Exercices: Série 6 - Pointeurs - ICC-C 2025-2026

Rappel : Pour faire lire des valeurs d'un fichier texte à votre programme, vous pouvez utiliser la redirection de l'entrée :

```
./exo < fichier.txt
```

Votre code ne saura pas qu'il lit depuis un fichier. Utilisez `scanf` et/ou `fgets` sans intercaler des messages avec `printf`. Affichez seulement le résultat à la fin du programme avec `printf`.

1. Lecture et compréhension

Qu'affiche ce code ? Décrivez ce qui se passe à chaque étape de l'exécution. Vous pouvez dessiner des boîtes et les modifier au fur et à mesure.

```c
int plus10(int a) {
  a += 10;
  printf("plus 10 = %d\n", a);
  return a;
}

int main() {
  int v = 5;
  printf("v = %d\n", v);
  plus10(plus10(v));
  printf("v = %d\n", v);
}
```

Même question avec le code suivant. En quoi l'usage de pointeurs change-t-il votre réponse ?

```c
int* ptr_plus10(int *a) {
  *a += 10;
  printf("plus 10 = %d\n", *a);
  return a;
}

int main() {
  int v = 5;
  printf("v = %d\n", v);
  ptr_plus10(ptr_plus10(&v));
  printf("v = %d\n", v);
}
```

2. Échauffement : échanger trois variables

Écrivez une fonction `cycle` qui accepte 3 pointers sur des entiers, `a`, `b`, `c`. Elle doit faire « cycler » les valeurs pointées de sorte que, après la fonction, `a` ait pris l'ancienne valeur de `b`, `b` celle de `c`, et `c` celle de `a`.

Testez votre fonction avec le programme suivant :

```c
int main() {
  int x = 5, y = 7, z = 17;
  cycle(...); // à vous de transmettre x, y, z pour les paramètres a, b, c
  printf("%d %d %d\n", x, y, z);
}
```

qui doit afficher

```
7 17 5
```

3. Stack (pile)

Dans cet exercice nous allons implémenter une pile d'entiers. Comme une pile d'assiettes, où l'on ne peut ajouter et reprendre des assiettes que par le dessus.

Nous utiliserons un tableau pour stocker les éléments de la pile, donc une `int stack[]`. En plus de sa taille **maximale** `max_stack_size` (la taille totale du tableau, comme on l'a créé), on manipulera la taille **courante** `stack_size` de la pile. La taille courante ne pourra jamais dépasser la taille maximale (imaginez que s'il y a plus de 50 assiettes dans la pile, la pile s'écroule !).

La taille maximale pourrait donc être 50. La taille courante vaut 0 au départ. Ceci signifie que la pile est vide.

Implémentez une fonction

```
bool push(int stack[], size_t *stack_size, size_t max_stack_size, int elem);
```

qui empile un nouvel élément.

Dans notre implémentation nous considérons que le bas de la pile correspond à l'indice 0 du tableau `stack` (« à gauche »). Quand la pile n'est pas vide, et donc la valeur de `stack_size` est strictement supérieure à zéro, le haut de la pile se trouve à l'indice `stack_size - 1` (« à droite »). On peut rajouter un entier sur la pile si `stack_size < max_stack_size`. Dans ce cas, nous allons stocker l'entier à l'indice `stack_size` du tableau `stack` et ensuite nous allons incrémenter la variable `stack_size`. Cette fonction retourne `true` si l'élément a bien été stocké sur la pile, ou `false` si la pile était déjà remplie.

Nous aurons aussi besoin d'une fonction

```
bool pop(int stack[], size_t *stack_size, int *elem);
```

qui enlève un élément du haut de la pile et le stocke à l'emplacement de mémoire où pointe le paramètre `elem`. Quand `stack_size > 0`, l'entier du haut de la pile est `stack[stack_size - 1]`. Cette fonction retourne `true` quand l'opération réussit, et dans ce cas décrémente la variable `stack_size`. Elle retourne `false` quand la pile est vide, et dans ce cas elle ne touche pas à `elem`.

Pour tester votre code, dans la fonction `main` lisez un entier représentant la taille maximale de la pile `max_stack_size`, suivi d'un deuxième entier K. S'ensuivent K lignes contenant

- soit la chaîne `"pop"` ;
- soit la chaîne `"push"` suivie d'un entier.

Au début la pile est vide.

On aimerait afficher les résultats des opérations `pop`. Si la i-ème opération est `push` et elle échoue (car la pile est remplie), alors on aimerait afficher le message

```
Étape i : pile remplie
```

Si la i-ème opération est `pop` et elle échoue (car la pile est vide), alors on aimerait afficher le message

```
Étape i : pile vide
```

Rappel : Pour vérifier qu'une chaîne est égale à une autre il faut vérifier l'égalité élément par élément. Vous aviez déjà écrit du code similaire (et plus compliqué) pour `a_sous_chaine` la semaine dernière. Cette fois, vous aurez besoin d'une fonction

```
bool equalstr(const char s1[], const char s2[])
```

qui prend deux chaînes et qui vérifie que les éléments des deux chaînes sont égaux deux par deux.

Pour lire un *mot* (sans espace) depuis l'entrée, vous pouvez utiliser `%s` avec `scanf`. Il ne faut alors *pas* mettre de `&` devant le nom du tableau.

```
char mot[100];
scanf("%s", mot);
```

Cela s'intègre mieux que `fgets` s'il y a d'autres `scanf`.

Exemple 1 Pour l'entrée suivante (taille max de 3 et 6 étapes)

```
3 6
push 3
push 5
push 8
pop
pop
pop
```

on devrait afficher

```
8
5
3
```

car on effectue les étapes suivantes :

0. push 3
1. push 5
2. push 8
3. pop -> 8
4. pop -> 5
5. pop -> 3

Exemple 2 Pour l'entrée suivante (taille max de 3 et 12 étapes)

```
3 12
push 100
push 13
push -5
push 42
pop
pop
push 85
pop
pop
pop
push 100
pop
```

on devrait afficher

```
Étape 3: pile remplie
-5
13
85
100
Étape 9: pile vide
100
```

car on effectue les étapes suivantes :

0. push 100
1. push 13
2. push −5
3. push 42 - échec, la pile a taille max de 3 et elle est déjà remplie
4. pop -> −5
5. pop -> 13
6. push 85
7. pop -> 85
8. pop -> 100
9. pop -> ? - échec, la pile est vide !
10. push 100
11. pop -> 100

4. « The floor is lava »

Alice joue à un jeu vidéo depuis des années. Dans ce jeu, elle contrôle un adorable lapin qui s'appelle Bix. Alice veut nous prouver à quel point elle est forte. Elle arrive à un endroit où Bix doit traverser un champ dans une zone volcanique. Sans même regarder l'écran, Alice écrit sur un bout de papier la séquence de touches qu'elle veut utiliser afin de faire traverser le champ à Bix. A-t-elle une aussi bonne mémoire qu'elle le croit ?

Bix se trouve sur un rocher et peut sauter sur les rochers voisins (gauche, droite, haut, bas). Tant que Bix reste sur les rochers, il est en sécurité. Par contre, s'il fait un faux pas et qu'il tombe dans la lave, le jeu est terminé. On nous donne une matrice binaire de taille $M \times N$ qui représente le plan du champ. Les rochers sont représentés par des `1` et la lave par des `0`. On lit d'abord M et N et ensuite les M lignes de N valeurs binaires.

Bix se trouve à la position $(0, 0)$. Il doit arriver à la sortie qui se trouve à la position $(M-1, N-1)$. On lit une chaîne de caractères (sans espace vide) représentant la suite d'instructions qu'Alice veut utiliser. Les caractères dans cette chaîne peuvent être `'h'` (pour « haut »), `'b'` (pour « bas »), `'g'` (pour « gauche »), ou `'d'` (pour « droite »).

Si Bix essaye de sortir de l'écran (quand il est au bord), il ne bouge pas et on passe à l'instruction suivante.

Si la suite d'instructions est juste et mène Bix jusqu'à la sortie sans quitter les rochers, alors affichez « Bravo Bix ! ».

Si la suite d'instructions ne mène pas Bix à la sortie, mais il ne tombe quand même pas dans la lave, alors affichez « Bix se trouve à (L, C) », ou L et C représentent la position finale de Bix. Il se peut qu'il passe par la case de sortie, mais si la suite des instructions est trop longue, il risque de retourner dans le champ.

Enfin, si Bix tombe dans la lave pendant son déplacement, affichez le texte « Game over X. Bix est tombé à (L, C) », ou X est l'indice de l'instruction qui a tué Bix, et L et C représentent l'endroit où Bix est tombé dans la lave.

Exemple : si la matrice est

```
5 6
1 0 0 0 0 0
1 0 0 1 0 1
1 0 1 1 0 1
1 1 1 0 1 0
0 0 1 1 1 1
```

alors on a les cas de figures suivants :

Chaîne des mouvements	Affichage final
`hbbbdgddhdh`	`Bix se trouve à (1, 3)`
`bbbddbdddghh`	`Game over 11. Bix est tombé à (2, 4)`
`ddddd`	`Game over 0. Bix est tombé à (0, 1)`
`bbbddbddd`	`Bravo Bix !`

Vous pouvez structurer votre solution presque comme vous l'entendez. Une seule contrainte : vous devez avoir une fonction `execute_one_move` qui prend en paramètres :

- la grille ;
- la position de Bix ;

- et un caractère (`char`) de déplacement.

Cette fonction doit exécuter le déplacement donné si possible (si on ne sort pas du champ) en modifiant la position de Bix. Elle peut gérer la chute dans la lave ou pas, au choix, et retourner la valeur qui vous arrange.