Semaine 13
Arbres - 1/2

Vous pouvez jeter un oeil à :

`cours5_arbres.ps` et éventuellement `Arbres_transparents.ppt` (transparents).

A distribuer aux étudiants lors de ce TD :

- `ficheTAD2.pdf`

La consultation de cette fiche sera autorisée lors des évaluations.

1 Parcours en profondeur : applications

Exercice 35 : Expression arithmétique codée sous forme d'un AB
Cf cours pour replacer le contexte.
Ecrire une action permettant d'écrire l'expression correspondante correctement, sous forme complètement parenthésée. Pour l'exemple vu plus haut, on veut : ((2 * a) + (b / (a + 1))).
Réponse : parcours infixe pour afficher les sommets + parenthèses pour les sommets internes : (en descendant,) en remontant. On ne veut pas de parenthèses autour des opérandes (feuilles), donc il faut pouvoir déterminer quand on est sur une feuille ou non. On peut rajouter une primitive `estUneFeuille` qui teste si un sommet est une feuille.

```
Fonction estUneFeuille( E A : TArbBin, Adr : TAdresse ) : booléen
// primitive pas définie si Adr est NULL
Début
    Retourner( adresseFilsGauche( A, Adr )=NULL et adresseFilsDroit(A, Adr )=NULL )
Fin

Action ParcoursAfficheExpr( E  A : TArbBin, Adr : TAdresse )
Début
    Si Adr <> NULL
    Alors Début
            Si estUneFeuille( A, Adr)
            Alors écrire( valeurSommet( A, Adr ) )
            Sinon Début
                    écrire( "(" )
                    ParcoursAfficheExpr( A, adresseFilsGauche(A, Adr) )
                    écrire( valeurSommet( A, Adr ) )
                    ParcoursAfficheExpr( A, adresseFilsDroit(A, Adr) )
                    écrire( ")" )
                  Fin
          Fin
```

```
Fin

Utilisation : ParcoursAfficheExpr( A, adresseRacine(A) )
```

Exercice 36 : Compter le nombre de sommets d'un AB
Ecrire une fonction qui retourne le nombre de sommets d'un AB.
Réponse : On peut s'éloigner un peu de l'algo de parcours générique pour mettre en avant la définition simple suivante : nb sommets = 1 + nb sommets sous-arbres gauche + nb sommets sous-arbre droit.

```
Fonction CompterSommets( E  A : TArbBin, Adr : TAdresse ) : entier
Var nb : entier
Début
    Si Adr = NULL
    Alors nb <-- 0
    Sinon Début
            nb <-- 1 + CompterSommets( A, adresseFilsGauche(A, Adr) )
                     + CompterSommets( A, adresseFilsDroit(A, Adr) )
          Fin
    Retourner(nb)
Fin

Utilisation : n <-- CompterSommets( A, adresseRacine(A) )
```

Exercice 37 : Compter le nombre de feuilles d'un AB
Ecrire une fonction qui retourne le nombre de feuilles d'un AB.
Réponse : ici, la définition simple est la suivante : nb feuilles = nb feuilles sous-arbres gauche + nb feuilles sous-arbre droit. On ne doit compter 1 que lorsqu'on est sur une feuille.

```
Fonction CompterFeuilles( E  A : TArbBin, Adr : TAdresse ) : entier
Var nb : entier
Début
    Si Adr = NULL
    Alors nb <-- 0
    Sinon Début
            Si estUneFeuille( A, Adr)
            Alors nb <-- 1
            Sinon Début
                    nb <-- CompterFeuilles( A, adresseFilsGauche(A, Adr) )
                           + CompterFeuilles( A, adresseFilsDroit(A, Adr) )
                  Fin
          Fin
    Retourner(nb)
Fin

Utilisation : n <-- CompterFeuilles( A, adresseRacine(A) )
```

Remarque : quand on arrive sur une feuille, on ne descend pas dans ses sommets "fictifs" (=NULL), par contre pour un sommet qui n'a qu'un seul fils, on descend dans son sommet "fictif" et on retourne 0.

2 Arbres binaires de recherche (ABR)

Exercice 38 : Insertion d'une feuille dans un ABR
Écrivez une action `inserer` qui, étant donnés un arbre (supposé structuré en ABR) et un élément, insère l'élément dans l'arbre sous forme d'une feuille "bien placée" (i.e. respectant la structure d'ABR).
Réponse :

```
Action inserer( ES  A : TArbBin, E Val : TInfo )
Var Trouvé : booléen
    Adr : TAdresse
Début
    Adr <-- adresseRacine( A )
    Trouvé <-- Faux
    Tant Que non Trouvé
    Faire Début
            Si Val < valeurSommet( A, Adr )
            Alors Si adresseFilsGauche( A, Adr ) = NULL
                  Alors Début
                          Trouvé <-- Vrai
                          insérerFilsGauche( A, Adr, Val )
                        Fin
                  Sinon Adr <-- adresseFilsGauche( A, Adr )
            Sinon Si adresseFilsDroit( A, Adr ) = NULL
                  Alors Début
                          Trouvé <-- Vrai
                          insérerFilsDroit( A, Adr, Val )
                        Fin
                  Sinon Adr <-- adresseFilsDroit( A, Adr )
          Fin
Fin
```

On peut aussi proposer une solution récursive

```
Action inserer( ES  A : TArbBin, Adr: TAdresse, E Val : TInfo )

Début

    Si Val < valeurSommet( A, Adr )
    Alors Si adresseFilsGauche( A, Adr ) = NULL
                  Alors insérerFilsGauche( A, Adr, Val )
                  Sinon Début
```

```
                        Adr <-- adresseFilsGauche( A, Adr )
                        inserer(A, Adr, Val)
                        Fin

    Sinon Si adresseFilsDroit( A, Adr ) = NULL
                  Alors insérerFilsDroit( A, Adr, Val )
                  Sinon Début
                        Adr <-- adresseFilsDroit( A, Adr )
                        inserer(A, Adr, Val)
                        Fin
Fin
```

Exercice 39 : Construction d'un ABR à partir d'une collection d'éléments
Utilisez l'action précédente `inserer` pour construire un ABR avec les éléments contenus dans

1. un tableau : action `tabVersABR`,
2. une liste : action `listeVersABR`.

Réponse : le tableau et la liste sont forcément non vide avant l'appel, sinon, on ne peut pas construire un arbre.

1. ```
   Const CMAX=100
   Type TTab : tableau[CMAX] de TInfo
   ...
   Action tabVersABR( S  A : TArbBin, E Tab : TTab, n : entier )
   Var i : entier
   Début
       créerArbre( A, tab[0] )
       Pour i de 1 à n-1
       Faire inserer( A, tab[i] )
   Fin
   ```
2. ```
   Type TListe : Liste de TInfo
   ...
   Action listeVersABR( S  A : TArbBin, E L : TListe )
   Var Adr : TAdresse // d'une liste, pas d'un arbre
   Début
       Adr <-- adressePremier( L )
       créerArbre( A, valeurElement( L, Adr ) )
       Adr <-- adresseSuivant( Adr )
       Tant Que Adr <> NULL
       Faire Début
               inserer( A, valeurElement( L, Adr ) )
               Adr <-- adresseSuivant( Adr )
             Fin
   Fin
   ```

Exercice 40 : Construction d'une liste triée à partir d'un ABR
Le parcours infixe de l'ABR fournit les éléments triés. Écrivez une action `aBRVersListe`

qui, étant donné un ABR, construit la liste triée des éléments contenus dans l'arbre.
Réponse :

1. Solution 1 : parcours à main gauche infixe, mais bof bof... car il faut gérer une adresse pour insérerAprès, et rajouter un élément fictif. Voici l'algo ci-dessous, mais ne pas l'écrire avec les étudiants, juste montrer sur un exemple pour comparer avec une meilleure solution qui suit.

```
Action aBRVersListe( E A : TArbBin, S L : TListe )
Var fictif : TInfo
Début
    créerListe( L )
    insérerEnTête( L, fictif )
    construireListe( A, adresseRacine( A ), L, adressePremier( L ) )
    supprimerEnTête( L )
Fin

Action construireListe( E A : TArbBin, AdrABR : TAdresse,
                        ES L : TListe, AdrL : TAdresse )
Début
    Si AdrABR <> NULL
    Alors Début
            construireListe( A, adresseFilsGauche( AdrABR ), L, AdrL )
            insérerAprès( L, valeurSommet( A, AdrABR ), AdrL )
            AdrL <-- adresseSuivant( AdrL )
            construireListe( A, adresseFilsDroit( AdrABR ), L, AdrL )
          Fin
Fin
```

2. Voici une solution qui utilise un parcours A MAIN DROITE en profondeur infixe.

```
Action aBRVersListe( E A : TArbBin, S L : TListe )
Début
    créerListe( L )
    construireListe( A, adresseRacine( A ), L )
Fin

Action construireListe( E A : TArbBin, AdrABR : TAdresse,
                        ES L : TListe )
Début
    Si AdrABR <> NULL
    Alors Début
            construireListe( A, adresseFilsDroit( AdrABR ), L )
            insérerEnTete( L, valeurSommet( A, AdrABR ) )
            construireListe( A, adresseFilsGauche( AdrABR ), L )
          Fin
Fin
```