Université Ibn Tofail **A.U. 2018/2019**
Faculté des Sciences **Filière : SMI/SMA**
Département d'Informatique **Semestre 4**
Kénitra **Structures de Données.**

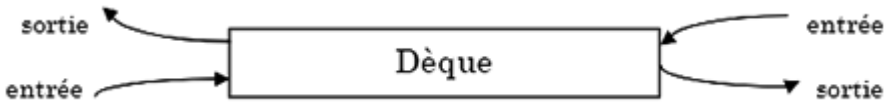
Examen (1h30)

Exercice 1 :

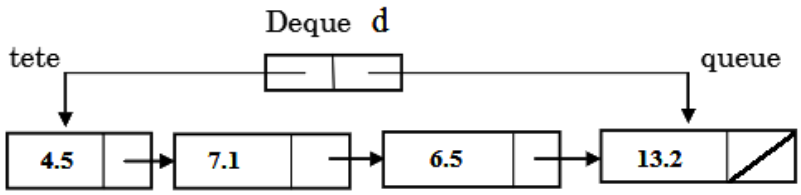
1. Construire l'arbre binaire de recherche obtenu en insérant les éléments de la liste suivante dans leur ordre d'arrivée: **30, 60, 10, 35, 50, 6, 65, 55, 8**.
2. Appliquer sur l'arbre précédent les rotations nécessaires pour le transformer en un arbre équilibré AVL. Pour cela étiqueter l'arbre binaire de recherche en calculant pour chaque nœud son facteur d'équilibre.
3. Définir la structure de données **ArBin** pour représenter les arbres binaires d'entiers.
4. Ecrire la fonction **fusion**(ArBin *R1, ArBin *R2) qui prend deux arbres binaires de recherches de racines R1, R2 et retourne la racine d'un arbre binaire de recherche contenant les deux. Utiliser les deux fonctions **suppr**(R, v) et **inser**(R, v) qui permettent respectivement la suppression et l'insertion de clé v dans l'arbre binaire de racine R.

Exercice 2 :

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



On considère une liste chainée de réels manipulée comme une Dèque.



Les opérations de manipulation de la Dèque sont les suivantes :

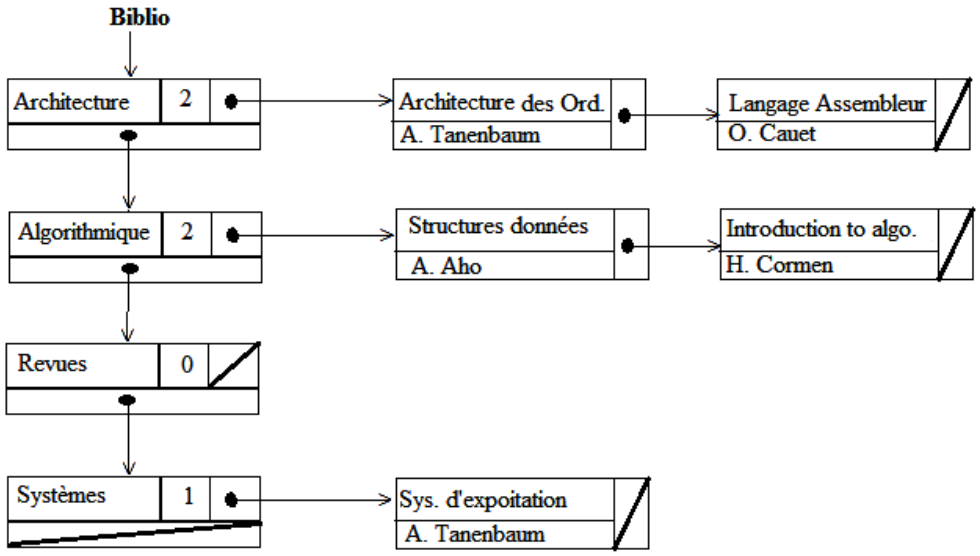
- **initDeque**() qui initialise une dèque.
- **DequeVide**(Deque d) qui vérifie si une dèque est vide.
- **EnDequer** (d, e, sens) permet de rajouter un élément **e** à la dèque **d** soit en tête soit en queue; **sens** est défini pour désigner l'extrémité à utiliser et sera égal à 1 si l'ajout est en tête et -1 si l'ajout est en queue.
- **DeDequer** (d, sens) permet de supprimer un élément soit en tête soit en queue et le retourner.

1. Donner la spécification abstraite de la structure **Deque**.

2. Définir le type **Maillon** de la liste de réels et le type **Deque**.
3. Implémenter opérations de manipulation de la Dèque.

Exercice 3 :

Dans cet exercice, un étudiant en informatique souhaite représenter sa bibliothèque personnelle en utilisant une structure dynamique. La structure proposée est représentée dans la figure suivante :



La liste verticale **Biblio** qui contient les catégories des livres **(ctLiv :** chaîne de 10 caractères) avec le nombre de livres (**nbLiv :** un entier) dans chacune et l'adresse de la liste des livres (**ListeLiv**), tandis que les listes horizontales (**ListeLiv)** contiennent les titres des livres (**titre :** chaîne de 20 caractères) avec leurs auteurs (**auteur :** chaîne de 16 caractères) dans chaque catégorie.

Les opérations prévues sur cette structure de données sont :

- **initBiblio**() : permet de créer une structure vide.
- **rechCat**(Biblio , ctLiv) : permet de vérifier l'existence d'une catégorie ctLiv dans la liste Biblio. Elle retourne l'adresse du maillon contenant la catégorie s'il existe sinon elle retourne NULL.
- **ajoutCat**(Biblio, ctLiv) : permet l'ajout à la fin de la liste d'une nouvelle catégorie.
- **ajoutLivre**(Biblio , ctLiv, titre, auteur) : permet l'insertion d'un nouveau livre dans la liste des livres de catégorie ctLiv (l'ajout se fait au début de la liste).
- **supprimCat**(Biblio , ctLiv) : supprime une catégorie avec tous ses livres.
- **afficheLivre**(Biblio , ctLiv) : affiche des livres d'une catégorie donnée ctLiv.
- **totalLivres**(Biblio) : retourne le nombre total de livre dans la bibliothèque.

1. Donner la spécification abstraite de la structure de données **Biblio**.
2. Donner la déclaration des structures de données nécessaires à l'implémentation de cette bibliothèque.
3. Implémenter les opérations précédentes.