Arbres Généraux

1 Des représentations

Exercise 1.1 (Statique-dynamique : par *n*-uplets de pointeurs)

```
constantes
  NbMaxFils = 10
types
  /* déclaration du type t_element */
  t_arbre_nuplets = ^t_noeud_nuplet
  t_tab_ag        = NbMaxFils t_arbre_nuplets
  t_noeud_nuplet  = enregistrement
    t_element           cle
    t_tab_ag            fils
    entier              nbFils
  fin enregistrement t_noeud_nuplet
```

Exercise 1.2 (Premier Fils - Frère Droit)

```
types
  /* déclaration du type t_element */
  t_arbre_dyn = ↑ t_noeud_ag
  t_noeud_ag  = enregistrement
    t_element           cle
    t_arbre_dyn         fils, frere
  fin enregistrement t_noeud_ag
```

2 Des parcours

Exercise 2.3 (Parcours en profondeur)

```
algorithme procedure parcours_prof
  parametres locaux
   t_arbre_nuplets  A
  variables
   entier  i
debut
  si A↑.nbFils = 0 alors
    /* terminaison */
  sinon
    /* traitement préfixe */
   pour i ← 1 jusqu'a A↑.nbFils-1 faire
     parcours_prof(A↑.fils[i])
       /* traitement intermédiaire i */
   fin pour
   parcours_prof(A↑.fils[A↑.nbFils])
    /* traitement postfixe */
  fin si
fin algorithme procedure parcours_prof
```

```
algorithme fonction recherche_ag : booleen
  parametres locaux
    t_element     x
    t_arbre_nuplets  A
  variables
    entier  i
debut
  si x = A↑.cle alors
    retourne vrai
  sinon
    i ← 1
    tant que (i <= A↑.nbFils) et non(recherche_ag (x, A↑.fils[i])) faire
      i ← i+1
    fin tant que
    retourne (i <= A↑.nbFils)
  fin si
fin algorithme fonction recherche_ag
```

```
algorithme procedure parcours_prof_ff
  parametres locaux
   t_arbre_dyn  A
  variables
   t_arbre_dyn  T
debut
  si A↑.fils = NUL alors
     /* terminaison */
  sinon
     /* traitement préfixe */
   T ← A↑.fils
   tant que T↑.frere <> NUL faire
     parcours_prof_ff (T)
       /* traitement intermédiaire i */
     T ← T↑.frere
   fin tant que
   parcours_prof_ff(T)
     /* traitement postfixe */
  fin si
fin algorithme procedure parcours_prof_ff
```

Exercise 2.4 (Parcours en largeur)

```
algorithme procedure affichage_largeur_nuplets
  parametres locaux
    t_arbre_nuplets   T
  variables
    t_queue           q
    entier            i
debut
  q ← enfiler(T, file_vide())
  q ← enfiler(NUL, q)
  faire
    T ← defiler(q)
    si (T = NUL) alors
      ecrire("\ n")
      si non est_vide(q) alors
        q ← enfiler(NUL, q)
      fin si
    sinon
      ecrire(T↑.cle, " ")
      pour i ← 1 jusqu'a T^.nbFils faire
        q ← enfiler(T^.fils[i], q)
      fin pour
    fin si
  tant que non est_vide(q)
fin algorithme procedure affichage_largeur_nuplets
```

```
algorithme procedure affichage_largeur_dyn
  parametres locaux
    t_arbre_dyn       T
  variables
    t_queue           q
debut
  q ← enfiler(T, file_vide())
  q ← enfiler(NUL, q)
  faire
    T ← defiler(q)
    si (T = NUL) alors
      ecrire("\ n")
      si non est_vide(q) alors
        q ← enfiler(NUL, q)
      fin si
    sinon
      ecrire(T↑.cle, " ")
      T ← T↑.fils
      tant que (T <> NUL) faire
        q ← enfiler(T, q)
        T ← T↑.frere
      fin tant que
    fin si
  tant que non est_vide(q)
fin algorithme procedure affichage_largeur_dyn
```

3 Applications

Exercise 3.5 (Hauteur d'un arbre général – *cont. 01 nov. 2001*)

```
algorithme fonction hauteur_nuplet : entier
  parametres locaux
    t_arbre_nuplets T
  variables
    entier h,  i
debut
  h ← - 1
  pour i ← 1 jusqu'a T↑.nbFils faire
    h ← max(h, hauteur_nuplet(T↑.fils[i]))
  fin pour
  retourne ((h + 1))
fin algorithme fonction hauteur_nuplet
```

```
algorithme fonction hauteur_dyn : entier
  parametres locaux
    t_arbre_dyn T
  variables
    entier h
debut
  h ← - 1
  T ← T↑.fils
  tant que (T <> NUL) faire
    h ← max(h, hauteur_dyn(T))
    T ← T↑.frere
  fin tant que
  retourne ((h + 1))
fin algorithme fonction hauteur_dyn
```

```
algorithme fonction hauteur_dyn2 : entier
  parametres locaux
    t_arbre_dyn T
  variables
    entier h
debut
  si (T = NUL) alors
    retourne (- 1)
  sinon
    retourne max(1 + hauteur_dyn2(T↑.fils), hauteur_dyn2(T↑.frere))
  fin si
fin algorithme fonction hauteur_dyn2
```

Exercise 3.6 (Arité moyenne d'un arbre général – *cont. 01 nov. 2012*)

```
algorithme procedure rec_arity_tuple
  parametres locaux
    t_tree_tuples          A
  parametres globaux
    integer            nodes, children
  variables
    integer            i
debut
  si (A↑.children <> 0) alors
    nodes ← (nodes + 1)
    children ← (children + A↑.children)
    pour i ← 1 jusqu'a A↑.children faire
      rec_arity_tuple(A↑.fils[i], nodes, children)
    fin pour
  fin si
fin algorithme procedure rec_arity_tuple

algorithme fonction arity_tuple : real
  parametres locaux
    t_tree_tuples     T
  variables
    entier            nodes, children
    real              r
debut
  children ← 0
  nodes ← 0
  rec_arity_tuple(T, nodes, children)
  r ← children
  retourne (r / nodes)
fin algorithme fonction arity_tuple
```

```
algorithme procedure rec_arite_dyn
  parametres locaux
    t_arbre_dyn       A
  parametres globaux
    integer            nodes, children
  variables
    integer            i
debut
  si (A↑.fils <> NUL) alors
    nodes ← (nodes + 1)
    A ← A↑.fils
    tant que (A <> NUL) faire
      rec_arite_dyn(A, nodes, children)
      children ← (children + 1)
      A ← A↑.frere
    fin tant que
  fin si
fin algorithme procedure rec_arite_dyn
```

Exercise 3.7 (Dynamique ↔ Statique)

```
algorithme fonction dyn2stat : t_arbre_nuplets
  parametres locaux
    t_arbre_dyn       T
  variables
    t_arbre_nuplets   R
debut
  allouer(R)
  R↑.cle ← T↑.cle
  R↑.nbFils ← 0
  T ← T↑.fils
  tant que (T <> NUL) faire
    R↑.nbFils ← (R↑.nbFils + 1)
    R↑.fils[R↑.nbFils] ← dyn2stat(T)
    T ← T↑.frere
  fin tant que
  retourne (R)
fin algorithme fonction dyn2stat
```

```
algorithme fonction rec_stat2dyn : t_arbre_dyn
  parametres locaux
    t_arbre_nuplets   T
    entier            pos
  variables
    t_arbre_dyn       R
    entier            i
debut
  si (pos > T↑.nbFils) alors
    retourne (NUL)
  fin si
  allouer(R)
  R↑.cle ← T↑.fils[pos]↑.cle
  R↑.fils ← rec_stat2dyn(T↑.fils[pos], 1)
  R↑.frere ← rec_stat2dyn(T, (pos + 1))
  retourne (R)
fin algorithme fonction rec_stat2dyn

algorithme fonction stat2dyn : t_arbre_dyn
  parametres locaux
    t_arbre_nuplets   T
  variables
    t_arbre_dyn       R
debut
  allouer(R)
  R↑.cle ← T↑.cle
  R↑.frere ← NUL
  R↑.fils ← rec_stat2dyn(T, 1)
  retourne (R)
fin algorithme fonction stat2dyn
```

Exercise 3.8 (Représentation par listes)

```
algorithme fonction arbre_to_chaine : chaine
  parametres locaux
    t_arbre_nuplets A
  variables
    entier    i
    chaine    liste
debut
  liste ← '(' + elt_to_chaine(A↑.cle)
  pour i ← 1 jusqu'a A↑.nbFils faire
    liste ← liste + arbre_to_chaine(A↑.fils[i])
  fin pour
  liste ← liste + ')'
  retourne liste
fin algorithme fonction arbre_to_chaine
```

```
algorithme fonction arbre_to_chaine : chaine
  parametres locaux
    t_arbre_dyn    A
  variables
    chaine    liste
debut
  si A = NUL alors
    retourne ""
  sinon
    liste ← '(' + elt_to_chaine(A↑.cle)
    liste ← liste + arbre_to_chaine(A↑.fils)
    liste ← liste + ')'
    liste ← liste + arbre_to_chaine(A↑.frere)
    retourne liste
  fin si
fin algorithme fonction arbre_to_chaine
```

```
algorithme fonction arbre_to_chaine : chaine
  parametres locaux
    t_arbre_dyn    A
debut
  si A = NUL alors
    retourne ""
  sinon
    retourne ( '(' + elt_to_chaine(A↑.cle)
                   + arbre_to_chaine(A↑.fils) + ')'
                   + arbre_to_chaine(A↑.frere) )
  fin si
fin algorithme fonction arbre_to_chaine
```

```
algorithme fonction creer_noeud : t_arbre_dyn
  parametres locaux
    entier              cle
    t_arbre_dyn         fils, frere
  variables
    t_arbre_dyn         A
debut
  allouer(A)
  A↑.cle ← cle
  A↑.fils ← fils
  A↑.frere ← frere
  retourne A
fin algorithme fonction creer_noeud
```

```
algorithme fonction transforme : t_arbre_dyn
  parametres locaux
    chaine    ch
  parametres globaux
    entier    i
  variables
    chaine        cle
    t_arbre_dyn   fils, frere
debut
  si (i <= longueur(ch)) et (ch[i] = '(' ) alors
    i ← i+1
    cle ← ""
    tant que (ch[i] <> ')' ) et (ch[i] <> '(' ) faire
      cle ← cle + ch[i]
      i ← i+1
    fin tant que
    fils ← transforme (ch,i)
    i ← i+1
    frere ← transforme (ch,i)
    retourne (creer_noeud (cle,fils,frere))
  sinon
    retourne NUL
  fin si
fin algorithme fonction transforme
```

```
algorithme fonction chaine_to_arbre : t_arbre_dyn
  parametres locaux
    chaine    liste
  variables
    entier i
debut
  i ← 1
  retourne (transforme (ch,i))
fin algorithme fonction chaine_to_arbre
```