DEVUIP Service Scolarité	**ANNEE UNIVERSITAIRE 2012/2013** **SESSION D'AUTOMNE** PARCOURS / ETAPE : LIIN300 Code UE : J1IN3001 **Epreuve :** Algorithmique et structures de données 1 **Date :** 7 Janvier 2013 **Heure :** 8H30 **Durée :** 1H30 Documents : non autorisés Epreuve de Mme : DELEST **Vous devez répondre directement sur le sujet qui comporte 8 pages. Insérez ensuite votre réponse dans une copie d'examen comportant tous les renseignements administratifs**	

Indiquez votre code **d'anonymat** : N° : .

La notation tiendra compte de la clarté de l'écriture des réponses.

Barème indicatif

- Question 1 – Connaissances générales : 2 points
- Question 2 – Containeur liés aux arbres binaires : 6 points
- Question 3 – Connaissance des structures de données : 3 Points
- Question 4 – Connaissance des structures de données : 3 Points
- Question 5 – Utilisation des structures de données : 6 points -

Question 1. Cochez les affirmations correctes et quelle que soit la réponse justifiez sur les lignes en pointillé.

☒ Pour une structure de type containeur, il y a au moins 5 primitives.

Il y a 5 primitives : créer, détruire, ajouter, supprimer, valeur. Il peut exister d'autres primitives correspondant à la spécificité du containeur.

☐ Dans un tasMin, un parcours en ordre infixe donne les objets dans l'ordre croissant.

Dans un tasMin, l'ensemble des valeurs des sous arbres de la racine sont supérieures à la valeur de la racine.

☐ La structure de file permet l'accès au dernier élément entré dans le containeur.

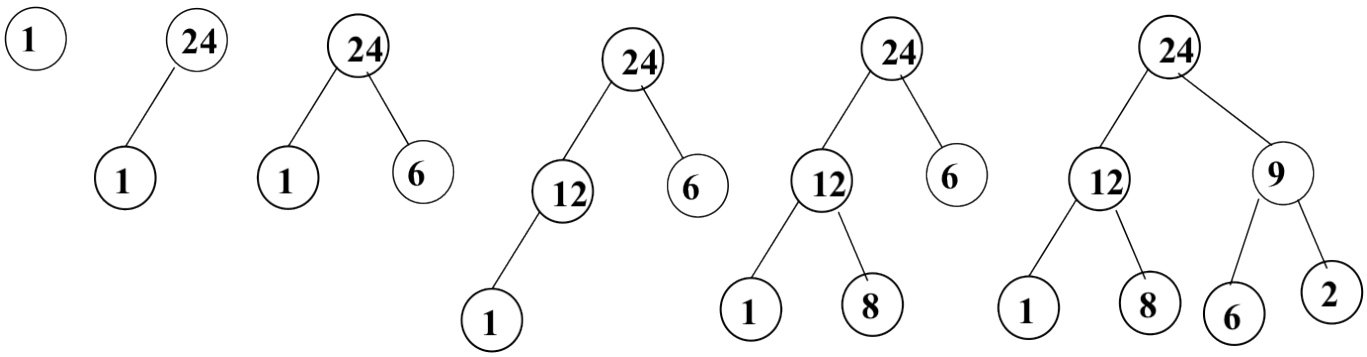
Non. La file donne accès au premier élément entré (First In First Out). La définition ci-dessus correspond à une pile (Last In First Out)

☐ Si s est un pointeur vers un structure contenant un champs p qui est un pointeur vers un entier, on accède à l'entier par s^.p?

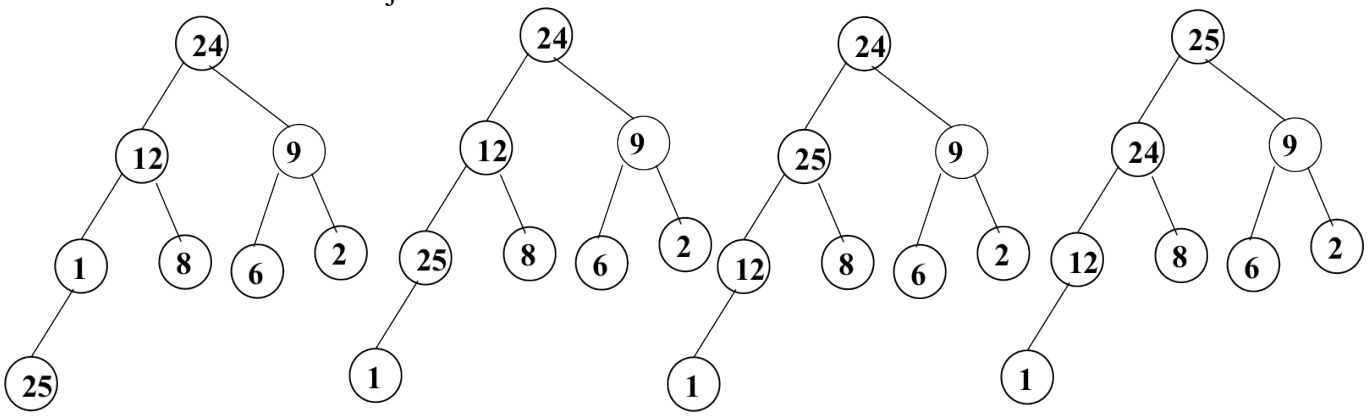
s^ désigne la structure, s^.p désigne le champ donc c'est un pointeur pas un entier.

Question 2. Soit la liste de clé A=(1,24,6,12,8,9,2).

1 - Construire le tas Max T correspondant à l'insertion consécutive des clés de la liste A. On dessinera l'arbre après chacune des quatre premières insertions ainsi que l'arbre final.



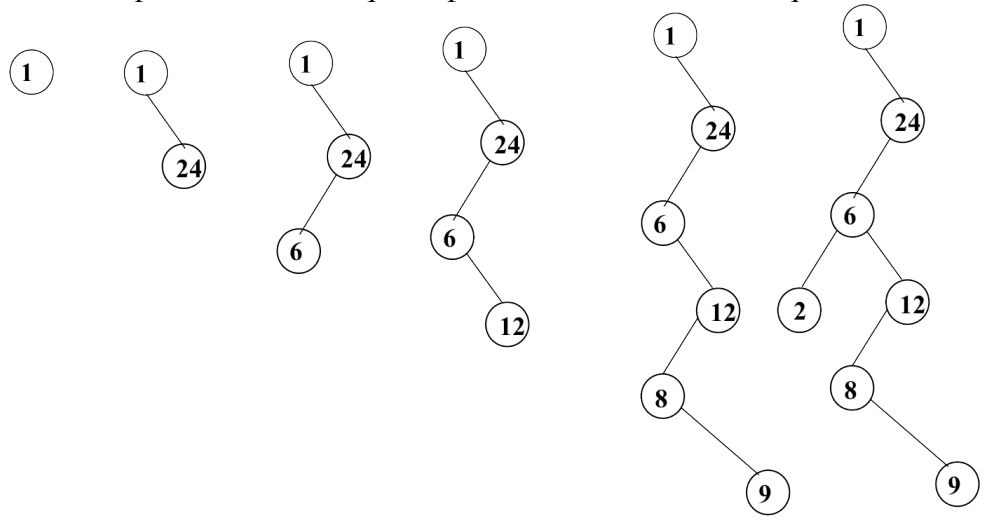
2 - Montrez l'exécution de l'ajout de la clé 25 sur l'arbre T.



3 - Donner la fonction *ajouter* qui ajoute un élément dans un tas Max.

```
fonction ajouter(ref T:tas d'objet, val v:entier):vide ;
   début
     T.tailleTas=T.tailleTas+1;
     T.arbre[T.tailleTas]=v;
     reorganiseTasMontant(T,tailleTas);
   fin
fonction reorganiseTasMontant(ref T: tas d'objet;val x:sommet):vide;
   var p:sommet;
   var signal:booléen;
   début
     p=père(x);
     signal=vrai;
     tantque x!=1 et getValeur(T,x)>getValeur(T,p)faire
       échanger(T.arbre[p],T.arbre[x])
       x=p;
       p=père(x);
     fintantque
   fin
```

4 – Construire l'arbre binaire de recherche B correspondant à l'insertion consécutive des clés de la liste A. On dessinera l'arbre après chacune des quatre premières insertions ainsi que l'arbre final.



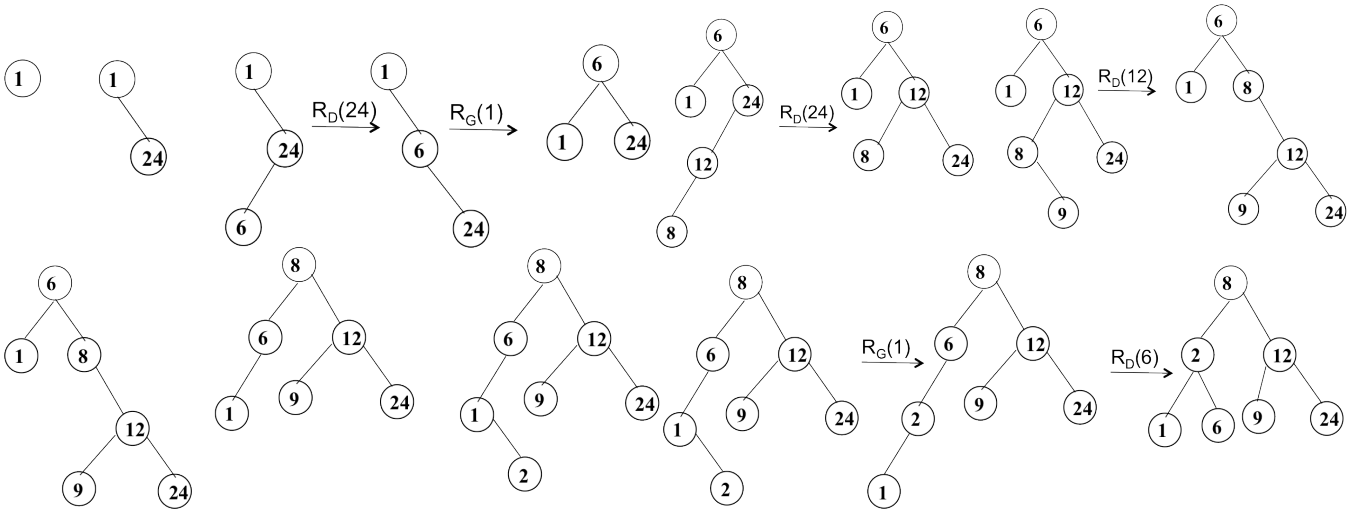
5 – Donner la liste des sommets en ordre préfixe, en ordre infixe et en ordre suffixe de l'arbre B.

Ordre préfixe : 1,24,6,2,12,8,9

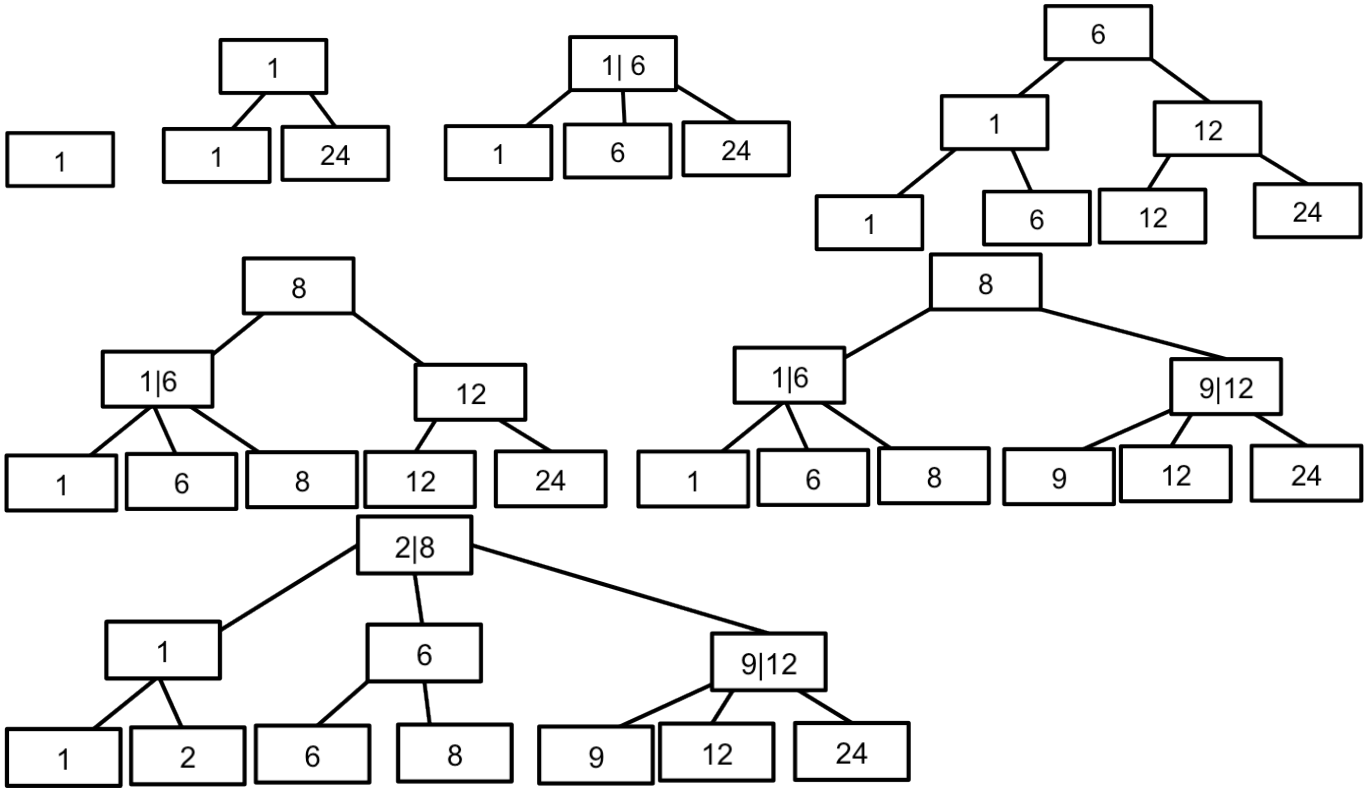
Ordre infixe : 1,2,6,8,9,12,24

Ordre suffixe : 2,9,8,12,6,24,1

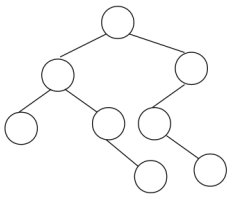
6 – Construire l'arbre binaire de recherche AVL correspondant à l'insertion consécutive de la liste A. On dessinera l'arbre après chacune des quatre premières insertions ainsi que l'arbre final.



7 - Construire le 2-3 arbre correspondant à l'insertion consécutive de la liste A. On dessinera l'arbre après chacune des quatre premières insertions ainsi que l'arbre final.



Question 3. On définit la longueur de cheminement interne d'un arbre binaire comme la somme des hauteurs des sommets interne de l'arbre (nombre d'arêtes de la racine au sommet). Ecrire la fonction *cheminementInterne* qui prend en paramètre un arbre et calcule cette longueur en utilisant uniquement les primitives du type abstrait. Par exemple, la fonction renvoie 6 sur l'arbre ci-dessous



```
Fonction cheminementInterne(ref A :arbreBinaire d'objets ;val h :entier) :entier
    Var c :entier ;
    Début
        Si estFeuille(A) alors
            retourner(0)
        sinon
            c=h ;
            si filsGauche(A) !=NIL alors
                c=c+ cheminementInterne (filsGauche(A),h+1)
            finsi
            si filsDroit(A) !=NIL alors
                c=c+ cheminementInterne (filsDroit(A),h+1)
            finsi
            retourner( c)
        finsi
    fin
```

L'appel se fait avec cheminementInterne (A,0).

Question 4. Ecrire une fonction qui prend en entrée une file d'entiers et qui fournit en sortie une file telle que tous les nombres multiples de 3 se retrouvent en tête de la file et dans le même ordre que la file originale, les autres éléments sont dans la file en ordre inverse. La file de départ reste inchangée. Par exemple, [1,3,5,4,2,6,8[fournit en sortie en [3,6,8,2,4,5,1 [.

```
Fonction mul3File(val  F :file de entier) :file de entier ;
    var FC :file d'entier ;
    var P :  pile d'entier ;
    var v : entier ;
    Début
        creerPile(P) ;
        creerFile(FC) ;
        enfiler(F,NULL) ;
        tantque valeur(F) !=NULL faire
            v=valeur(F) ;
            si v  est multiple de 3 alors
                enfiler(FC,v) ;
            sinon
                empiler(P,v) ;
            finsi
            défiler(F) ;
        fintantque
        defiler(F) ;
        tantque !pileVide(P) faire
            enfiler(FC,valeur(P)) ;
```

```
        depiler(P)
    fintantque
    detruirePile(P)
    retourner(FC)
fin
```

Question 5. On souhaite enregistrer pour une gare donnée l'ensemble des trains avec les informations suivantes : provenance ou destination (30 caractères), horaire de départ ou d'arrivée (heure,minute), retard estimé (minutes), voie (entier). Dans la suite, on décrira précisément chaque structure de donnée. On précisera s'il s'agit ou non d'un containeur. On ne demande pas d'écrire les primitives mais uniquement de donner l'en-tête de la primitive et son rôle.

1 – Décrivez la structure de données *heure* permettant de stocker un horaire. En donner les primitives.

heure=structure
H :entier ;
M :entier ;
Finstructure ;

Ce n'est pas un containeur. Les primitives sont des get et set sur les champs.
Fonction getH(val h :heure) :entier / récupération du champ H */*
Fonction getM(val h :heure) :entier / récupération du champ M */*
Fonction setH(val h :heure;val e :entier) :vide / modification du champ H */*
Fonction setM(val h :heure;val e :entier) :vide / modification du champ M */*

2 – Décrivez la structure de données *train* permettant de décrire les informations liées à un train. En donner les primitives.

Train=structure
Ville : tableau[1..30]de car ;
Provenance :booléen ; / vrai si le train vient de la ville faux si c'est sa destination*/*
Htrain : heure;
Retard :entier ;
Voie : entier
Finstructure ;

Ce n'est pas un containeur. Les primitives sont des get et set sur les champs.
Fonction get<X>(val T :train) :<typeX> / récupération du champ <X> */*
Fonction set<X>(val T :train ;val e : <typeX>) :vide / modification du champ <X> */*
<X>==Ville et <typeX>== tableau[1..30]de car
ou <X>== Provenance et <typeX>== booléen
ou <X>== Htrain et <typeX>== heure
ou <X>== Retard et <typeX>== entier
ou <X>== Voie et <typeX>== entier

3 – Quelles sont les structures de données que l'on peut utiliser pour l'ensemble des trains d'une gare si on souhaite répondre en temps constant à la question « Trains en provenance de x » où x est une ville. Justifiez. Donnez les avantages et les inconvénients de chacune.

On peut choisir une table de hashage dont la clé est constituée par les champs ville et heure. Il faut considérer l'heure pour distinguer deux trains différents desservant une même ville. On peut choisir un dictionnaire, le mot enregistré étant la concaténation des champs ville et heure. Le type de la donnée satellite sera ^polygone

Un dictionnaire est à la fois plus coûteux en mémoire et en temps pour l'adjonction d'un train que le hashage. Cependant les trains ne changeant que peu au cours du temps (à part le retard et les horaires été/hiver), le dictionnaire peut être construit et gardé au cours du temps et on aura une garantie d'accès à la donnée satellite en temps constant. Les tables de hashage peuvent devenir coûteuse en temps en cas de mauvaise résolution des conflits.

Dans les deux cas ce sont des containeurs. Les primitives sont celles vu en cours et modifiées pour prendre en compte une donnée satellite.

4 – On souhaite accéder en O(1) au premier train en partance quelle que soit sa destination. Quelle structure de données utiliser.

On choisit une file de priorité à cause des retards. Dans les noeuds on ne stocke que les trains tels que le champ Provenance est à faux. Quand un retard est annoncé, la filede priorité peut être modifiée.

La clé sera constituée par la valeur Htrain+Retard avec une fonction d'ordre un peu particulière puisque elle dépend de l'heure dans la journée. Par exemple si il est 16h30, un train qui part à 0h30 est « plus tard » qu'un train partant à 17h30. On stockera dans le nœud l'adresse de la donnée satellite. C'est donc un containeur avec les primitives habituelles.

5 – Décrire la structure de données *gare* qui permet de décrire un ensemble de trains en prenant en compte les questions précédentes.

gare=structure

Dtrain :dico ;
Ttrain : tas ;
Finstructure

Ce n'est pas un containeur au sens propre puisque c'est une structure mais comme elle contient deux containeurs elle joue le rôle de containeur.

6 – Ecrire la fonction *ajoutertrain* qui prend comme paramètre une variable de type train et permet d'ajouter le train dans une variable de type *gare*. On ne demande pas de reimplémenter les primitives vues en cours mais de les utiliser.

```
Fonction ajoutertrain(ref G :gare ;val T :train) :vide ;
   début
      ajouter(Dtrain,T) ;
      ajouter(Ttrain,T)
   fin
```

Afin que les fonctions appelées, il faut les modifier :

- *pour le dico prise en compte du mot constitué des 2 champs*
- *pour le tas, construction et prise en compte d'une fonction de comparaison des heures*

Dans les deux cas, il faudra ajouter dans les cellules la gestion des données satellites via T.

Listes simplement chainées (listeSC)

fonction valeur(val L:liste d'objet):objet;
fonction debutListe(val L:liste d'objet) :vide ;
fonction suivant(val L:liste d'objet) :vide ;
fonction listeVide(val L:liste d'objet): booleen;
fonction créerListe(ref L:liste d'objet):vide;
fonction insérerAprès(ref L:liste d'objet; val x:objet;):vide;
fonction insérerEnTete(ref L:liste d'objet val x:objet):vide;
fonction supprimerAprès(ref L:liste d'objet):vide;
fonction supprimerEnTete(ref L:liste d'objet):vide;
fonction detruireListe(ref L:liste d'objet):vide;

Listes doublement chainées (listeDC)

fonction finListe(val L:liste d'objet):vide;
fonction précédent(val L::liste d'objet): vide;

Piles

fonction valeur(ref P:pile de objet):objet;
fonction pileVide(ref P:pile de objet):booléen;
fonction créerPile(P:pile de objet) :vide
fonction empiler(ref P:pile de objet;val x:objet):vide;
fonction dépiler(ref P:pile de objet):vide;
fonction detruirePile(ref P:pile de objet):vide;

Files

fonction valeur(ref F:file de objet):objet;
fonction fileVide(ref F:file de objet):booléen;
fonction créerFile(F:file de objet);vide;
fonction enfiler(ref F:file de objet;val x:objet):vide;
fonction défiler (ref F:file de objet):vide;
fonction detruireFile(ref F:file de objet):vide;

Arbres binaires

fonction getValeur(val S:sommet):objet;
fonction filsGauche(val S:sommet):sommet;
fonction filsDroit(val S:sommet):sommet;
fonction pere(val S:sommet):sommet;
fonction setValeur(ref S:sommet;val x:objet):vide;
fonction ajouterFilsGauche(ref S:sommet,val x:objet):vide;
fonction ajouterFilsDroit(ref S:sommet,x:objet):vide;
fonction supprimerFilsGauche(ref S:sommet):vide;
fonction supprimerFilsDroit(ref S:sommet):vide;
fonction detruireSommet(ref S:sommet):vide;
fonction créerArbreBinaire(val Racine:objet):sommet;
fonction detruireArbreBinaire(val Racine:objet):sommet;

Arbres planaires

fonction valeur(val S:sommetArbrePlanaire):objet;
fonction premierFils(val S:sommetArbrePlanaire):sommetArbrePlanaire;
fonction frere(val S:sommetArbrePlanaire):sommetArbrePlanaire;
fonction pere(val S:sommetArbrePlanaire):sommetArbrePlanaire;
fonction créerArborescence(val racine:objet):sommetArbrePlanaire;
fonction ajouterFils(ref S:sommetArbrePlanaire,val x:objet):vide;
fonction supprimerSommet(ref S:sommetArbrePlanaire):vide;
fonction detruireArborescence(val racine:objet):sommetArbrePlanaire;

Arbres binaire de recherche

fonction ajouter(ref A:arbreBinaire d'objets, val v:objet):vide;
fonction supprimer(val A: arbreBinaire d'objets, val v:objet):vide;

Tas

fonction valeur(ref T:tas d'objet): objet;
fonction ajouter(ref T:tas de objet, val v:objet):vide;
fonction supprimer(val T:tas de objet):vide;
fonction creerTas(ref T:tas,val:v:objet):vide;
fonction detruireTas(ref T:tas):vide;

File de priorité

fonction changeValeur(ref T:tas d'objet,val s:sommet,val v:objet):vide;

Dictionnaire

fonction appartient(ref d:dictionnaire,val M::mot):booléen;
fonction creerDictionnaire(ref d: dictionnaire):vide ;
fonction ajouter(ref d:dictionnaire,val M::mot):vide;
fonction supprimer(ref d:dictionnaire,val M:mot):vide;
fonction detruireDictionnaire(ref d:dictionnaire):vide;

Table de Hachage

fonction chercher(ref T:tableHash de clés, val v:clé):curseur;
fonction créerTableHachage(ref T: tableHash de clé, ref h:fonction):vide;
fonction ajouter(ref T:tableHash de clé,val x:clé):booleen;
fonction supprimer((ref T:tableHash de clé,val x:clé):vide;
fonction detruireTableHachage(ref T: tableHash de clé, ref h:fonction):vide;

FIN