Algorithmique en C

Franck Pommereau

franck.pommereau@univ-evry.fr
Licence 1, Université d'Évry Val d'Essonne

Résumé Les exercices signalés par le symbole ☑ sont des questions de cours destinées à vérifier que les *points essentiels* en sont connus. Les autres exercices sont signalés avec des étoiles indiquant leur difficulté : ⋆ indique un exercice facile qui devrait être résolu directement, ⋆⋆ indique un exercice qui nécessite une petite réflexion préalable et ⋆⋆⋆ indique un exercice long ou qui demande une vraie réflexion. Enfin, ◷ indique un exercice optionnel pour ceux et celles à qui il reste du temps à occuper (ou à faire chez soi).

☑ Exercice 1

1. Reprenez le programme `hello.c` vu en cours, compilez-le, et executez-le.
2. Que se passe-t-il si vous oubliez la ligne commençant par `#include` ?
3. Pourquoi ?
4. Reprenez le programme `hello-name.c` vu en cours, compilez-le, et exécutez-le. La bibliothèque `readline` n'étant pas disponible sur les machines de TP, remplacez :
   ```
   name = readline("What's your name? ");
   ```
   par :
   ```
   printf("What's your name? ");
   scanf("%ms", &name);
   ```

⋆ Exercice 2

Reprenez le programme `hello.c` et modifiez-le pour qu'il affiche plusieurs messages.

⋆ Exercice 3

Écrivez un programme qui lit au clavier un nom, un prénom, et un âge puis affiche le message "`Bonjour PRÉNOM NOM, vous avez ÂGE ans.`" où les mots en majuscules sont remplacés par les valeurs lues au clavier.

⋆ Exercice 4

Écrivez un programme qui lit un `int` au clavier puis le copie dans une variable de type `unsigned int`. Cette affectation est-elle légale ? Le compilateur proteste-t-il ? Que se passe-t-il si l'entier lu au clavier est négatif ? Affichez la valeur du `unsigned int` après son affectation. (Utilisez `%u` dans `printf` pour l'afficher.) Que pouvez-vous dire de la valeur affichée.

⋆ Exercice 5

Écrivez un programme lisant deux entiers a et b au clavier puis affichant les valeurs de $a + b$, $a - b$, $a * b$, a/b. Que se passe-t-il si vous saisissez $b = 0$.

⋆ Exercice 6

Écrivez un programme lisant deux entiers t et p au clavier représentant respectivement un nombre total t d'étudiants inscrits, et un nombre p d'étudiants présents. Faites afficher le pourcentage d'étudiants présents par votre programme.

⋆ **Exercice 7**

1. Écrivez une fonction prenant deux entiers en paramètres et renvoyant le plus petit des deux.
2. Ajoutez un programme principale qui lit deux entiers au clavier et affiche le plus petit des deux en faisant appel à cette fonction.

⋆⋆ **Exercice 8**

Les nombres de Fibonacci forment une suite $(F_n)_n$ telle que : $F_0 = 0$, $F_1 = 1$ et $F_n = F_{n-1} + F_{n-2}$.

1. Écrivez une fonction récursive pour calculer le $n^{\text{ème}}$ nombre de Fibonacci.
2. Lancez cette fonction pour de petites, puis de grandes valeurs de n.
3. Que constatez-vous ?
4. Expliquez ce constat en listant tous les appels à votre fonction.

⋆⋆ **Exercice 9**

On a vu en cours une implantation récursive de la fonction factorielle, basée sur la définition : $\mathsf{fact}(n) = 1$ si $n = 0$, $n \times \mathsf{fact}(n-1)$ sinon. Cependant :

— on peut remplacer la partie récursive de cette définition par $\mathsf{fact}(n) = \prod_{1 \leq i \leq n} i$;
— on sait que la multiplication est associative et donc : $1{\times}2{\times}3{\times}\cdots = ((1{\times}2){\times}3){\times}\cdots$, c'est-à-dire qu'on peut réaliser les multiplications une à une.

Exploitez ces deux propriétés pour écrire une version itérative du calcul de la factorielle. (Cette fonction a été vue en cours, vous devez la reconstruire et non de la recopier.)

⋆⋆⋆ **Exercice 10**

On a vu à l'exercice 8 une façon simple mais peu efficace d'implanter la suite de Fibonacci. On se propose de l'améliorer.

1. Sur une feuille de papier, calculez dans l'ordre tous les six premiers termes de la suite de Fibonacci : F_0, F_1, F_2, F_3, F_4, F_5, et enfin F_6.
2. Quels sont les valeurs que vous avez utilisé pour calculer F_6 ?
3. Exploitez ce constat pour proposer une implantation itérative de cette suite.
4. Testez-la pour de grandes valeurs de n. Faites-vous le même constat que pour la version récursive ? Pourquoi ?

Remarque : ceci n'est pas une condamnation de la récursivité, ce n'est tout simplement pas le même algorithme qui est utilisé dans les deux cas. D'ailleurs, le meilleur algorithme pour calculer la suite de Fibonacci est récursif et plus efficace que celui qu'on vient d'implanter.

⋆⋆⋆ **Exercice 11**

En vous inspirant des deux exercices précédents, proposez une implantation itérative du calcul de PGCD avec l'algorithme d'Euclide présenté en cours. (Cette fonction a été vue en cours, vous devez la reconstruire et non de la recopier.)

⋆⋆ **Exercice 12**

1. Écrivez une fonction qui teste si un nombre est premier en lui cherchant un diviseur.
2. Utilisez cette fonction pour afficher les 100 premiers nombres premiers.

⋆⋆ **Exercice 13**

1. Écrivez une fonction prenant un paramètre un entier naturel n et dessinant un carré d'étoiles de côté n.

2. Même chose en supprimant la première diagonale.
3. Même chose en supprimant la seconde diagonale.
4. Même chose en supprimant la moitié supérieur droite.
5. Même chose en supprimant la moitié supérieur gauche.

Exemples, pour $n = 4$, de gauche à droite :

```
* * * *          * * *         * * *          *                    *
* * * *        *   * *         * *   *        * *                * *
* * * *        * *   *         *   * *        * * *            * * *
* * * *        * * *             * * *        * * * *        * * * *
```

Exercice 14

Dessinez les mêmes motifs qu'à l'exercice précédent, mais présentés sur une seule ligne comme dans l'illustration ci-dessus.

⋆⋆ **Exercice 15**

L'algorithme du crible d'Érathosthène permet de déterminer les nombres premiers inférieurs à un entier N. Il est assez simple : on écrit dans un tableau (le crible) tous les entiers entre 2 et N. On répète ensuite les opérations suivantes :

— prendre le premier entier du crible, il est premier ;
— l'effacer ainsi que ses multiples.

On s'arrête quand le crible est vide, on a alors récupéré tous les nombres premiers qu'il contenait. Proposez un type pour représenter un crible de taille quelconque en permettant facilement l'effacement d'un entier. Écrivez un programme lisant un entier N et affichant tous les nombres premiers inférieurs ou égaux à N au moyen de l'algorithme du crible d'Érathosthène. Que pensez-vous de son efficacité par rapport à une boucle exploitant la fonction de l'exercice 12 ?

⋆⋆ **Exercice 16**

On rappelle qu'une chaîne de caractères en C de type `char*` et qu'un caractère nul (`'\0'`) en marque la fin. Écrivez les fonctions suivantes.

1. Une fonction qui prend une chaîne en paramètre et renvoie sa longueur. (Nombre de caractères utiles, c'est-à-dire sans compte le `'\0'` final.)
2. Une fonction qui prend une chaîne en paramètre et renvoie son miroir.
3. Une fonction qui vérifie si une chaîne est une palindrome.
4. Une fonction qui vérifie si deux chaînes sont miroir l'une de l'autre.

Exercice 17

Mêmes questions que dans l'exercice précédent mais en n'utilisant que des accès par pointeurs à la place des accès de type tableau (avec la notation `s[i]`).

⋆⋆ **Exercice 18**

1. Écrivez une fonction faisant l'addition en binaire de deux chaînes de caractères `'0'` ou `'1'` représentant des entiers naturels, dont le bit de poids faible sera positionné à l'indice 0.
2. Écrivez une fonction qui affiche un nombre binaire ainsi codé, mais en présentant le bit de poids faible à gauche comme c'est l'habitude.
3. Écrivez une fonction calculant la valeur d'un entier ainsi représenté en binaire.
4. Écrivez une fonction calculant la représentation binaire d'un entier naturel.

◷ Exercice 19

Mêmes questions mais en écrivant le bit de poids fort au début de la chaîne. (L'affichage d'un nombre en binaire devient un simple `printf`, mais le reste se complique.)

⋆⋆⋆ Exercice 20

On se donne le type `Tab` défini en cours par :

dicho.c

```
3  typedef struct {
4    unsigned int len;
5    int* val;
6  } Tab;
```

1. Écrivez une fonction qui renverse l'ordre des éléments d'un tel tableau (en place, c'est-à-dire sans faire une copie du tableau contrairement au miroir d'une chaîne vu plus tôt).
2. Écrivez une fonction qui renvoie la plus petite valeur d'un tableau.
3. Écrivez une fonction qui renvoie la deuxième plus petite valeur d'un tableau, en ne faisant qu'un parcours de ce tableau.
4. Écrivez une fonction qui teste si aucun élément d'un tableau n'y est dupliqué. (Autrement dit, la fonction renvoie `1` si chaque élément du tableau n'y est présent qu'une seule fois, `0` sinon.)
5. Écrivez une fonction qui prend un tableau en argument et remplace chaque valeur par son nombre d'occurrences.
6. Écrivez une fonction qui prend un tableau en argument et renvoie un nouveau tableau tel que :
   — toute valeur dans le tableau de départ est présente dans le tableau résultat ;
   — le tableau résultat ne contient aucun élément dupliqué.

◷ Exercice 21 (Tri à bulles)

Le principe du tri à bulles est de faire "remonter" dans le tableau les valeurs les plus grandes, comme des bulles. Un parcours traverse le tableau en échangeant les valeurs successives qui n'ont pas le bon ordre. À la suite de ce parcours, la plus grande valeur se trouve à sa place. En recommençant le processus sur la partie non triée du tableau, on va remettre successivement toutes les valeurs à leur place. Il est possible de vérifier au cours d'un passage qu'on ne fait aucun échange et donc que le tableau est déjà trié, ce qui permet d'arrêter le tri. Implantez le tri à bulles sur un tableau de type `Tab` comme à l'exercice 20.

⋆⋆⋆ Exercice 22

(Tri par insertion) On a vu en cours le principe du tri par insertion : on sélectionne la première valeur de la partie non triée du tableau (tout le tableau sauf sa première case au départ), et on l'insère à se place dans la partie triée (une seule case au départ, mais qui grandit d'une case à chaque insertion). On va le réaliser en plusieurs étapes, sur des tableaux de type `Tab` comme présenté à l'exercice 20.

1. Reprenez la fonction `slice` vue en cours qui renvoie une "tranche" de tableau, écrivez une fonction qui recopie une tranche sur une autre de même taille.
2. Écrivez une fonction qui recherche par dichotomie la place d'un entier dans un tableau trié.

3. Écrivez finalement le tri par sélection en insérant un à un les éléments non triés d'un tableau. L'insertion utilise la recopie de tranches pour décaler les éléments qui suivent la position d'insertion.
4. Pensez-vous que l'insertion dans un tableau est une opération efficace ? Comparez au tri par sélection vu en cours.

⋆⋆ **Exercice 23 (Tri compteur)**
Le tri compteur (ou tri de comptage) est applicable si on trie des valeurs d'un type qui en comporte peu de valeurs distinctes. Par exemple, les caractères d'une chaîne : chacun est codé sur un seul octet et il n'y en a donc que 256 possibles. On va donc supposer qu'on trie des chaînes de caractères. L'algorithme se déroule en trois étapes :

1. On déclare un tableau indexé par les valeurs possibles pour un `char`, c'est-à-dire les entiers entre 0 et 255. On initialise toutes les cases à zéro.
2. On parcours la chaîne une première fois, et la case de chaque caractère rencontré est incrémentée.
3. Pour obtenir les valeurs dans l'ordre, il suffit de parcourir le tableau de compteurs et, pour chaque cellule d'index `c` et de valeur `n`, on doit placer `n` fois le caractère `c` dans la chaîne finale.

Implantez ce tri.

⋆⋆ **Exercice 24**
Écrivez une fonction qui teste si deux chaînes sont des anagrammes l'une de l'autre.