+1/1/60+

EPFL

Ens. : O. Lévêque, D. Tomozei
CS-119a : Information, Calcul, Communication - (n/a)
Lundi 24 juin 2024
Durée : 180 minutes

n/a

n/a

SCIPER : **999999** SALLE : **BLANK** Signature :

Attendez le début de l'épreuve avant de tourner la page. Ce document est imprimé recto-verso, il contient 16 pages, les dernières pouvant être vides. Ne pas dégrafer.

- Posez votre carte d'étudiant sur la table.
- Document autorisé pour cet examen : un formulaire constitué de deux pages A4 recto-verso, manuscrites (ou préparées avec stylet+tablette).
- L'utilisation de tout appareil électronique (calculatrice, ordinateur, smartphone/watch, tablette) est interdite pendant l'épreuve.
- L'examen est composé de deux parties:
  - une partie avec 16 questions à choix multiple, valant en tout 32 points ; chaque question admet une seule réponse correcte parmi 4 possibilités : la réponse correcte vaut 2 points ; toute autre option (pas de réponse, réponse fausse, ou plusieurs cases cochées) vaut 0 point.
  - une partie avec 6 questions de type ouvert, valant en tout 36 points.
- Merci d'avance de soigner la présentation de vos réponses !
- Si une question est erronée, les enseignants se réservent le droit de l'annuler.

Respectez les consignes suivantes | Read these guidelines | Beachten Sie bitte die unten stehenden Richtlinien

choisir une réponse | select an answer Antwort auswählen	ne PAS choisir une réponse | NOT select an answer NICHT Antwort auswählen	Corriger une réponse | Correct an answer Antwort korrigieren

ce qu'il ne faut **PAS** faire | what should **NOT** be done | was man **NICHT** tun sollte

Première partie, questions à choix multiple

Pour chaque question, marquer la case correspondant à la réponse correcte sans faire de ratures. Il n'y a qu'**une seule** réponse correcte par question.

Question 1 :
On considère l'algorithme suivant :

algo
entrée : *L liste de n lettres*
sortie : *valeur binaire (oui/non)*

```
s ⟵ oui
Pour i allant de 1 à ⌊n/2⌋
    Si L(i) ≠ L(n+1−i)
        s ⟵ non
Sortir: s
```

Pour quelle liste de lettres en entrée la sortie de l'algorithme ci-dessus est-elle un oui?

☐ CHACHACHA ☐ MALAYALAM ☐ HAHAAH ☐ ALABAMA

Question 2 :
On considère l'algorithme suivant :

algo
entrée : *n nombre entier positif*
sortie : *nombre entier positif*

```
s ⟵ 0
Pour i allant de 1 à n
    Pour j allant de 1 à i
        Pour k allant de j à i
            s ⟵ s+1
Sortir: s
```

Quelle est la sortie de **algo**(3) ?

☐ 9 ☐ 27 ☐ 10 ☐ 6

Question 3 :
Quelle est la complexité temporelle de l'algorithme de la question 2 ?

☐ $\Theta(n^4)$ ☐ $\Theta(n)$ ☐ $\Theta(n^3)$ ☐ $\Theta(n^2)$

Question 4 :
On considère l'algorithme suivant :

algo

entrée : A, B *deux nombres entiers positifs*
sortie : *nombre entier positif*

Si $B = 0$
  Sortir: 1
Si B *est pair*
  Sortir: $(\textbf{algo}(A, B/2))^2$
Sinon
  Sortir: $(\textbf{algo}(A, (B-1)/2))^2 \cdot A$

Quelle est la sortie de $\textbf{algo}(A, B)$?

☐ A^B ☐ B^A ☐ $A \cdot B$ ☐ A^2 si B est pair; A^3 si B est impair

Question 5 :
Si A, B sont chacun des nombres à n chiffres, combien d'instructions seront-elles lues par l'algorithme de la question 4 jusqu'à ce qu'il termine ? (ceci correspond à la complexité temporelle de l'algorithme, si on ne tient pas compte de la complexité des opérations en elles-mêmes)

☐ $\Theta(2^n)$ ☐ $\Theta(n^n)$ ☐ $\Theta(n)$ ☐ $\Theta(10^n)$

Question 6 :
On considère deux algorithmes $\textbf{algo1}(n)$ et $\textbf{algo2}(n)$ dont les complexités temporelles sont respectivement données par $\frac{n}{2}$ et $2n^2$ (on suppose ici que ces complexités temporelles sont *exactes*).

L'exécution de $\textbf{algo1}(1'000)$ sur une machine donnée (disons la machine A) termine en 1 minute.

L'exécution de $\textbf{algo2}(1'000)$ sur une autre machine (disons la machine B) termine également en 1 minute.

Si on exécute maintenant $\textbf{algo1}(10'000)$ sur la machine A et $\textbf{algo2}(10'000)$ sur la machine B, alors :

☐ $\textbf{algo1}(10'000)$ termine en 5 minutes et $\textbf{algo2}(10'000)$ termine en 200 minutes
☐ $\textbf{algo1}(10'000)$ termine en 10 minutes et $\textbf{algo2}(10'000)$ termine en 200 minutes
☐ $\textbf{algo1}(10'000)$ termine en 10 minutes et $\textbf{algo2}(10'000)$ termine en 100 minutes
☐ $\textbf{algo1}(10'000)$ termine en 5 minutes et $\textbf{algo2}(10'000)$ termine en 100 minutes

Question 7 :
Soient $x = 108$ et $y = 124$ deux nombres entiers positifs, chacun représenté en binaire avec la représentation des nombres entiers positifs sur 8 bits.

On effectue une opération XOR bit-à-bit sur ces deux nombres (c'est-à-dire qu'on effectue l'opération XOR sur le premier bit de x et le premier bit de y, puis la même opération sur le second bit de x et le second bit de y, etc., sans jamais considérer de retenue). On obtient ainsi une nouvelle séquence de 8 bits. Quel est le nombre entier positif z représenté par cette nouvelle séquence de 8 bits ?

☐ 222 ☐ 16 ☐ 8 ☐ 232

Question 8 :
On considère le problème suivant :

Etant donné une liste L de n nombres entiers relatifs, identifier le plus grand sous-ensemble $S \subset \{1, \ldots, n\}$ (i.e., le sous-ensemble avec le plus grand nombre d'éléments) tel que $\sum_{i \in S} L(i) \geq 0$.

Parmi les affirmations ci-dessous, laquelle est vraie?

☐ Ce problème fait partie de la classe P.
☐ Ce problème fait partie de la classe NP, mais on ne sait pas s'il fait partie de la classe P.
☐ Ce problème fait partie de la classe NP, donc il ne fait pas partie de la classe P.
☐ On ne sait pas si ce problème fait partie de la classe P, ni s'il fait partie de la classe NP.

Question 9 : Soient trois variables définies comme suit:

```
int u = 12, v[100];
float score;
```

Plus loin dans le même bloc il y a un appel de fonction:

```
score = calculer(u, v);
```

Quelles sont toutes les déclarations possibles, i.e., qui ne déclenchent pas d'erreur de compilation, pour la fonction `calculer` parmi les déclarations suivantes?

(a) `int calculer(long a, int* b);`

(b) `float calculer(int a, int& b);`

(c) `double calculer(int a, int b[]);`

(d) `float calculer(long a, int** b);`

☐ (a) et (d) ☐ Uniquement (c) ☐ (a) et (c) ☐ (b) et (c)

Question 10 : Qu'affiche ce code?

```
int tab[3] = {1, 2, 3};
tab[++tab[0]]--;
printf("%d %d %d\n", tab[0], tab[1], tab[2]);
```

- ☐ `2 2 2`
- ☐ Il ne compile pas.
- ☐ `1 2 3`
- ☐ `2 1 3`

Question 11 : Qu'affiche le code suivant?

```
int a[] = {1, 2, 3, 4, 5};
for (int* b = a + 4; b >= a; b--)
{
    printf("%d ", b[0]);
}
```

- ☐ Il ne compile pas.
- ☐ `5 5 5 5 5`
- ☐ `5 4 3 2 1`
- ☐ `9 8 7 6 5`

Question 12 : Qu'affiche ce code?

```
#define T 3
void f(const char *txt, char *out, int b, int u, int t)
{
    out[u] = txt[b];
    if (txt[b] == '\0')
    {
        return;
    }
    if (txt[b] == 'u' && t > 0)
    {
        f(txt, out, b, u + 1, t - 1);
    }
    else
    {
        f(txt, out, b + 1, u + 1, T);
    }
}

int main()
{
    char string[10];
    f("but", string, 0, 0, T);
    printf("%s\n", string);
}
```

- ☐ `tub`
- ☐ `but`
- ☐ Ne termine pas.
- ☐ `buuuut`

On définit les structs suivantes:

```
typedef struct _point
{
    double x, y;
} point_t;
```

```
typedef struct _polygone
{
    int n;
    point_t *sommets;
} polygone_t;
```

Dans la fonction `main()` on définit les variables suivantes:

```
point_t carré[] =
{
  {0, 0}, {0, 5}, {5, 5}, {5, 0}
};

polygone_t p;
p.n = 4;
p.sommets = carré;
```

```
polygone_t *copie1 = &p;

polygone_t *copie2 = malloc(sizeof(polygone_t));
copie2->n = p.n;
copie2->sommets = p.sommets;

polygone_t *copie3 = malloc(sizeof(polygone_t));
copie3->n = p.n;
copie3->sommets = malloc(p.n * sizeof(point_t));
for (int i=0; i<p.n; i++)
{
  copie3->sommets[i] = p.sommets[i];
}
```

Question 13 : Le code suivant se trouve dans `main()` juste après les instructions ci-dessus. Qu'affiche-t-il?

```
p.n = 3;
printf("%d %d %d\n", copie1->n, copie2->n, copie3->n);
```

☐ 3 4 4 ☐ 4 4 4 ☐ 3 3 4 ☐ 3 3 3

Question 14 : La fonction **double** `perimetre(polygone_t *poly)` calcule le périmètre d'un polygone. Qu'affiche le code suivant? Il se trouve dans `main()` juste après le code de la question précédente.

```
p.n = 4;
for (int i = 0; i < 4; i++)
{
    carré[i].x *= 2;
    carré[i].y *= 2;
}
printf("%.1lf %.1lf %.1lf\n", perimetre(copie1), perimetre(copie2), perimetre(copie3));
```

☐ 40.0 40.0 20.0 ☐ 40.0 40.0 40.0 ☐ 20.0 20.0 20.0 ☐ 40.0 20.0 20.0

Question 15 : Voici quelques définitions de fonctions:

```
void swap1(int u, int v)
{
    int aux = u;
    u = v;
    v = aux;
}
```

```
void swap2(int *u, int *v)
{
    int *aux = u;
    u = v;
    v = aux;
}
```

```
void swap3(int *u, int *v)
{
    int aux = u[0];
    u[0] = v[0];
    v[0] = aux;
}
```

```
void swap4(int *u, int *v)
{
    int **aux = &u;
    &u = &v;
    &v = aux;
}
```

On définit les variables `int a = 1, b = 2;` Quel appel échange leur contenu?

- ☐ `swap3(&a, &b);`
- ☐ `swap1(a, b);`
- ☐ `swap2(&a, &b);`
- ☐ `swap4(&a, &b);`

Question 16 : Qu'affiche le code suivant?

```
char *tag = "My name is xxx", *bob = "Robert", result[100], *r = result;

for (char *p = tag, *q = bob; *p != '\0' && *q != '\0'; p++)
{
    if (*p == 'x')
    {
        *r = *q;
        q++;
    }
    else
    {
        *r = *p;
    }
    r++;
}
*r = '\0';

printf("%s\n", result);
```

- ☐ `My name is Robert`
- ☐ `My name is Bob`
- ☐ `My name is Rob2edmsaere2323`
- ☐ `My name is Rob`

Deuxième partie, questions de type ouvert

Répondre dans l'espace dédié. Votre réponse doit être soigneusement justifiée, toutes les étapes de votre raisonnement doivent figurer dans votre réponse. Laisser libres les cases à cocher : elles sont réservées aux correcteurs.

Question 17: *Cette question est notée sur 8 points*

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 *Réservé au correcteur*

Soient m, n deux nombres entiers positifs, A un tableau de $m \times n$ nombres entiers tels que

$$A(i,j) \leq A(i,\ell), \quad \text{pour tout} \quad 1 \leq i \leq m, \quad 1 \leq j < \ell \leq n$$

et

$$A(i,j) \geq A(k,j), \quad \text{pour tout} \quad 1 \leq i < k \leq m, \quad 1 \leq j \leq n$$

On considère l'algorithme suivant :

mystère

entrée : *A un tableau de $m \times n$ nombres entiers vérifiant les propriétés ci-dessus et x un nombre entier*
sortie : *valeur binaire (oui/non)*

```
i ⟵ 1
j ⟵ 1
Tant que i ≤ m et j ≤ n
    Si A(i,j) = x
        Sortir: oui
    Si A(i,j) < x
        j ⟵ j + 1
    Sinon
        i ⟵ i + 1
Sortir: non
```

a) (1 point) Quelle est la sortie de l'algorithme **mystère** si les données d'entrée sont les suivantes ?

$$m = 3, \quad n = 4, \quad A = \begin{pmatrix} 13 & 32 & 40 & 100 \\ 12 & 17 & 39 & 65 \\ 5 & 14 & 31 & 38 \end{pmatrix} \quad \text{et} \quad x = 31$$

b) (1 point) En général, dans quel cas la sortie de l'algorithme **mystère** est-elle un oui ?

c) (1 point) Supposons que $m = n$ en entrée. Quelle est alors la complexité temporelle de l'algorithme **mystère** en fonction de n ? (utiliser la notation $\Theta(\cdot)$)

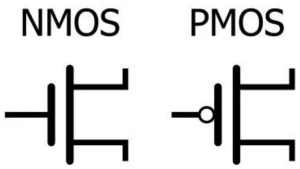
d) (1 point) Supposons maintenant que $m = 2$ soit un nombre fixé. Quelle est alors la complexité temporelle de l'algorithme **mystère** en fonction de n ? (utiliser à nouveau la notation $\Theta(\cdot)$)

e) (4 points) Dans ce dernier cas ($m = 2$), existe-t-il un algorithme dont la sortie soit identique à celle de l'algorithme **mystère**, mais dont la complexité temporelle soit *significativement* moindre (c'est-à-dire avec un ordre de grandeur inférieur en n) ? Si oui, écrire un tel algorithme (en spécifiant quelle est sa complexité temporelle) ; si non, expliquer pourquoi un tel algorithme n'existe pas.

Question 18: *Cette question est notée sur 4 points.*

☐0 ☐1 ☐2 ☐3 ☐4 *Réservé au correcteur*

En utilisant les transistors n-mos et p-mos vus au cours:



construire une porte NOR (=NON OU) dont la sortie s vaut 1 si et seulement si les deux entrées x et y valent chacune 0.

Rappel: Un transistor n-mos laisse passer le courant entre l'émetteur et le collecteur si et seulement si la tension d'entrée à la base est haute; il se passe exactement le contraire dans un transistor p-mos.

Question 19: *Cette question est notée sur 4 points.*

$\square_0$	$\square_1$	$\square_2$	$\square_3$	$\square_4$	*Réservé au correcteur*

Rappel: Si une sinusoïde pure $Y(t) = \sin(2\pi ft)$ passe à travers un filtre à moyenne mobile de période T_c, alors le signal sortant du filtre est donné par

$$\widehat{Y}(t) = \frac{\sin(\pi f T_c)}{\pi f T_c} \cdot \sin(2\pi ft - \pi f T_c)$$

a) (1 point) On considère le signal $X(t) = \sin(6\pi t) + \sin(8\pi t)$. Quel est le signal $\widehat{X}(t)$ sortant après passage à travers un filtre à moyenne mobile de période $T_c = 0.25$ sec ?

b) (1 point) Quelle est la bande passante du signal filtré $\widehat{X}$?

c) (1 point) A quelle fréquence f_e faut-il échantillonner la signal $\widehat{X}$ si on désire éviter l'effet stroboscopique lors de sa reconstruction au moyen de la formule d'interpolation ?

d) (1 point) Est-ce que la réponse à la question c) change si la période du filtre $T_c = 0.5$ sec ? Justifier.

Question 20: *Cette question est notée sur 4 points.*

☐0 ☐1 ☐2 ☐3 ☐4 *Réservé au correcteur*

Soit un message formé de 2^n lettres (avec $n \geq 3$), contenant 2^{n-2} fois la lettre A, 2^{n-2} fois la lettre B et 2^{n-3} fois la lettre C, le reste étant composé d'espaces (qu'on considère ici comme des lettres à part entière).

a) (2 points) On encode ce message en une suite de 0 et de 1 en utilisant l'algorithme de Huffman. Ecrire ci-dessous le dictionnaire obtenu et dessiner l'arbre correspondant.

b) (1 point) Calculer le nombre moyen de bits par lettre utilisé par cet encodage.

c) (1 point) En utilisant l'approximation $\log_2(3) \simeq 1.6$, calculer (approximativement) l'entropie du message.

Question 21: *Cette question est notée sur 4 points.*

$\square_0$ $\square_1$ $\square_2$ $\square_3$ $\square_4$ *Réservé au correcteur*

On considère le code correcteur d'erreurs suivant : pour tout envoi de 4 bits x_1, x_2, x_3, x_4, ceux-ci sont protégés par les 3 bits de parité suivants :

$$x_5 = x_1 \oplus x_2, \quad x_6 = x_3 \oplus x_4, \quad x_7 = x_5 \oplus x_6$$

a) (2 points) Quelle est la distance minimale de ce code correcteur d'erreurs ? Justifier.

b) (1 point) Combien d'effacements et d'erreurs un tel code permet-il de corriger ? (donner des réponses *numériques*)

c) (1 point) Si on sait qu'au plus une erreur est survenue lors de la transmission et qu'on reçoit la séquence 1111011, que sait-on alors sur le ou les mots de code envoyés à l'origine (donner toutes les possibilités) ?

Question 22: *Cette question est notée sur 12 points.*

0	1	2	3	4	5	6	7	8	9	10	11	12	*Réservé au correcteur*

On veut construire une liste chaînée qui contient la suite des mots d'un texte dans l'ordre de leur apparition. Vous pouvez utiliser la fonction `strlen()` pour obtenir la longueur d'une chaîne de caractères, mais vous ne pouvez pas utiliser d'autres fonctions prédéfinies, comme `strcpy`, `memcpy`, ou `strtok` par exemple.

a) (2 points) Écrivez une fonction qui reçoit en paramètre une chaîne de caractères `string` et qui retourne le nombre de caractères qui correspondent au plus long mot qui commence à `string[0]`. Un mot est une suite contiguë de caractères alphanumériques. Vous pouvez utiliser la fonction `isalnum(c)` qui retourne `1` si le caractère `c` est alphanumérique et `0` sinon.
Par exemple, `longueur_mot("le langage C")` et `longueur_mot("le?!lele")` doivent retourner `2` = la longueur du mot "le". Par contre l'appel `longueur_mot("... sofferte onde serene ...")` doit retourner `0`, car le premier caractère n'est pas alphanumérique. Pour la chaîne vide la valeur de retour doit également être `0`.

```
int longueur_mot(const char *string) {
```

```
}
```

b) (2 points) Écrivez une fonction qui reçoit en paramètre une chaîne de caractères `string` et un entier `n` positif et qui retourne un pointeur vers une nouvelle chaîne de caractères contenant une copie des `n` premiers caractères de `string`. Si `string` contient moins que `n` caractères, alors on copie tout le contenu de `string`.
Par exemple `copier("supercalifragilisticexpialidocious", 5)` retourne la chaîne `"super"`.

```
char* copier(const char *string, int n) {
```

```
}
```

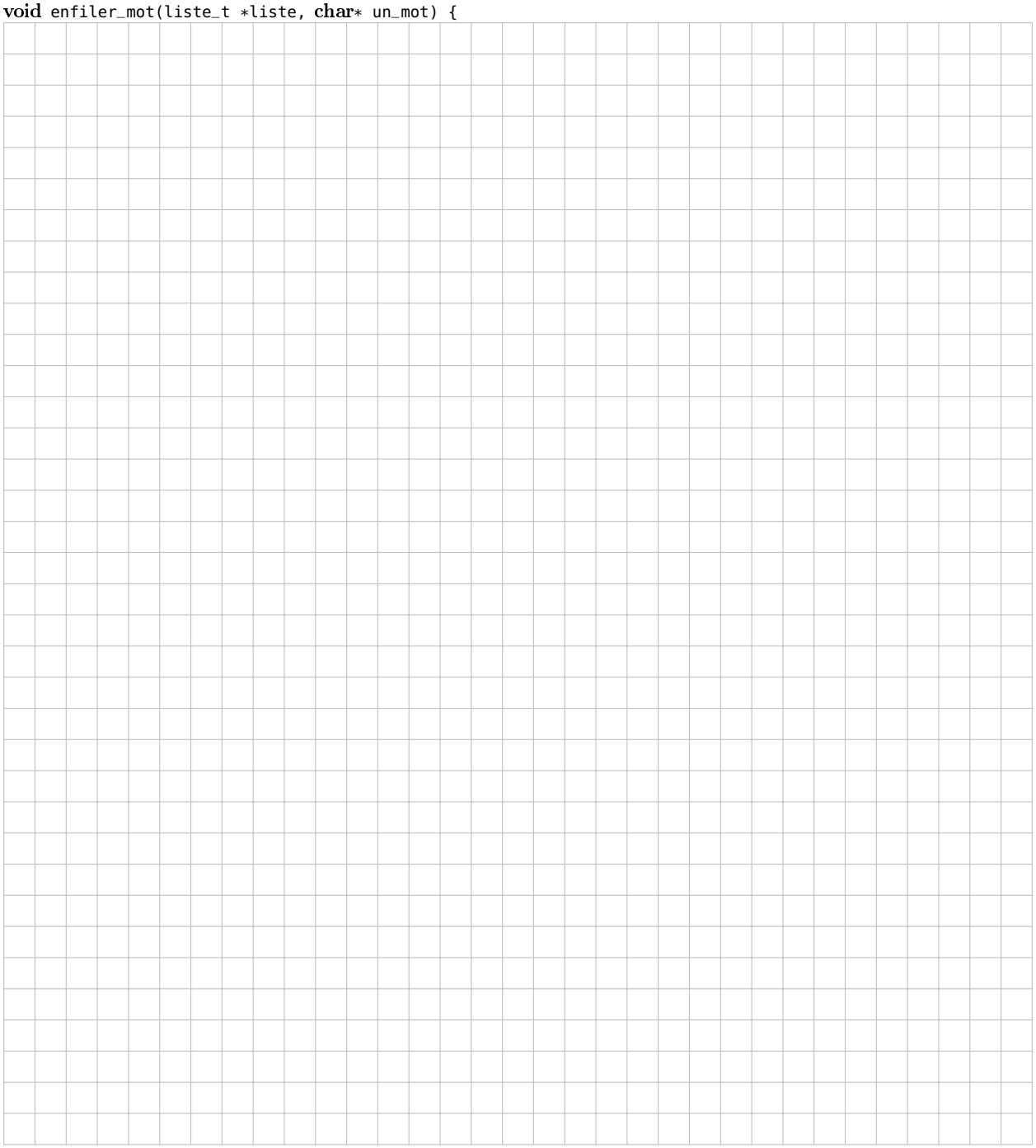
On définit les deux struct suivantes:

```
typedef struct _cell
{
    char *mot;
    struct _cell *next;
} cell_t;
```

```
typedef struct _liste
{
    cell_t *premier;
    cell_t *dernier;
} liste_t;
```

c) (3 points) Écrivez une fonction qui reçoit une liste et un pointeur vers une chaîne de caractères contenant un seul mot. Cette fonction rajoute une nouvelle cellule ayant pour contenu le pointeur `un_mot` **à la fin** de la liste chaînée `liste`. Cette fonction doit mettre à jour les champs `premier` et `dernier`. Pour une liste vide ces champs auront la valeur `NULL`.

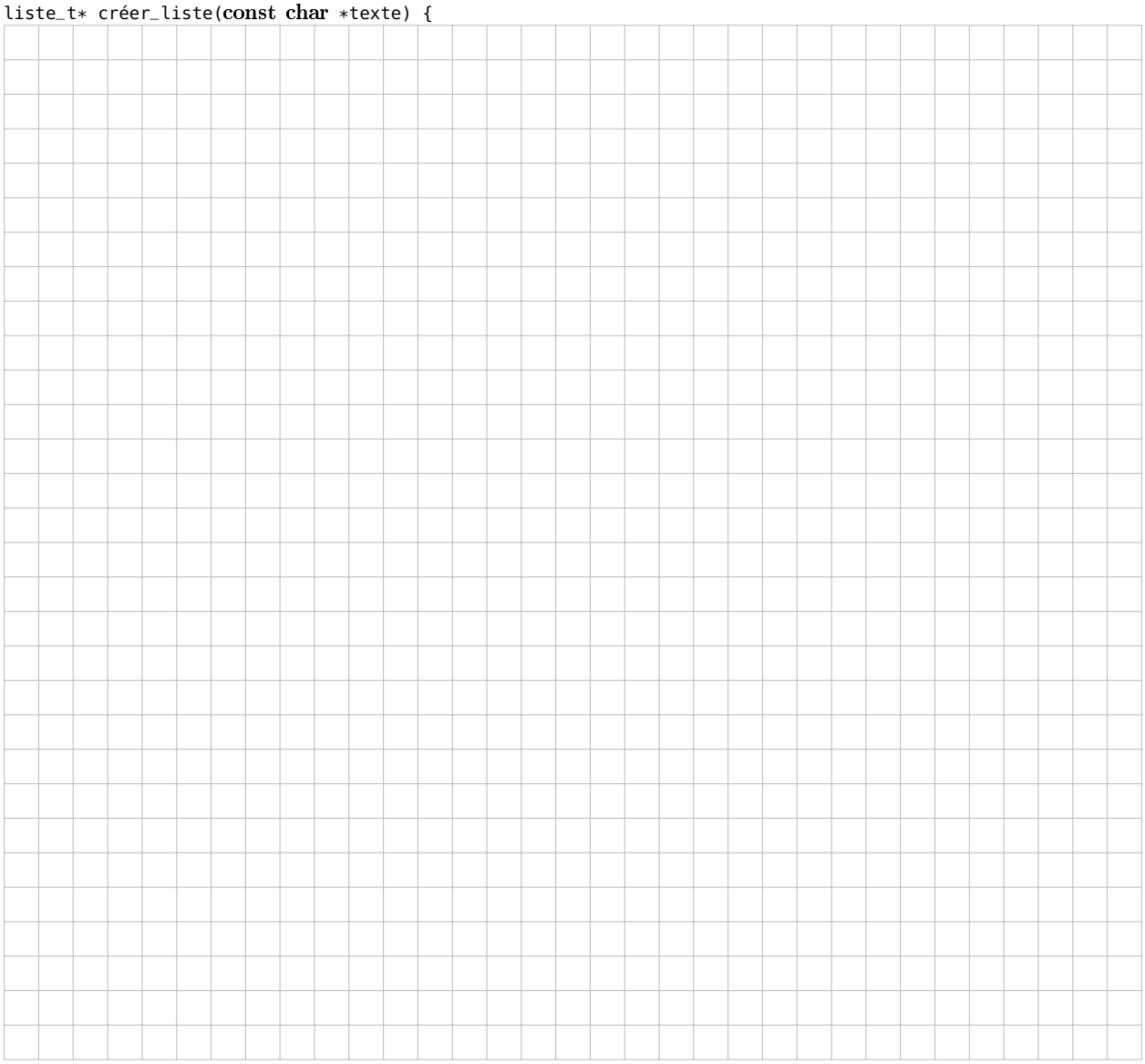
```
void enfiler_mot(liste_t *liste, char* un_mot) {
```



```
}
```

d) (3 points) Utilisez les trois fonctions ci-dessus pour implémenter une fonction qui prend en paramètre une chaîne de caractères `texte` et retourne la liste de mots qui s'y trouvent dans l'ordre de leur apparition.

```
liste_t* créer_liste(const char *texte) {
```



```
}
```

e) (2 points) Implémentez une fonction qui libère la mémoire d'une liste.

```
void libérer(liste_t *pliste) {
```

```
}
```