PROGRAMMATION AVANCÉE
TP1

1. Rappels et Tableaux à une dimension

Exercice 1 Écrivez une fonction `AfficheTab` qui :

(1) Prend en paramètre un pointeur sur un tableau d'entiers, et la taille du tableau
(2) Affiche l'ensemble des éléments du tableau, les uns à la suite des autres. Les éléments seront séparés par des virgules ;
(3) Quitte immédiatement si le pointeur du tableau est à `NULL`.

Questions :
Que se passe-t'il si on utilise cette fonction sur un tableau qui contient des `unsigned short int *` ?
Même chose si on utilise un tableau qui contient des `unsigned int *` ?

Exercice 2 Écrivez une fonction qui :

(1) Prend pour paramètre un entier positif n.
(2) Qui crée un tableau de n entiers.
(3) Qui remplie avec les cases avec les n premiers termes de la fonction de Fibonnacci.
(4) Qui renvoie le Tableau.

Exercice 3 Écrivez une fonction qui affiche la taille des types suivants :

(1) `int *`
(2) `int*`
(3) `void`
(4) `void *`
(5) `long int`
(6) `long int *`

Interprétez les résultats. Dans quels cas

Exercice 4 Est-ce que les instructions suivantes sont valides ?

```
void * Z = malloc(10*4) ;
int * X = Z ;
AfficheTab(X,10) ;
```

Où `AfficheTab` est la fonction qui affiche le contenu d'un tableau d'entiers.
Que se passe-t-il si l'on écrit :

```
AfficheTab((int *)((char *)X+8),8) ;
```

Est-ce correct ? Si oui qu'est ce qui est affiché ? Donnez une explication.

Exercice 5 Écrivez une fonction qui prend en paramètre un tableau d'entiers (`int *`) et n la taille du tableau. Et qui détermine si le tableau passé en paramètre correspond à une permutation. Une permutation, dans notre cas, sera un tableau où toutes les valeurs contenues dans le

tableau sont comprises entre 0 et $n-1$ (bornes incluses) et chaque valeur apparaît exactement une fois.

Exercice 6 Écrivez une fonction qui crée un tableau d'entiers de taille n. Et qui remplit le tableau avec une permutation aléatoire. Vous pourrez utilisez le générateur aléatoire de la librairie standard. Cela fonctionne en deux temps : L'initialisation du processus `srand(time(NULL)) ;` se fait une seule fois et permet d'initialiser la *graine* (seed en Anglais) qui sert pour les générations aléatoires à venir. Afin d'avoir un peu de variation sur l'initialisation, nous utilisons l'heure du système (`time(NULL)`.

Ensuite on peut utiliser la fonction `rand()` qui renvoie une nombre aléatoire entier. Pour rester entre O (inclus) et n (exclus) on peut utiliser `rand()% n`.

Pour générer une permutation aléatoire on peut partir de la permutation identitée et effectuer des échanges entre les positions i et j, où i et j sont des valeurs aléatoires comprises entre 0 et $n-1$.

Exercice 7 Écrivez une fonction qui prend en paramètre un tableau de caractères `T` et une permutation de même taille Et qui permute les éléments.

Et qui procède à la permutation des éléments. Par exemple : si le Tableau de caractère est "`Bonjourx!`" et la permutation est $(8,7,6,5,4,3,2,1)$ alors le tableau de caractère résultant sera "`!xruojnoB`".

De manière plus générale pour la case `T[i]` la valeur de `permutation[i]` indiquera la nouvelle position dans le tableau résultat. Ainsi le 'B' de T se retrouve en dernière position.

2. Tableaux à 2 dimensions

Exercice 8 Écrivez une fonction qui prendra pour paramètre n et m deux entiers positifs et qui créera un tableau avec n lignes et m colonnes. Les valeurs stockées seront des flottants. On veillera à renvoyer un pointeur non nul si et seulement si tous les `malloc` ont réussi...

Exercice 9 Écrivez une fonction qui permet de remplir la matrice en demandant à l'utilisateur de rentrer des valeurs. (On pourra utiliser `scanf()`).

Vous écrirez également une fonction qui fait l'affichage.

Exercice 10 Écrivez une fonction qui fait la rotation à droite de la matrice. Pour la matrice suivante : $\begin{array}{|c|c|c|}\hline 1&2&3\\\hline 4&5&6\\\hline\end{array}$ sa rotation à droite est : $\begin{array}{|c|c|}\hline 4&1\\\hline 5&2\\\hline 6&3\\\hline\end{array}$

Exercice 11 Faites une fonction qui Libère la matrice. En pensant à gérer les pointeurs qui peuvent être nuls

Exercice 12 Écrivez une fonction qui prend pour paramètre un entier positif n. et qui crée un tableau à 2 dimensions qui contiendra les n premières lignes du triangle de Pascal.

Faites une fonction qui effectue l'affichage et une autre qui effectue la libération de la mémoire.

Le Triangle de Pascal pour les 5 premières lignes.

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

Chaque ligne commence par un 1 et finit par un 1. Le nombre d'éléments à la ligne i est i (i commence à 1). Un élément qui n'est ni au début ni à la fin de la ligne est obtenue en additionnant l'élément au dessus de lui avec l'élément qui est à au dessus et à gauche de lui.

3. Liste

Vous devez créer une structure de Liste doublement chaînée qui contiendra des entiers.
La structure sera articulée en deux parties :
— La structure de liste.
— La structure d'éléments.
La structure de liste sera de la forme :

```
struct LListe{
  int nb_elements_ ;
  Element * head_ ;
  Element * tail_ ;
} ;

typedef struct LListe Liste ;
```

La structure Element sera de la forme :

```
struct LElement{
  /**
   * Element suivant / NULL si dernier element
   */
  Element * next_ ;
  /**
   * Element précédente
   */
  Element * prev_ ;
  /**
   * Valeur stockée dans la cellule.
   */
  int val_ ;
}
typedef struct LElement Element ;
```

Les définitions précédentes seront stockées dans un même fichier `.h` .
Vous écrirez les fonctions suivantes :

(1) Création d'une liste vide.
(2) Longueur de la liste.
(3) Ajout d'un élément à la fin.
(4) Ajout d'un élément au début.
(5) Valeur du dernier élément.
(6) Valeur du premier élément.
(7) Faire une structure permettant le parcours de la liste sans forcément avoir accès directement à la cellule. (structure appelée itérateur)

4. Arbre Binaire

Le but maintenant est de coder une structure capable de représenter les arbres binaires et les arbres binaires de recherche (Pour rappel un arbre binaire de recherche est tel que la valeur r de la racine est supérieure ou égale à la valeur maximum du sous-arbre gauche et inférieure strictement à la valeur minimum du sous-arbre droit.)

La structure d'arbre sera la suivante :

```
struct Arbre {
  int val_ ;
  struct SousArbre * sa_ ;
} ;

struct SousArbre{
  // Sous arbre gauche
    struct Arbre * sag ;
  //  sous arbre droit
   struct Arbre * sad ;
} ;
```

Vous devez écrire les fonctions suivantes :

(1) Ajouter un élément dans l'arbre

(2) Compter le nombre d'éléments

(3) Afficher les éléments de l'arbre ordre préfixe/suffixe/infixe [1].

(4) Copier les valeurs d'un arbre dans un tableau (selon l'ordre infixe).

Pour les arbres binaire de recherche on devra pouvoir faire les opérations spécifiques :

(1) Chercher l'élément maximum (resp minimum)

(2) Insérer un élément à la bonne position.

(3) Détecter si l'arbre est équilibré.

1. préfixe = on affiche la valeur de la racine puis le sous-arbre gauche et ensuite le sous-arbre droit. Suffixe= sous-arbre gauche, puis la racine, puis sous-arbre droit...