Partiel de Rattrapage
Gestion d'un Parking Intelligent

B. Mnassri – 24 juin 2025

Durée : 1h30
Support autorisé : Aucun
Total : 20 points

Avertissement

Pour être prise en compte, chaque réponse doit être clairement numérotée et lisiblement rédigée.

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

Contexte

Un petit parking de 10 places est géré par un système informatisé. Les voitures arrivent et attendent leur tour dans une file. Certaines sont prioritaires (ambulances, pompiers...) et doivent être autorisées à entrer avant les autres, même si elles ne sont pas en tête de file.
Chaque voiture est identifiée par :

- une plaque d'immatriculation (chaîne dynamique)
- une heure d'arrivée (entier 0–23)
- un champ `prioritaire` (0 ou 1)

Pour manipuler la file, on utilise une structure chaînée composée de maillons contenant un pointeur vers les données et un pointeur vers le maillon suivant.

```c
// Maillon de la file
typedef struct maillon {
    void* data;
    struct maillon* suivant;
} t_maillonF;

// Structure représentant une file
typedef struct {
    t_maillonF* tete; // élément en tête (à défiler)
    t_maillonF* fin;  // dernier maillon (où enfiler)
} t_file;
```

Fonctions disponibles pour la gestion de la file

Les fonctions suivantes sont supposées disponibles et correctement implémentées. Vous pouvez les utiliser directement dans vos réponses.

- `void init_file(t_file *f);` — Initialise une file vide (tête et fin à `NULL`).
- `int fileVide(t_file *f);` — Retourne `1` si la file est vide, `0` sinon.
- `void enfiler(t_file *f, void *data);` — Ajoute un nouvel élément à la fin de la file (après `fin`).
- `void* defiler(t_file *f);` — Retire et retourne le contenu de l'élément en tête. La mémoire du maillon est libérée.
- `void* tete(t_file *f);` — Retourne (sans retirer) le contenu de l'élément en tête de la file.

Partie 1 : Structures et concepts (2 pts)

Q1. (1 pt) Quelle structure est la plus adaptée pour une file avec priorité possible ?

(a) Pile (LIFO)
(b) Liste triée
(c) Liste chaînée avec recherche conditionnelle
(d) Tableau dynamique trié

Q2. (1 pt) Justifiez brièvement votre choix.

Partie 2 : Complétion de code (5 pts)

Q3. (1,5 pt) Considérant les informations décrites dans le contexte, complétez et commentez la structure suivante sachant que ses trois champs sont nommés `plaque`, `heure_arrivee` et `prioritaire` :

```
typedef struct {
    ____________________;  // ____________________
    int heure_arrivee;    // de 0 à 23
    ____________________;  // ____________________
} t_voiture;
```

Q4. (3,5 pts) On souhaite écrire une fonction `initVoiture` permettant de saisir les informations d'une voiture depuis le clavier, avec des vérifications renforcées.

— La plaque d'immatriculation peut provenir de différents pays. Elle doit contenir entre 5 et 15 caractères, en utilisant uniquement des lettres, des chiffres, des tirets ou des espaces.
— L'heure d'arrivée doit être un entier entre 0 et 23.
— Le champ `prioritaire` vaut 1 (oui) ou 0 (non).

Fonctions utilitaires disponibles :

Les fonctions suivantes sont supposées disponibles et correctement implémentées. Elles peuvent être utilisées librement dans vos réponses.

— `int plaqueValide(const char *p);` — Vérifie si la plaque passée en paramètre est conforme aux critères définis :
– longueur entre 5 et 15 caractères,
– uniquement lettres, chiffres, tirets ou espaces.
Retourne 1 si la plaque est valide, 0 sinon.
— `void viderBuffer();` — Vide le tampon d'entrée standard après un `scanf`, pour éviter des lectures parasites.
— `void toUpperStr(char *s);` — Convertit une chaîne de caractères en majuscules, lettre par lettre.

(a) (0,5 pt) Expliquer brièvement l'utilité de `toUpperStr(...)` dans ce contexte.
(b) (1 pt) Complétez la ligne ayant comme commentaire `// Convertit en majuscules`.
(c) (1 pt) La fonction `scanf(...)` retourne le nombre de valeurs lues avec succès. Il est donc recommandé d'utiliser `if (scanf(...) != 1)` pour vérifier que la saisie a bien été interprétée. Complétez le paramètre manquant dans le `scanf(...)` pour lire l'heure.
(d) (1 pt) Complétez les deux lignes à la fin de la fonction pour vider le buffer après la saisie de la priorité.

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

```

```c
    // 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
        ________________; // 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", ________________) != 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");
            ________________;
        } else {
            ________________;
            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` ?

(a) Pour respecter le format français `AB-123-CD`
(b) Pour limiter la taille des plaques à exactement 9 caractères
(c) Pour pouvoir stocker des plaques de longueur variable, notamment étrangères
(d) Pour permettre au programme de se terminer plus rapidement

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

(a) Le programme continue normalement

(b) Le programme plante immédiatement
(c) Le caractère `\n` reste dans le tampon et fausse les lectures suivantes
(d) Rien, car `scanf()` gère automatiquement le tampon

Q7. (1 pt) Lors de l'appel à la fonction `initVoiture()`, l'utilisateur est invité à saisir la plaque d'immatriculation lorsqu'il voit le message suivant :

```
Plaque (5 à 15 caractères, lettres/chiffres/tirets/espaces) :
```

Que se passe-t-il si l'utilisateur saisit la chaîne suivante : `eB-231kc` ?
Expliquez précisément les étapes de traitement effectuées par le programme.

Q8. (1 pt) Libération de mémoire dynamique et portée de vie
Pour rappel, dans la fonction `initVoiture()`, la plaque d'immatriculation d'une voiture est allouée dynamiquement avec `malloc` :

```
v->plaque = malloc(strlen(temp) + 1);
```

Supposons maintenant qu'on ajoute la ligne suivante à la fin de la fonction `initVoiture()` :

```
free(v->plaque);
```

(a) Que se passe-t-il si cette ligne est réellement exécutée alors que le programme continue d'accéder à la plaque après l'appel à `initVoiture()` ?

(a.1) Cela libère correctement la mémoire, sans effet secondaire.
(a.2) La plaque reste accessible normalement après l'appel à `initVoiture`.
(a.3) L'accès à la plaque après l'appel provoque un comportement indéfini (accès mémoire invalide).
(a.4) Cela force la réallocation automatique de la mémoire au prochain appel.

(b) Quelle est la bonne stratégie pour éviter une fuite mémoire sans provoquer de comportement indéfini ?

(b.1) Libérer la mémoire dans `initVoiture`, juste après l'avoir allouée.
(b.2) Ne jamais la libérer, car le système s'en charge à la fin du programme.
(b.3) Libérer la mémoire une fois que la structure `t_voiture` n'est plus utilisée (par exemple après un appel à `autoriserEntree`).
(b.4) La réallouer à chaque lecture avec `realloc` automatique.

Q9. (1 pt) Voici une fonction à analyser. Quelle est la sortie produite à l'écran ?

```c
void afficher(t_voiture* v) {
    printf(">> %s | %dh | %s\n",
           v->plaque,
           v->heure_arrivee,
           v->prioritaire ? "PRIO" : "standard");
}

t_voiture v = {"AB123CD", 14, 1};
afficher(&v);
```

Q10. (1 pt) Écrivez les instructions nécessaires en langage C (déclaration, initialisation et appel de fonction) pour obtenir exactement la sortie suivante :

```
>> CF-478-CD | 18h | standard
```

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

On souhaite simuler l'enregistrement et le traitement de l'entrée de véhicules dans le parking. Les véhicules prioritaires doivent être autorisés à entrer avant les autres.

Q11. (3 pts) Complétez la fonction `autoriserEntree` suivante, qui recherche dans la file la première voiture prioritaire (champ `prioritaire` à 1). Si aucune n'est trouvée, on autorise la voiture en tête.

```
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 ==
        __________________) {
        precedent = __________________;
        courant = __________________;
    }

    // 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
    __________________;
    if (courant == f->fin)
        f->fin = precedent;

    void* data = courant->data;
    __________________;
    return __________________;
}
```

Q12. (1,5 pt) Sachant que `defiler(f)` retourne NULL si la file est vide :

(a) Que retourne `autoriserEntree()` dans ce cas ?

(b) Que risque-t-il de se passer si le retour est utilisé sans vérification dans `main()` (par exemple accès à `prochaine->plaque`) ?

(c) Proposez une ligne de code (ou un bloc) à ajouter dans le `main()` pour éviter ce risque.

Q13. (3,5 pts) Complétez la fonction `main()` suivante, qui enregistre 5 voitures puis les autorise à entrer selon leur priorité. Utilisez la fonction `initVoiture()`.

```
int main() {
    t_file file;
    init_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);
        __________________ v = malloc(sizeof(t_voiture));
        if (!v) exit(1);

        // saisie interactive
        initVoiture(__________________);
        // insérer dans la file
        __________________;
    }

    while (!fileVide(&file)) {
```

```
        // autoriser entrée
        t_voiture* prochaine = ____________________;
        printf(">> %s | %dh | %s\n",
            prochaine->plaque,
            prochaine->heure_arrivee,
            prochaine->prioritaire ? "PRIO" : "standard"
        );

        // libérer mémoire
        ____________________;
        ____________________;
    }

    return 0;
}
```