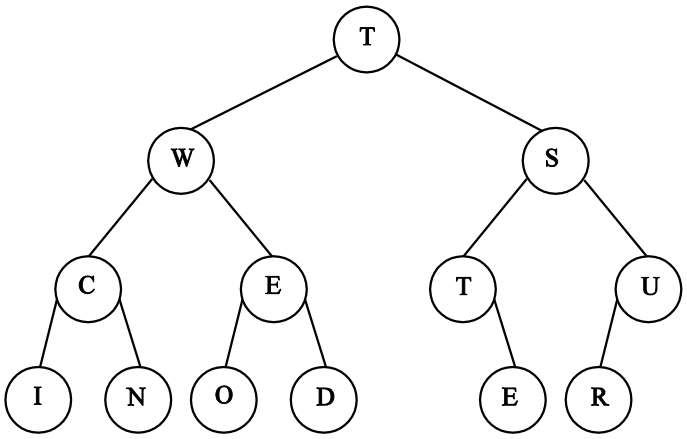
CORRECTIONS DES EXERCICES DU CONCOURS D'ENTREE EPITA 2012

Pour toutes les fonctions demandées, plutôt que de fournir des solutions en C, Caml ou Pascal, nous avons préféré fournir une solution algorithmique en pseudo-langage facilement implémentable dans un de ces langages.

A) Arbre : Incontestablement …

1) L'arbre est celui de la figure 1.



Suffixe : "INCODEWETRUST"

Figure 1.

2) Le parcours *suffixe* de l'arbre B donne : INCODEWETRUST *qui si l'on place correctement les espaces devient : « IN CODE WE TRUST »*). ce que je vous laisse traduire ☺

B) Arbre : Poids...

Spécifications :

La fonction ***calcule_poids(arbrebinaire B):entier*** retourne la somme des étiquettes de l'arbre B.

Principe :

On utilise un algorithme basé sur le parcours profondeur main gauche et la définition récurrente de la taille d'un arbre binaire. C'est à dire que la somme calculée pour un arbre vide sera 0 et pour un arbre non vide sera la valeur de l'étiquette du nœud augmentée du poids du sous-arbre gauche (*appel récursif*) et du poids du sous-arbre droit (*appel récursif*).

```
algorithme fonction calcule_poids : entier
parametres locaux
   arbrebinaire B
debut
   si B = arbrevide alors
      retourne 0
   sinon
      retourne contenu(racine(B)) + calcule_poids(g(B)) +
calcule_poids(d(B))
   fin si
```

```
fin algorithme fonction calcule_poids
```

Pour les algorithmes qui suivent, on utilisera le type t_liste (représentation statique) définit ci-dessous :

```
constantes
   LMax = …

types
   /* déclaration du type des éléments */
   t_element =  …
   /* déclaration du type  vecteur d'éléments */
   t_vectLMaxElts = LMax t_element

   /* déclaration du type t_liste */
   t_liste = enregistrement
      t_vectLMaxElts   elts
      entier           longueur
   fin enregistrement t_liste
```

C) Liste en progression arithmétique …

Spécifications :

La fonction ***arithmetique (t_liste L) : entier*** vérifie si la liste ***L*** suit une progression arithmétique. Elle retourne la valeur de la raison si c'est le cas, la valeur 0 sinon (de même si la liste contient moins de 2 éléments).

Principe :

Si la liste possède au moins deux éléments, la raison potentielle est donnée par la différence entre les deux premiers éléments. On parcourt la liste tant que l'élément suivant est égal à l'élément courant augmenté de la raison. Si le parcours atteint le dernier élément de la liste alors la propriété est vraie, fausse sinon. A noter que si les deux premiers éléments sont égaux, le parcours n'est pas effectué.

```
algorithme fonction arithmetique : entier
parametres locaux
   t_liste L
variables
   entier  raison, i
debut
   raison <- 0
   si (L.longueur > 1) ou (L.elts[2] - L.elts[1] <> 0) alors
      raison <- L.elts[2] - L.elts[1]
      i <- 2
      tant que (i < L.longueur) et (L.elts[i] + raison =
   L.elts[i+1]) faire
         i <- i+1
      fin tant que
      si i < L.longueur alors
         raison <- 0
      fin si
   fin si
   retourne (raison)
fin algorithme fonction arithmetique
```

D) Liste : Insertion et tri …

1) Procédure insertion

Spécifications :

La procédure ***insertion (t_element e, t_liste L)*** insère ***e*** à sa place dans ***L*** (triée en ordre croissant).

Principe :

Si la liste contenant ***L*** est pleine, l'insertion ne peut pas avoir lieu, il ne se passe rien.
Sinon, on recherche la place ***i*** de ***e*** : on parcourt la liste depuis le début jusqu'à atteindre ou dépasser ***e***, ou arriver après le dernier élément.
Ensuite, les valeurs de la liste de la fin jusqu'à la place ***i*** sont décalées d'une place vers la droite et on peut insérer ***e*** en place ***i***.

```
algorithme procedure insertion
parametres locaux
   t_element e
parametres globaux
   t_liste   L
variables
   entier    i, j
debut
   si L.longueur <= LMax alors
      /* recherche de la place de x */
      i <- 1
      tant que (i <= L.longueur) et (x > L.elts[i]) faire
         i <- i+1
      fin tant que
      /* décalages pour placer x à la place i */
      pour j <- L.longueur jusqu'à i decroissant faire
         L.elts[j+1] <- L.elts[j]
      fin pour
      L.elts[i] <- x
      L.longueur <- L.longueur + 1
   fin si
fin algorithme procedure insertion
```

2) Procédure tri_insert

Spécifications :

La procédure ***tri_insert (t_liste Lorig, t_liste Lres)*** construit la liste ***Lres*** version triée croissante de la liste ***Lorig***.

Principe :

La liste ***Lres*** est initialement vide. On insère chaque élément de la liste ***Lorig***, parcourue séquentiellement, à sa place dans ***Lres*** en utilisant la fonction ***insertion***.

```
algorithme procedure tri_insert
parametres locaux
   t_liste Lorig
parametres globaux
   t_liste Lres
variables
   entier  i
debut
```

```
   Lres.longueur <- 0
   pour i <- 1 jusqu'à Lorig.longueur faire
      insertion(Lorig.elts[i],Lres)
   fin pour
fin algorithme procedure tri_insert
```