Systeme de Gestion de Parking

Projet de fin de module

Projet de fin de module

Etudiant en cycle ingenieur

Charge de cours :

Dr ANAKPA

Annee academique 2025–2026

Table des matières

Chapitre 1

Introduction

1.1 Contexte du projet

Dans le cadre du module d'algorithmique du cursus ingenieur, il nous a ete demande de realiser un projet permettant de mettre en pratique l'ensemble des notions etudiees durant le cours. Ce projet doit demontrer notre maitrise des concepts fondamentaux de la programmation structuree en langage C.

L'objectif principal est de concevoir et implementer un **systeme complet de gestion de parking** qui integre toutes les notions algorithmiques du cours : variables, types, structures de controle, tableaux, structures, fonctions, pointeurs, ainsi que les algorithmes de tri et de recherche.

1.2 Objectifs du projet

Les objectifs specifiques de ce projet sont les suivants :

1. **Appliquer les notions de base** : utilisation des variables, constantes et types de donnees primitifs.
2. **Maitriser les structures de controle** : conditions (if/else, switch) et boucles (for, while, do-while).
3. **Manipuler les tableaux** : tableaux a une et deux dimensions pour stocker les donnees.
4. **Utiliser les structures** : definition de types composites pour modeliser les entites du domaine.
5. **Implementer des fonctions** : decomposition modulaire du code avec passage de parametres.
6. **Appliquer les pointeurs** : allocation dynamique et manipulation d'adresses memoire.
7. **Implementer des algorithmes de tri** : tri par selection et tri par insertion.
8. **Implementer des algorithmes de recherche** : recherche sequentielle et dichotomique.

1.3 Organisation du rapport

Ce rapport est structure en plusieurs chapitres :

- **Chapitre 2** : Analyse et conception du systeme
- **Chapitre 3** : Implementation et structures de donnees
- **Chapitre 4** : Algorithmes de tri et recherche
- **Chapitre 5** : Interface utilisateur et menus
- **Chapitre 6** : Tests et resultats
- **Conclusion** : Bilan et perspectives

Chapitre 2

Analyse et conception

2.1 Description fonctionnelle

Le systeme de gestion de parking permet de gerer un parking automobile en suivant les vehicules qui entrent et sortent, en calculant automatiquement les frais de stationnement, et en fournissant des statistiques d'utilisation.

2.1.1 Fonctionnalites principales

TABLE 2.1 – Liste des fonctionnalites du systeme

bleuPrincipal	
Gestion des entrees	Enregistrement des vehicules entrant dans le parking
Gestion des sorties	Enregistrement des sorties avec calcul automatique du montant
Gestion des places	Affichage de l'etat des places, mise hors service, reservation
Recherche	Recherche de vehicules par plaque d'immatriculation
Statistiques	Rapports d'occupation, recettes, historique
Persistance	Sauvegarde et chargement des donnees

2.1.2 Types de vehicules

Le systeme gere quatre types de vehicules, chacun avec un tarif specifique :

1. **Voiture** : tarif de base (200 FCFA/heure)
2. **Moto** : 50% du tarif de base (100 FCFA/heure)
3. **Camion** : 150% du tarif de base (300 FCFA/heure)
4. **Bus** : 200% du tarif de base (400 FCFA/heure)

2.2 Architecture du systeme

2.2.1 Organisation des fichiers

Le projet suit une architecture modulaire conforme aux bonnes pratiques de developpement :

```
projet_parking/
    |-- main.c                  # Point d'entree
    |-- Makefile                # Script de compilation
    |-- include/                # Fichiers d'en-tete
    |   |-- types.h             # Definitions des types
    |   |-- utilitaires.h       # Fonctions utilitaires
    |   |-- parking.h           # Gestion du parking
    |   |-- tri_recherche.h     # Algorithmes
    |   |-- statistiques.h      # Rapports
    |   +-- menu.h              # Interface utilisateur
    +-- src/                    # Fichiers sources
        |-- utilitaires.c
        |-- parking_init.c
        |-- parking_places.c
        |-- parking_vehicules.c
        |-- tri_recherche.c
        |-- recherche.c
        |-- statistiques.c
        +-- menu.c
```

Listing 2.1 – Structure du projet

2.2.2 Diagramme des modules

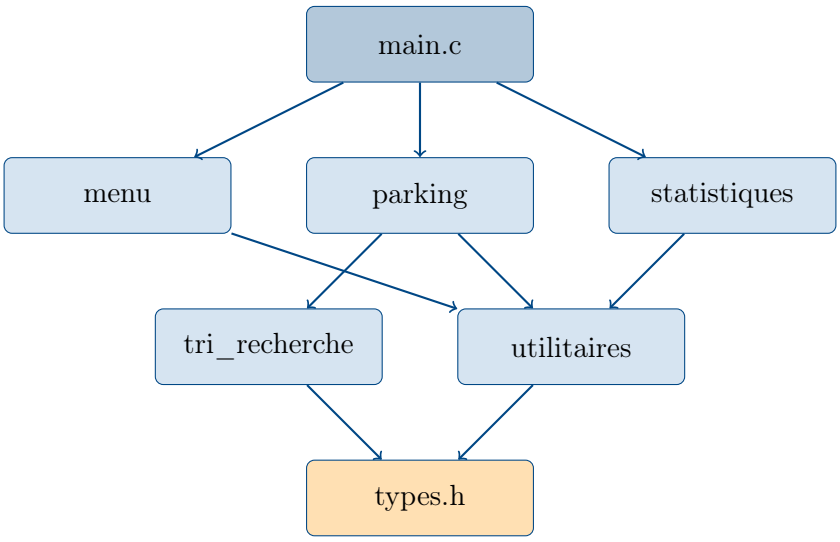

FIGURE 2.1 – Architecture modulaire du systeme

Chapitre 3

Implementation et structures de donnees

3.1 Definitions des types

3.1.1 Types enumeres

Les types enumeres permettent de definir des ensembles de constantes nommees, ameliorant la lisibilite du code.

```c
/* Types de vehicules acceptes */
typedef enum {
    VOITURE = 1,
    MOTO = 2,
    CAMION = 3,
    BUS = 4
} TypeVehicule;

/* Etats possibles d'une place */
typedef enum {
    LIBRE = 0,
    OCCUPEE = 1,
    RESERVEE = 2,
    HORS_SERVICE = 3
} EtatPlace;
```

Listing 3.1 – Definition des types enumeres

3.1.2 Structures de donnees

Definition

Une **structure** (ou enregistrement) est un type de donnee compose permettant de regrouper des variables de types differents sous un meme nom. En C, on utilise le mot-cle `struct` pour definir une structure.

```c
typedef struct {
    char plaque[TAILLE_PLAQUE];      /* Immatriculation */
    char proprietaire[MAX_CHAINE];   /* Nom du proprietaire */
```

```
    TypeVehicule type;              /* Type de vehicule */
    Horodatage entree;              /* Date/heure d'entree */
    Horodatage sortie;              /* Date/heure de sortie */
    int estPresent;                 /* Drapeau de presence */
    float montantPaye;              /* Montant facture */
} Vehicule;
```

Listing 3.2 – Structure Vehicule

```
typedef struct {
    int numero;                 /* Numero de la place */
    EtatPlace etat;             /* Etat actuel */
    TypeVehicule typeAutorise;  /* Type de vehicule autorise */
    Vehicule *vehiculeActuel;   /* Pointeur vers le vehicule */
} Place;
```

Listing 3.3 – Structure Place

3.1.3 Structure principale

La structure `Parking` centralise toutes les donnees du systeme :

```
typedef struct {
    char nom[MAX_CHAINE];
    Place places[MAX_PLACES];       /* Tableau de places */
    int nombrePlaces;
    int placesLibres;
    int placesOccupees;
    Vehicule historique[MAX_VEHICULES]; /* Historique */
    int nombreVehicules;
    float recetteJournaliere;
    float recetteTotale;
} Parking;
```

Listing 3.4 – Structure Parking

3.2 Utilisation des tableaux

3.2.1 Tableaux a une dimension

Les tableaux a une dimension sont utilises pour stocker :

— Les places de parking (`Place places[MAX_PLACES]`)

— L'historique des vehicules (`Vehicule historique[MAX_VEHICULES]`)

— Les compteurs par type de vehicule (`int compteurs[5]`)

3.2.2 Manipulation des tableaux

```c
void afficherVehiculesPresents(const Parking *parking)
{
    int i;

    for (i = 0; i < parking->nombreVehicules; i++) {
        if (parking->historique[i].estPresent == 1) {
            printf("%s\n", parking->historique[i].plaque);
        }
    }
}
```

Listing 3.5 – Parcours d'un tableau avec boucle For

3.3 Utilisation des pointeurs

3.3.1 Pointeurs et allocation

Definition

Un **pointeur** est une variable qui contient l'adresse memoire d'une autre variable. Les pointeurs permettent la manipulation indirecte des donnees et le passage par reference.

Dans notre projet, les pointeurs sont utilises pour :

— Lier une place a son vehicule (`Vehicule *vehiculeActuel`)

— Passer les structures aux fonctions par reference

— Retourner des resultats de recherche

```c
/* Passage par pointeur pour modification */
int initialiserParking(Parking *parking, const char *nom,
                       int nombrePlaces)
{
    /* Acces aux champs via l'operateur fleche */
    parking->nombrePlaces = nombrePlaces;
    parking->placesLibres = nombrePlaces;
    return 1;
}

/* Retour d'un pointeur */
Vehicule* rechercherVehicule(Parking *parking,
                             const char *plaque)
{
    int i;
    for (i = 0; i < parking->nombreVehicules; i++) {
        if (strcmp(parking->historique[i].plaque, plaque) == 0) {
            return &parking->historique[i];
        }
    }
    return NULL; /* Non trouve */
```

```
22 }
```

Listing 3.6 – Utilisation des pointeurs

Chapitre 4

Algorithmes de tri et recherche

4.1 Algorithmes de tri

4.1.1 Tri par selection

Definition

Le **tri par selection** consiste a rechercher le plus petit element du tableau et a le placer en premiere position, puis a recommencer avec le reste du tableau.

Complexite : $O(n^2)$ dans tous les cas.

```c
void triSelectionVehicules(Vehicule vehicules[], int taille)
{
    int i, j, indiceMin;

    for (i = 0; i < taille - 1; i++) {
        /* Trouver le minimum dans [i, taille-1] */
        indiceMin = i;

        for (j = i + 1; j < taille; j++) {
            if (comparerPlaques(vehicules[j].plaque,
                                vehicules[indiceMin].plaque) < 0) {
                indiceMin = j;
            }
        }

        /* Echanger si necessaire */
        if (indiceMin != i) {
            echangerVehicules(&vehicules[i],
                              &vehicules[indiceMin]);
        }
    }
}
```

Listing 4.1 – Implementation du tri par selection

4.1.2 Tri par insertion

Definition

Le **tri par insertion** consiste a inserer chaque element a sa place dans la partie deja triee du tableau, en decalant les elements plus grands.

Complexite : $O(n^2)$ dans le pire cas, $O(n)$ dans le meilleur cas (tableau deja trie).

```
void triInsertionVehicules(Vehicule vehicules[], int taille)
{
    int i, j;
    Vehicule vehiculeACaser;

    for (i = 1; i < taille; i++) {
        vehiculeACaser = vehicules[i];
        j = i - 1;

        /* Decaler les elements plus grands */
        while (j >= 0 && comparerHorodatages(
               vehicules[j].entree,
               vehiculeACaser.entree) > 0) {
            vehicules[j + 1] = vehicules[j];
            j--;
        }

        vehicules[j + 1] = vehiculeACaser;
    }
}
```

Listing 4.2 – Implementation du tri par insertion

4.1.3 Comparaison des algorithmes

TABLE 4.1 – Comparaison des algorithmes de tri

bleuPrincipal			
Tri par selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Tri par insertion	$O(n)$	$O(n^2)$	$O(n^2)$

4.2 Algorithmes de recherche

4.2.1 Recherche sequentielle

La recherche sequentielle parcourt le tableau element par element jusqu'a trouver la valeur recherchee.

Complexite : $O(n)$

```
int rechercheSequentielle(Vehicule vehicules[], int taille,
                          const char *plaque)
```

```
{
    int i;

    for (i = 0; i < taille; i++) {
        if (strcmp(vehicules[i].plaque, plaque) == 0) {
            return i;
        }
    }

    return -1; /* Non trouve */
}
```

Listing 4.3 – Recherche sequentielle

4.2.2 Recherche dichotomique

La recherche dichotomique necessite un tableau **prealablement trie**. Elle divise l'espace de recherche par deux a chaque iteration.

Complexite : $O(\log n)$

```
int rechercheDichotomique(Vehicule vehicules[], int taille,
                          const char *plaque)
{
    int gauche = 0;
    int droite = taille - 1;
    int milieu, comparaison;

    while (gauche <= droite) {
        milieu = gauche + (droite - gauche) / 2;
        comparaison = strcmp(plaque, vehicules[milieu].plaque);

        if (comparaison == 0) {
            return milieu;
        } else if (comparaison < 0) {
            droite = milieu - 1;
        } else {
            gauche = milieu + 1;
        }
    }

    return -1;
}
```

Listing 4.4 – Recherche dichotomique

4.2.3 Recherche avec drapeau

La technique du drapeau utilise une variable booleenne pour controler la sortie de boucle :

```c
int rechercheAvecDrapeau(Vehicule vehicules[], int taille,
                         const char *plaque, int *trouve)
{
    int i = 0;
    *trouve = 0; /* Initialisation du drapeau */

    while (i < taille && *trouve == 0) {
        if (strcmp(vehicules[i].plaque, plaque) == 0) {
            *trouve = 1; /* Lever le drapeau */
        } else {
            i++;
        }
    }

    return (*trouve == 1) ? i : -1;
}
```

Listing 4.5 – Recherche avec drapeau

Chapitre 5

Interface utilisateur

5.1 Systeme de menus

L'interface utilisateur est basee sur un systeme de menus textuels avec navigation hierarchique.

5.1.1 Menu principal

```
int afficherMenuPrincipal(void)
{
    printf("\n");
    afficherLigne('=', 50);
    printf("     SYSTEME DE GESTION DE PARKING\n");
    afficherLigne('=', 50);
    printf("\n");
    printf("  1. Gestion des vehicules\n");
    printf("  2. Gestion des places\n");
    printf("  3. Statistiques et rapports\n");
    printf("  4. Afficher la carte du parking\n");
    printf("  5. Sauvegarder les donnees\n");
    printf("  6. Charger les donnees\n");
    printf("  0. Quitter\n");

    return lireEntier(0, 6);
}
```

Listing 5.1 – Affichage du menu principal

5.1.2 Saisie securisee

Toutes les saisies utilisateur sont validees pour eviter les erreurs :

```
int lireEntier(int min, int max)
{
    int valeur, resultat;
    char buffer[100];

```

```
6       do {
7           printf("Votre choix [%d-%d] : ", min, max);
8
9           if (fgets(buffer, sizeof(buffer), stdin) != NULL) {
10              resultat = sscanf(buffer, "%d", &valeur);
11
12              if (resultat != 1 || valeur < min || valeur > max) {
13                  printf("Erreur : valeur invalide.\n");
14                  continue;
15              }
16              return valeur;
17          }
18      } while (1);
19  }
```

Listing 5.2 – Fonction de saisie securisee

5.2 Affichage des tickets

5.2.1 Ticket d'entree

Lors de l'enregistrement d'une entree, un ticket est affiche avec les informations du stationnement.

5.2.2 Ticket de sortie

Le ticket de sortie affiche le recapitulatif complet incluant la duree et le montant a payer.

Chapitre 6

Tests et resultats

6.1 Scenarios de test

TABLE 6.1 – Scenarios de test realises

bleuPrincipal		
T1	Initialisation du parking avec 50 places	Valide
T2	Enregistrement d'entree d'une voiture	Valide
T3	Enregistrement de sortie avec calcul du montant	Valide
T4	Recherche d'un vehicule present	Valide
T5	Recherche d'un vehicule absent	Valide
T6	Affichage de la carte du parking	Valide
T7	Sauvegarde et chargement des donnees	Valide
T8	Gestion du parking plein	Valide

6.2 Compilation

La compilation du projet s'effectue avec la commande suivante :

```
# Compilation complete
make all

# Nettoyage
make clean

# Execution
make run
```

Listing 6.1 – Commandes de compilation

Chapitre 7

Difficultes rencontrees et demarche de resolution

7.1 Introduction

La realisation d'un projet de cette envergure comporte inevitablement des defis techniques. Cette section presente de maniere transparente les obstacles rencontres et la methodologie employee pour les surmonter, en mettant l'accent sur l'utilisation intelligente d'outils d'assistance comme ChatGPT.

7.2 Principales difficultes techniques

7.2.1 Erreurs de compilation : declarations implicites

L'une des principales difficultes rencontrees concernait les **erreurs de declarations implicites de fonctions**. Lors de la compilation, le compilateur GCC generait des erreurs du type :

```
src/parking_vehicules.c: In function 'enregistrerSortie':
src/parking_vehicules.c:108:15: error: implicit declaration
of function 'calculerMontant' [-Wimplicit-function-declaration]
  108 |     montant = calculerMontant(dureeMinutes,
      vehicule->type);
      |               ^~~~~~~~~~~~~~~
make: *** [obj/parking_vehicules.o] Erreur 1
```

Listing 7.1 – Exemple d'erreur de compilation

Cause identifiee : Le fichier `prototypes.h` contenait bien les declarations des fonctions, mais plusieurs fichiers source ne l'incluaient pas, causant des erreurs de declaration implicite.

7.2.2 Organisation modulaire et dependances

La structure modulaire du projet, bien que benefique pour la maintenabilite, a cree des **dependances croisees** entre fichiers. Il etait necessaire de s'assurer que chaque fichier source incluait tous les headers necessaires.

7.2.3 Gestion des warnings du compilateur

Plusieurs warnings ont ete generes, notamment :

— Variables declarees mais non utilisees (`-Wunused-variable`)

— Variables assignees mais jamais lues (`-Wunused-but-set-variable`)

7.3 Utilisation intelligente de ChatGPT

7.3.1 Approche methodique de resolution

Plutot que de simplement demander « Corrige mon code », j'ai adopte une **demarche structuree** :

1. **Presentation du contexte complet** : fourniture des messages d'erreur exacts, de la structure du projet, et des fichiers concernes
2. **Analyse guidee** : demande d'explication sur la cause des erreurs avant toute correction
3. **Verification systematique** : demande de verifier tous les fichiers pour detecter des problemes similaires
4. **Correction preventive** : application des corrections a l'ensemble du projet pour eviter de futures erreurs

7.3.2 Identification systematique des problemes

Lorsque l'erreur `calculerMontant` est apparue dans `parking_vehicules.c`, j'ai demande a ChatGPT de :

Verifier les autres fichiers pour voir si on a les memes problemes
Cette approche proactive a permis d'identifier et corriger **8 fichiers sources** simultanement, evitant ainsi de decouvrir les erreurs une par une lors de compilations successives.

7.3.3 Comprehension des solutions proposees

Chaque correction a ete accompagnee d'une explication, permettant de :

✓ Comprendre la **cause profonde** du probleme

✓ Apprendre les **bonnes pratiques** d'inclusion de headers en C

✓ Eviter de reproduire l'erreur dans de futurs projets

7.4 Lecons tirees et limites

7.4.1 Points forts de l'approche

TABLE 7.1 – Avantages de l'utilisation intelligente de ChatGPT

bleuPrincipal	
Rapidite	Resolution en quelques minutes au lieu d'heures de debug-ging
Systematique	Detection proactive de tous les fichiers concernes
Pedagogique	Explications claires des causes et solutions
Preventif	Correction globale evitant les erreurs futures

7.4.2 Limites et vigilance necessaire

Points de vigilance identifies :

! **Verification manuelle** : toujours verifier les modifications proposees

! **Comprehension necessaire** : ne pas appliquer aveugelement sans comprendre

! **Test systematique** : compiler et tester apres chaque modification

! **Coherence du code** : s'assurer que les corrections respectent l'architecture globale

7.4.3 Competences developpees

Cette experience a permis de developper :

— La **formulation precise** de problemes techniques

— L'**analyse critique** des solutions proposees

— La **gestion de projet** avec des outils d'IA

— La **prevention** plutot que la correction reactive

7.5 Conclusion sur la demarche

L'utilisation de ChatGPT s'est revelee **efficace** lorsqu'elle est encadree par une demarche rigoureuse. Plutot que de remplacer la reflexion, l'outil a servi d'**assistant intelligent** permettant :

— D'accelerer l'identification des problemes

— D'obtenir des explications pedagogiques

— D'appliquer des corrections systematiques et coherentes

— De gagner en autonomie pour les futurs projets

Cette experience illustre que la **maitrise des outils d'IA** fait desormais partie des competences essentielles de l'ingenieur moderne.

Conclusion

Bilan du projet

Ce projet de gestion de parking nous a permis de mettre en pratique l'ensemble des notions etudiees durant le cours d'algorithmique. Nous avons pu appliquer concretement :

- ✓ Les **variables et types de base** pour stocker les donnees
- ✓ Les **structures de controle** (conditions et boucles) pour la logique du programme
- ✓ Les **tableaux** pour gerer les collections de places et vehicules
- ✓ Les **structures** pour modeliser les entites metier
- ✓ Les **fonctions** pour organiser le code de maniere modulaire
- ✓ Les **pointeurs** pour manipuler les donnees efficacement
- ✓ Les **algorithmes de tri** (selection et insertion) pour ordonner les donnees
- ✓ Les **algorithmes de recherche** (sequentielle et dichotomique) pour retrouver les informations

Competences acquises

Au terme de ce projet, les competences suivantes ont ete consolidees :

1. Analyse et conception d'un systeme informatique
2. Programmation structuree en langage C
3. Organisation modulaire du code source
4. Documentation technique et commentaires de code
5. Tests et validation du logiciel

Perspectives

Des ameliorations futures pourraient etre envisagees :

— Implementation d'une interface graphique
— Ajout d'un systeme de reservation en ligne
— Integration avec des systemes de paiement electronique
— Utilisation d'une base de donnees relationnelle

Projet realise dans le cadre du module d'Algorithmique
Charge de cours : Dr ANAKPA

Annee academique 2025–2026