TD 10
Construction et évaluation d'une expression algébrique postfixée

TAD
Semestre 2

Type fonctionnelle C'est un type ou on ne propose que des fonctions

Type impératif C'est un type ou on ne propose au moins une procédure

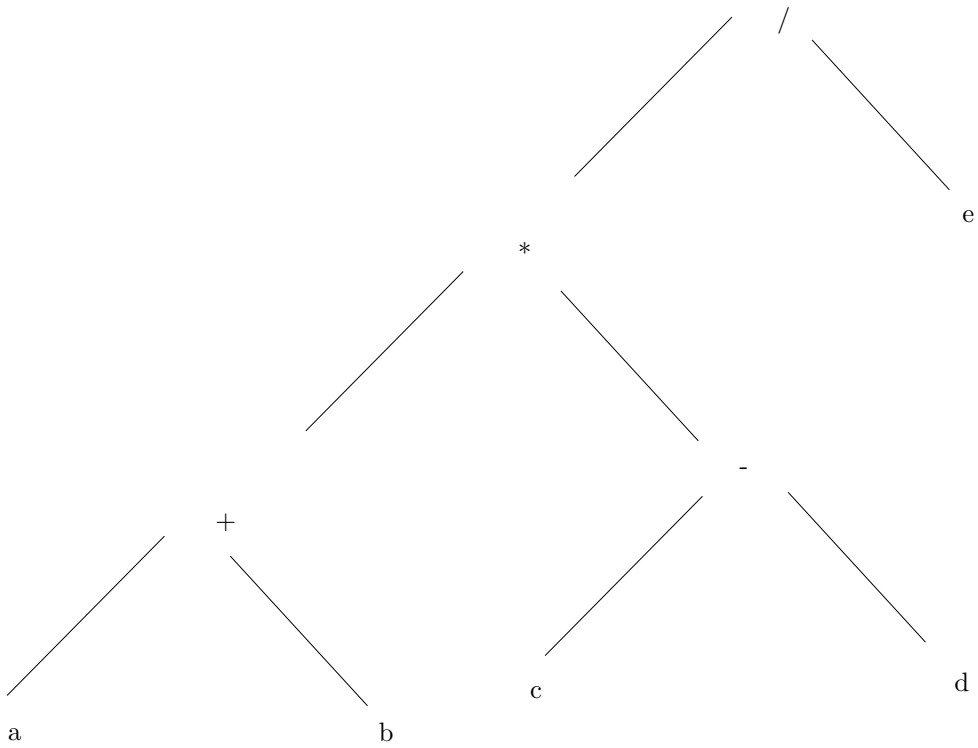
1 Spécification algorithmique des opérations du TAD

```
1  -- construit un arbre vide
2  fonction arbreVide() retourne <ArbreBinaire[T]>;
3
4  -- détermine si un arbre est vide
5  fonction estVide (entree arbre <ArbreBinaire[T]>) retourne <Booleen>;
6
7  -- racine d'un arbre binaire
8  -- necessite non arbreVide(a)
9  fonction racine (entree arbre <ArbreBinaire[T]>) retourne <T>
10   declenche arbreVide;
11
12 -- sous-nombre gauche d'un arbre a
13 -- Nécessite non estVide(a)
14 fonction gauche (entree arbre <ArbreBinaire[T]>) retourne <ArbreBinaire[T]>
15   declenche estVide;
16
17 -- sous-nombre droit d'un arbre a
18 -- Nécessite non estVide(a)
19 fonction droite (entree arbre <ArbreBinaire[T]>) retourne <ArbreBinaire[T]>
20   declenche estVide;
21
22 -- Construit un nouvel arbre à partir de r, g et d
23 fonction nouvelArbre(entree r <T>, entree g <ArbreBinaire[T]>,
24            entree t <ArbreBinaire[T]>, entree d <ArbreBinaire[T]>)
25   retourne <ArbreBinaire[T]>
26   declenche arbrePlein;
```

Listing 1 – sécification algorithmiques

2 Algorithmes de parcours d'un arbre binaire

2.1



- Parcours GRD : $a + b * c - d/e$
- Parcours RGD : $/ * + a\ b - c\ d\ e$
- Parcours GDR : $a\ b + c\ d - * e/$

2.2

```
1  -- afficher le contenu d'un arbre a selon le parcours RGD
2  procedure afficherRGD(entree arbre <ArbreBinaire[Caractere]>)
3  debut
4    si  non estVide(arbre) alors
5      ecrire(racine(a));
6      afficher(RGD(gauche(a));
7      afficherRGD(droit(a));
8    fin si;
9  fin
```

Listing 2 – Affichage d'un arbre binaire de parcours RGD

3 Utilisation du TAD ArbreBinaire[T]

3.1

```
1  procedure construireArbre (sortie a <ArbreBinaire[Caractere]>)
2  glossaire
3    p <Pile[ArbreBinaire[Caractere]]>;
4    c <Caractere>;
5    sd <ArbreBinaire[Caractere]]>;
6    sg <ArbreBinaire[Caractere]]>;
7
```

```
8  debut
9    creerPile(p);
10   lire(c);
11   tantque c /= '.' faire
12     si operande(c) alors
13       empiler(p, nouvelArbre(c, arbreVide, arbreVide));
14     sinon
15       sd <- dernier(p);
16       depiler(p);
17       sg <- dernier(p);
18       depiler();
19       empiler(p, nouveArbre(c, sg, sd));
20     fin si;
21     lire(c);
22   fin tantque;
23   a <- dernier(p);
24   depiler(p);
25 fin
26
27
28 fonction valeurArbre(entree arbre <ArbreBinaire[Caractere]>à
29     retourne <Entier>
30 debut
31   --cas d'arret
32     --à compléter
33   -- calcul la valeur du  de a
34   valGauche <- valeurArbre(gauche(arbre)):
35   -- calcul la valeur du suit de a
36   valDroit <- valeurArbre(droit(arbre));
37   -- appliquer l'opérateur à valgauche de valDroit
38   retourner (appliquerOperateur(racine(arbre), valGauche, valDroit));
39 fin
40
41 fonction valeurArbre(entree arbre <ArbreBinaire[Caractere]>)
42 retourne <Reel>
43 declenche arbreVide
44 glossaire
45   ValGauche <Reel>;
46   valDroite <Reel>;
47 debut
48   si estVide(gauche(arbre)) et estVide(droite(arbre)) alors
49     retourner (valeur(racine(arbre));
50   sinon
51     valeurArbre(droit(arbre));
52     valeurArbre(gauche(arbre));
53   fin si;
54 fin
```

Listing 3 – construireArbre

4 Représentation physique du TAD ArbreBinaire[T]

```
1 type ArbreBinaire[T] pointeur sur <Noeud[T]>;
2 type Noeud[T] : enregistrement
3   racine <T>,
4   sag <ArbreBinaire[T]>,
5   sad <ArbreBinaire[T]>;
```

```
6
7  -- créer un arbre vide
8  fonction arbreVide()
9    retourne ArbreBinaire[T]
10 debut
11   retourner (NULL);
12 fin
13
14 -- retourne vrai si l'arbre 'arbre' est vide
15 fonction estVide(entree arbre <ArbreBinaire[T]>)
16   retourne <Booleen>
17 debut
18   retourner (arbre = NULL);
19 fin
20
21 -- retourne la racine de l'arbre 'arbre'
22 fonction racine(entree arbre <ArbreBinaire[T]>)
23   retourne <T>
24   declenche arbreVide
25 debut
26   si estVide(arbre) alors
27     declencher arbreVide;
28   fin si;
29
30   retourner (arbre↑.racine);
31 fin
32
33 -- retourne le sous arbre gauche de l'arbre 'arbre'
34 fonction gauche(entree arbre <ArbreBinaire[T]>)
35   retourne <T>
36   declenche arbreVide
37 debut
38   si estVide(arbre) alors
39     declencher arbreVide;
40   fin si;
41
42   retourner (arbre↑.sag);
43 fin
44
45 -- retourne le sous arbre droit de l'arbre 'arbre'
46 fonction droite(entree arbre <ArbreBinaire[T]>)
47   retourne <T>
48   declenche arbreVide
49 debut
50   si estVide(arbre) alors
51     declencher arbreVide;
52   fin si;
53
54   retourner (arbre↑.sad);
55 fin
56
57 --construit un nouvel arbre à partir de la racine, sag et sad
58 fonction nouvelArbre(entree racine <T>, entree sag <ArbreBinaire[T]>, entree
       sad <ArbreBinaire[T]>)
59   retourne <ArbreBinaire[T]>
60   declenche arbrePlein
61 glossaire
62   nouveau <ArbreBinaire[T]>;
```

```
63 debut
64   allouer(nouveau);
65   si nouveau = NULL alors
66     declencher arbrePlein;
67   fin si;
68
69   nouveau↑.racine <- r;
70   nouveau↑.sag <- sag;
71   nouveau↑.sad <- sad;
72
73   retourner (nouveau);
74 fin
```

Listing 4 – Représentation du type