Algorithmique et Programmation 1
IMAC 1ere année

TP 11

Tableaux à deux dimensions et listes doublement chaînées

Dans cette séance de travaux dirigés, on travaillera sur les tableaux à deux dimensions statiques et dynamiques ainsi que sur les listes doublement chaînées.

Exercice 1. (Tableaux statique à deux dimensions)

1. Défininissez une constante `N` (par exemple 4) en début de programme. Cette constante sera la taille des tableaux utilisés dans tout l'exercice.
2. Définir une fonction `initialiseTab(int tab[N][N])` qui remlit `tab` par des valeurs scannées par l'utilisateur.
3. Définir une fonction `afficheTab(int tab[N][N])` qui affiche les valeurs de `tab` en effectuant des retour à la ligne à chaque nouvelle ligne de `tab`. Par exemple, si

```
1   tab = { {0, 1, 1, 1}, {1, 0, 1, 1}, {1, 1, 0, 1}, {1, 1, 1, 0} }
```

   votre fonction d'affichage doit donner :

```
1   0 1 1 1
2   1 0 1 1
3   1 1 0 1
4   1 1 1 0
```

4. Définir une fonction `remplitDiagonale(int tab[N][N])`, qui met à `0` toutes les valeurs de la diagonale de `tab`.
5. Définir une fonction `remplitPartieSup(int tab[N][N])` qui remplit toutes les cases de coordonnées `(i, j)` avec `i < j` par des `1`.
6. Définir une fonction `sym(int tab[N][N])` qui rend le tabeau `tab` symétrique : les valeurs de cases de coordonnées `(i, j)` avec `j < i` sont mises à `tab[j][i]`.
7. Modifiez vos codes des trois fonctions précédentes de sorte qu'elles ne contiennent aucun `if`.
8. En utilisant les fonctions déjà crées, écrivez un `main` dans lequel vous construisez le tableau donné en exemple de la question 3.

Exercice 2. (Tableaux dynamniques à deux dimensions)

1. Dans un main, défininissez un pointeur `tab` de pointeurs sur `int`. Dans la suite, on utilise chaque pointeur de `tab` comme un tableau simple et `tab` comme un tableau double.

2. Scannez la taille de `tab` et initialisez `tab` grâce à un `malloc`.
3. Scannez la taille des pointeurs de `tab` et initialisez-les grâce à un `malloc`. On suppose que chacun de ces pointeurs ont la même taille.
4. Remplissez `tab` en scannant ses valeurs. Affichez `tab` ligne à ligne.
5. En utilisant un `free`, libérez les espaces mémoires occupés par les pointeurs de `tab` puis libérez l'emplacement mémoire occupé par `tab`.

Exercice 3. (Listes doublement chaînées)

Dans cet exercice, on aborde les listes doublement chaînées : il s'agit de listes où chaque cellule pointe à la fois sur la cellule qui lui succède mais également sur celle qui la précède.

1. Définir les types structurés

```
1        typedef struct cellule{
2          int valeur;
3          struct cellule *suivante, *precedent;
4        }Cellule, *ListeD;
```

   Le champ `precedent` de la cellule d'entrée d'une liste est égal à `NULL`.

2. Définir les fonctions `insereTete` et `afficheListeD`.
3. Définir une fonction `insereApres(ListeD lst, int n int m)` qui prend en arguments une liste ainsi que deux entiers `n` et `m` et ajoute une cellule contenant la valeur `m` après la cellule contenant la valeur `n`. Si `n` n'est pas présente dans `lst`, alors la nouvelle cellule est ajoutée à la fin de `lst`.