Programmation Impérative – TP 3
Structures de données et allocation dynamique

Guillaume Mercier
mercier@enseirb-matmeca.fr
Augusta Mukam
augusta.mukam@u-bordeaux.fr
Philippe Swartvagher
philippe.swartvagher@enseirb-matmeca.fr

2023 – 2024

Pour les exercices demandant d'écrire une fonction, on écrira également une fonction `main()`, pour s'assurer que le code écrit compile et que la fonction a le comportement attendu.

1 Dernier retour sur la division euclidienne

Reprendre l'exercice sur la division euclidienne du TP précédent.

Écrire une fonction `division` qui à partir de deux entiers a et b renvoie une structure qui contient le quotient et le reste de la division euclidienne de a par b.

Écrire une seconde version de cette fonction qui prend comme troisième paramètre un pointeur sur la structure et remplit les champs de cette structure. On pourra en profiter pour renvoyer une valeur indiquant le succès ou une erreur dans la fonction. Quelle erreur peut-il arriver ?

2 Tri d'entiers

Écrire un programme qui prend en paramètres n'importe quel nombre d'entiers et affiche l'ensemble des nombres dans l'ordre croissant.

On convertira les nombres dans un tableau de `int`, puis on utilisera la fonction `qsort()` pour trier ce tableau.

Exemple d'exécution :

```
./tri 2 1 4 3 0
0 1 2 3 4
./tri 3 2 10
```

```
2 3 10
```

Bonus Proposer une solution sans convertir les nombres vers un tableau de `int`, qui manipule directement le tableau de paramètres `argv` de la fonction `main()`.

3 Miroir d'un texte

Écrire une fonction `miroir()` qui prend en paramètres deux chaînes de caractères et écrit dans la deuxième chaîne de caractères le miroir de la première.

On testera cette fonction en écrivant un programme qui affiche le miroir de la chaîne de caractères passée en paramètre. Cette chaîne de caractères peut avoir n'importe quelle taille.

Exemple d'exécution :

```
./miroir "1A Telecom"
moceleT A1
```

4 Représentation de dates

On va créer une structure qui permet de stocker une date et implémenter un ensemble de fonctions qui permet de manipuler ces dates. Une fois toutes les questions lues, on commencera par réfléchir aux membres que doit posséder la structure.

Question 1 Écrire une fonction
`struct date_s date_from_string(const char* s)`
qui prend en argument une date sous forme de chaîne de caractères `dd/mm/yyyy` et qui retourne une structure `struct date_s`. On supposera que la date passée en paramètre est valide et correctement formatée.

On fera une version sans `sscanf`, puis une version avec.

Question 2 Écrire une fonction
`void date_to_string(const struct date_s* date, char sep, char* str)`
qui écrit dans la chaîne de caractères `str` la date `date` suivant le format `ddcmmcyyyy` où `c` correspond au séparateur `sep`.

- Avec un tel prototype, qui s'occupe d'allouer la mémoire pour `str` : la fonction ou bien le code qui appelle la fonction ?
- Quelle taille doit être allouée pour `str` ?

La fonction `sprintf` sera sans doute utile...

Question 3 Écrire une fonction
`int date_compare(const struct date_s* const d1, const struct date_s* const d2)`
qui retourne un nombre négatif si `d1` est plus petite que `d2`, 0 si les deux dates sont égales ou un nombre positif si `d1` est plus grande que `d2`.

On fera une version qui utilise des conditions et une version qui n'en utilise pas.

Question 4 Écrire une fonction
`int date_get_value(const struct date_s* const date, ??????? value)`
qui permet de récupérer la valeur du jour, du mois ou de l'année, selon la valeur du paramètre `value`.

- Quel sera le type du paramètre `value` ?

Question 5 Écrire une fonction
`char* month_to_str(const int month)`
qui renvoie un pointeur vers une chaîne de caractères contenant le nom du mois dont le numéro est passé en paramètre (par exemple : `1` → `"January"`).

On écrira cette fonction sans utiliser de boucle ou de condition (sauf pour tester si le paramètre est valide).

Question 6 (bonus) Reprendre l'exercice du TP 1 pour tester si une année est bissextile et l'adapter pour que la fonction
`int date_is_bissextile(const struct date_s* const date)`
renvoie `1` si `date` appartient à une année bissextile, `0` sinon.

5 Représentation et manipulation de matrices

On va créer un ensemble de fonctions pour manipuler des matrices. Les éléments composant la matrice seront de type `float`, stockés dans un tableau de tableaux de flottants. Le tableau parent contient les pointeurs vers les tableaux représentant les différentes lignes.

Question 1 Comment allouer et libérer un tableau de tableaux ? Comment accéder à l'élément de coordonnées (i, j) ($i^{\text{ème}}$ ligne, $j^{\text{ème}}$ colonne) d'un tel tableau ?

Question 2 Écrire une fonction
`struct matrix_s* matrix_create(const int m, const int n)`
qui prend en argument les dimensions de la matrice (m lignes et n colonnes), alloue les tableaux nécessaires pour stocker les valeurs de la matrice et retourne un pointeur vers une structure représentant la matrice. Les valeurs de la matrice ne seront pas initialisées.

- Quels attributs vont composer la structure `struct matrix_s` ?

- La fonction renvoie un pointeur : que faut-il faire dans la fonction pour que l'adresse renvoyée pointe vers une zone mémoire toujours valide une fois sorti de la fonction ?
- Comment gérer le cas où la fonction échoue (par exemple : taille incorrecte, erreur lors de l'allocation de mémoire, ...) ?

Question 3 Écrire une fonction
`void matrix_destroy(struct matrix_s* const matrix)`
qui libère les ressources allouées lors de la création de la matrice.

Question 4 Écrire une fonction
`int matrix_set_value(const struct matrix_s* const matrix, const int i, const int j, const float value)`
qui associe à l'élément (i, j) de la matrice la valeur `value`. Que va retourner cette fonction ?

Question 5 Écrire une fonction
`int matrix_get_value(struct matrix_s* matrix, const int i, const int j, float * value)`
qui place à l'adresse pointée par `value` la valeur de l'élément (i, j) de la matrice.

- Que va retourner cette fonction ? Pourquoi ne retourne-t-elle pas la valeur de l'élément (i, j) ?
- Pourquoi définit-on les fonctions `matrix_set_value` et `matrix_get_value` plutôt que directement accéder au tableau stockant les valeurs de la matrice ?

Question 6 Écrire une fonction
`void matrix_print(const struct matrix_s* const matrix)`
qui affiche les valeurs de la matrice.

Question 7 Écrire une fonction

```
struct matrix_s* matrix_operation2(
  const struct matrix_s* const matrix_a,
  const struct matrix_s* const matrix_b,
  const ?????? operation
)
```

qui applique une opération entre deux matrices passées en paramètres et renvoie une matrice contenant le résultat de l'opération. On implémentera l'addition ($C_{i,j} = A_{i,j} + B_{i,j}$) et la multiplication ($C_{i,j} = \sum_{k=0}^{n} A_{i,k} B_{k,j}$ avec A, B et C de dimensions respectivement $m \times n$, $n \times p$ et $m \times p$).

- Comment gérer le cas où la fonction échoue ?
- Quel sera le type du paramètre `operation` ?

Créer des matrices, les remplir de valeurs et s'assurer que le résultat des opérations est correct.

Question 8 Il est possible de stocker toutes les valeurs que contient la matrice dans un unique tableau de taille $m \times n$. Dans ce cas, comment accéder à l'élément de coordonnées (i, j) ? Modifier l'interface de manipulation de matrices pour stocker les valeurs de cette façon.

Question 9 (bonus) Écrire les fonctions suivantes :

- Renvoie une copie de matrice :
  `struct matrix_s* matrix_copy(const struct matrix_s* const m)`
- Renvoie une copie de la matrice avec tous les éléments multipliés par un facteur :
  `struct matrix_s* matrix_scale(const struct matrix_s* const m, const float scalar)`
- Renvoie la transposée de la matrice :
  `struct matrix_s* matrix_transpose(const struct matrix_s* const m)`
- Renvoie une matrice remplie de zéros :
  `struct matrix_s* matrix_zeros(const int n, const int m)`
- Renvoie la matrice identité de dimensions $n \times n$:
  `struct matrix_s* matrix_identity(const int n)`

6 Bonus : grand morpion

Reprendre l'exercice bonus du TP précédent pour faire un morpion et ajouter un paramètre au programme qui correspond à la taille du plateau.

Exemple de début d'exécution :

```
./morpion 5
  1 2 3 4 5
1 . . . . .
2 . . . . .
3 . . . . .
4 . . . . .
5 . . . . .
> Joueur 1 : numéro de ligne ?
```