Algo Prog 2021-2022: TP1

tags: `algo` `bourg` `c++` `iut` `tp`

Exercice 1

Ecrire une fonction `char lettre_max(char ch[])` qui retourne la lettre minuscule la plus présente dans la chaine de caractères `ch`. Il se peut que vous ayez besoin d'un tableau d'occurrences (càd un tableau d'entiers de taille 26) qui compte le nombre de fois où chaque lettre minuscule est présente dans le tableau.
Testez votre fonction dans votre main à partir d'une phrase lue par l'utilisateur.

Exercice 2 : Carrés magiques

Le but de cet exercice est la réalisation d'algorithmes de détection/création de carrés magiques.

Le nombre de cases d'un coté de notre carré sera déclaré comme une constante N en début de programme. On pourra commencer avec des carrés de taille 3.

```
const int N=3;
```

Dans la suite, on définira un *carré* comme étant un tableau d'entiers de taille $N \times N$.

Lecture et affichage d'un carré de nombres entiers

On demande d'écrire une fonction qui demande à l'utilisateur de renseigner le contenu d'un carré de côté N (on dit aussi d'ordre N) et de l'afficher sous la forme d'un carré (cf exemple ci dessous). Le dialogue programme-utilisateur doit suivre le modèle de l'exemple, où les réponses de l'utilisateur sont les entiers soulignés suivis d'un retour à la ligne. L'entête pourra être la suivante:

```
void init_carre(int carre[N][N]) ;
```

Exemple (avec $N=3$):

```
case 1, 1 = 4
case 1, 2 = 9
case 1, 3 = 2
case 2, 1 = 3
case 2, 2 = 5
case 2, 3 = 7
case 3, 1 = 8
case 3, 2 = 1
case 3, 3 = 6

Vous avez fourni le carré :
4 9 2
```

```
3 5 7
8 1 6
```

Carré presque magique

Un carré presque magique d'ordre n, vérifie que les sommes des valeurs de chacune des n lignes, n colonnes et des 2 diagonales sont toutes égales. On demande d'écrire une fonction qui teste si un carré passé en paramètre est presque magique et renvoie un booléen vrai ou faux. Testez votre fonction.

On peut constater que le carré ci-dessous est, suivant cette définition, presque magique car chaque somme fait 6.

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

Carré magique

Un carré magique d'ordre n

- comporte n^2 cases, contenant tous les entiers de 1 à n^2 (par conséquent 2 cases distinctes contiennent 2 entiers distincts)
- et il vérifie que les sommes des valeurs de chacune de n lignes, n colonnes et des 2 diagonales sont toutes égales

Il est clair que ces sommes sont toutes égales à $n(n^2+1)/2$.

Exemple : carré magique d'ordre 3

```
4 9 2
3 5 7
8 1 6
```

Ecrivez une fonction qui teste si un carré passé en paramètre est magique.

Verification des doublons

On propose de modifier le programme de la question `init_carre` pour qu'il vérifie, dès la lecture, les données fournies par l'utilisateur.
Pour un carré d'ordre n, chaque fois que l'utilisateur fournit la valeur d'une nouvelle case, le programme doit contrôler que cette valeur est comprise entre 1 et n^2 et différente des valeurs précédemment fournies. Pour effectuer cette dernière vérification, on suggère d'utiliser un tableau de booléens `DEJA_FOURNI` dont les indices sont les valeurs entre 1 et n^2. Le fait que `DEJA_FOURNI[v]` contient la valeur VRAI (resp. FAUX), indiquera que la valeur v a déjà été fournie (resp. n'a pas déjà été fournie).

Construction automatique de carrés magiques

Au cours de l'histoire, il y a eu de nombreux procédés pour construire automatiquement des carrés magiques. Nous présentons une méthode trouvée dans un livre grec du byzantin Manuel Moschopoulos (XIV° siècle) qui permet la construction de carrés magiques d'ordre impair.

Numérotons les lignes de haut en bas et les colonnes de gauche à droite.

Notons $L(x)$L(x) le numéro de ligne et $C(x)$C(x) le numéro de colonne de la case sur laquelle se trouve la valeur x (avec $L(x)$L(x) et $C(x)$C(x) compris entre 1 et N). Soit $n=2*k+1$n=2*k+1 l'ordre du carré.
Algorithme:

(1) La valeur 1 est placée dans la case (k+2, k+1) autrement dit
L(1) = k+2
C(1) = k+1

(2) Lorsque la position de la valeur x est connue, la position de la valeur x+1 est définie par les règles cidessous.

- Si x n'est pas multiple de n alors
  L(x+1) = 1 + (L(x) mod n)
  C(x+1) = 1 + (C(x) mod n)
- Si x est multiple de n alors
  L(x+1) = 1 + ((L(x) + 1) mod n)
  C(x+1) = C(x)

Exemple : pour n = 3 (donc k = 1)

x	L(x)	C(x)
1	3	2
2	1	3
3	2	1
4	1	1
5	2	2

x	L(x)	C(x)
6	3	3
7	2	3
8	3	1
9	1	2

Ce qui donne le carré magique présenté à la question (3)

Ecrire un programme qui construit et affiche le carré magique obtenu par le procédé ci-dessus. Testez-le en faisant varier N.

Compléter ce programme afin qu'en plus il vérifie, après l'affichage du carré, que ce carré est bien magique, en utilisant pour cela des éléments du programme des questions précédentes.

La preuve que cet algorithme construit un carré magique n'est pas triviale, on peut la trouver dans le livre Histoire d'algorithmes, du caillou à la puce, Collection Regards sur la science, Editeur Belin (1994)

Pour aller plus loin : carrés latins

Un carré latin est une variante des carrés magiques. Il s'agit d'un carré où chaque ligne et chaque colonne contient toutes les valeurs entre 1 et N. Par exemple, un sudoku est un carré latin avec N=9 (cependant, le sudoku a des contraintes supplémentaires avec les sous-carrés 3×3).

Par exemple, le carré ci-dessous est un carré latin
1 2 3
3 1 2
2 3 1

Ecrivez une fonction qui prend un carré en paramètre et retourne un booléen indiquant si le carré est latin ou pas.

Ecrivez une fonction qui compte le nombre de carrés latins de taille 3. Combien y en a-t-il ?

Pour cela, vous pouvez tester tous les carrés possibles (en faisant varier les valeurs entre 1 et N), et ne garder que ceux pour lesquels la fonction précédente retourne vraie. Il pourra vous être utile de générer toutes les permutations de 1 à

N via la fonction `next_permutation` dont la documentation est ici (https://www.cplusplus.com/reference/algorithm/next_permutation/). Cette fonction s'utilise sur des tableaux 1D et dans un tableau 2D, tab[i] correspond à la ligne i du tableau sous la forme de tableau 1D.

Testez votre fonction pour N=4 ou 5.