Correction Partiel de Rattrapage
Gestion d'un Parking Intelligent

B. Mnassri – 24 juin 2025

Partie 1 : Structures et concepts (2 pts)

Q1. (1 pt) Quelle structure est la plus adaptée pour une file avec priorité possible ?
Réponse correcte : `(c) Liste chaînée avec recherche conditionnelle`

Justification des autres choix :

(a) *Pile (LIFO)* : ne respecte pas l'ordre d'arrivée ni la gestion des priorités.
(b) *Liste triée* : nécessiterait un tri à chaque insertion, ce qui est inefficace.
(d) *Tableau dynamique trié* : les opérations d'insertion/suppression seraient coûteuses.

Q2. (1 pt) Justifiez brièvement votre choix.
Correction attendue :
Une **liste chaînée avec recherche conditionnelle** permet de parcourir efficacement la file pour détecter la première voiture prioritaire sans trier l'ensemble. Cela permet de respecter à la fois la logique d'une file et la nécessité de gérer les priorités dynamiquement.

Partie 2 : Complétion de code (5 pts)

Q3. (1,5 pt) Considérant les informations décrites dans le contexte, complétez et commentez la structure suivante :

```c
typedef struct {
    char* plaque;          // chaîne dynamique (allocation mémoire)
    int heure_arrivee;     // de 0 à 23
    int prioritaire;       // 0 = non, 1 = oui
} t_voiture;
```

Q4. (3,5 pts)

(a) **(0,5 pt) Expliquer brièvement l'utilité de `toUpperStr(...)`**
Elle permet de normaliser la saisie utilisateur (ex : "ab-123-cd" devient "AB-123-CD") afin de comparer les chaînes de manière uniforme, sans tenir compte de la casse.

(b) **(1 pt) Compléter la ligne ayant comme commentaire // Convertit en majuscules**

```c
toUpperStr(temp); // Convertit en majuscules
```

(c) **(1 pt) Compléter le paramètre manquant dans le `scanf(...)` pour lire l'heure**

```c
&v->heure_arrivee
```

(d) **(1 pt) Compléter les deux lignes à la fin de la fonction pour vider le buffer après la saisie de la priorité**

```c
viderBuffer();
```

à placer dans les deux blocs 'if' et 'else' du test sur la priorité.

Code complété :

```c
void initVoiture(t_voiture *v) {
    char temp[100];

    // Saisie de la plaque (dynamique)
    do {
        printf("Plaque (5 à 15 caractères, lettres/chiffres/tirets/espaces) : ");
        fgets(temp, sizeof(temp), stdin);
        temp[strcspn(temp, "\n")] = '\0'; // Enlève le \n
        toUpperStr(temp); // Convertit en majuscules

        if (plaqueValide(temp)) {
            v->plaque = malloc(strlen(temp) + 1);
            if (v->plaque == NULL) {
                fprintf(stderr, "Erreur d'allocation mémoire.\n");
                exit(EXIT_FAILURE);
            }
            strcpy(v->plaque, temp);
            break;
        } else {
            printf("Plaque invalide. Veuillez réessayer.\n");
        }
    } while (1);

    // Heure d'arrivée
    do {
        printf("Heure d'arrivée (0-23) : ");
        if (scanf("%d", &v->heure_arrivee) != 1 || v->heure_arrivee < 0 || v->
            heure_arrivee > 23) {
            printf("Heure invalide. Veuillez entrer un nombre entre 0 et 23.\n");
            viderBuffer();
        } else {
            viderBuffer();
            break;
        }
    } while (1);

    // Prioritaire ?
    do {
        printf("Prioritaire ? (0 = non, 1 = oui) : ");
        if (scanf("%d", &v->prioritaire) != 1 || (v->prioritaire != 0 && v->prioritaire !=
             1)) {
            printf("Valeur invalide. Entrez 0 (non) ou 1 (oui).\n");
            viderBuffer();
        } else {
            viderBuffer();
            break;
        }
    } while (1);
}
```

Partie 3 : Compréhension de code (5 pts)

Q5. (0,5 pt) Pourquoi la fonction `initVoiture()` utilise-t-elle une allocation dynamique (`malloc`) pour le champ `plaque` ?

Réponse correcte : (c) Pour pouvoir stocker des plaques de longueur variable, notamment étrangères.

Explication : Une allocation dynamique permet d'ajuster la taille du tableau à la longueur réelle de la plaque (entre 5 et 15 caractères), contrairement à un tableau statique.

Q6. (0,5 pt) Que se passe-t-il si l'on oublie d'appeler `viderBuffer()` après un `scanf()` ?

Réponse correcte : (c) Le caractère `\n` reste dans le tampon et fausse les lectures suivantes.

Explication : Cela empêche les `fgets()` ou `scanf()` suivants de fonctionner correctement car ils lisent directement le `\n` laissé par l'entrée précédente.

Q7. (1 pt) Que se passe-t-il si l'utilisateur saisit `eB-231kc` ?

Étapes :

- L'utilisateur saisit "eB-231kc" suivi d'un `\n`.
- La fonction `fgets()` lit cette chaîne et remplace le `\n` par `'\0'`.
- `toUpperStr(temp)` transforme la chaîne en "EB-231KC".
- `plaqueValide(temp)` vérifie que :
  - la longueur est entre 5 et 15 caractères : OK (8 caractères)
  - les caractères sont alphanumériques : OK
- La plaque est donc considérée comme valide, et copiée dynamiquement dans `v->plaque`.

Q8. (1 pt) Libération mémoire après `initVoiture()`

(a) Réponse correcte : **(a.3)** *L'accès à la plaque après l'appel provoque un comportement indéfini.*

Explication : Libérer `v->plaque` dans `initVoiture()` rend son contenu inaccessible pour les appels ultérieurs, ce qui peut entraîner des erreurs ou un crash.

(b) Réponse correcte : **(b.3)** *Libérer la mémoire une fois que la structure `t_voiture` n'est plus utilisée.*

Explication : Cela garantit que l'information reste utilisable pendant toute sa durée de vie, tout en évitant les fuites mémoire.

Q9. (1 pt) Sortie produite par la fonction `afficher`

```
>> AB123CD | 14h | PRIO
```

Explication :

- `v->plaque` = "AB123CD"
- `v->heure_arrivee` = 14
- `v->prioritaire` = 1 donc affichage de `"PRIO"`

Q10. (1 pt) Exemple le plus simple d'instructions en langage C pour produire la sortie :

```c
t_voiture v = {"CF-478-CD", 18, 0};
afficher(&v);
```

Partie 4 : Simulation du système de gestion (8 pts)

Q11. (3 pts) Complétion de la fonction `autoriserEntree()`

```c
void* autoriserEntree(t_file* f) {
    t_maillonF *courant = f->tete;
    t_maillonF *precedent = NULL;

    // Recherche de la première voiture prioritaire
    while (courant && ((t_voiture*)courant->data)->prioritaire == 0) {
        precedent = courant;
        courant = courant->suivant;
    }

    // Si aucune voiture prioritaire trouvée ou voiture prioritaire est en tête
    if (courant == NULL || courant == f->tete)
        return defiler(f);

    // Suppression du maillon prioritaire non en tête
    precedent->suivant = courant->suivant;
    if (courant == f->fin)
        f->fin = precedent;

    void* data = courant->data;
    free(courant);
    return data;
}
```

Q12. (1,5 pt) Comportement de `autoriserEntree()` si la file est vide

(a) **Elle retourne `NULL`.**

(b) **Un accès direct à `prochaine->plaque` dans `main()` provoquerait un comportement indéfini (souvent un `segfault`), car `prochaine == NULL`.**

(c) **Exemples de vérification à insérer dans `main()` pour éviter ce risque :**

Version avec `continue` :

```c
t_voiture* prochaine = autoriserEntree(&file);
if (prochaine == NULL) {
    printf("Erreur : aucune voiture à autoriser.\n");
    continue;
}
```

Version alternative avec `break` dans la condition de boucle :

```c
while (!fileVide(&file)) {
    t_voiture* prochaine = autoriserEntree(&file);
    if (!prochaine) break;

    // traitement de la voiture...
}
```

Q13. (3,5 pts) Complétion de la fonction `main()`

```c
int main() {
    t_file file;
    init_file(&file);

    int nb_voitures = 5;
    printf("=== Enregistrement de %d voitures ===\n", nb_voitures);

    for (int i = 0; i < nb_voitures; i++) {
        printf("Voiture %d :\n", i + 1);
        t_voiture* v = malloc(sizeof(t_voiture));
        if (!v) exit(1);

        // saisie interactive
        initVoiture(v);
        // insérer dans la file
        enfiler(&file, v);
    }

    while (!fileVide(&file)) {
        // autoriser entrée
        t_voiture* prochaine = autoriserEntree(&file);
        if (!prochaine) break;

        printf(">> %s | %dh | %s\n",
            prochaine->plaque,
            prochaine->heure_arrivee,
            prochaine->prioritaire ? "PRIO" : "standard"
        );

        // libérer mémoire
        free(prochaine->plaque);
        free(prochaine);
    }

    return 0;
}
```