Correction du 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 :

- ☐ `s.age`
- ☒ `s->age`
- ☐ `*s.age`
- ☐ `*s->age`

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 `t++`.
- ☒ `t[i]` est équivalent à `*(t + i)`
- ☐ `sizeof(t)` est égal à `sizeof(&t[0])`.

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 :

- ☐ `x = y`
- ☒ `y == x`
- ☒ `x == y`
- ☐ `equals(x,y)`

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

- ☐ `Point`
- ☐ `struct anonyme`
- ☐ `struct`
- ☒ 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 ?

- ☐ if
- ☒ while
- ☒ for
- ☐ switch

Q17. (0.2 pt) Un algorithme est principalement :

- ☐ Une suite d'instructions dépendant du langage
- ☒ Une suite logique et ordonnée d'instructions
- ☐ Un programme compilé
- ☐ 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 le code machine exécutable
- ☐ Contenir les résultats d'exécution
- ☐ 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.
- ☒ Une erreur de compilation empêche la création de l'exécutable.
- ☐ Un programme peut être exécuté sans être compilé.
- ☒ 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 ?

- ☐ `#define PI = 3.14`
- ☒ `#define PI 3.14`
- ☐ `const float PI = 3.14;`
- ☐ `macro PI 3.14`

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 :

```c
int main(void) {
    char nomFichier[] = "mots.txt";
    afficherMotPlusLong(nomFichier);
    return 0;
}
```

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(chaine, "\n")` renvoie la position du premier caractère `'\n'` dans la chaîne.

```c
int lireLigne(FILE *f, char *chaine, int taille) {
    /* Lire une ligne du fichier. Si aucune ligne n'a pu être lue, retourner 0 */

    if (fgets(chaine, taille, f) == NULL) {

        return 0;
    }

    /* Supprimer le '\n' s'il existe */

    chaine[strcspn(chaine, "\n")] = '\0';

    return 1;
}
```

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

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

.....

```c
int longueurChaine(const char *s) {
    int i = 0;

    while (s[i] != '\0') {
        i++;
    }

    return i;
}
```

.....

.....

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 :

```c
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`.

.....

.....

.....

```c
void afficherMotPlusLong(const char *nomFichier) {
    FILE *f = fopen(nomFichier, "r");
    if (f == NULL) {
        printf("Erreur ouverture fichier\n");
        return;
    }

    char mot[50];
    char max[50] = "";
    int len, maxLen = 0;

    while (lireLigne(f, mot, 50)) {
        len = longueurChaine(mot);
        if (len > maxLen) {
            maxLen = len;
            strcpy(max, mot);
        }
    }

    fclose(f);

    printf("Mot le plus long : %s\n", max);
    printf("Longueur : %d\n", maxLen);
}
```

.....

.....

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 :

```c
typedef struct {
    char nom[25];
    char pays[25];
} Fournisseur;

typedef struct Produit {
    char libelle[25];
    float prix;
    int quantite;
    Fournisseur fournisseur;
    struct Produit *associe;
} 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.

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

.....

```c
void afficherProduit(const Produit *p) {
    if (p == NULL) {
        return;
    }
    printf("Libelle : %s\n", p->libelle);
    printf("Prix : %.2f\n", p->prix);
    printf("Quantite : %d\n", p->quantite);
    printf("Fournisseur : %s\n", p->fournisseur.nom);
    printf("Pays : %s\n", p->fournisseur.pays);
}
```

.....

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.

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

.....

```c
void associerProduits(Produit *p1, Produit *p2) {
    if (p1 != NULL && p2 != NULL && p1 != p2 && p1->associe == NULL)
    {
        p1->associe = p2;
    }
}
```

.....

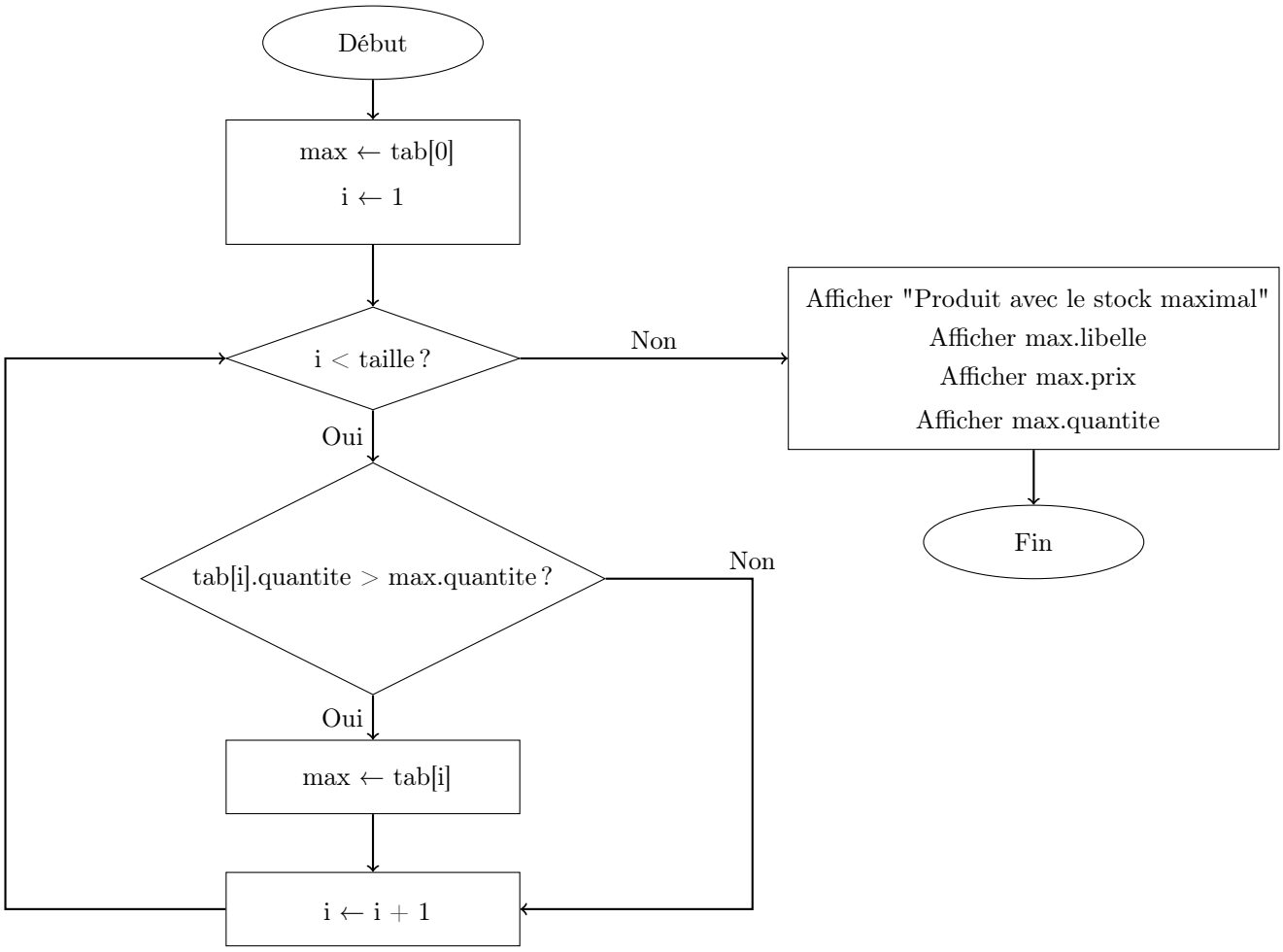
Q3. (1.75 pt) Compléter le code suivant afin de saisir un tableau de 7 produits en demandant leurs informations à l'utilisateur (libellé, prix, quantité et informations du fournisseur), puis d'associer le premier élément du tableau au second élément et d'afficher les informations du premier élément du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

```c
int main(void) {
    Produit tab[7];
    for (int i = 0; i < 7; i++) {
        printf("Produit %d\n", i + 1);

        printf("Libelle : ");
        fgets(tab[i].libelle, 25, stdin);
        tab[i].libelle[strcspn(tab[i].libelle, "\n")] = '\0';

        printf("Prix : ");
        scanf("%f", &tab[i].prix);

        printf("Quantite : ");
        scanf("%d", &tab[i].quantite);
        getchar(); // vider le '\n'

        printf("Nom du fournisseur : ");
        fgets(tab[i].fournisseur.nom, 25, stdin);
        tab[i].fournisseur.nom[strcspn(tab[i].fournisseur.nom, "\n")] = '\0';

        printf("Pays du fournisseur : ");
        fgets(tab[i].fournisseur.pays, 25, stdin);
        tab[i].fournisseur.pays[strcspn(tab[i].fournisseur.pays, "\n")] = '\0';

        tab[i].associe = NULL;
    }
    associerProduits(&tab[0], &tab[1]);

    afficherProduit(&tab[0]);

    return 0;
}
```

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 ← tab[0]

    i ← 1

    TANT QUE i < taille FAIRE

        SI tab[i].quantite > max.quantite ALORS

            max ← tab[i]
        FIN SI

        i ← i + 1
    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 :

```c
#define N 5
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 `@9000`. Les adresses des éléments suivants devront être déduites en conséquence.

Adresse mémoire	Valeur décimale	Élément	Stockage en mémoire (hexadécimal)
@9000	4	`t[0]`	`00 00 00 04`
@9004	-2	`t[1]`	`FF FF FF FE`
@9008	7	`t[2]`	`00 00 00 07`
@9012	0	`t[3]`	`00 00 00 00`
@9016	-3	`t[4]`	`FF FF FF FD`

On donne les représentations hexadécimales (sur 4 octets) des valeurs : $-2 \rightarrow$ `FF FF FF FE` et $-3 \rightarrow$ `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)`	20	`t` est un tableau de 5 entiers, chaque entier occupant 4 octets ($5 \times 4 = 20$).
`sizeof(p)`	8	`p` est un pointeur vers `int` ; la taille d'un pointeur est de 8 octets sur une architecture 64 bits.
`sizeof(*p)`	4	`*p` est la valeur pointée par `p`, de type `int`, qui occupe 4 octets.
`sizeof(t[0])`	4	`t[0]` est un élément du tableau, de type `int`, qui occupe 4 octets.

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 :

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

.....

```c
int compterPositifs(const int *t, int n) {
    int compteur = 0;
    const int *p = t;

    for (int i = 0; i < n; i++) {
        if (*p > 0) {
            compteur++;
        }
        p++;
    }

    return compteur;
}
```

.....

.....

Q4. (1 pt) Compléter la fonction suivante pour déterminer le minimum d'un tableau à l'aide d'un pointeur :

```c
int minimum(const int *t, int n) {

    const int *p = t;

    int min = *p;

    for (int i = 1; i < n; i++) {

        if (*(p + i) < min) {

            min = *(p + i);
        }
    }
    return min;
}
```