

## Programmation en C et structures de données

guillaume.revy@univ-perp.fr

---

CT, Mercredi 4/01/2023 de 14h à 16h

---

**Aucun document n'est autorisé.  
Les calculatrices, ordinateurs, et téléphones portables sont interdits.**

### Exercice 1. Importance de l'indentation, malgré tout

0.75 + 1.25 = 2 pts

Contrairement à certains autres langages, l'indentation d'un programme C n'a pas d'impact sur le résultat de son exécution. Mais elle a une importance pour sa lisibilité.

- 1. Que fait le programme ci-dessous ?

```
int main(void) {float ct, cc, note;scanf("%f", &ct);scanf("%f", &cc);note=.5f*cc+.5f*ct;printf("Resultat=%f\n", note);if(note<10){printf("... non validee\n");}else{printf("... validee\n");}return 0;}
```

- 2. Écrire un programme qui lit une valeur entière au clavier, puis qui indique si elle est positive, négative, ou nulle (en distinguant les 3 cas).

### Exercice 2. Trois boucles simples

1 + 1 + 0.5 = 2.5 pts

- 1. Écrire un programme qui lit une valeur entière  $n \geq 1$ , puis qui calcule de manière itérative et affiche la somme des  $n$  premiers entiers strictement positifs. (Par exemple, pour  $n = 3$ , la valeur affichée sera 6.)
- 2. Écrire un programme qui lit une valeur entière  $n \geq 1$ , puis qui affiche le motif donné ci-dessous pour la valeur  $n = 5$ , en utilisant une (des) boucle(s) **for**.

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

- 3. Modifier la (les) boucle(s) pour utiliser une (des) boucle(s) **while**.

### Exercice 3. Fréquence d'apparition dans un tableau d'entiers

1 + 1 + 1 = 3 pts

Soient  $T$  un tableau de taille  $n$  d'entiers, ne contenant que des chiffres (càd, des entiers dans l'intervalle  $[0, 9]$ ), et  $F$  un tableau d'entiers représentant la fréquence d'apparition de chaque chiffre dans  $T$ .

- 1. Quelle doit être la taille de  $F$ ? Pourquoi.
- 2. Écrire une fonction qui prend en paramètre  $T$  et  $F$ , puis qui construit les valeurs de  $F$ .
- 3. Illustrer l'utilisation de cette fonction dans un programme principal.

### Exercice 4. Une structure date

1 + 1 + 1 + 1 = 4 pts

Dans cette exercice, nous souhaitons manipuler et afficher une date, en utilisant une structure.

- 1. Écrire une structure **date** représentant une date sous la forme jour/mois/année, et où chaque champ est stocké sur un entier 16 bits.
- 2. Écrire une fonction **afficher\_date** qui affiche une date passée en paramètre.

Pour la création d'une date, votre voisin propose la fonction ci-dessous. D'après lui, le programme compile parfaitement sous `gcc-12`, mais à l'exécution il plante (*Segmentation fault : 11*).

```

struct date*
creation(unsigned short j, unsigned short m, unsigned short a)
{
    struct date D = {j, m, a};
    return &D;
}

```

- 3. Proposer une explication, en vous aidant d'un dessin.
- 4. Corriger la fonction, sans modifier son prototype.

### Exercice 5. Fonction d'allocation

$1 + 1 = 2 \text{ pts}$

Nous souhaitons ici écrire une fonction qui alloue un tableau de taille  $n$  de **float**. Votre voisin propose la fonction ci-dessous, qui compile parfaitement sous `gcc-12`. Mais il n'arrive pas à savoir si elle fonctionne correctement.

```

void
allouer_float(float *T, int n)
{
    T = (float *)malloc(n * sizeof(float));
    // ...
}

```

- 1. Illustrer sur un schéma le fonctionnement de cette fonction pour  $n = 4$ .
- 2. Un programme utilisant cette fonction s'exécute-t-il correctement? Si oui, justifier la réponse. Si non, expliquer pourquoi et proposer une correction.

### Exercice 6. Le compte est bon

$1 + 1 + 1 + 1 + 2 + 0.5 = 6.5 \text{ pts}$

Le *compte est bon* est un jeu télévisé où un ensemble  $P$  de 6 plaques différentes est choisi aléatoirement parmi les 24 plaques de départ suivantes :

$$\{1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10, 25, 50, 75, 100\},$$

chaque nombre pouvant donc apparaître dans  $P$  au plus 2 fois, à l'exception de 25, 50, 75, et 100 qui peuvent apparaître au plus 1 fois. Ensuite, un nombre  $N$  est tiré aléatoirement entre 100 et 999. Le joueur doit alors trouver une suite d'opérations arithmétiques permettant de calculer  $N$  à partir de ces 6 plaques et des 4 opérations arithmétiques de bases ( $+, -, \times, /$ ). Par exemple :

$$P = \{4, 3, 5, 8, 100, 9\} \quad \text{et} \quad N = 650 = (100 / (9 - (4 + 3))) \times (5 + 8).$$

Nous allons représenter la suite d'opérations arithmétiques par une liste chaînée, où chaque maillon contient trois entiers (2 opérandes et 1 résultat) et un caractère pour l'opération arithmétique proprement dite.

- 1. Définir la structure utilisée pour manipuler cette liste chaînée.
- 2. Écrire une fonction `insérer` qui, étant donnés une liste  $L$ , et 2 opérandes, 1 résultat et 1 caractère, insère une nouvelle opération arithmétique en fin de la liste chaînée  $L$ .
- 3. Écrire enfin une fonction `vider` qui supprime tous les éléments d'une liste chaînée  $L$ .

*Aucune de ces deux fonctions ne retourne explicitement la liste modifiée.*

Ensuite un algorithme pour résoudre ce problème est le suivant :

- On choisit 2 plaques  $p_1$  et  $p_2$  parmi les 6, que l'on supprime de  $P$ .
- On additionne  $p_1$  et  $p_2$ , et on ajoute le résultat à  $P$ : on obtient donc un nouvel ensemble  $P$  de 5 plaques.

On résout alors le nouveau problème de manière récursive sur ce nouvel ensemble  $P$  de 5 plaques.

Sur l'exemple précédent, avec  $p_1 = 4$  et  $p_2 = 3$ , on obtient :

$$P = \{7, 5, 8, 100, 9\} \quad \text{et} \quad N = 650.$$

- Si l'addition de  $p_1$  et  $p_2$  ne permet pas de résoudre le problème, on essaie de manière successive avec la soustraction, la multiplication et la division.

Si le problème ne peut pas être résolu, l'algorithme doit retourner l'entier le plus proche de  $N$  qu'il est possible de calculer et la suite d'opérations arithmétiques permettant de le calculer.

*Attention tout de même :*

- L'addition et la multiplication sont commutatives, mais pas la soustraction ni la division.
- Seule la division entière est ici prise en compte.
- Si le résultat d'une opération est nul, cette opération est ignorée.
- Enfin, multiplier ou diviser par 1 n'a aucun effet.

- 4. Écrire une fonction `generer_probleme` qui construit l'ensemble  $P$  et le nombre à trouver  $N$ .
- 5. Écrire une fonction `resoudre` qui, étant donnés  $P$  et  $N$ , construit la suite d'opérations arithmétiques permettant de calculer  $N$ , ou l'entier le plus proche de  $N$  qu'il est possible de calculer si  $N$  ne l'est pas.
- 6. Illustrer enfin leur utilisation dans un programme principal.

## Exercice 7. Matrice diagonale

$2 + 1 + 1 = 4 \text{ pts}$

En algèbre linéaire, une matrice diagonale  $M$  est une matrice carrée dont les coefficients en dehors de la diagonale principale sont nuls, les coefficients de cette diagonale pouvant être nuls ou non nuls. L'exemple ci-dessous donne une matrice diagonale de taille  $3 \times 3$  :

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 8 \end{pmatrix}.$$

*On représentera une matrice par un tableau 2D de taille  $n \times n$ .*

- 1. Écrire une fonction `allocation_2d` sans retour (avec un retour `void`) qui, étant donné un entier  $n$ , alloue l'espace mémoire nécessaire au stockage d'une matrice carrée de taille  $n \times n$  d'entiers.
- 2. Écrire une fonction `est_diagonale` qui, étant donnée une matrice  $M$ , vérifie et retourne un entier indiquant si la matrice est diagonale (retour = 1) ou non (retour = 0).
- 3. Écrire enfin une fonction `liberation_2d` qui, étant donnée une matrice  $M$ , libère la mémoire utilisée pour le stockage de  $M$ .