CS-119(a) – ICC-C Série 6

2025-03-25

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

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

Votre code ne saura pas qu'il lit depuis un fichier. Utilisez tout simplement `scanf` sans intercaler des messages pour l'utilisateur avec `printf`. Affichez seulement le résultat à la fin du programme.

Exo1 Plus

Qu'affiche ce code ? Décrivez ce qui se passe à chaque étape de l'exécution.

```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);
}
```

Solution de l'exercice 1

D'abord on appelle la fonction `plus10` ce qui affiche `15` et retourne `15`. La valeur de retour est passée en argument du deuxième appel de la fonction `plus10` qui affiche `25` et retourne `25`. Comme vu en cours, vu que le paramètre `a` est une variable locale de la fonction `plus10`, la valeur de `v` ne change pas, et donc reste égale à `5`.

```
v = 5
plus 10 = 15
plus 10 = 25
v = 5
```

Exo2 Plus*

Qu'affiche ce code ? Décrivez ce qui se passe à chaque étape de l'exécution.

```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);
}
```

Solution de l'exercice 2

Le premier appel à `ptr_plus10` prend en argument l'adresse de `v` et modifie le contenu de cet emplacement de mémoire. Donc on affiche `15` et la variable `v` vaut aussi `15`. La fonction retourne l'adresse de `v` qui est à son tour passée en argument du deuxième appel de la fonction `ptr_plus10`. Ce deuxième appel met à jour la variable `v` par le même mécanisme et affiche `25`. A la fin, la variable `v` vaut aussi `25`.

```
v = 5
plus 10 = 15
plus 10 = 25
v = 25
```

Exo3 Stack

Dans cet exercice nous allons implémenter une pile d'entiers.

Nous utiliserons un tableau pour stocker les éléments de la pile. Définissez donc une variable globale `int pile[100]`. Définissez également une variable globale `int taille` qui représente la taille de la pile, ainsi qu'une autre variable globale `int taille_max` qui représente la taille maximale qu'on autorise à la pile. Initialisez la variable `taille` à `0`. Ceci signifie que la pile est vide.

Implémentez une fonction

```
int push(int objet);
```

qui empile un nouvel objet. Dans notre implémentation nous considérons que le bas de la pile correspond à l'indice `0` du tableau `pile` ("à gauche"). Quand la pile n'est pas vide, et donc la valeur de `taille` est strictement supérieure à zéro, le haut de la pile se trouve à l'indice `taille - 1` ("à droite"). On peut rajouter un entier sur la pile si `taille < taille_max`. Dans ce cas, nous allons stocker l'entier à l'indice `taille` du tableau `pile` et ensuite nous allons incrémenter la variable `taille`. Cette fonction retourne `1` (vrai) si l'objet a bien été stocké sur la pile, ou `0` (faux) si la pile est remplie, donc si `taille == taille_max`.

Nous aurons aussi besoin d'une fonction

```
int pop(int *p_objet);
```

qui enlève un objet du haut de la pile et le stocke à l'emplacement de mémoire où pointe le paramètre `p_objet`. Quand `taille > 0`, l'entier du haut de la pile est `pile[taille-1]`. Cette fonction retourne `1` (vrai) quand l'opération réussit, et dans ce cas décrémente la variable `taille`, ou alors retourne `0` quand la pile est vide, et dans ce cas elle ne touche pas à `p_objet`.

Pour tester votre code, dans la fonction `main` lisez un entier représentant la taille maximale de la pile `taille_max` (valeur inférieure à 100), suivi d'un deuxième entier `K`. S'ensuivent `K` lignes contenant soit

— la chaine `pop`, soit

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

Attention ! Pour vérifier qu'une chaîne est égale à une autre il faut vérifier l'égalité **élément par élément**. Ecrivez donc une fonction

```
int equalstr(const char s1[], const char s2[], int max_length)
```

qui prend deux chaines et une taille maximale et qui vérifie que les éléments des deux chaînes sont égaux deux par deux en utilisant une boucle.

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

```
Etape 3: pile remplie
-5
13
85
100
Etape 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`

Solution de l'exercice 3

```c
#include <stdio.h>

int pile[100];
int taille, taille_max;

int push(int objet)
{
    if (taille < taille_max)
    {
        // Empiler
        pile[taille++] = objet;
        return 1;
    }
    // Pas possible, la pile est remplie
```

```
    return 0;
}

int equalstr(char *s1, char *s2, int max_length)
{
    for (int i = 0; i < max_length; i++)
    {
        if (s1[i] != s2[i])
        {
            return 0;
        }
        if (s1[i] == '\0')
        {
            return 1;
        }
    }
    return 1;
}

int pop(int *p_objet)
{
    if (taille > 0)
    {
        // Enlever du haut de la pile
        *p_objet = pile[--taille];
        return 1;
    }
    // Pas possible, la pile est vide
    return 0;
}

int main()
{
    int n_operations;

    scanf("%d", &taille_max);
    scanf("%d", &n_operations);

    for (int i = 0; i < n_operations; i++)
    {
        char operation[5];
        scanf("%s", operation);
        if (equalstr(operation, "push", 5))
```

```
        {
            int valeur;
            scanf("%d", &valeur);

            // Push valeur
            if (!push(valeur))
            {
                printf("Etape %d: pile remplie\n", i);
            }
        }
        else if (equalstr(operation, "pop", 4))
        {
            // Pop
            int resultat;
            if (!pop(&resultat))
            {
                printf("Etape %d: pile vide\n", i);
            }
            else
            {
                printf("%d\n", resultat);
            }
        }
        else
        {
            printf("Operation inconnue %s\n", operation);
        }
    }
}
```

Exo4 The floor is lava

Bob joue à un jeu vidéo depuis des années. Dans ce jeu il contrôle un adorable lapin qui s'appelle Bix. Bob veut nous prouver à quel point il est fort. Il arrive à un endroit où Bix doit traverser un champ dans une zone volcanique. Sans même regarder l'écran, Bob écrit sur un bout de papier la séquence de touches qu'il veut utiliser afin de faire traverser Bix le champ. A-t-il une aussi bonne mémoire qu'il 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 que Bob 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 n'y a rien qui se passe et on passe à l'instruction suivante.

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

Si la suite de 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. C'est possible 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
```

pour la chaîne `hbbbdgddhdh` on devrait afficher

```
Bix se trouve à (1, 3)
```

pour la chaîne `bbbddbdddghh` on devrait afficher

```
Game over 11. Bix est tombé à (2, 4)
```

pour la chaîne `ddddd` on devrait afficher

```
Game over 0. Bix est tombé à (0, 1)
```

et pour la chaîne `bbbddbddd` on devrait afficher

```
Bravo Bix!
```

Solution de l'exercice 4

```c
#include <stdio.h>
#include <string.h>

int M, N, map[100][100];

void haut(int *l)
{
    if (*l > 0)
    {
        (*l)--;
    }
}

void bas(int *l)
{
    if (*l < M - 1)
    {
        (*l)++;
    }
}

void gauche(int *c)
{
    if (*c > 0)
    {
        (*c)--;
    }
}

void droite(int *c)
{
    if (*c < N - 1)
    {
        (*c)++;
    }
}
int bouge(int *l, int *c, char touche)
{
    if (touche == 'h')
    {
        haut(l);
    }
```

```
    else if (touche == 'b')
    {
        bas(l);
    }
    else if (touche == 'g')
    {
        gauche(c);
    }
    else
    {
        droite(c);
    }

    return map[*l][*c];
}

int main()
{
    scanf("%d %d", &M, &N);
    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
            scanf("%d", &map[i][j]);

    char touche[100];

    int bix_ligne = 0, bix_col = 0;

    scanf("%s", touche);
    int K = strlen(touche);

    for (int i = 0; i < K; i++)
    {
        if (!bouge(&bix_ligne, &bix_col, touche[i]))
        {
            printf("Game over %d. "
                   "Bix est tombé à (%d, %d)\n",
                   i, bix_ligne, bix_col);
            return 0;
        }
    }
    if (bix_ligne == M - 1 && bix_col == N - 1)
    {
        printf("Bravo Bix!\n");
```

```
    }
    else
    {
        printf("Bix se trouve à (%d, %d)\n",
               bix_ligne, bix_col);
    }
}
```