si el resultat d'avaluar una expressió és el valor especial `indefinit`, el sistema escriurà la paraula `indefinit`. Això pot passar o bé perquè apareguin variables o operadors indefinits o bé perquè els arguments dels operadors no compleixin les restriccions de tipus i condicions que hem especificat a la llista anterior d'operacions primitives, o perquè el nombre d'arguments és diferent de l'esperat per algun dels operadors (primitius o no) que apareixen a l'expressió.

En el següent exemple d'ús del sistema, suposeu que la variable `x` no està inicialitzada.

```
> (+ 10 15)
25
> (+ (+ 1 2) 7)
10
> (- (- 5))
5
> (+ x 1)
indefinit
> (cons 3 (2 1))
(3 2 1)
> (head (tail (10 20 30)))
20
> (if 1 (+ 0 1) (+ 2 3))
1
```

2.2 Definicions

Les instruccions que corresponen a definicions de variables o d'operacions són les següents:

1. `(define <var> <exp>)`. Defineix el valor de la variable `<var>` com el resultat d'avaluar l'expressió `<exp>`. En el cas que el resultat d'avaluar `<exp>` sigui indefinit, `<var>` quedarà indefinida.

2. `(define <nom> <llista1> <exp>)`. Crea una operació nova anomenada `<nom>`, amb els paràmetres especificats a `llista1` i que té com a codi l'expressió avaluable `exp`. La llista `llista1` només contindrà noms vàlids de paràmetres sense repeticions.

Després de definir el valor d'una variable, el sistema escriurà el nom de la nova variable seguit d'un espai i del seu valor. Aquesta variable amb el seu corresponent valor es podrà fer servir posteriorment en les expressions a avaluar. En cas que el resultat d'avaluar `<exp>` sigui indefinit, no hi haurà cap efecte i s'escriurà `indefinit` en lloc del valor.

Si el que es defineix és una operació, el sistema escriurà el nom de l'operació seguit d'un espai, el caràcter `#` i el nombre de paràmetres de l'operació. Aquesta nova operació es podrà fer servir posteriorment en les expressions a avaluar amb uns paràmetres concrets. La definició d'una operació no comportarà cap avaluació del seu codi i, per tant, no pot produir un resultat

indefinit. Tanmateix, sí podrà ser indefinit el resultat d'avaluar aquesta operació amb uns determinats paràmetres posteriorment.

Dins del codi d'una operació definida (`exp`) no hi haurà mai cap definició de variables ni d'operacions i les úniques variables que poden aparèixer són els paràmetres de l'operació. Les variables definides no seràn visibles dins del codi de les operacions definides. Per tant, qualsevol aparició a `exp` d'una variable que no és paràmetre de l'operació donarà el valor indefinit quan el codi s'avalua.

Una variable o una operació, excepte les primitives, es pot redefinir usant el mateix nom que ja tenia. L'efecte és que la definició antiga quedarà substituïda per la nova excepte en el cas que la nova tingui un resultat indefinit, cas en el qual es mantindrà la definició antiga.

3 Exemples

Variables i valors.

```
> (define x 10)
x 10
> (define z (+ (head (tail (1 2 3 4))) 10))
z 12
> (+ (head (1 2 3 4)) z)
13
> (if (< z x) (1) ())
()
```

Definició de la diferència de dos nombres.

```
> (define diff (x y) (+ x (- y)))
diff #2
> (diff 10 20)
-10
```

Definició del producte.

```
> (define * (x y) (if (= x 0) 0 (+ y (* (diff x 1) y))))
* #2
> (* 4 5)
20
```

Definició del quadrat d'un nombre.

```
> (define quadrat (x) (* x x))
quadrat #1
> (quadrat 5)
25
```

Definició de la divisió entera.

```
> (define / (x y) (if (< y x) (+ 1 (/ (diff x y) y))
                              (if (= x y) 1 0)))
/ #2
> (/ 21 4)
5
```

Definició d'una operació que retorna la suma dels n primers naturals.

```
> (define sum-first (n) (if (< 0 n) (+ n (sum-first (diff n 1))) 0))
sum-first #1
> (sum-first 10)
55
```

Definició d'una operació que calcula la mitjana entera dels n primers naturals.

```
> (define mitjana (n) (/ (sum-first n) n))
mitjana #1
> (mitjana 10)
5
```

4 Entrada de dades al programa

La nostra calculadora rebrà una seqüència d'instruccions pel canal estàndard d'entrada amb les següents restriccions:

1. Una instrucció és o bé una expressió que s'haurà d'avaluar o bé una definició, sigui de variable o d'operació, o bé `****`, que indicarà que s'acaba l'execució de la calculadora.

2. Una instrucció parentitzada (és a dir, que comença per parèntesi) pot ocupar una línia o més. Una instrucció no parentitzada (és a dir, una expressió avaluable atòmica o la comanda d'acabar `****`) ocuparà una única línia.

3. Tota línia és una instrucció o part d'una única instrucció o una línia en blanc.

4. La primera (i sovint única) línia de tota instrucció comença per un caràcter no blanc.

5. Totes les instruccions estaran ben parentitzades. En particular, qualsevol expressió, sigui una instrucció a avaluar, formi part de la definició d'una variable o de la definició d'una operació, estarà ben parentitzada.

6. Després del darrer parèntesi d'una instrucció no hi haurà cap caràcter o només caràcters blancs.

7. Dins de qualsevol instrucció parentitzada, entre dos elements consecutius d'una llista hi haurà un o més separadors. Els separadors poden ser blancs o salts de línia.

8. Els noms de les variables i dels paràmetres formals començaran sempre per un caràcter lletra (minúscula o majúscula).

9. Els noms de les operacions definides no començaran per un caràcter dígit ni pel caràcter `'-'`.

10. Cap nom de variable, de paràmetre formal o d'operació contindrà parèntesis.

11. Totes les definicions estaran ben formades:

    (a) Una definició de variable constarà d'un parèntesi obert immediatament seguit de la paraula `define`, un o més separadors, un nom vàlid de variable, un o més separadors, una expressió avaluable i un parèntesi tancat.

    (b) Una definició d'operació constarà d'un parèntesi obert immediatament seguit de la paraula `define`, un o més separadors, un nom vàlid d'operació, un o més separadors, una llista ben parentitzada amb els noms dels paràmetres formals de l'operació, un o més separadors, una expressió avaluable i un parèntesi tancat.

12. En una llista de paràmetres formals qualsevol no hi haurà noms repetits ni cap d'ells coincidirà amb el nom d'una operació (primitiva o definida). Sí podran coincidir amb els noms de les variables definides.

13. Un mateix nom no s'usarà mai per definir una variable i una operació alhora.

5 Sortida esperada del programa

1. Per a cada instrucció que sigui una expressió avaluable, el programa escriurà només el resultat de l'avaluació seguit d'un salt de línia. Si el resultat és una llista d'enters, s'escriurà normalitzada de manera que entre dos elements consecutius només hi haurà un caràcter blanc, el primer element s'escriurà immediatament després del parèntesi d'obrir i l'últim element s'escriurà immediatament abans del parèntesi de tancar. Si la llista és buida, s'escriurà el parèntesi d'obrir i el de tancar seguits.

2. Per a cada instrucció que sigui una definició de variable, s'escriurà el nom de la variable, seguit d'un caràcter blanc, del resultat d'avaluar l'expressió associada i d'un salt de línia.

3. Per a cada instrucció que sigui una definició d'operació, s'escriurà el nom de l'operació, seguit d'un caràcter blanc, del caràcter `#`, del nombre d'arguments (aritat o mida de la llista de paràmetres formals associada) i d'un salt de línia.

A l'acabar, quan es llegeixi l'instrucció `****`, s'escriuran seguint (2) totes les variables vigents per ordre lexicogràfic de nom (que és l'ordre estàndard entre strings de C++), i després totes les funcions vigents seguint (3) per ordre lexicogràfic de nom.

6 Tractament d'errors

Els únics errors que es poden produir són durant l'avaluació de les expressions, i són els següents:

- Consultar una variable indefinida;
- Consultar dins del codi d'una operació definida qualsevol variable que no sigui un paràmetre formal de l'operació;
- Cridar una operació que no sigui primitiva i no hagi estat definida prèviament;
- Cridar una operació amb un nombre incorrecte de paràmetres;
- Cridar una operació amb algun paràmetre incorrecte (això inclou tant un tipus de paràmetre incorrecte com un valor del tipus correcte però que no compleix un requisit de l'operació).

- Obtenir un resultat no enter com a element d'una llista d'enters

En qualsevol d'aquests casos l'avaluació és indefinida i la calculadora escriu `indefinit` un cop, independentment del nombre d'errors que hi pugui haver.

7 Referència

Entre l'abundant bibliografia sobre LISP, pot resultar especialment útil llegir l'inici de la secció 1.1 (fins la subsecció 1.1.6 inclosa) del llibre *Structure and Interpretation of Computer Programs* de H. Abelson i J. Sussman, 2a edició (Cambridge, MIT Press), que es pot trobar com a recurs electrònic al consorci de biblioteques de la UPC.