Travaux Dirigés – TD 7

Exercice 46. Opérations sur les tableaux

1. Écrire une procédure pour saisir un vecteur des éléments réels positifs de taille N (la valeur de N sera demandée à l'utilisateur)
2. Écrire trois fonctions pour calculer :
   — La somme de deux vecteurs saisis par l'utilisateur
   — La soustraction de ces deux vecteurs
   — Le produit scalaire de ces vecteurs
3. Écrire une procédure qui affiche le résultat des opérations.

Exercice 47. Pointeurs et structures

1. Donner la déclaration de la structure suivante :
   etudiant
   — le nom et le prénom
   — la note du médian
   — la note du final
   — la note moyenne
2. En utilisant un tableau dynamique type "etudiant", écrire un programme pour saisir une liste de N étudiants (la valeur de N sera saisie par l'utilisateur). La note moyenne est donnée en pondérant par 0.4 et 0.6 les notes du médian et du final, respectivement.
3. Ecrire une procédure/fonction permettant de trier cette liste d'étudiants par ordre croissant selon le champ nom et prénom

Exercice 48. Opérations matricielles

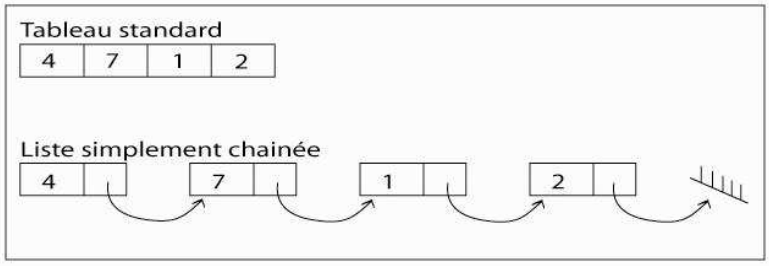
Écrire une procédure qui permet de lire deux matrices de taille quelconque (l'utilisateur présice la taille) et de faire l'addition, la soustraction et la multiplication des deux matrices.

Écrire une procédure qui permet de faire la transposée d'une matrice de taille quelconque (l'utilisateur présice la taille).

Exercice 49. Listes linéaires chaînées

Le conteneur de données le plus utilisé est le tableau. Un tableau devient peu pratique si les éléments du tableaux sont sujets à des changements au cours du programme. Une suppression d'un élément au milieu forcera l'utilisateur à décaler tous les éléments qui sont après l'élément supprimé pour conserver la cohérence du conteneur.

Une liste chaînée est un conteneur différent dans le sens où les éléments de la liste sont répartis dans la mémoire et reliés entre eux par des pointeurs. On peut ajouter et enlever des éléments d'une liste chaînée à n'importe quel endroit, à n'importe quel instant, sans devoir recréer la liste entière.



La liste est caractérisée par un pointeur qui pointe vers le premier élément. La liste est vide au départ, c'est à dire, elle contient aucun élément et le pointeur est mis à `NULL`. Un élément de la liste est composé des données et d'un pointeur qui pointe vers le prochain élément. Le dernier élément n'a pas de successeur et son pointeur est mis à `NULL`. Pour simplifier l'implémentation, on considère que chaque élément contient un entier.

1. Écrire la structure `element` d'un élément de la liste et définir le type `liste`
2. Écrire la procédure `void insererElement(liste l, struct element e)` qui permet d'insérer l'élément `e` dans la liste
3. Écrire la fonction `struct element * successeur(liste l, struct element e)` qui retourne un pointeur vers le successeur de l'élément `e`. Si l'élément est le dernier ou n'existe pas dans la liste, la fonction doit retourner `NULL`
4. Écrire la fonction `struct element * predecesseur(liste l, struct element e)` qui retourne un pointeur vers le predecesseur de l'élément `e`. Si l'élément est le premier ou n'existe pas dans la liste, la fonction doit retourner `NULL`
5. Écrire la procédure `void supprimerElement(liste l, struct element e)` qui permet de retirer l'élément `e` de la liste. La liste reste inchangée si l'élément n'existe pas dans la liste.
6. Écrire la procédure `void trierListe(liste l)` qui permet de trier la liste dans un ordre décroissant.
7. Modifier la fonction `insererElement` pour permettre une insertion ordonnée.