# Partiel Informatique – Semestre 2

Correction des exercices

B. Mnassri

01/06/2025

## Exercice 1 : Questions de cours/TPs (files, piles) (6 points)

### Question 1 (1 pt)

— **Pile** : **C. Dernier entré, premier sorti (LIFO)**
— **File** : **A. Premier entré, premier sorti (FIFO)**

### Question 2 (1,5 pt)

**Opérations :**

1. Enfiler : 12, 7, 23
2. Défiler deux fois ⇒ reste : 23
3. Enfiler : 14, 8, 35

**2a. État final de la file circulaire (taille 5) :**

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 35 | _ | 23 | 14 | 8 |

Tête = 2, Queue = 1
**2b. Prochain élément défilé :** 23

### Question 3 (1 pt)

— Enfilage : **Fin de liste**
— Défilage : **Début de liste**

### Question 4 (1 pt)

**Réponse correcte : C** Après empilements X, H, P puis un dépilement, la pile contient H → X

### Question 5 (1,5 pt)

**Réponse correcte : A**
— Empiler tous les éléments dans une pile
— Puis dépiler et enfiler à nouveau

## Exercice 2 : Compréhension de code. Utilisation d'une pile (3 points)

### Question

La fonction `toPost()` transforme une expression arithmétique parenthésée (forme infixée) en une version postfixée (notation polonaise inverse), en utilisant une pile pour stocker temporairement les opérateurs.

On vous demande ce que retourne l'appel suivant :

```
toPost("(14*((9-3)+(817/2)))")
```

### Analyse étape par étape

— Les chiffres sont copiés directement dans la chaîne de sortie.
— Les opérateurs sont empilés, puis insérés lors de la rencontre d'une parenthèse fermante `')'`.
— Des espaces sont ajoutés entre chaque opérande/opérateur.

### Résultat final

```
14 9 3 - 817 2 / + *
```

### Tableau des caractères produits (1 caractère par case)

| **Indice** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Caractère** | 1 | 4 | ␣ | 9 | ␣ | 3 | ␣ | - | ␣ | 8 | 1 | 7 | ␣ | 2 | ␣ | / | ␣ | + | ␣ | * | \0 |

## Exercice 3 : Structures, files et files de priorité (11 points)

### Question 1 (1 pt)

```c
typedef struct {
    char num_secu[14];    // 13 caractères + '\0'
    char *nom;            // chaîne dynamique
    int priorite;         // priorité attribuée
} t_patient;
```

### Question 2 (2 pts)

```c
void initPatient(t_patient *p) {
    char buffer[100];

    // Saisie du numéro de sécurité sociale
    printf("Entrez le numéro de sécurité sociale (13 chiffres) : ");
    fgets(buffer, sizeof(buffer), stdin);
    buffer[strcspn(buffer, "\n")] = '\0'; // Enlever le \n éventuel
    strncpy(p->num_secu, buffer, 14);
```

```
    p->num_secu[13] = '\0'; // sécurité

    // Saisie du nom
    printf("Entrez le nom du patient : ");
    fgets(buffer, sizeof(buffer), stdin);
    buffer[strcspn(buffer, "\n")] = '\0'; // Enlever le \n éventuel

    // Allocation dynamique du champ nom
    p->nom = malloc(strlen(buffer) + 1);
    if (p->nom != NULL) {
        strcpy(p->nom, buffer);
    } else {
        fprintf(stderr, "Erreur d'allocation mémoire pour le nom.\n");
        exit(1);
    }

    // Initialisation de la priorité
    p->priorite = 0;
}
```

### Question 3 (1 pt)

```
void saisirPriorite(t_patient *p) {
    int prio;

    do {
        printf("Saisissez la priorité du patient (entre 1 et 4) : ");
        if (scanf("%d", &prio) != 1) {
            while (getchar() != '\n'); // Vider le buffer en cas d'erreur
            prio = 0; // valeur invalide pour rester dans la boucle
        }
    } while (prio < 1 || prio > 4);

    p->priorite = prio;
}
```

### Question 4 (2 pts)

```
void gererPatient(t_file *attente, t_tas *priorites) {
    if (fileVide(attente)) {
        printf("La file d'attente est vide.\n");
        return;
    }

    // 1. Retirer le prochain patient (pointeur générique -> t_patient*)
    t_patient *p = (t_patient *)defiler(attente);

    if (p == NULL) {
        printf("Erreur : patient nul.\n");
        return;
    }
```

```
    // 2. Demander au médecin de saisir la priorité
    saisirPriorite(p);

    // 3. Créer un noeud pour le tas
    t_noeud nouveau;
    nouveau.cle = p->priorite;  // la priorité détermine la position dans
       le tas
    nouveau.data = p;           // on insère le pointeur vers la structure
        patient

    // 4. Ajouter au tas de priorité
    ajouter_au_tas(priorites, nouveau);
}
```

### Question 5 (5 pts)

```
int main()
{
    // déclaration d'une file et d'un tas
    t_file f;
    t_tas t;

    // initialiser la file
    init_file(&f);

    // initialiser le tas avec un nombre de niveaux égal à 5
    init_tas(&t, 5);

    // déclaration de variables
    t_patient *p;
    int i;

    // insertion de 10 patients dans la file d'attente
    for(i = 0; i < 10; i++) {
        // allouer une structure patient
        p = (t_patient *)malloc(sizeof(t_patient));

        // saisie des informations sur le patient
        initPatient(p);

        // enfiler le patient dans la file d'attente
        enfiler(&f, p);
    }

    printf("\n");

    // tant que la file n'est pas vide
    while(fileVide(&f) != 1) {
        // gérer un patient
        gererPatient(&f, &t);
    }

    // libérer correctement la mémoire
```

```
    // tant que le tas n'est pas vide
    while(!tasVide(&t)) {
        // supprimer le prochain patient du tas
        p = (t_patient *)supprimer_du_tas(&t);

        // libérer la chaîne pour le nom du patient
        free(p->nom);

        // libérer la structure patient
        free(p);
    }

    return 0;
}
```