

ZZ1 – C – TP n°1

Tableau statique à 2 dimensions - SODOKU

Le principe du SODOKU est très simple. Sur une grille 9x9, il faut placer les chiffres de 1 à 9 tels que toutes les lignes, toutes les colonnes et toutes les régions ne doivent voir ces chiffres qu'une seule fois. La première figure montre un sodoku résolu. Les nombres en diagonale ont été donnés au départ. La deuxième figure nomme chacune des neuf régions. Les figures viennent de la Wikipedia.

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

4	1	5	6	3	8	9	7	2
0,0	0,1	0,2						
1,0	1,1	1,2						
2,0	2,1	2,2						

On vous propose de réaliser le **moteur de jeu texte** du sodoku. Pour cela, vous devrez compiler et tester régulièrement. Sous gcc, vous devez utiliser les options :

```
-Wall -ansi -pedantic
```

Les éléments devront être déclarés et bien documentés : vous devez mettre ensemble les constantes en début de programme.

Pour définir une matrice de taille N.M, cela se fait très simplement :

```
float matrice [N][M] ;
```

Un tableau à deux dimensions ne peut exister en C, il s'agit en fait d'un tableau à une dimension où les lignes sont stockées les unes après les autres. C'est pour cela, qu'il faut toujours donner la deuxième dimension lors de passage de paramètres (La deuxième dimension est utilisée dans le calcul de l'emplacement des éléments du tableau).

```
void traiter(float m[][M]) ;
```

1. Ecrire le squelette du programme : la fonction `int main()` qui affiche "SODOKU".
2. Déclarer une matrice carrée de N cases de côté appelée **grille** dans la fonction `main()`. La taille N est une constante évaluée par le préprocesseur. Déclarer et initialiser une variable entière **remplissage**, elle représente le nombre d'éléments dans la grille.
3. Ecrire une fonction **initialiser()** qui prend en paramètre **grille** et qui initialise tous les éléments de la grille à 0 et qui renvoie un entier 0.

4. Ecrire une fonction **afficher()** qui prend en paramètre la grille et qui l'affiche en texte. Vérifier que tout marche jusque là.
5. Ecrire une fonction **generer()** qui permet de placer un certain nombre de chiffres sur la grille. Elle prend en paramètre la grille et un entier, le germe du générateur du nombre aléatoire. Elle renvoie le nombre d'éléments placés dans le tableau. Dans un premier temps, cette fonction ne fera rien mais vous placerez ici les valeurs utiles aux tests de fin de jeu.
6. Ecrire une fonction **saisir()** qui demande à l'utilisateur de saisir au clavier les indices **i** et **j** et la valeur **v** à placer à l'emplacement **(i,j)**. La fonction doit vérifier la validité des indices et de la valeur. Si la case n'est pas occupée, la valeur doit être placée dans la grille. **remplissage** est alors incrémentée.
7. Ecrire la fonction **verifierLigne()** qui prend en paramètre un numéro et un sens (HORIZ ou VERT) qui vérifie que la ligne ou la colonne (suivant les cas) numéro est bien remplie. On pourra utiliser un tableau intermédiaire pour vérifier cela. La fonction retournera 1 si tout s'est bien passé, 0 sinon. Les constantes HORIZ de valeur 0 et VERT de valeur 1 sont à définir.
8. Ecrire la fonction **verifierRegion()** qui prend en paramètre deux indices **k** et **l** qui correspondent à la région **(k,l)** et qui renvoie 1 si la région est correctement remplie, 0 sinon.
9. Ecrire la fonction **verifierGrille()** qui renvoie 1 si la grille est correctement remplie et 0 sinon.
10. Ecrire le programme principal, en supposant que la seule condition d'arrêt est la réussite du sodoku (ce test ne devra être fait que si nécessaire).
11. Un petit problème subsiste : on peut réécrire sur les nombres qui ont été donnés au départ. Est-ce que vous pourriez donner une idée simple (avec uniquement ce que l'on a vu en cours pendant les semaines bloquées) qui permettrait de contourner ce problème ?

POINTEURS

A : Passage par valeur et par adresse

1. Ecrire une fonction **echangeParValeur** qui 'échange' deux variables **a** et **b** entières et que l'on passe **par valeur**. Dans le corps de la fonction, les valeurs sont affichées avant et après l'échange. Rappel : le passage par valeur est le passage par défaut en langage C, on travaille sur des copies de variables. Ecrire un programme principal qui teste cette fonction en affichant les valeurs de 2 variables entières **i** et **j** (déclarées localement dans le `main()`) avant et après l'appel à la fonction **echangeParValeur**.
2. Ajouter une fonction **echangeParAdresse** qui échange vraiment les deux variables **a** et **b** entières. Pour ce faire on fait un passage **par adresse**. Dans le corps de la fonction, les valeurs sont également affichées avant et après l'échange. Dans le programme principal on ajoute un affichage avant et après l'échange de valeurs de 2 variables entières **i** et **j** par la fonction **echangeParAdresse**.

B: Opérations sur les pointeurs (tester et comprendre le fonctionnement du code ci-dessous)

```
#include <stdio.h>

main()
{
    int    i, * ptri = &i;
    char   c1, * ptrc1 = &c1;

    printf("ptri = %d ptrc1 = %d \n",ptri, ptrc1);
    printf("ptri = %x ptrc1 = %x \n",ptri, ptrc1);
    ptri++;                ptrc1++;
    printf("ptri = %d ptrc1 = %d \n",ptri, ptrc1);

    return 0 ;
}
```

Après avoir testé ce code, vous allez l'adapter en faisant les modifications suivantes :

- Ajouter une nouvelle variable de type double (d) et d'un pointeur sur ce double (ptrd).
- Afficher pour chaque pointeur, la valeur pointée, l'adresse contenue dans le pointeur et l'adresse du pointeur (attention à ne pas confondre ces 2 dernières).
- Ajouter 2 au pointeur sur la variable double précision et affichez à nouveau la valeur contenue dans ce pointeur (constatez et comprenez l'écart entre cette valeur et la valeur précédente).
- Ensuite, déclarez encore une variable de type caractère (c2) et un pointeur sur cette variable (ptrc2).
- Ecrire une fonction **swap** qui fait un échange effectif des 2 pointeurs, de manière à placer la valeur '1' dans la variable c2 grâce au pointeur ptrc1 - et la valeur '2' dans la variable c1.
- Vérifiez les échanges des adresses contenues par les pointeurs et de valeurs de c1 et c2 grâce à des affichages avant et après l'échange. Est-ce que les adresses des pointeurs ont changé ? (Vérifiez-le toujours grâce à des affichages – attention à la précision du langage)

C: Saisie d'une chaîne de caractères

```
int main()                /* Tester et corriger le programme suivant */
{
    char *chaine;

    printf("Entrez une chaîne de caractères : ");
    scanf("%s", chaine);

    if (chaine == "Benny") printf("Deux chaînes sont identiques!\n");
}
```

Tester avec DDD

Après avoir compris les corrections à apporter au code ci-dessus, écrire un programme qui déclare un pointeur sur des caractères et qui saisit au clavier une chaîne de caractères, qui affiche cette chaîne, qui ajoute "The end !" en fin de chaîne, puis qui affiche à nouveau cette chaîne qui a été modifiée. Les débutants chercheront à minimiser le nombre de (Segmentation Fault / Erreur de Segmentation).

D: Allocation dynamique de tableau

Ecrire un programme qui déclare un pointeur sur des réels flottants double précision, puis qui alloue dynamiquement 1000000 éléments à partir de cette adresse. Initialiser ce tableau dynamique avec les carrés des indices de boucle. Afficher le tableau, puis rendre la mémoire allouée.