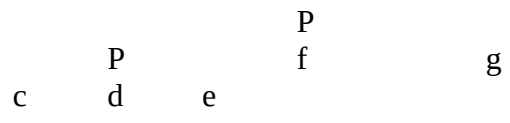
Deug Mias-2
Informatique
Institut Galilée.
2000-2001

TD-9
Arbres

-I- Représentation linéaire d'un arbre.

Chaque noeud d'un arbre sera représenté par un enregistrement constitué d'une valeur (une étiquette), d'un pointeur sur son premier fils (filsDeG) ainsi que d'un pointeur vers son premier frère à droite (frèreDeD). L'arbre sera représenté par un pointeur sur le noeud racine.

a) Ecrire une procédure qui à partir d'un arbre quelconque construit une expression parenthèsée représentant cet arbre :
exemple: P(P(c,d,e), f, g) représente



Remarque : pour obtenir une expression complètement parenthésée (comme en Lisp) on ajouterait une paire de parenthèses supplémentaires à l'expression parenthésée.
Ici on obtiendrait (P (P (c,d,e), f, g)).

correction

```
Type    pNoeud = ^noeud
        noeud =Enregistrement
                filsDeG : pNoeud
                label :chaine
                frereDeD : pNoeud
        finEnregistrement
Type arbre = Pnoeud

(* L 'expression parenthesee se definit de la manière suivante:
exp= label
exp=label(listeDexp)
listeDexp=exp
listeDexp=exp, ListeDexp
*)


Procedure ConstruitExp (a : arbre ; var s : chaine )
debut
        (* ajouter "(" ici et ")" à la fin pour avoir une expression complètement
parenthesee*)
        s<--""
        construitExpR(a , s)
```

```
fin

Procedure ConstruitExpR(a : arbre ; var s : chaine )
var f: pNoeud
debut
        si a <> nil alors
                ajouteChaine(a^.label,s)
                si a^.filsDeG<> nil alors
                        ajouteChaine('(', s)
                        ConstruitExpR(a^.filsDeG, s)
                        f<--a^.filsDeG
                        Tantque f^.frereDeD<>nil faire
                                ajouteChaine(',', s)
                                ConstruitExpR(f^.frereDeD, s)
                                f<--f^.frereDeD
                        fintq
                        ajouteChaine(')', s)
                finsi
        finsi
fin
```

fincorrection

b) Ecritre une procédure qui crée un arbre à partir d'une expression parenthèsée (on se limitera aux labels d'une seule lettre).

correction

```
Procedure ConstruitArbre (var s : chaine ;var  a : arbre)
debut
                si longeur(s)=0 alors
                        a<--nil
                sinon
                        ConstruitArbreR(s, a,1)
                fsi

Procedure ConstruitArbreR(var s : chaine ;var  a : arbre ; var courant : entier)
var f: pNoeud
debut
                (*courant est la position courante dans la chaine. A la sortie courant
                contient la position suivante à examiner*)
                (* on suppose que l'arbre à construire (ou le sous-arbre) a au moins un
                noeud *)
                new(a)
                a^.label <--s[courant]
                courant<--courant+1
                si courant > longueur(s) OU s[courant]<> "(" alors
                        a^.filsDeG<--nil
                sinon
                        (*il y a au moins un fils à construire*)
```

```
                courant<--courant+1 (*on saute la "(" *)
                construitArbreR(s, a^.filsDeG,courant)
                f<--a^.filsDeG
                tantQue s[courant] =',' faire
                        ConstruitArbreR(s,f^.frereDeD,courant)
                        courant<--courant+1
                        f<--f^.frereDeD
                fintq
                (*f est le dernier fils de a*)
                f^.frereDeD<-- nil
                courant<--courant+1
        fsi
fin
```
fincorrection

c) Evaluer une expression parenthésée en supposant qu'aux feuilles on a des chiffres (entre 0 et 9) et que les deux opérateurs sont l'addition(P) et la multiplication(M). On utilisera la fonction num(caractère) qui renvoie l'entier correspondant au chiffre représenté par le caractère.

correction

```
Fonction EvalR (a:arbre):entier
var l:pNoeud
debut
        (* si a contient un operateur on évalue les arguments puis on applique
        l'opérateur, sinon on evalue le noeud*)
        si a ^.val ='P' alors
                res<--0
                l<--a
                tantque l <>nil faire
                        res<--res+evalR(l)
                        l<--l^.FrereDeD
                fintq
                retourner(res)
        fsi
        si a ^.val ='M' alors
                res<--1
                l<--a
                tantque l <>nil faire
                        res<--res*evalR(l)
                        l<--l^.FrereDeD
                fintq
                retourner(res)
        fsi
        retourner(num(a^.val))
fin
```

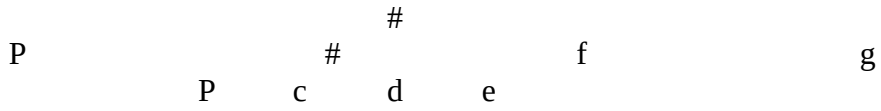
fincorrection

-II- Autre représentation

Une autre solution pour les expressions est de représenter la même expression

P (P (c,d,e), f, g)

par l'arbre suivant dont seules les feuilles ont une étiquette non vide (les étiquettes vides sont ici représentées par '#') :



Les opérateurs qui étaient auparavant (en I) dans les noeuds intérieurs sont maintenant eux aussi des feuilles. Plus précisément ils sont toujours les premiers fils.

Ecrire des procédures transformant les arbres de type-I- en arbres de type -II- (et *vice versa*.) .

correction

On prend la racine, si c'est une feuille l'arbre est identique, sinon (c'est un opérateur) on lui met une valeur # et on lui ajoute un premier fils contenant la valeur de la racine, attention ce fils n' a pas de descendant. Puis on modifie de la même manière les sous arbres fils du noeud racine.

fincorrection

Question supplémentaire:

-III- Ancêtres

a) Chercher l'ancêtre commun le plus proche de deux noeuds (dont on a l'accès) dans un arbre représenté comme en I. (on aura intérêt à utiliser une méthode récursive).

b) Même chose mais on a ajouté à la représentation un noeud père, ce qui permet de dresser la liste des ascendants de chacun des deux noeuds.