Algorithmique et Programmation – Examen (durée 1h30)

Première session du 11 mai 2026

Consignes pour la composition de l'examen sur feuille
L'utilisation du compilateur reste autorisée, mais uniquement depuis Moodle.

NOTES : Aucun document autorisé. Sont interdits les calculatrices, les téléphones, ainsi que tout autre ustensile de calcul ou de communication, en dehors de votre session Moodle via Safe Exam Browser depuis votre ordinateur.

Exercice 1 : Questions de connaissances générales (5 points)

Les réponses aux 5 questions ont déjà été données sur la page Moodle précédente.

Exercice 2 : Quatre fonctions et un programme à compléter (6 points)

Consignes des codes sources à compléter
Recopiez les codes sur votre copie en complétant vos réponses dans les petites cases grises. Chaque case grise correspond à un et un seul caractère à compléter.

Dans cet exercice, nous nous intéressons au chiffrement de messages, permettant d'encoder puis décoder un message à l'aide d'une clef. Une technique de chiffrage consiste à transposer la position des caractères du message, selon une permutation obtenue à partir de la clef, obtenant ainsi un anagramme du message originel. Une amélioration de ce type de chiffrage est le fractionnement préalable des caractères du message. Pour faciliter la compréhension, nous expliquons d'abord ces deux principes séparément, avant de les combiner.

Le principe de transposition

Par exemple, prenons le message "Bonjour tout le monde." et la clef "ZEBRAS".

La permutation est obtenue en calculant la séquence à partir de l'ordre alphabétique des lettres de la clef, puis en ordonnant la séquence selon l'indice de chaque lettre. Par exemple, la première lettre de la clef dans l'ordre alphabétique est le 'A' (càd 1 dans la séquence), qui est placé à l'indice 5 de la clef, alors la première composante de la permutation sera 5. La lettre suivante dans l'ordre alphabétique est le 'B' qui est en position 3, alors la deuxième composante de la permutation sera 3. Et ainsi de suite, pour le 'E' (2), le 'R' (4), le 'S' (6), puis le 'Z' (1) :

Indices	1	2	3	4	5	6
Clef	Z	E	B	R	A	S
Séquence	6	3	2	4	1	5
Permutation	5	3	2	4	6	1

Le chiffrement du message consiste à regrouper les caractères par blocs de 6 (càd la longueur de la clef), puis à lire tour à tour dans chaque bloc, en prenant à chaque fois le caractère indiqué par la permutation, et de répéter pour chaque composante de la permutation. Par exemple, pour le chiffrement du message "Bonjour_tout_le_monde." :

	Indices																								
Permutation	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6	**Résultat**
5	B	o	n	j	o	u	r	_	t	o	u	t	_	l	e	_	m	o	n	d	e	.	*	*	oum*
3	B	o	n	j	o	u	r	_	t	o	u	t	_	l	e	_	m	o	n	d	e	.	*	*	ntee
2	B	o	n	j	o	u	r	_	t	o	u	t	_	l	e	_	m	o	n	d	e	.	*	*	o_ld
4	B	o	n	j	o	u	r	_	t	o	u	t	_	l	e	_	m	o	n	d	e	.	*	*	jo_.
6	B	o	n	j	o	u	r	_	t	o	u	t	_	l	e	_	m	o	n	d	e	.	*	*	uto*
1	B	o	n	j	o	u	r	_	t	o	u	t	_	l	e	_	m	o	n	d	e	.	*	*	Br_n
	Message																								

Ainsi, le chiffrement obtenu pour le message est : "oum*nteeo_ldjo_.uto*Br_n".

Le principe de fractionnement

Le fractionnement consiste à remplacer les caractères du message par une représentation à plusieurs caractères (càd fractionnement), sur laquelle la transposition s'appliquera alors. Par exemple, considérons le nombre dans la table ASCII, que nous prenons sur trois chiffres, associé à chaque caractère :

B	o	n	j	o	u	r	_	t	o	u	t	_	l	e	_	m	o	n	d	e	.	*	*
066	111	110	106	111	117	114	032	116	111	117	116	032	108	101	032	109	111	110	100	101	046	042	042

Le message devient donc trois fois plus long :

`066111110106111117114032116111117116032108101032109111110100101046042042`

Chiffrage par fractionnement puis transposition

Maintenant, nous combinons les deux principes par deux étapes successives en appliquant la transposition sur les caractères fractionnés (toujours avec la clef "ZEBRAS") :

`Bonjour_tout_le_monde.**`

⤋ **Fractionnement**

`066111110106111117114032116111117116032108101032109111110100101046042042`

⤋ **Transposition**

`101311031044601467219012611111300104111011101100167216821062011111011110`

Déchiffrage

Nous proposons maintenant d'écrire un programme de déchiffrement, que nous appliquerons à décoder la phrase suivante de Paulo Coelho, obtenue par fractionnement ASCII sur 3 chiffres puis transposition à partir de la même clef "ZEBRAS" que ci-avant :

```
14103139110100303031110911110301131319000044450240179021140478617515215512048531762958262260
30121113100149010100131100310000211390131441011010011111101010111101111011101101011110000491
21276596532824208419024552120212174115622010111111011110011111110111101111111100110100
```

`exercice2.c`

```
1  #include <stdio.h>
2  #include <string.h>
```

1) Tout d'abord, nous avons besoin d'une fonction permettant de calculer une permutation `prm` à partir d'une clef `key` (de longueurs `k_len`). L'idée est d'initialiser la permutation avec l'identité, puis de la transposer en même temps que de trier les caractères de la clef par ordre croissant (par le tri à bulles). Complétez le code de cette fonction :

`exercice2.c`

```
4  ____ make_permutation(int *prm, char *key, int k_len) _
5    int i, j, t _
6    char tmp _
7
8    ___ (i=0 _ i < _____ _ i++) _
9      prm[i] = i _
10   _
11
12   ___ (j=0 _ j < _____ _ j++) _
13     ___ (i=0 _ i < _____ - j - 1 _ i++) _
14       __ (___[___] _ ___[_]) _
15         tmp = ___[_] _
16         ___[_] = ___[___] _
17         ___[___] = tmp _
```

```
18
19          t = ███[▒] ▒
20          ███[▒] = ███[███] ▒
21          ███[███] = ▒ ▒
22        ▒
23      ▒
24    ▒
25  ▒
```

2) Ensuite, nous avons besoin d'une fonction permettant de rétablir l'ordre de lecture des caractères, c'est-à-dire annulant la transposition, en recopiant dans `dst` les caractères provenant de la chaîne src (de longueur `s_len`) selon la permutation `prm` (de longueur `p_len`). Complétez le code de cette fonction :

exercice2.c

```
27  ████ untransposing (char *dst, char *src, int s_len, int *prm, int p_len) ▒
28    ███ i, j, k=0 ▒
29
30    ███ (███ ▒ j < █████ ▒ ███) ▒
31      ███ (███ ▒ i < (█████ / █████) ▒ ███) ▒
32        ███[▒ * █████ + ███[▒]] = ███[k++] ▒
33      ▒
34    ▒
35
36    dst[k] = '\0' ▒
37  ▒
```

3) Puis, nous avons besoin d'une fonction permettant de rétablir les codes ASCII des caractères d'origine, en les calculant à partir de la séquence de chiffres de la chaîne de caractères `src` (de longueur `s_len`) et en les recopiant dans la chaîne de caractères `dst`. Complétez le code de cette fonction :

exercice2.c

```
39  void defractionation (char *dst, char *src, int s_len) ▒
40    ███ i, k=0 ▒
41
42    ███ (███ ▒ i < ████ ▒ ████) ▒
43      ███[k++] = (███[▒] - '0') * 100 + (███[███] - '0') * 10 + (███[███] - '0') ▒
44    ▒
45
46    dst[k] = '\0' ▒
47  ▒
```

4) Maintenant, nous créons un programme combinant les deux fonctions pour décoder la phrase de Paulo Coelho. Complétez la fonction principale de ce programme :

exercice2.c

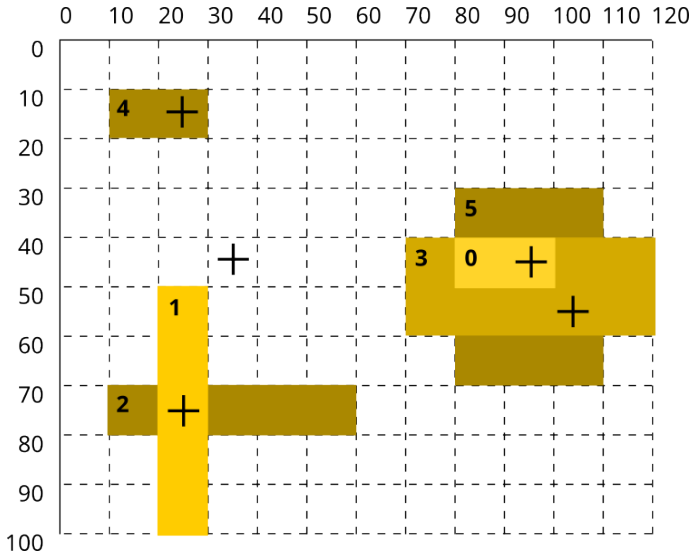
```
49  ███ ████() ▒
50    char messagebytes[1000] = "14103139110100303031110911110301131319000044450240179021 1404"
51     "7861751521551204853176295826226030121113100149010100131100310000211390131441011010011"
52     "1111010101111011111011101010111100004912127659653282420841902455212021217411562201011"
53     "1111011110011111110111101111111100110100" ;
54    char untransposed[1000] ;
55    char unfractioned[500] ;
```

```
56    char key[30] = "ZEBRAS" ;
57    int permutation[30] ;
58    int p_len = strlen(key), m_len = strlen(messagebytes) ;
59
60    ████████████████(████████████, ███, ██████) ;
61
62    ██████████████(████████████, ████████████, ██████, ████████████, ██████) ;
63
64    ███████("untransposed = %s\n", ████████████) ;
65
66    ████████████████(████████████, ████████████, ██████) ;
67
68    ███████("unfractioned = %s\n", ████████████) ;
69
70    ███████ 0 ;
71  ;
```

Exercice 3 : Une fonction et deux programmes à écrire (9 points)

Z buffer

Nous considérons une scène 2D dans laquelle des formes géométriques sont positionnées en x et y (en l'occurrence, des rectangles). La priorité d'affichage est déterminée par un ordre en z. Cet exercice se place donc dans le cadre de logiciels de dessin vectoriel (ex. PowerPoint, Illustrator, Draw.io). Un exemple de scène avec six rectangles est la suivante :



Chaque rectangle est numéroté par son ordre en z, de 0 à 5, sachant que la plus petite valeur est pour le rectangle le plus proche, et la plus grande pour le plus éloigné. Les rectangles sont codés en mémoire avec la structure de données suivante :

```
typedef struct rectangle {
  int pos_x, pos_y;
  int largeur, hauteur;
} rectangle;
```

Comme souvent en informatique, le point d'origine (0, 0) est en haut à gauche. De plus, la position du rectangle est donnée pour son point le plus haut et le plus à gauche. Cela donne alors le tableau suivant pour les six rectangles de la scène illustrée ci-dessus :

```
rectangle scene[6] = {
  {80, 40, 20, 10},
  {20, 50, 10, 50},
  {10, 70, 50, 10},
  {70, 40, 50, 20},
  {10, 10, 20, 10},
  {80, 30, 30, 40}
};
```

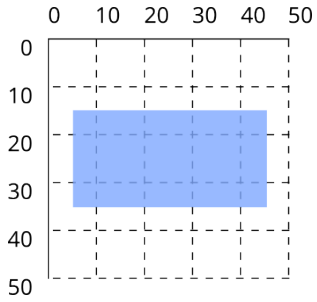
Nous voulons un logiciel qui permette de sélectionner des rectangles à partir d'une position de curseur de souris. Pour ce faire, nous commençons dans cet exercice par écrire un petit programme listant les rectangles qui incluent un point (x, y) donné par l'utlisateur.

(a) Tout d'abord, écrivez une fonction en langage C permettant de vérifier si un point est inclus dans un rectangle. La fonction renvoie la valeur 1 si le point est inclus, 0 sinon.

Le prototype de la fonction sera le suivant :
`char point_est_dans_rectangle (int point_x, int point_y, rectangle rect) ;`

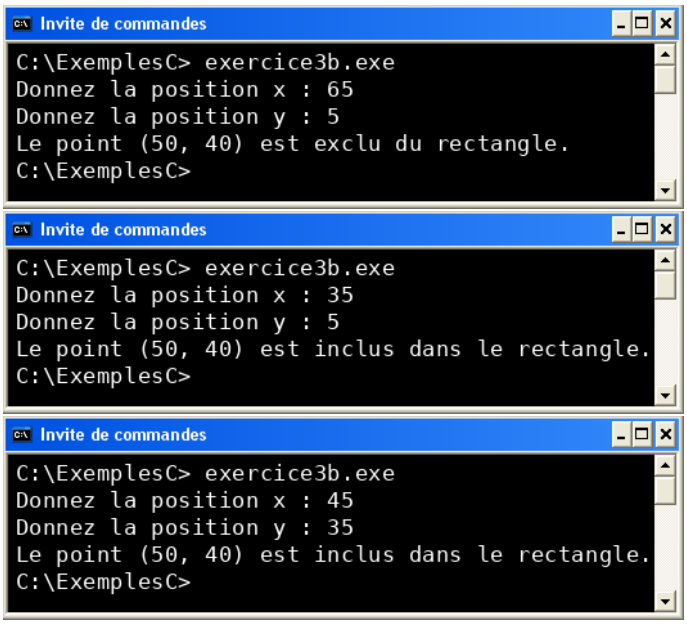
RAPPEL SUR LES STRUCTURES : En C, l'opérateur "." permet l'accès aux champs d'une structure. Donc, par exemple, si nous déclarons une variable `r1` de type `rectangle`, l'accès à ses quatre champs se fera comme suit pour un rectangle de largeur 40 et hauteur 20 placé en position (5, 15) :

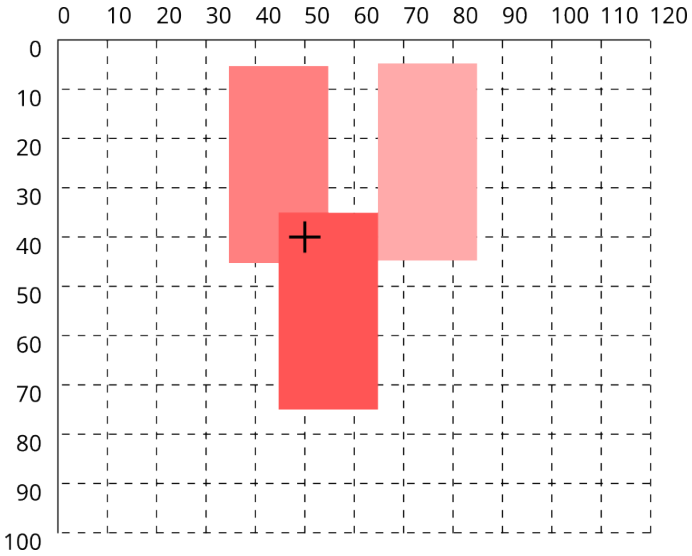
```
rectangle r1 ;
r1.pos_x = 5 ;
r1.pos_y = 15 ;
r1.largeur = 40 ;
r1.hauteur = 20 ;
```



(b) Pour tester la fonction (a), écrivez un programme en langage C qui utilise un rectangle de largeur 20 et hauteur 40, qui demande la position du rectangle à l'utilisateur, et qui teste l'inclusion du point (50, 40).

Voir trois exemples d'exécution de ce programme :





(c) Enfin, écrivez maintenant un programme en langage C qui demande les coordonnées d'un point à l'utlisateur, et qui retrouve les rectangles incluant ce point parmi les six rectangles de la scène déclarée ci-dessus, du plus proche au plus éloigné, en distinguant le premier rectangle atteint en Z. Ce programme doit lui aussi utiliser la fonction du (a).

Voir cinq exemples d'exécution de ce programme :