DOCUMENTATION

Program de gestion des motos

Créé par

DAIBBAR MOHAMED & CHIGR SAAD

TABLEAU DE CONTENU

I. Introduction

Notre programme est composé de plusieurs fonctions, chacune ayant un rôle spécifique dans la gestion d'une base de données de motos et d'adhérents. Le code permet de gérer l'ajout, la modification, la suppression et l'emprunt de motos, ainsi que l'affichage des adhérents.

NOTE IMPORTANTE : L'utilisation du programme est entièrement automatique. Vous n'avez pas besoin d'ouvrir les fichiers du tout. Lorsque vous ajoutez une moto, elle est ajoutée automatiquement. Lorsque vous fermez le programme et que vous le réouvrez, la liste des motos ajoutées est affichée. De même, lorsqu'un client prend une moto, elle devient automatiquement indisponible. Ainsi, si quelqu'un d'autre souhaite la prendre, le programme indiquera que la moto n'est pas disponible jusqu'à ce que le premier client la ramène. Plus d'informations sont disponibles dans le manuel d'utilisation.

II. Fonctions du Programme

1. rendreMoteur: Permet à un utilisateur de retourner un moteur emprunté et met à jour le stock.
2. miseAJourAdherent: Met à jour les informations de location de moteur pour un adhérent donné.
3. comparerNom: Compare deux adhérents par nom pour le tri**.(qsort)**
4. methodePaiement: Gère le processus de paiement et l'emprunt de moteur pour un adhérent.
5.comparerNoms: Permet de trier les adherents par ordre alphabetique
6. afficherAdherents: Affiche une liste triée de tous les adhérents ayant emprunté des moteurs.
7.miseAJourFichierAdherents: Écrit la liste mise à jour des adhérents dans le fichier des adhérents.
8. afficherStockMoteurs: Affiche le stock actuel de moteurs avec leurs détails.
9. supprimerMotor: Supprime un moteur du stock en fonction du numéro de commande.
10. modifierMoteur: Permet de modifier les détails d'un moteur dans le stock.
11. rechercherMoteur: Recherche un moteur dans le stock par nom.
12. remplirFichier: Écrit le stock actuel de moteurs dans le fichier de stockage**.**
13. remplirStock: Charge le stock de moteurs depuis le fichier de stockage en mémoire.
14. ajouterMoteur: Ajoute un nouveau moteur au stock et met à jour le fichier de stockage.

Le programme commence par définir deux structures, **motors** et **adherents**, pour stocker les informations des moteurs et des adhérents respectivement. Ensuite, différentes fonctions sont définies pour gérer les opérations sur les motos et les adhérents. La fonction **main** contient une boucle principale qui affiche le menu et permet à l'utilisateur de sélectionner une opération à effectuer.

Les structures

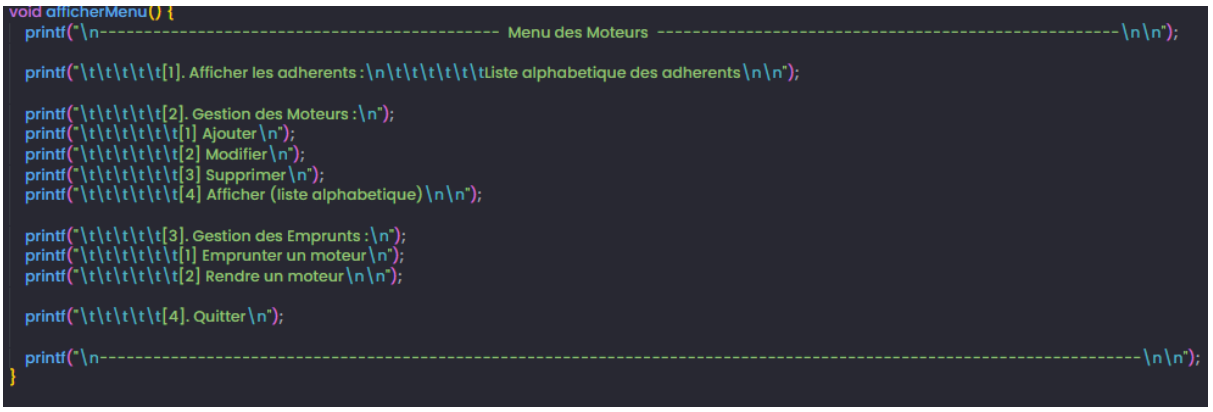
- **struct motors:** Définie pour stocker les informations sur les motos, comprenant le nom (*name*), le prix (*price*), et la disponibilité (*disponible*).

```
struct motors {
    char nom[100];
    int prix;
    int disponible;
};
```

- **struct adherents:** Définie pour stocker les informations sur les adhérents, comprenant le nom de l'adhérent (*name*) et le nom de la moto emprunté (*motorname*).

```
struct adherents {
    char nom[100];
    char nomMoteur[100];
};
```

1.Fonction afficherMenu:



```
void afficherMenu() {
    printf("\n---------------------------------------------- Menu des Moteurs ----------------------------------------------\n\n");

    printf("\t\t\t\t\t\t[1]. Afficher les adherents :\n\t\t\t\t\t\t\tListe alphabetique des adherents\n\n");

    printf("\t\t\t\t\t\t[2]. Gestion des Moteurs :\n");
    printf("\t\t\t\t\t\t\t[1] Ajouter\n");
    printf("\t\t\t\t\t\t\t[2] Modifier\n");
    printf("\t\t\t\t\t\t\t[3] Supprimer\n");
    printf("\t\t\t\t\t\t\t[4] Afficher (liste alphabetique)\n\n");

    printf("\t\t\t\t\t\t[3]. Gestion des Emprunts :\n");
    printf("\t\t\t\t\t\t\t[1] Emprunter un moteur\n");
    printf("\t\t\t\t\t\t\t[2] Rendre un moteur\n\n");

    printf("\t\t\t\t\t\t[4]. Quitter\n");

    printf("\n------------------------------------------------------------------------------------------------------------\n\n");
}
```

Affiche le menu principal avec les différentes options disponibles pour l'utilisateur.

2.Fonction Ajoutermoteur:

```
void ajouterMoteur(struct moteurs* moteur, int* pborne, int taille) {
    if (*pborne >= taille) {
        printf("Le stockage est plein!\n");
        return;
    }

    printf("Veuillez entrer le nom du nouveau moteur: ");
    fgets(moteur[*pborne].nom, 100, stdin);
    moteur[*pborne].nom[strcspn(moteur[*pborne].nom, "\n")] = '\0';

    printf("Veuillez entrer le prix du moteur: ");
    scanf("%d", &moteur[*pborne].prix);
    getchar();

    printf("Si le moteur est disponible, entrez 1, sinon entrez 0: ");
    scanf("%d", &moteur[*pborne].disponible);
    getchar();

    FILE* file = fopen("store.txt", "a");
    if (file == NULL) {
        perror("Erreur d'ouverture du fichier");
        exit(1);
    }
    fprintf(file, "%s %d %d\n", moteur[*pborne].nom, moteur[*pborne].prix, moteur[*pborne].disponible);
    fclose(file);

    (*pborne)++;

    FILE* fb = fopen("pborne.txt", "w");
    if (fb == NULL) {
        perror("Erreur d'ouverture de pborne.txt");
        exit(1);
    }
    fprintf(fb, "%d", *pborne);
    fclose(fb);
}
```

Ajoute un nouveau moteur au stock et met à jour le fichier de stockage.

- **Étapes:**
  -Vérifie si le stockage est plein.
  -Demande les détails de la nouvelle moto à l'utilisateur

  -Ajoute la moto au fichier <*store.txt*>.
  -Met à jour le compteur de motos et sauvegarde dans <*pborne.txt*>.

3.Fonction remplirstock:

```
void remplirStock(struct moteurs* moteur, int* pborne) {
    FILE* file = fopen("store.txt", "r");
    if (file == NULL) {
        perror("Erreur d'ouverture du fichier");
        exit(1);
    }

    for (int i = 0; i < *pborne; i++) {
        fscanf(file, "%s %d %d", moteur[i].nom, &moteur[i].prix, &moteur[i].disponible);
    }
    fclose(file);
}
```

Charger les informations des motors à partir du fichier <*store.txt*> dans le tableau de motos.

- **Entrées:**
  motor: Tableau de motos.
  pborne: Pointeur vers le nombre actuel de motos.
- **Étapes:**
  -Ouvre le fichier <*store.txt*> en lecture.
  -Charge les informations des motos dans le tableau **motor**.
  -Ferme le fichier.

4.Fonction remplirfichier:

```c
void remplirFichier(struct moteurs* moteur, int* pborne) {
    FILE* file = fopen("store.txt", "w");
    if (file == NULL) {
        perror("Erreur d'ouverture du fichier");
        exit(1);
    }

    for (int i = 0; i < *pborne; i++) {
        fprintf(file, "%s %d %d\n", moteur[i].nom, moteur[i].prix, moteur[i].disponible);
    }
    fclose(file);
}
```

Sauvegarder les informations actuelles des motos dans le fichier <*store.txt*>.

Entrées:

motor: Tableau de motos

pborne: Pointeur vers le nombre actuel de motos.

Étapes:

Étapes:-Ouvre le fichier <*store.txt*> en écriture.

-Écrit les informations des motos dans le fichier.

-Ferme le fichier.

5.Fonction rechercherMoteur:

Rechercher une moto par son nom.

```c
void rechercherMoteur(struct moteurs* moteur, int* pborne) {
    char nomMoteur[100];

    if (*pborne == 0) {
        printf("Le stock est vide!\n");
        return;
    }

    printf("Entrez le nom du moteur que vous voulez: ");
    fgets(nomMoteur, 100, stdin);
    nomMoteur[strcspn(nomMoteur, "\n")] = '\0';

    for (int i = 0; i < *pborne; i++) {
        if (strcmp(nomMoteur, moteur[i].nom) == 0) {
            printf("Nom du moteur: %s\n", moteur[i].nom);
            printf("Prix du moteur: %d\n", moteur[i].prix);
            printf("Numero de commande: %d\n", i);
            return;
        }
    }

    printf("Ce moteur n'est pas disponible. Veuillez verifier si le nom est correct.\n");
}
```

Entrées:

motor: Tableau de motos.

pborne: Pointeur vers le nombre actuel de motos.

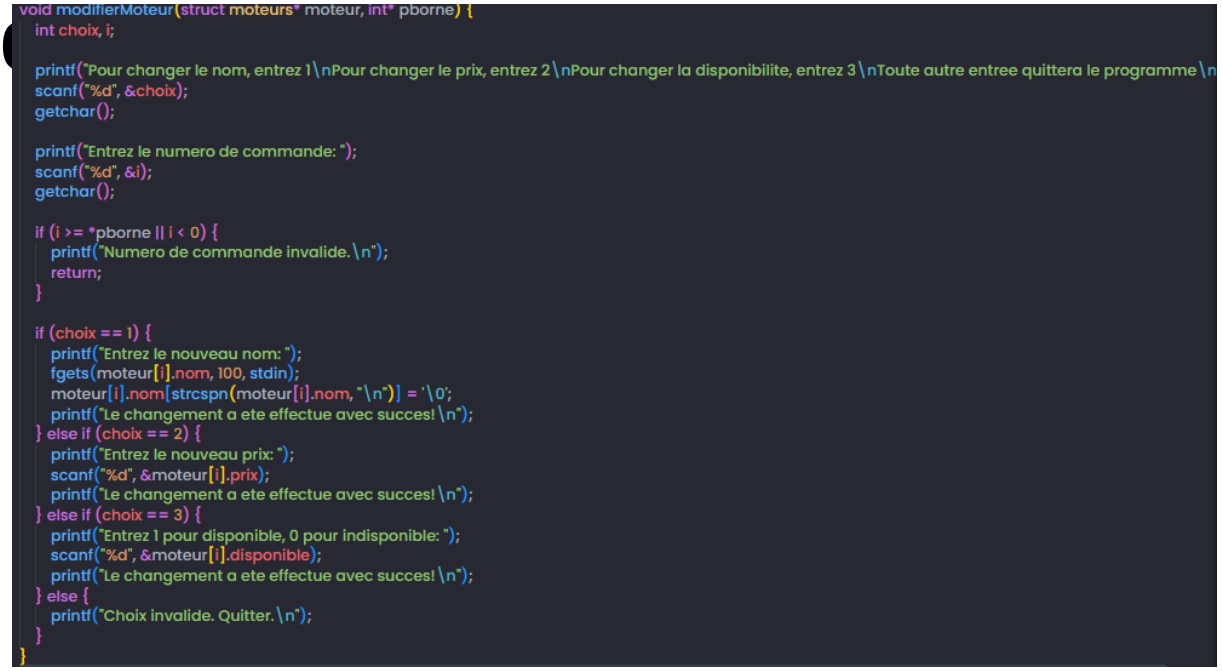
Étapes:

-Vérifie si le stock est vide.

-Demande le nom de la moto à rechercher.

-Parcourt le tableau pour trouver la moto correspondant et affiche ses détails.

-Informe l'utilisateur si la moto n'est pas trouvée.



```c
void modifierMoteur(struct moteurs* moteur, int* pborne) {
    int choix, i;

    printf("Pour changer le nom, entrez 1\nPour changer le prix, entrez 2\nPour changer la disponibilite, entrez 3\nToute autre entree quittera le programme\n
    scanf("%d", &choix);
    getchar();

    printf("Entrez le numero de commande: ");
    scanf("%d", &i);
    getchar();

    if (i >= *pborne || i < 0) {
        printf("Numero de commande invalide.\n");
        return;
    }

    if (choix == 1) {
        printf("Entrez le nouveau nom: ");
        fgets(moteur[i].nom, 100, stdin);
        moteur[i].nom[strcspn(moteur[i].nom, "\n")] = '\0';
        printf("Le changement a ete effectue avec succes!\n");
    } else if (choix == 2) {
        printf("Entrez le nouveau prix: ");
        scanf("%d", &moteur[i].prix);
        printf("Le changement a ete effectue avec succes!\n");
    } else if (choix == 3) {
        printf("Entrez 1 pour disponible, 0 pour indisponible: ");
        scanf("%d", &moteur[i].disponible);
        printf("Le changement a ete effectue avec succes!\n");
    } else {
        printf("Choix invalide. Quitter.\n");
    }
}
```

Modifier les détails d'une moto existante.(le fichier est aussi modifier automatiquement lorsque l'utilisateur ouvre le program la prochaine fois il va trouver les nouveaux informations bien modifier)

- **Entrées:**

  motor: Tableau de motos.

  pborne: Pointeur vers le nombre actuel de motos.

- **Étapes:**

  -Recherche une moto par son nom.

  -Demande le type de modification (nom, prix, disponibilité).

-Applique la modification et affiche un message de confirmation.

7.Fonction supprimerMoteur:

Supprimer une moto de la base de données.

```
void supprimerMoteur(struct moteurs* moteur, int* pborne) {
  int commande;
  printf("Entrez le numero de commande du moteur a supprimer: ");
  scanf("%d", &commande);
  getchar();

  if (commande >= *pborne || commande < 0) {
    printf("Numero de commande invalide.\n");
    return;
  }

  for (int i = commande; i < *pborne - 1; i++) {
    moteur[i] = moteur[i + 1];
  }
  (*pborne)--;

  remplirFichier(moteur, pborne);
  FILE* fb = fopen("pborne.txt", "w");
  if (fb == NULL) {
    perror("Erreur d'ouverture de pborne.txt");
    exit(1);
  }
  fprintf(fb, "%d", *pborne);
  fclose(fb);

  printf("Moteur supprime avec succes.\n");
}

Codiumate: Options | Test this function
void afficherStockMoteurs(struct moteurs* moteur, int* pborne) {
  FILE* file = fopen("store.txt", "r");
  if (file == NULL) {
    perror("Erreur d'ouverture du fichier");
    return;
  }
```

- **Entrées:**motor: Tableau de motos.**pborne:** Pointeur vers le nombre actuel de motos.
- **Étapes:**
  -Recherche la moto à supprimer.
  -Demande le numéro d'ordre de la moto à supprimer.

-Supprime la moto du tableau et décale les entrées restantes.
-Met à jour le fichier de stockage et le compteur de motos.

8.Fonction afficherstockoteurs:

Afficher toutes les motos en stock.

```
void afficherStockMoteurs(struct moteurs* moteur, int* pborne) {
    FILE* file = fopen("store.txt", "r");
    if (file == NULL) {
        perror("Erreur d'ouverture du fichier");
        return;
    }

    for (int i = 0; i < *pborne; i++) {
        fscanf(file, "%s %d %d", moteur[i].nom, &moteur[i].prix, &moteur[i].disponible);
        printf("Moteur %d: %s,    Prix: %d\n", i + 1, moteur[i].nom, moteur[i].prix);
        printf("\n-------------------------------------------------------------\n");
    }
    fclose(file);
}
```

- **Entrées:**
  motor: Tableau de motos.
  pborne: Pointeur vers le nombre actuel de motos.
- **Étapes:**
  -Ouvre le fichier <*store.txt*> en lecture.
  -Parcourt le fichier et affiche les détails de chaque moto.
  -Ferme le fichier.

```c
void methodePaiement(struct moteurs* moteur, struct adherents* adherent, int* pborne, int* borno) {
    int commandeMoteur, prix;

    printf("Si vous voulez le prendre, veuillez entrer le numero de commande pour verifier: ");
    scanf("%d", &commandeMoteur);
    getchar();

    if (commandeMoteur >= *pborne || commandeMoteur < 0) {
        printf("Numero de commande invalide.\n");
        return;
    }

    if (moteur[commandeMoteur].disponible == 1) {
        printf("Le prix est: %d\n", moteur[commandeMoteur].prix);
        printf("Veuillez payer le prix: ");
        scanf("%d", &prix);
        getchar();

        printf("Veuillez entrer votre nom: ");
        fgets(adherent->nom, 100, stdin);
        adherent->nom[strcspn(adherent->nom, "\n")] = '\0';

        FILE* fm = fopen("borno.txt", "w+");
        (*borno)++;
        fprintf(fm, "%d", *borno);
        fclose(fm);

        FILE* f = fopen("baseAdherents.txt", "a");
        if (f == NULL) {
            perror("Erreur d'ouverture de baseAdherents.txt");
            exit(1);
        }
        fprintf(f, "%s %s\n", adherent->nom, moteur[commandeMoteur].nom);
        fclose(f);

        moteur[commandeMoteur].disponible = 0;
        remplirFichier(moteur, pborne);

        printf("Merci pour votre paiement!\n");
    } else {
        printf("Ce moteur n'est pas disponible.\n");
    }
```

- **Entrées:**

  motor: Tableau de motos.
  adherent: Structure des adhérents.
  pborne: Pointeur vers le nombre actuel de motos.
- **Étapes:**

  -Demande le numéro d'ordre de la moto à emprunter.
  -Vérifie la disponibilité de la moto.

-Demande le paiement et les détails de l'adhérent.-Enregistre l'emprunt dans le fichier <*adherentbase.txt*>.-Met à jour la disponibilité de la moto.

10.Fonction comparerNoms:

Cette fonction est utilisée pour trier la liste des adhérents par ordre alphabétique

```c
int comparerNoms(const void *a, const void *b) {
    struct adherents *adherent1 = (struct adherents *)a;
    struct adherents *adherent2 = (struct adherents *)b;
    return strcmp(adherent1->nom, adherent2->nom);
}
```

11.Fonction afficherAdherents:

```c
void afficherAdherents(int borno) {
    struct adherents adherento[borno];
    FILE* ff = fopen("baseAdherents.txt", "r");
    if (ff == NULL) {
        perror("Erreur d'ouverture de baseAdherents.txt");
        return;
    }
    for(int i=0; i<borno; i++){
        fscanf(ff,"%s %s",adherento[i].nom,adherento[i].nomMoteur);
    }
    qsort(adherento,borno,sizeof(struct adherents),comparerNoms);

    for(int i =0 ; i<borno; i++){
        printf("\t\t\t\t\t (char [122])"----------
        printf("----------\n"
    }
    fclose(ff);
}
```

Affiche une liste triée de tous les adhérents ayant emprunté des moteurs.

- **But:** Afficher la liste des adhérents et les moteurs qu'ils ont empruntés.
- **Entrées:** Aucune (utilise le fichier <*adherentbase.txt*>).
- **Étapes:**
  -Ouvre le fichier <*adherentbase.txt*> en lecture.

-Parcourt le fichier et affiche les informations de chaque adhérent.
-Ferme le fichier

12.Fonction miseAJourAdherent:

```
void miseAJourAdherent(char* nomAdherent) {
    struct adherents listeAdherents[100];
    int count = 0;

    FILE* file = fopen("baseAdherents.txt", "r");
    if (file == NULL) {
        perror("Erreur d'ouverture de baseAdherents.txt");
        return;
    }

    while (fscanf(file, "%s %s", listeAdherents[count].nom, listeAdherents[count].no
        if (strcmp(listeAdherents[count].nom, nomAdherent) == 0) {
            strcpy(listeAdherents[count].nomMoteur, "None");
        }
        count++;
    }
    fclose(file);

    miseAJourFichierAdherents(listeAdherents, count);
}
```

Met à jour le fichier *<adherentbase.txt>* avec les données actuelles des adhérents. Elle renvoie les données de tous les adhérents vers le fichier.

13.FonctionmiseAJourFichierAdherents:

Met à jour les informations d'un adhérent spécifique.
Permet à l'utilisateur de modifier les détails d'un adhérent (notamment lors du retour d'un moteur)

```
void miseAJourFichierAdherents(struct adherents* listeAdherents, int count) {
    FILE* file = fopen("baseAdherents.txt", "w");
    if (file == NULL) {
        perror("Erreur d'ouverture de baseAdherents.txt");
        return;
    }

    for (int i = 0; i < count; i++) {
        fprintf(file, "%s %s\n", listeAdherents[i].nom, listeAdherents[i].nomMoteur);
    }
    fclose(file);
}
```

14.

Fonction Rendremotor:

Gère le retour d'une moto emprunté par un adhérent. Met à jour le statut de la moto et les détails de l'adhérent en mémoire et dans les fichiers.

```c
void rendreMoteur(struct moteurs* moteur, int* pborne) {
    char nomAdherent[100];

    printf("Entrez votre nom: ");
    fgets(nomAdherent, 100, stdin);
    nomAdherent[strcspn(nomAdherent, "\n")] = '\0';

    miseAJourAdherent(nomAdherent);

    for (int i = 0; i < int moteurs::disponible
        if (moteur[i].disponible == 0) {
            moteur[i].disponible = 1;
            printf("Le moteur a ete retourne avec succes.\n");
            remplirFichier(moteur, pborne);
            return;
        }
    }
    printf("Moteur non trouve.\n");
}
```

15.Fonction main:

Point d'entrée principal du programme
gère l'exécution et les interactions avec l'utilisateur.

Étapes:

-Initialise les variable
s et les structures.
-Charge le nombre de motos à
partir du fichier
<pborne.txt>.
Affiche le menu et entre dans une boucle
infinie pour traiter les choix de l'utilisateur.

-Exécute la fonction en fonction
du choix de
l'utilisateur.

III. Manuel du programme

Utilisation du menu :

Lorsque vous ajoutez une moto à notre système, elle est automatiquement enregistrée. Vous n'avez pas à vous soucier de sauvegarder ou de manipuler des fichiers. Le programme le gère pour vous, de manière transparente et sans tracas.
De même, lorsque vous fermez le programme et que vous le rouvrez plus tard, vous verrez automatiquement la nouvelle liste de motos ajoutées. Pas besoin de rechercher ou d'ouvrir des fichiers. Tout est là, prêt à être consulté.
De plus, lorsque vous ou un client prenez une moto, le programme met automatiquement à jour sa disponibilité. Si la moto est déjà empruntée, le programme vous le dira immédiatement. Vous n'avez pas à vous soucier de gérer manuellement les disponibilités ou de vérifier les listes.
Et si vous supprimez une moto, le programme veillera à ce qu'elle soit supprimée de manière transparente de tous les enregistrements. Vous n'avez pas à vous soucier de supprimer manuellement des fichiers ou de nettoyer quoi que ce soit.

- Le menu principal s'affiche avec plusieurs options :
  Afficher les adhérents
  Gestion des Motors
  Gestion des emprunts
  Quitter le programme
  Gestion des Motos:
- **Ajouter une moto :** Fournir le nom, le prix et la disponibilité.
- **Modifier une moto :** Choisir le moteur par son numéro d'ordre et modifier ses attributs.
- **Supprimer une moto :** Choisir la moto par son numéro d'ordre et le supprimer.
- **Afficher toutes les motos :** Affiche toutes les motos disponibles avec leurs détails.
  Gestion des emprunts:
- **Emprunter une moto :** Rechercher une moto, vérifier sa disponibilité, payer et enregistrer l'adhérent.
- **Rendre une moto :** Fournir le nom de l'adhérent et mettre à jour le statut de la moto.
  Lorsque l'utilisateur entre une commande ou modifie les informations d'une moto ou d'un adhérent le programme modifie cette information automatiquement en cours d'exécution.
  Les données sont sauvegardées automatiquement par le programme donc la modification via le fichier n'est pas nécessaire.
  Toute modification sera sauvegardée même après la fermeture du programme.

IV. Bilan qualitatif du travail

Ce projet était une opportunité pour nous instruire et nous approfondir dans le domaine de la programmation et du langage C. C'était aussi une opportunité de faire parler notre imagination. Il nous a aussi appris à résoudre des problèmes complexes et à réfléchir de manière logique.
On a rencontrés plusieurs problèmes lors du travail de celui-ci:
-l'organisation
-la fonction supprimermotor() qui a été tres complexe a realiser
-l'organisation de la fonction main()
-debug de programme apres la creation des fonctions