Le 04 janvier 2016

Examen final (durée 1h30)

Exercice 1: *Tri Gnome* (4pts)

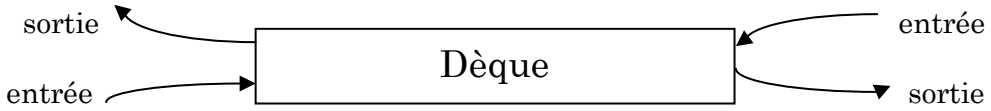
Le but de cet exercice est d'écrire l'algorithme du tri Gnome et de déterminer le nombre d'échanges au meilleur cas et au pire cas effectués au cours d'un tri gnome.

Méthode : Dans le tri gnome, on commence par le début du tableau, on compare deux éléments consécutifs (i, i+1): s'ils sont dans l'ordre on se déplace d'un cran vers la fin du tableau (incrémente) ou on s'arrête si la fin est atteinte ; sinon, on les permute et on se déplace d'un cran vers le début du tableau (décrémente) ou si on est au début du tableau alors on se déplace d'un cran vers la fin (incrémente).

a. Ecrire une procédure effectuant le tri gnome.
b. Donnez la complexité du tri gnome au meilleur cas et au pire cas? Justifiez votre réponse.

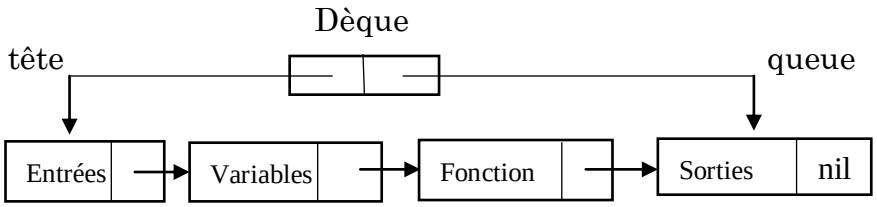
Exercice 2: *Une Dèque* (10pts)

Une « Dèque »[1] possède à la fois les propriétés d'une pile et d'une file. Une Dèque est définie par deux points d'entrée *tete* et *queue*. On peut donc ajouter ou supprimer un élément à chaque extrémité de la structure.



On considère une liste chainée de mots de longueur ≤ 20 manipulée comme une Dèque.

Exemple :



1. Définir le type Liste de mots et le type Dèque.
2. Ecrire les actions paramétrées des opérations de manipulation de la Dèque suivantes[2] :
   a. InitDèque() qui initialise une dèque
   b. DèqueVide(Dèque) de type booléen ; qui vérifie si une dèque est vide
   c. EnDéquer (Dèque, élément (mot), sens) permet de rajouter un élément soit en tête soit en queue ; **sens** est défini pour désigner l'extrémité à utiliser et sera égal à **1** si ajout en tête et **-1** si ajout en queue.
   d. DéDéquer (Dèque, sens) permet de supprimer un élément soit en tête soit en queue et le retourne ;

[1] Le mot dèque vient de l'anglais « double ended queue »
[2] La procédure **CopieMot** qui copie un mot dans un autre est considérée prédéfinie.

e. Extrémité (Dèque, sens) permet de retourner l'élément qui se trouve soit en tête soit en queue (sans le supprimer).

3. Ecrire une fonction <u>récursive</u> **Identique** qui vérifie si deux mots m1, m2 sont identiques.

4. Ecrire l'algorithme qui :
   - Initialise une Dèque D ;
   - Remplit (initialise) la dèque D avec n mots quelconques saisis au clavier en alternant une fois à gauche (tête) et une fois à droite (queue) ;
   - Supprime la <u>première</u> occurrence d'un mot m1 et la <u>dernière</u> occurrence d'un mot m2 de la Dèque D ;
   - Affiche D.

<u>Exercice 3</u> Arbre (6pts)

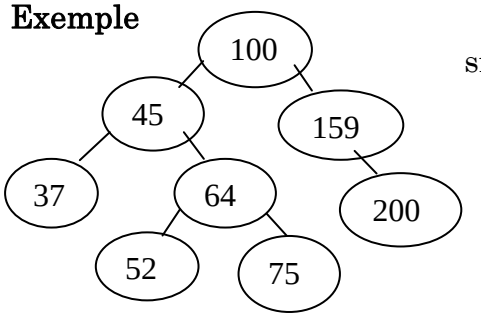
Soit A un Arbre binaire ordonné (de recherche) contenant des valeurs entières.

1. Donner la déclaration du type Arbre (d'entiers)
2. Etant donnée une valeur val, écrire une procédure récursive **Insère** qui insère val dans A (supposé non vide).

On voudrait extraire un sous-arbre binaire ordonné A1 de A de la manière suivante :
On considère <u>une valeur x dans A</u>, x sera la racine du sous-arbre A1 et toutes les valeurs de A <u>inférieures</u> strictement à x seront mises dans A1.

3. En utilisant les actions précédentes, écrire une fonction **Sous-arbre** qui construit le sous-arbre A1 (de racine x) à partir de A. On suppose que x existe dans A.
4. On suppose que l'arbre A est déjà construit. Soit une valeur x existant dans A, écrire un algorithme qui construit A1 à partir de A et affiche les éléments de A1 dans l'ordre croissant.

Exemple



si x=64, A1 contiendra les valeurs {64, 45, 37, 52}

<u>Bon courage</u>

Corrigé examen final

Exercice 1: *Tri Gnome* **(4pts) (2+1+1)**

a.

```
procédure TriGnome(E/S T : tableau[100] entier ; E/ n :entier)
var i, X : entier ;
Début
  i←1;
  tantque (i<n) faire
     si (T[i]<=T[i+1]) alors i←i+1
     sinon
          X←T[i] ; T[i]←T[i+1] ;  T[i+1]←X ;
          Si (i>1) alors i←i-1
                   Sinon i←i+1 ;
          finsi ;
      Finsi ;
  fait;
Fin;
```

b. Complexité du tri gnome au meilleur cas et au pire cas.

- La complexité au meilleur cas est de l'ordre de $O(n)$ quand le tableau est déjà trié on ne fait aucune permutation.
- La complexité au pire cas est de l'ordre de $O(n^2)$ quand le tableau est trié dans le sens inverse.

Exercice 2: *Une Dèque* **(10pts) (1+0,75+0,75+1,5+1,5+1+1,5+3)**

1. Définir le type Liste de mots et le type Dèque.

```
Type liste=^nœud ;
      nœud=enregistrement
                  element :chaine ;
                  suivant :liste ;
            finenreg ;

       Dèque=enregistrement
                  tete, queue :liste ;
            finenreg ;
```

2.

a. Fonction InitDèque() : Dèque //qui initialise une dèque

```
Var d :dèque ;
Début
  d.tete←nil ; d ;queue←nil
  retourner d ;
Fin ;
```

b. Fonction DèqueVide(E/d :Dèque) :booléen ; //qui vérifie si une dèque est vide

```
Debut
   Si (d.tete=nil et d.queue=nil) alors retourner vrai
                                  Sinon retourner faux ;
   Finsi ;
Fin ;
```

c. Procédure EnDéquer (E/S d :Dèque ; E/ mot :chaine ; E/ sens :entier)

```
Var nouv :liste ;
Début  nouv←Allouer(tailleDe(nœud) ;
       CopieMot(^nouv.element, mot) ; ^nouv.svt←nil ;
 Si (DèqueVide(d)) alors d.tete←nouv ; d.queue←nouv ;
 Sinon  Si (sens=1) alors ^nouv.suivant←d.tete ;  d.tete←nouv ;
        Sinon ^(d.queue).svt ←nouv ;   d.queue←nouv ;
        Finsi ;
 Finsi ;
Fin ;
```

d. Procédure DéDéquer (E/S d : Dèque ; E/ sens :entier ; E/S mot chaine)

```
Var p, r : liste ;
Début
  Si (d.tete=d.queue) alors p←d.tete ;
                            CopieMot(mot , ^(d.tete).element) ;
                            d.tete=nil;   d.queue=nil
  Sinon
    Si (sens =1) alors CopieMot(mot , ^(d.tete).element) ; p←d.tete ;
                       d.tete←^(d.tete).suivant
    sinon CopieMot(mot , ^(d.queue).element) ; p←d.queue ;
          r←d.tete ;
          tantque(^r.suivant !=d.queue) r←^r.suivant ;
          d.queue←r ;   ^r.svt←nil ;
    finsi ;
  finsi ;
  libérer(p) ;
Fin ;
```

e. Fonction Extrémité (E/ d : Dèque, sens :entier) :chaine

```
Début
  Si (sens=1) alors retourner (^(d.tete).element )
  Sinon retourner (^(d.queue.element) ;
  Finsi ;
Fin ;
```

3. fonction Identique(E/ m1, m2 : chaine ; E/ i : entier) : booléen
Début //1er appel i =longueur(m1)=longueur(m2)
si (i=0) alors retourner vrai
sinon si (m1[i] ≠ m2[i]) alors retourner faux
sinon retourner (Identique(m1, m2, i-1)) ;
finsi ;
finsi ;
Fin ;

4. Algorithme Ex2 ;

```
Var d, di : Dèque ; n : entier ; ch, mot, m1, m2 : chaine ; l1, l2, sens : entier ;  trouve :booléen ;
Début
 d←InitDèque() ;
 lire(n) ;  sens=1 ;
 pour i←1 à n faire
     lire(mot) ;
     Endéquer(d, mot, sens) ;
     sens←sens*(-1) ;
 fait ;
```

// Suppression

```
lire(m1, m2) ; di←InitDèque() ;   trouve ←faux
tantque ( ! DèqueVide(d) et (trouve=faux) faire
   ch← Extrémité(d, 1) ;
   l1← longueur(m1) ; l2← longueur( ch);
   si  (l1=l2) et (identique(m1, ch, l1) alors  Dédéquer(d,1, mot) ; trouve← vrai
                                          sinon  Dédéquer(d, 1, mot) ; Endéquer(di, mot, 1) ;
   finsi ;
fait ;
si (DèqueVide(d) alors écrire("m1 n'existe pas") ; finsi ;
tantque ( !DèqueVide(di))  faire Dédéquer(di, 1, mot) ; EnDéquer(d, mot,1) ;    fait ;

trouve ←faux ;
tantque ( ! DèqueVide(d) et (trouve=faux) faire
   ch← Extrémité(d, -1) ;
   l1← longueur(m2) ; l2← longueur( ch);
   si  (l1=l2) et (identique(m2, ch, l1) alors  Dédéquer(d,-1, mot) ; trouve← vrai
                                          sinon  Dédéquer(d, -1, mot) ; Endéquer(di, mot, -1) ;
   finsi ;
fait ;
si (DèqueVide(d) alors  écrire("m2 n'existe pas");
tantque ( !DèqueVide(di))  faire  Dédéquer(di, -1, mot) ; EnDéquer(d, mot,-1) ;   fait ;
```

// Affichage de la dèque

```
Tantque ( !DèqueVide()) faire  Dédéquer(d, 1, mot) ; Ecrire(mot) ; Fait ;  // Dèque vide
```

Fin. *// Fin de l'algorithme*

Exercice 3: *Arbre* **(0,5+1,5+2,5+1,5)**

```
Type arbre = ^nœud ;
Type nœud = Enregistrement
                info : entier ;
                succ_gauche : arbre ;
                succ_droit : arbre ;
            Finenreg ;
```

```
Procédure Insere (E/ a :arbre ;  E/ val : entier ; E/ père :arbre ) ;
Var   p: arbre ;
Debut
  Si ( non Vide(a)) alors
       Si (^a.info = val)  alors  Ecrire(" La valeur existe déjà");
       Sinon   pere ← a ;
            Si (^a.info > val)  alors   Insere (Fils_gauche(a), val, père) ;
                                Sinon  Insere (Fils_droit(a), val, père) ;
            Fsi ;
       Fsi ;
   Fsi ;
  Sinon / * créer le noeud* /
    p ←Allouer (Taillede (nœud)) ;   /* création du nouvel élément */
    ^ p.info ← val ; ^p.succ_gauche ← nil ;  ^p. succ_droit ← nil ;
      /* raccorder au nœud père */
       Si  (^père. info >val)  alors  ^père.succ_gauche ← p ;
                               Sinon ^père.succ_droit ← p ;
       Fsi ;
Fin ;
```

```
Procedure SousArbre(E/ A : arbre ; E/ x : entier, E/S A1 : arbre)
Debut
     Si ( ArbreVide(A1)) alors
          A1←Allouer(TailleDe(nœud) ;
          ^A1.info←x ;  ^A1.succ_gauche←nil ;  ^A1.succ_droit←nil ;
     Fsi ;
     Si non ArbreVide (A)
          alors     SousArbre(FilsGauche(A),x,A1) ;
                    Si (^A.info < x ) alors Insérer (A1,^A.info, nil) ;
                                            SousArbre(FilsDroit(A),x,A1)
                    Fsi ;
     Fsi ;
Fin ;
```

Algorithme Ex3 ;

```
Var A, A1:arbre ; x : entier ;
Début
   // On suppose que A est déjà créé
   Ecrire("donnez x ") ;  lire(x) ; A1←nil ;
   Sous-arbre( A, x, A1) ;
   AfficheInfixe(A1) ; // dans l'ordre croissant
Fin.
```

```
Procédure AfficheInfixe (E/ a : arbre);
  Début
    Si  (non Vide(a))  alors
          AfficheInfixe (Fils_gauche(a)) ;
          Ecrire (^a.info) ;
          Affiche Infixe (Fils_droit(a)) ;
     fsi ;
  Fin ;
```