



Partiel – Algorithmique et programmation structurée en C

B. Mnassri, J. Palasi, J.P. Segado

Durée : 1h30

Support autorisé : Aucun

Total : 20 points

NOM :
Prénom :
Date :
Groupe :

Avertissement

Toute réponse ambiguë, non repérable ou illisible sera considérée comme incorrecte (note : 0).

Toute tentative de triche, sous quelque forme que ce soit, entraînera un signalement immédiat et pourra conduire à une procédure disciplinaire ainsi qu'à un avertissement officiel.

Exercice 1 — QCM (4 points)

Cochez la ou les bonnes réponses pour chaque question. **Toute réponse incomplète ou comportant au moins une erreur est considérée comme fausse.**

Q1. (0.2 pt) Lorsque `t` est un tableau d'entiers, l'affectation `int *p = t;` initialise `p` avec l'adresse de `t[0]`.

☐ Vrai

☐ Faux

Q2. (0.2 pt) La fonction qui lit une ligne entière dans un fichier texte est :

☐ `fgetc`

☐ `fscanf`

☐ `fgets`

☐ `readline`

Q3. (0.2 pt) `fopen("data.txt", "a")` écrase le fichier s'il existe.

☐ Vrai

☐ Faux

Q4. (0.2 pt) En langage C, toute valeur différente de 0 est considérée comme vraie dans une condition.

☐ Vrai

☐ Faux

Q5. (0.2 pt) L'oubli de `'\0'` dans une chaîne peut provoquer un comportement indéfini.

☐ Vrai

☐ Faux

Q6. (0.2 pt) Quelles propositions permettent à `f` de modifier une donnée du programme appelant ?

☐ `void f(int x);`

☐ `void f(float y);`

☐ `void f(int *x);`

☐ `void f(char *s);`

Q7. (0.2 pt) `void convertirMajuscules(char *s);` est un prototype qui correspond à :

- | | |
|--|---|
| ☐ Une fonction | ☐ Une procédure |
| ☐ Une fonction qui renvoie une chaîne | ☐ Une procédure qui renvoie un caractère |

Q8. (0.2 pt) On considère un pointeur `s` vers une structure contenant un champ `age`. Pour accéder à ce champ via le pointeur, on écrit :

- | | |
|--|--|
| ☐ <code>s.age</code> | ☐ <code>s->age</code> |
| ☐ <code>*s.age</code> | ☐ <code>*s->age</code> |

Q9. (0.2 pt) À propos des tableaux, `t` étant un tableau, quelles affirmations sont vraies ?

- | | |
|---|---|
| ☐ Dans une expression, un tableau est utilisé comme un pointeur sur son premier élément. | ☐ On peut écrire <code>t++</code> . |
| ☐ <code>t[i]</code> est équivalent à <code>*(t + i)</code> | ☐ <code>sizeof(t)</code> est égal à <code>sizeof(&t[0])</code> . |

Q10. (0.2 pt) Une boucle `do ... while` s'exécute :

- | | |
|---|---|
| ☐ Zéro fois si la condition est fausse | ☐ Un nombre fixe de fois |
| ☐ Au moins une fois | ☐ Tant que la condition est fausse |

Q11. (0.2 pt) En mémoire, un entier est stocké sur un nombre fixe de bits, déterminé par son type.

- | | |
|-------------------------------|-------------------------------|
| ☐ Vrai | ☐ Faux |
|-------------------------------|-------------------------------|

Q12. (0.2 pt) Pour tester l'égalité de deux entiers `x` et `y`, on peut écrire :

- | | |
|--|---|
| ☐ <code>x = y</code> | ☐ <code>y == x</code> |
| ☐ <code>x == y</code> | ☐ <code>equals(x,y)</code> |

Q13. (0.2 pt) Dans la déclaration suivante : `typedef struct { int x; } Point;` quel est le nom technique du type ?

- | | |
|--|--|
| ☐ <code>Point</code> | ☐ <code>struct anonyme</code> |
| ☐ <code>struct</code> | ☐ Il n'y a pas de nom technique |

Q14. (0.2 pt) Lors de la conception d'un programme, quel est l'ordre logique recommandé ?

- | | |
|---|---|
| ☐ Identifier les entrées, les sorties puis les traitements | ☐ Choisir le langage avant de comprendre le besoin |
| ☐ Écrire le code puis définir le problème | ☐ Compiler avant d'écrire l'algorithme |

Q15. (0.2 pt) Le cahier des charges d'un programme sert principalement à :

- | | |
|---|--|
| ☐ Décrire les fonctionnalités attendues et les contraintes | ☐ Compiler le programme |
| ☐ Écrire le code source | ☐ Tester les performances du processeur |

Q16. (0.2 pt) Parmi les instructions suivantes, lesquelles sont des boucles ?

- | | |
|---|--|
| ☐ <code>if</code> | ☐ <code>while</code> |
| ☐ <code>for</code> | ☐ <code>switch</code> |

Q17. (0.2 pt) Un algorithme est principalement :

- | | |
|--|--|
| ☐ Une suite d'instructions dépendant du langage | ☐ Un programme compilé |
| ☐ Une suite logique et ordonnée d'instructions | ☐ Un fichier exécutable |

Q18. (0.2 pt) Quel est le rôle d'un fichier `.c` dans un projet en langage C ?

- | | |
|--|---|
| ☐ Contenir le code source à compiler | ☐ Contenir les résultats d'exécution |
| ☐ Contenir le code machine exécutable | ☐ Gérer automatiquement la mémoire |

Q19. (0.2 pt) À propos de la compilation d'un programme en langage C, quelles affirmations sont vraies ?

- | | |
|---|--|
| ☐ La compilation transforme le code source en code machine. | ☐ Un programme peut être exécuté sans être compilé. |
| ☐ Une erreur de compilation empêche la création de l'exécutable. | ☐ La compilation permet de détecter certaines erreurs de syntaxe. |

Q20. (0.2 pt) Comment définit-on correctement une constante symbolique à l'aide du préprocesseur en langage C ?

- | | |
|---|--|
| ☐ <code>#define PI = 3.14</code> | ☐ <code>const float PI = 3.14;</code> |
| ☐ <code>#define PI 3.14</code> | ☐ <code>macro PI 3.14</code> |

Exercice 2 — Fichiers texte et chaînes de caractères (4.75 points)

On considère un fichier texte contenant des lignes, chacune constituée d'un mot (sans espaces) de longueur strictement inférieure à 50 caractères. On souhaite déterminer le mot le plus long contenu dans ce fichier à l'aide d'une procédure appelée depuis le programme principal comme suit :

```
1 int main(void) {  
2     char nomFichier[] = "mots.txt";  
3     afficherMotPlusLong(nomFichier);  
4     return 0;  
5 }
```

Les questions suivantes permettent de construire progressivement cette procédure.

Q1. (1.5 pt) Compléter la fonction suivante permettant de lire une ligne d'un fichier texte et de la stocker dans une chaîne de caractères de taille maximale `taille`. Si le caractère `'\n'` est présent à la fin de la ligne, il devra être supprimé. **La fonction renverra 1 si une ligne a été lue correctement, et 0 si la fin du fichier est atteinte.**

La fonction `fgets`, de syntaxe `char *fgets(char *str, int n, FILE *stream);`, lit au plus `n-1` caractères depuis le flux `stream` et les stocke dans `str`. Elle renvoie `str` en cas de succès et `NULL` si aucune ligne n'a pu être lue (fin de fichier ou erreur).

On rappelle que la fonction `strcspn(chaîne, "\n")` renvoie la position du premier caractère `'\n'` dans la chaîne.

```
1 int lireLigne(FILE *f, char *chaîne, int taille) {  
2     /* Lire une ligne du fichier. Si aucune ligne n'a pu être lue, retourner 0 */  
3  
4     if (.....) {  
5  
6         .....  
7     }  
8  
9     /* Supprimer le '\n' s'il existe */  
10  
11     .....  
12  
13     return 1;  
14 }
```

Q2. (1.25 pt) Écrire une fonction qui renvoie la longueur d'une chaîne de caractères sans utiliser la fonction `strlen`.
Voici le prototype :

```
1 int longueurChaine(const char *s);
```

[illegible]

Q3. (2 pts) Écrire une procédure qui, si le fichier peut être ouvert, lit l'intégralité du fichier dont le nom est donné en paramètre et affiche le mot le plus long contenu dans ce fichier ainsi que sa longueur. On pourra utiliser les fonctions écrites aux questions précédentes. Voici le prototype :

```
1 void afficherMotPlusLong(const char *nomFichier);
```

Note : Si le fichier ne peut pas être ouvert, la procédure affichera un message d'erreur puis exécutera simplement `return;` pour quitter, puisqu'elle est de type `void`.

This image shows a full page of white paper with horizontal dotted lines. The lines are evenly spaced and run across the width of the page, providing a guide for handwriting practice. There are no margins, text, or other markings on the page.

Exercice 3 — Structures et raisonnement algorithmique (6.5 points)

On souhaite gérer un produit contenant un libellé, un prix, une quantité en stock et des informations sur son fournisseur. Un produit peut également être associé à un autre produit (par exemple un produit complémentaire). On définit les structures suivantes :

```
1 typedef struct {
2     char nom[25];
3     char pays[25];
4 } Fournisseur;
5
6 typedef struct Produit {
7     char libelle[25];
8     float prix;
9     int quantite;
10    Fournisseur fournisseur;
11    struct Produit *associe;
12 } Produit;
```

Q1. (1.25 pt) Écrire une procédure qui affiche toutes les informations d'un produit (**hors produit associé**) à partir d'un pointeur sur `Produit`, y compris les informations relatives à son fournisseur. La procédure devra vérifier que le pointeur passé en paramètre n'est pas nul avant tout accès aux champs.

```
1 void afficherProduit(const Produit *p);
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Q2. (1.25 pt) Étant donné qu'un produit peut être associé à un autre produit, écrire une procédure qui associe deux produits en faisant pointer le champ `associe` du premier produit vers le second. L'association n'est possible que si les pointeurs sont valides et si le produit n'est pas déjà associé.

```
1 void associerProduits(Produit *p1, Produit *p2);
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Q3. (1.75 pt) Compléter le code suivant afin de saisir un tableau de 7 produits en demandant leurs informations à l'utilisateur (libellé, prix, quantité et informations du fournisseur), puis d'associer le premier élément du tableau au second élément et d'afficher les informations du premier élément du tableau.

```

1  int main(void) {
2      Produit tab[7];
3      for (int i = 0; i < 7; i++) {
4          printf("Produit %d\n", i + 1);
5
6          printf("Libelle : ");
7          fgets(....., 25, stdin);
8          tab[i].libelle[strcspn(tab[i].libelle, "\n")] = '\0';
9
10         printf("Prix : ");
11         scanf("%f", .....);
12
13         printf("Quantite : ");
14         scanf("%d", .....);
15         getchar(); // vider le '\n'
16
17         printf("Nom du fournisseur : ");
18         fgets(....., 25, stdin);
19         tab[i].fournisseur.nom[strcspn(tab[i].fournisseur.nom, "\n")] = '\0';
20
21         printf("Pays du fournisseur : ");
22         fgets(....., 25, stdin);
23         tab[i].fournisseur.pays[strcspn(tab[i].fournisseur.pays, "\n")] = '\0';
24
25         tab[i].associe = NULL;
26     }
27     associerProduits(.....);
28
29     afficherProduit(.....);
30
31     return 0;
32 }

```

Q4. (1.5 pt) On dispose du tableau de produits saisi précédemment. Compléter le pseudo-code ci-dessous afin de déterminer et afficher le produit dont la quantité en stock est la plus élevée.

Algorithme AfficherProduitStockMax

Entrées : tab (tableau de produits), taille (nombre de produits)

Sortie : affichage du produit ayant la quantité maximale

DEBUT

max ←

i ←

TANT QUE i < FAIRE

SI tab[i].quantite > ALORS

max ←

FIN SI

i ←

FIN TANT QUE

Afficher "Produit avec le stock maximal"

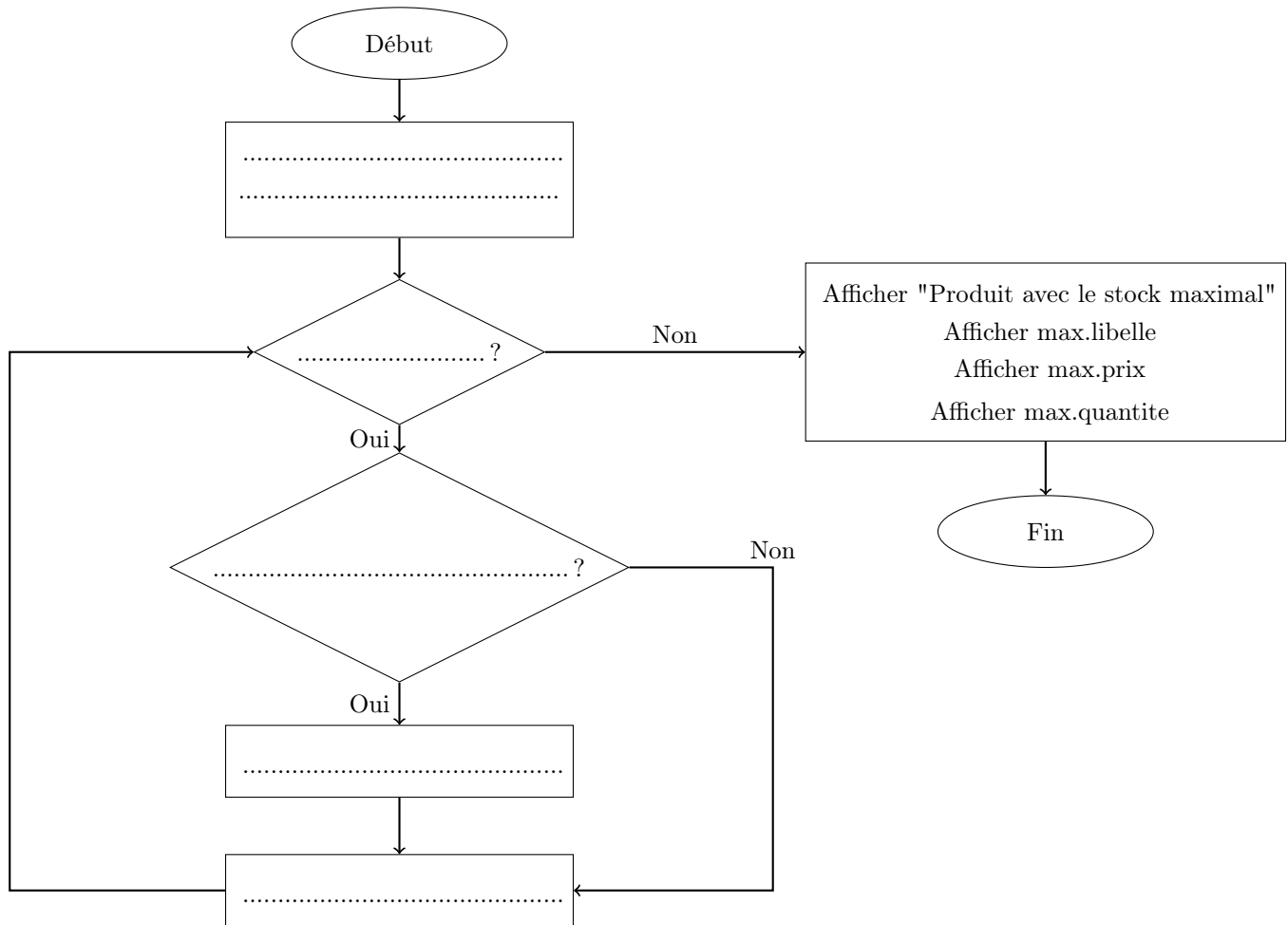
Afficher max.libelle

Afficher max.prix

Afficher max.quantite

FIN

Q5. (0.75 pt) Compléter le logigramme ci-dessous correspondant au pseudo-code de la question précédente, en renseignant les instructions manquantes dans chaque bloc.



Exercice 4 — Tableaux, pointeurs et représentation mémoire (4,75 points)

On considère le tableau d'entiers suivant :

```

1 #define N 5
2 int t[N] = {4, -2, 7, 0, -3};

```

Q1. (1 pt) Représenter l'organisation du tableau `t` en mémoire en indiquant, pour chaque élément, l'adresse mémoire du premier octet, la valeur décimale stockée, l'élément du tableau correspondant (`t[0]`, `t[1]`, ...) ainsi que la représentation hexadécimale de la valeur en mémoire sur 4 octets (**taille occupée par un entier**). On supposera que l'élément `t[0]` est stocké à l'adresse `09000`. Les adresses des éléments suivants devront être déduites en conséquence.

Adresse mémoire	Valeur décimale	Élément	Stockage en mémoire (hexadécimal)

On donne les représentations hexadécimales (sur 4 octets) des valeurs : $-2 \rightarrow \text{FF FF FF FE}$ et $-3 \rightarrow \text{FF FF FF FD}$. On rappelle qu'une valeur décimale positive inférieure ou égale à 9 est représentée en hexadécimal par le même chiffre, complété par des zéros afin d'occuper 4 octets.

Q2. (1 pt) On considère la déclaration suivante : `int *p = t;`. Compléter le tableau ci-dessous en indiquant, pour chaque expression, la valeur retournée par l'opérateur `sizeof` ainsi qu'une justification brève. On supposera qu'un `int` occupe 4 octets et que la taille d'un pointeur est de 8 octets sur une architecture 64 bits.

Expression	Valeur	Justification
sizeof(t)		
sizeof(p)		
sizeof(*p)		
sizeof(t[0])		

Q3. (1.75 pt) Écrire une fonction qui compte le nombre d'éléments strictement positifs d'un tableau, en le parcourant à l'aide d'un pointeur (sans utiliser la notation `t[i]`). Voici le prototype, où `n` est la taille du tableau :

```
1 int compterPositifs(const int *t, int n);
```

[illegible]

Q4. (1 pt) Compléter le code suivant afin de calculer le minimum d'un tableau à l'aide d'un pointeur :

```

1 int minimum(const int *t, int n) {
2
3     const int *p = .....;
4
5     int min = .....;
6
7     for (int i = 1; i < n; i++) {
8
9         if (..... < min) {
10
11             min = .....;
12         }
13     }
14     return min;
15 }

```