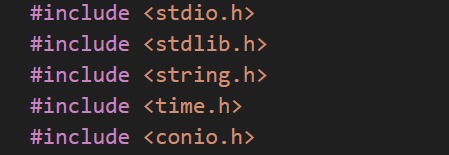


Plan :

1. **Les bibliothèques**
2. **Les structures**
3. **Les prototypes**
4. **Les fonctions**
5. **Les bibliothèques :**

****

**1. \*\*stdio.h\*\* :**

**Cette bibliothèque fournit les fonctions standards d'entrée/sortie, telles que `printf()` et `scanf()`, permettant d'afficher des messages à l'utilisateur et de récupérer des données à partir de l'entrée standard (le clavier par défaut).**

**2. \*\*stdlib.h\*\* :**

**Elle fournit des fonctions utilitaires telles que l'allocation dynamique de mémoire (`malloc()`, `calloc()`, `realloc()`) et la gestion de processus (`exit()`), entre autres. Elle est nécessaire pour l'utilisation de fonctions de gestion de la mémoire et d'autres fonctions standard de la bibliothèque C.**

**3. \*\*string.h\*\* :**

**Cette bibliothèque fournit des fonctions pour manipuler les chaînes de caractères en C, telles que `strcpy()`, `strcat()`, `strcmp()`, etc. Elle est utilisée pour effectuer des opérations de manipulation de chaînes de caractères, telles que la copie, la concaténation et la comparaison.**

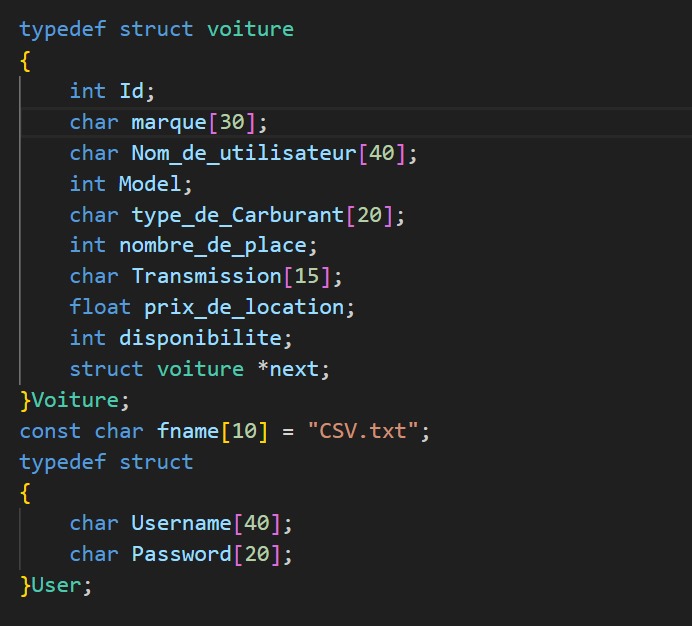
**4. \*\*time.h\*\* :**

**Cette bibliothèque fournit des fonctions pour manipuler le temps et la date en C, telles que `time()`, `strftime()`, `localtime()`, etc. Elle est utilisée pour obtenir des informations sur l'heure actuelle, la date et pour effectuer des opérations de chronométrage dans le programme.**

**5. \*\*conio.h\*\* :**

**Cette bibliothèque fournit des fonctions pour l'entrée/sortie de console dans les environnements DOS/Windows. Bien que largement utilisée dans le passé, elle n'est pas standard et n'est pas disponible sur tous les systèmes. Les fonctions les plus couramment utilisées de cette bibliothèque sont `getch()` et `clrscr()`, qui permettent respectivement de lire un caractère sans écho et de vider l'écran de la console.**

1. **Les structures :**

****

**1. \*\*Voiture\*\* :**

**Cette structure représente une voiture et contient les informations suivantes :**

**- `Id` : Identifiant de la voiture.**

**- `marque` : Marque de la voiture.**

**- `Nom\_de\_utilisateur` : Nom de l'utilisateur associé à la voiture.**

**- `Model` : Modèle de la voiture.**

**- `type\_de\_Carburant` : Type de carburant utilisé par la voiture.**

**- `nombre\_de\_place` : Nombre de places dans la voiture.**

**- `Transmission` : Type de transmission de la voiture.**

**- `prix\_de\_location` : Prix de location par jour de la voiture.**

**- `disponibilite` : Indicateur de disponibilité de la voiture (1 pour disponible, 0 pour non disponible).**

**- `next` : Pointeur vers la prochaine voiture dans une liste chaînée.**

**2. \*\*User\*\* :**

**Cette structure représente un utilisateur et contient les informations suivantes :**

**- `Username` : Nom d'utilisateur.**

**- `Password` : Mot de passe.**

**.** const char fname[10] = "CSV.txt";

**Cette constante est utilisée pour stocker le nom du fichier dans lequel les données des voitures sont enregistrées.**

1. **Les prototypes :**

****

**1. \*\*AppMenu(char[])\*\* :**

**Cette fonction gère le menu principal de l'application. Elle prend en paramètre le nom d'utilisateur pour personnaliser l'affichage du menu. Elle permet à l'utilisateur d'interagir avec les fonctionnalités de l'application telles que l'ajout, la modification, la suppression, l'affichage et la recherche des voitures.**

**2. \*\*Login(User\*)\*\* :**

**Cette fonction permet à un utilisateur de se connecter à l'application en vérifiant ses informations d'identification (nom d'utilisateur et mot de passe) par rapport à une liste d'utilisateurs enregistrés. Elle renvoie une valeur indiquant si l'authentification a réussi ou non.**

**3. \*\*InterfaceApp()\*\* :**

**Cette fonction affiche l'écran d'accueil de l'application, invitant l'utilisateur à entrer ses informations d'identification pour se connecter.**

**4. \*\*delay(int)\*\* :**

**Cette fonction met en pause l'exécution du programme pendant un certain nombre de secondes, ce qui peut être utilisé pour créer des délais ou des pauses dans l'exécution du programme.**

**5. \*\*UploadFileToList()\*\* :**

**Cette fonction lit les données des voitures à partir d'un fichier et les charge dans une liste chaînée de structures de voiture. Elle renvoie un pointeur vers le premier élément de cette liste.**

**6. \*\*AjoutVoiture(Voiture\*, Voiture\*)\*\* :**

**Cette fonction ajoute une nouvelle voiture à une liste existante de voitures. Elle prend en paramètre un pointeur vers la liste existante et un pointeur vers la nouvelle voiture à ajouter.**

**7. \*\*affiche(Voiture\*)\*\* :**

**Cette fonction affiche les détails de toutes les voitures présentes dans une liste chaînée de voitures.**

**8. \*\*UploadListToFile(Voiture\*)\*\* :**

**Cette fonction écrit les données des voitures à partir d'une liste chaînée dans un fichier.**

**9. \*\*CheckMatr(int , Voiture\*)\*\* :**

**Cette fonction vérifie si un certain matricule existe déjà dans une liste de voitures. Elle prend en paramètre le matricule à vérifier et un pointeur vers la liste de voitures.**

**10. \*\*ModifyMenu()\*\* :**

**Cette fonction affiche un menu permettant à l'utilisateur de choisir le type de modification à effectuer sur une voiture (par exemple, par matricule).**

**11. \*\*main()\*\* :**

**La fonction principale du programme, qui est le point d'entrée de l'application. C'est là que le programme commence son exécution.**

**12. \*\*SuppMenu()\*\* :**

**Cette fonction affiche un menu permettant à l'utilisateur de choisir le type de suppression à effectuer sur une voiture (par exemple, par matricule).**

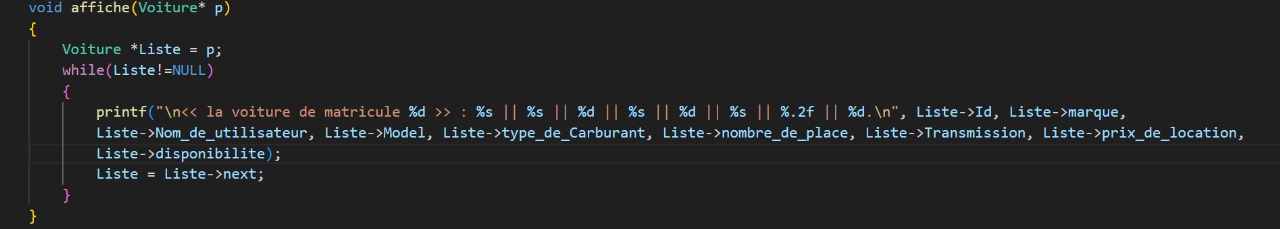
**13. \*\*SupprimerParMatricule(Voiture\*, int)\*\* :**

**Cette fonction supprime une voiture de la liste en utilisant son matricule comme référence.**

**14. \*\*RechercheMenu()\*\* :**

**Cette fonction affiche un menu permettant à l'utilisateur de choisir le type de recherche à effectuer sur les voitures (par exemple, par disponibilité ou par marque).**

1. **Les fonctions :**
2. **Fonction affiche() :**

****

* **La fonction `affiche(Voiture\* p)` prend en paramètre un pointeur vers la première voiture d'une liste chaînée de voitures (`p`). Elle affiche ensuite les détails de chaque voiture présente dans la liste.**

**Voici comment la fonction fonctionne :**

**1. Elle initialise un pointeur `Liste` avec la même valeur que le pointeur `p`. Cela permet de parcourir la liste tout en conservant une référence à son premier élément.**

**2. Elle commence une boucle `while` qui se poursuit tant que le pointeur `Liste` ne pointe pas vers NULL, ce qui signifie qu'il y a encore des éléments à afficher dans la liste.**

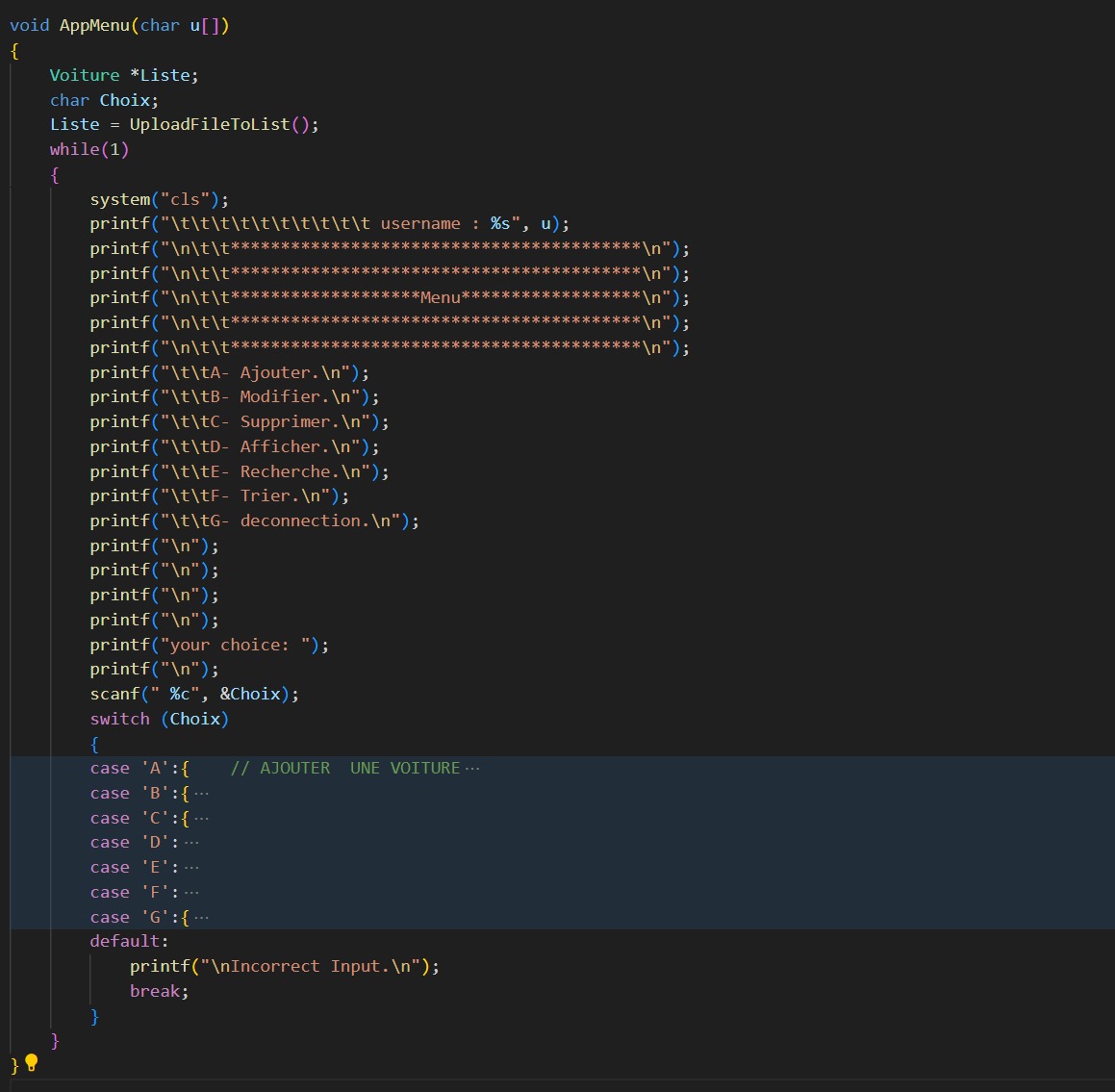
**3. À chaque itération de la boucle, la fonction utilise `printf()` pour afficher les détails de la voiture actuelle, y compris son matricule, sa marque, le nom de l'utilisateur, le modèle, le type de carburant, le nombre de places, le type de transmission, le prix de location et sa disponibilité.**

**4. Après avoir affiché les détails de la voiture actuelle, le pointeur `Liste` est mis à jour pour pointer vers la voiture suivante dans la liste, en utilisant le champ `next` de la structure de voiture.**

**5. La boucle se poursuit jusqu'à ce que toutes les voitures de la liste aient été affichées.**

**En résumé, la fonction `affiche()` est responsable de l'affichage des détails de toutes les voitures présentes dans une liste chaînée de voitures.**

1. **Fonction AppMenu() :**

****

**La fonction `AppMenu(char u[])` est une fonction qui affiche le menu principal de l'application et gère les choix de l'utilisateur. Voici un aperçu de ce que fait cette fonction :**

**1. Elle prend en paramètre le nom d'utilisateur (`u[]`) pour personnaliser l'affichage du menu.**

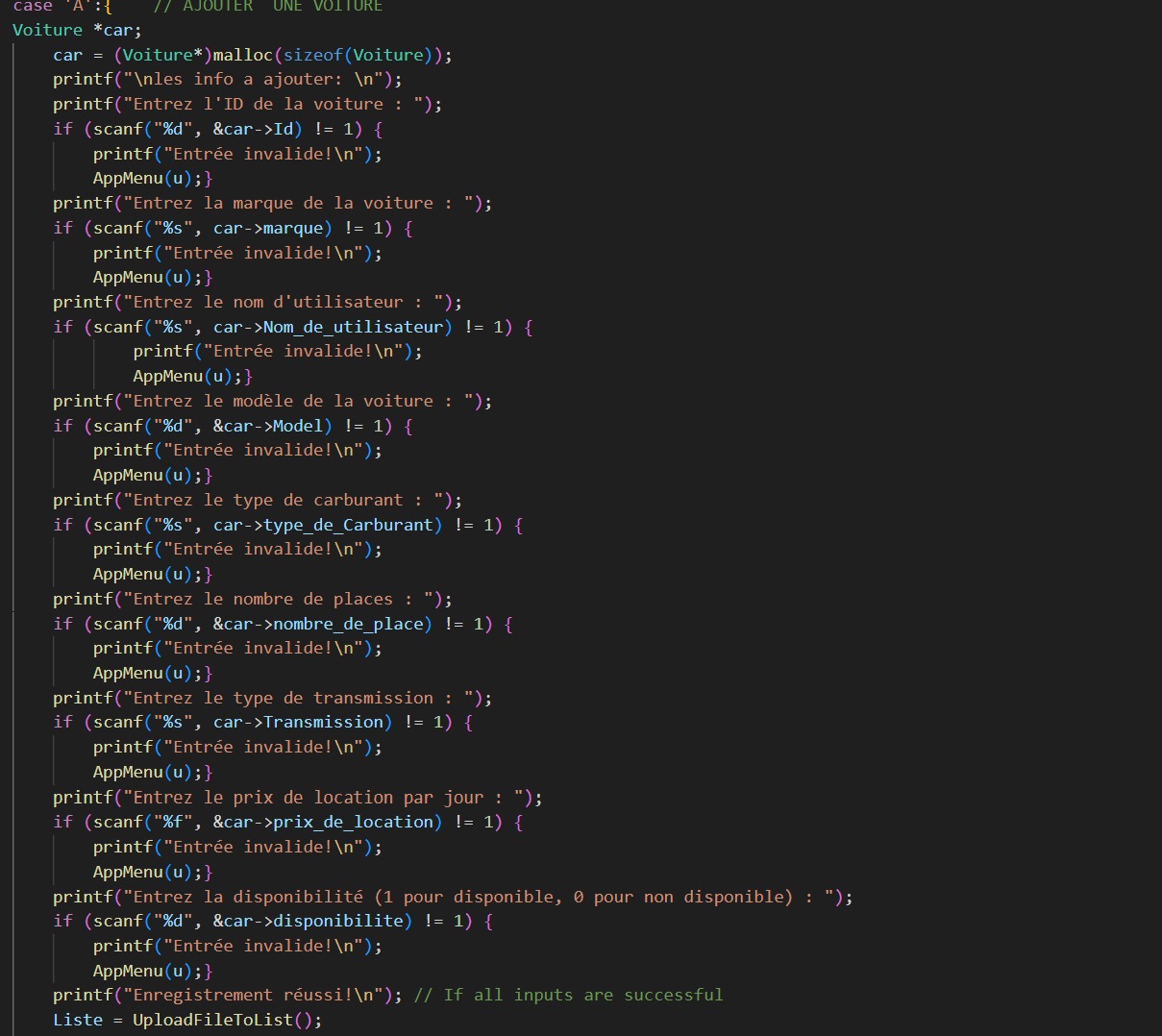
**2. Elle commence par charger les données des voitures à partir d'un fichier en appelant la fonction `UploadFileToList()` et stocke la liste dans une variable `Liste`.**

**3. Ensuite, elle entre dans une boucle `while(1)` qui se poursuit indéfiniment jusqu'à ce que l'utilisateur choisisse de se déconnecter.**

**4. À chaque itération de la boucle, elle efface l'écran de la console en utilisant `system("cls")`, puis affiche le menu principal de l'application avec différentes options telles que l'ajout, la modification, la suppression, l'affichage, la recherche, le tri et la déconnexion.**

**5. Elle attend que l'utilisateur saisisse son choix à l'aide de `scanf()`, stocke ce choix dans la variable `Choix`, puis exécute le code correspondant à l'option choisie par l'utilisateur à l'aide d'une structure de contrôle `switch`.**

**Case A :**

****

**1. `case 'A':` : C'est la partie du switch qui est activée lorsque l'utilisateur choisit d'ajouter une voiture (option A dans le menu principal).**

**2. `{` : Début du bloc de code exécuté lorsque l'option A est choisie.**

**3. `Voiture \*car;` : Déclare un pointeur vers une structure de voiture, qui sera utilisé pour stocker les informations de la nouvelle voiture.**

**4. `car = (Voiture\*)malloc(sizeof(Voiture));` : Alloue de la mémoire pour la structure de voiture à l'aide de `malloc`.**

**5. Le bloc de code entre les `printf` demande à l'utilisateur de saisir les informations de la nouvelle voiture, comme l'ID, la marque, le modèle, etc. Si l'utilisateur entre des données incorrectes, il est invité à réessayer.**

**6. `Liste = UploadFileToList();` : Charge la liste actuelle de voitures à partir du fichier.**

**7. `while(CheckMatr(car->Id, Liste)) { ... }` : Vérifie si l'ID de la voiture entré par l'utilisateur existe déjà dans la liste. Si oui, l'utilisateur est invité à entrer un nouvel ID.**

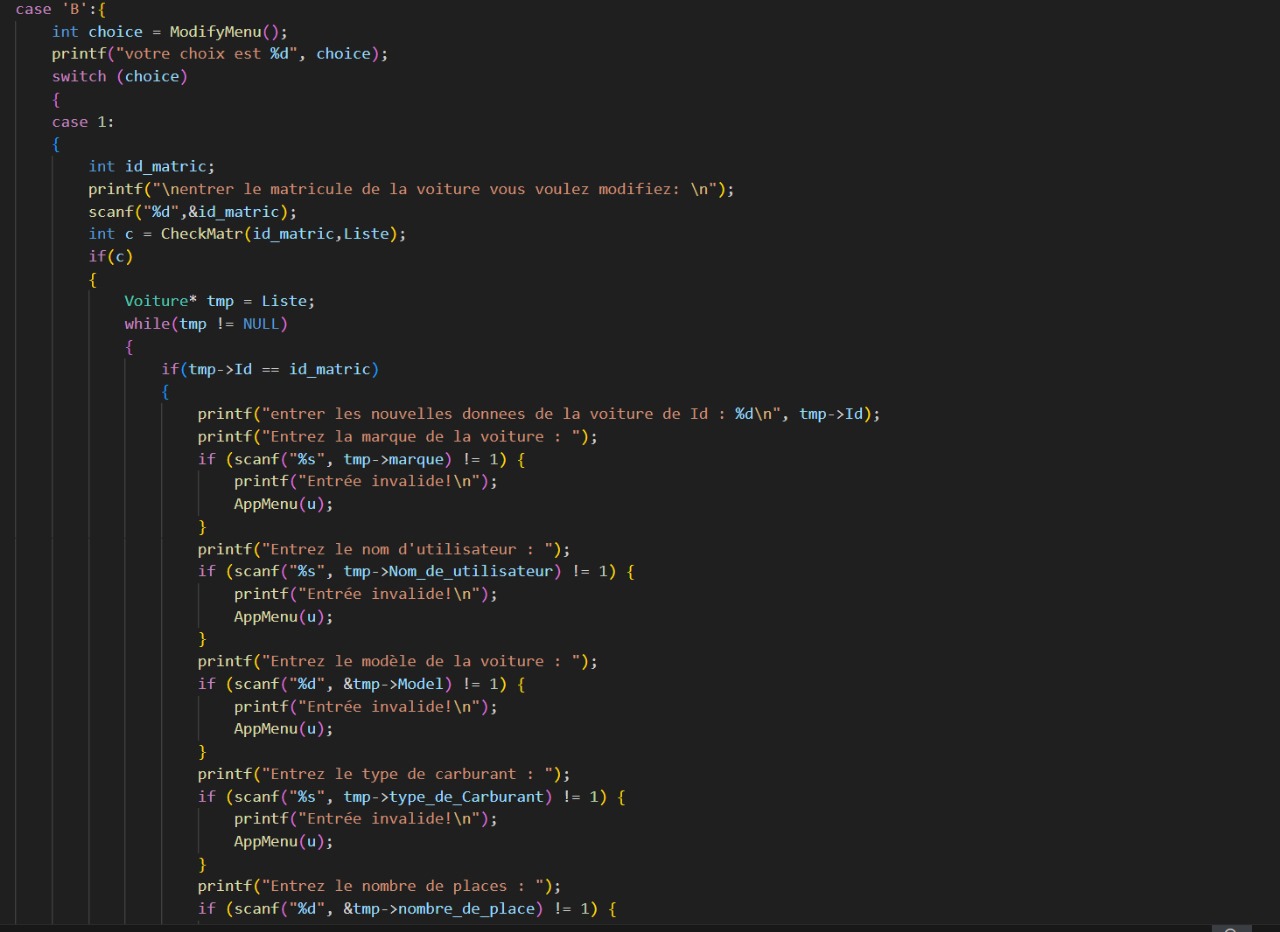
**8. Après que l'utilisateur ait entré toutes les informations correctement et qu'aucun ID en double n'a été détecté, les détails de la voiture sont affichés à l'écran pour confirmation.**

**9. `Liste = AjoutVoiture(Liste, car);` : Ajoute la nouvelle voiture à la liste existante.**

**10. `UploadListToFile(Liste);` : Enregistre la liste mise à jour dans le fichier.**

**11. `break;` : Termine l'exécution du bloc switch.**

**Case B :**

****

**1. `case 'B':` : C'est la partie du switch qui est activée lorsque l'utilisateur choisit de modifier une voiture existante (option B dans le menu principal).**

**2. `{` : Début du bloc de code exécuté lorsque l'option B est choisie.**

**3. `int choice = ModifyMenu();` : Appelle la fonction `ModifyMenu()` pour afficher un sous-menu qui permet à l'utilisateur de choisir comment il souhaite effectuer la modification (par matricule, par exemple).**

**4. `switch (choice)` : Démarre un autre switch en fonction du choix de l'utilisateur dans le sous-menu de modification.**

**5. `case 1:` : Si l'utilisateur choisit de modifier par matricule, ce bloc de code est exécuté.**

**6. `int id\_matric;` : Déclare une variable pour stocker le matricule de la voiture que l'utilisateur souhaite modifier.**

**7. `scanf("%d",&id\_matric);` : Demande à l'utilisateur d'entrer le matricule de la voiture à modifier.**

**8. `int c = CheckMatr(id\_matric,Liste);` : Vérifie si le matricule entré par l'utilisateur existe dans la liste de voitures.**

**9. Si le matricule existe (`c` est vrai), l'utilisateur est invité à entrer les nouvelles informations pour la voiture sélectionnée.**

**10. Les nouvelles informations sont lues à l'aide de `scanf()` et stockées dans les champs appropriés de la structure de voiture.**

**11. Une fois que les nouvelles informations ont été saisies, la liste mise à jour est enregistrée dans le fichier à l'aide de `UploadListToFile(Liste)`.**

**12. Si le matricule n'existe pas dans la liste, un message est affiché pour informer l'utilisateur.**

**13. `break;` : Termine l'exécution du bloc switch interne.**

**14. `break;` : Termine l'exécution du bloc switch externe.**

**Case C  :**

**Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement**

**1. `case 'C':` : C'est la partie du switch qui est activée lorsque l'utilisateur choisit de supprimer une voiture (option C dans le menu principal).**

**2. `{` : Début du bloc de code exécuté lorsque l'option C est choisie.**

**3. `int choice\_delete = SuppMenu();` : Appelle la fonction `SuppMenu()` pour afficher un sous-menu permettant à l'utilisateur de choisir comment il souhaite effectuer la suppression (par matricule, par exemple).**

**4. `switch (choice\_delete)` : Démarre un autre switch en fonction du choix de l'utilisateur dans le sous-menu de suppression.**

**5. `case 1:` : Si l'utilisateur choisit de supprimer par matricule, ce bloc de code est exécuté.**

**6. `Voiture \*sup1 = Liste, \*sup2;` : Déclare deux pointeurs `sup1` et `sup2` pour parcourir la liste de voitures à supprimer.**

**7. `int matri\_to\_delete;` : Déclare une variable pour stocker le matricule de la voiture à supprimer.**

**8. `scanf("%d", &matri\_to\_delete);` : Demande à l'utilisateur d'entrer le matricule de la voiture à supprimer.**

**9. `if(!CheckMatr(matri\_to\_delete,Liste))` : Vérifie si le matricule entré par l'utilisateur n'existe pas dans la liste de voitures. Si tel est le cas, un message est affiché pour informer l'utilisateur et le programme est mis en pause.**

**10. Sinon, si le matricule existe dans la liste, une boucle `while` est utilisée pour parcourir la liste de voitures à la recherche du matricule à supprimer.**

**11. Une fois que le matricule à supprimer est trouvé, les pointeurs sont ajustés pour retirer la voiture de la liste, puis la mémoire allouée dynamiquement est libérée à l'aide de `free()`.**

**12. La liste mise à jour est ensuite enregistrée dans le fichier à l'aide de `UploadListToFile(Liste)`.**

**13. `break;` : Termine l'exécution du bloc switch interne.**

**14. `break;` : Termine l'exécution du bloc switch externe.**

**Case D :**

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

**1. `case 'D':` : C'est la partie du switch qui est activée lorsque l'utilisateur choisit d'afficher les informations sur les voitures (option D dans le menu principal).**

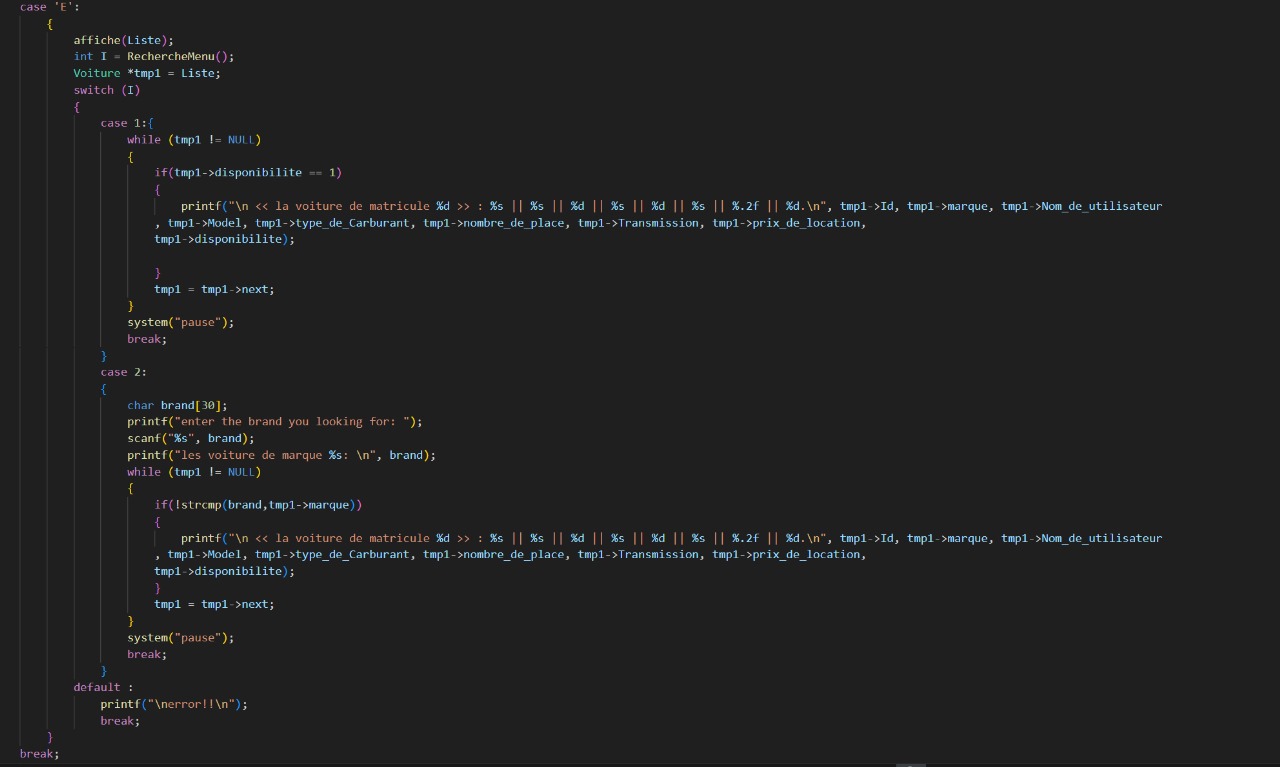
**2. `{` : Début du bloc de code exécuté lorsque l'option D est choisie.**

**3. `affiche(Liste);` : Appelle la fonction `affiche()` pour afficher les informations sur toutes les voitures dans la liste. La liste des voitures est transmise à cette fonction en tant que paramètre.**

**4. `system("pause");` : Met en pause l'exécution du programme jusqu'à ce que l'utilisateur appuie sur une touche. Cela permet à l'utilisateur de consulter les informations affichées avant que le programme ne revienne au menu principal.**

**5. `break;` : Termine l'exécution du bloc switch, ce qui signifie que l'exécution du programme passe au code après le switch dans la fonction `AppMenu()`.**

**Case E :**

****

**1. `case 'E':` : C'est la partie du switch qui est activée lorsque l'utilisateur choisit l'option "E" dans le menu principal, pour effectuer une recherche de voitures.**

**2. `{` : Début du bloc de code exécuté lorsque l'option E est choisie.**

**3. `affiche(Liste);` : Affiche toutes les voitures disponibles en appelant la fonction `affiche()` avec la liste des voitures en tant que paramètre.**

**4. `int I = RechercheMenu();` : Appelle la fonction `RechercheMenu()` pour obtenir le choix de recherche de l'utilisateur. Ce choix est stocké dans la variable `I`.**

**5. `Voiture \*tmp1 = Liste;` : Initialise un pointeur temporaire vers la liste des voitures.**

**6. `switch (I)` : Démarre une instruction switch pour traiter le choix de recherche de l'utilisateur.**

**7. `case 1:` : Si l'utilisateur choisit de rechercher les voitures disponibles.**

**- `while (tmp1 != NULL)` : Parcourt toutes les voitures dans la liste.**

**- `if(tmp1->disponibilite == 1)` : Vérifie si la voiture est disponible (disponibilité égale à 1).**

**- Affiche les informations de la voiture si elle est disponible.**

**8. `case 2:` : Si l'utilisateur choisit de rechercher les voitures par marque.**

**- `char brand[30];` : Déclare un tableau de caractères pour stocker la marque saisie par l'utilisateur.**

**- `printf("enter the brand you looking for: ");` : Invite l'utilisateur à saisir la marque recherchée.**

**- `scanf("%s", brand);` : Lit la marque saisie par l'utilisateur.**

**- Affiche toutes les voitures ayant la marque saisie par l'utilisateur.**

**9. `default:` : Si le choix de recherche de l'utilisateur n'est ni 1 ni 2, affiche un message d'erreur.**

**10. `break;` : Termine l'exécution du bloc switch.**

**11. `break;` : Termine l'exécution du bloc switch externe, ce qui signifie que l'exécution du programme passe au code après le switch dans la fonction `AppMenu()`.**

**Case F :**

**Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement**

**1. `case 'F':` : C'est la partie du switch qui est activée lorsque l'utilisateur choisit l'option "F" dans le menu principal, pour trier les voitures.**

**2. `{` : Début du bloc de code exécuté lorsque l'option F est choisie.**

**3. `int c;` : Déclare une variable entière pour stocker le choix de tri de l'utilisateur.**

**4. `printf("\nsort by:\n");` : Affiche un menu demandant à l'utilisateur de choisir le critère de tri.**

**5. `printf("1- price.\n");` : Affiche l'option de tri par prix.**

**6. `printf("2- brand. \n");` : Affiche l'option de tri par marque.**

**7. `printf("your choice: ");` : Invite l'utilisateur à saisir son choix de tri.**

**8. `scanf("%d", &c);` : Lit le choix de tri saisi par l'utilisateur.**

**9. `switch (c)` : Démarre une instruction switch pour traiter le choix de tri de l'utilisateur.**

**10. `case 1:` : Si l'utilisateur choisit de trier par prix.**

**- `SortListByPrice(Liste);` : Trie la liste des voitures par prix.**

**- `affiche(Liste);` : Affiche la liste triée.**

**- `UploadListToFile(Liste);` : Met à jour le fichier avec la liste triée.**

**- `system("pause");` : Attend une action de l'utilisateur avant de continuer.**

**11. `case 2:` : Si l'utilisateur choisit de trier par marque.**

**- `SortListByBrand(Liste);` : Trie la liste des voitures par marque.**

**- `affiche(Liste);` : Affiche la liste triée.**

**- `UploadListToFile(Liste);` : Met à jour le fichier avec la liste triée.**

**- `system("pause");` : Attend une action de l'utilisateur avant de continuer.**

**12. `break;` : Termine l'exécution du bloc switch.**

**13. `break;` : Termine l'exécution du bloc switch externe, ce qui signifie que l'exécution du programme passe au code après le switch dans la fonction `AppMenu()`.**

**Case G :**

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

**1. `case 'G':` : C'est la partie du switch qui est activée lorsque l'utilisateur choisit l'option "G" dans le menu principal, pour se déconnecter.**

**2. `{` : Début du bloc de code exécuté lorsque l'option G est choisie.**

**3. `system("cls");` : Efface l'écran pour une meilleure présentation.**

**4. `printf("\n\*\n");` : Affiche une ligne d'étoiles pour séparer le contenu.**

**5. `printf("\n\*disconnected\n");` : Affiche un message indiquant que l'utilisateur s'est déconnecté.**

**6. `printf("\n\*\*Succesfully\n");` : Affiche un message indiquant que la déconnexion a réussi.**

**7. `printf("\n\*\n");` : Affiche une autre ligne d'étoiles pour séparer le contenu.**

**8. `system("pause");` : Attend une action de l'utilisateur avant de continuer.**

**9. `system("cls");` : Efface à nouveau l'écran après que l'utilisateur a appuyé sur une touche pour revenir à l'écran de connexion.**

**10. `main();` : Appelle la fonction `main()` pour redémarrer l'application.**

**11. `default:` : Si l'entrée de l'utilisateur ne correspond à aucun des cas précédents, ce bloc est exécuté.**

**- `printf("\nIncorrect Input.\n");` : Affiche un message indiquant que l'entrée de l'utilisateur est incorrecte.**

**12. `break;` : Termine l'exécution du bloc switch.**

1. **Fonction delay :**

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

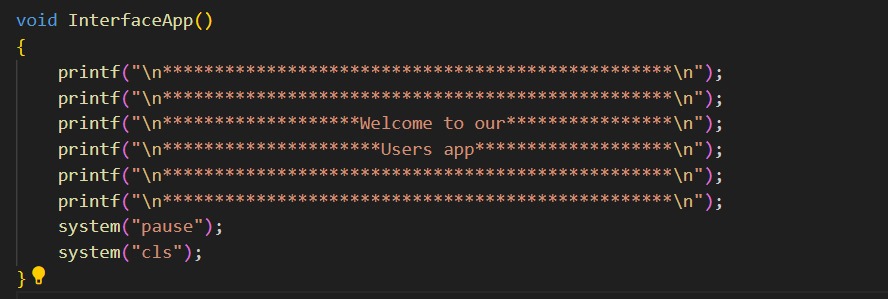
**1. `void delay(int number\_of\_seconds)` : C'est la signature de la fonction. Elle prend en paramètre le nombre de secondes pendant lesquelles le programme doit être mis en pause.**

**2. `int milli\_seconds = 1000 \* number\_of\_seconds;` : Convertit le nombre de secondes en millisecondes, car la fonction `clock()` mesure le temps en millisecondes.**

**3. `clock\_t start\_time = clock();` : Stocke l'heure de début de la pause en utilisant la fonction `clock()`, qui renvoie le nombre de millisecondes écoulées depuis le début de l'exécution du programme.**

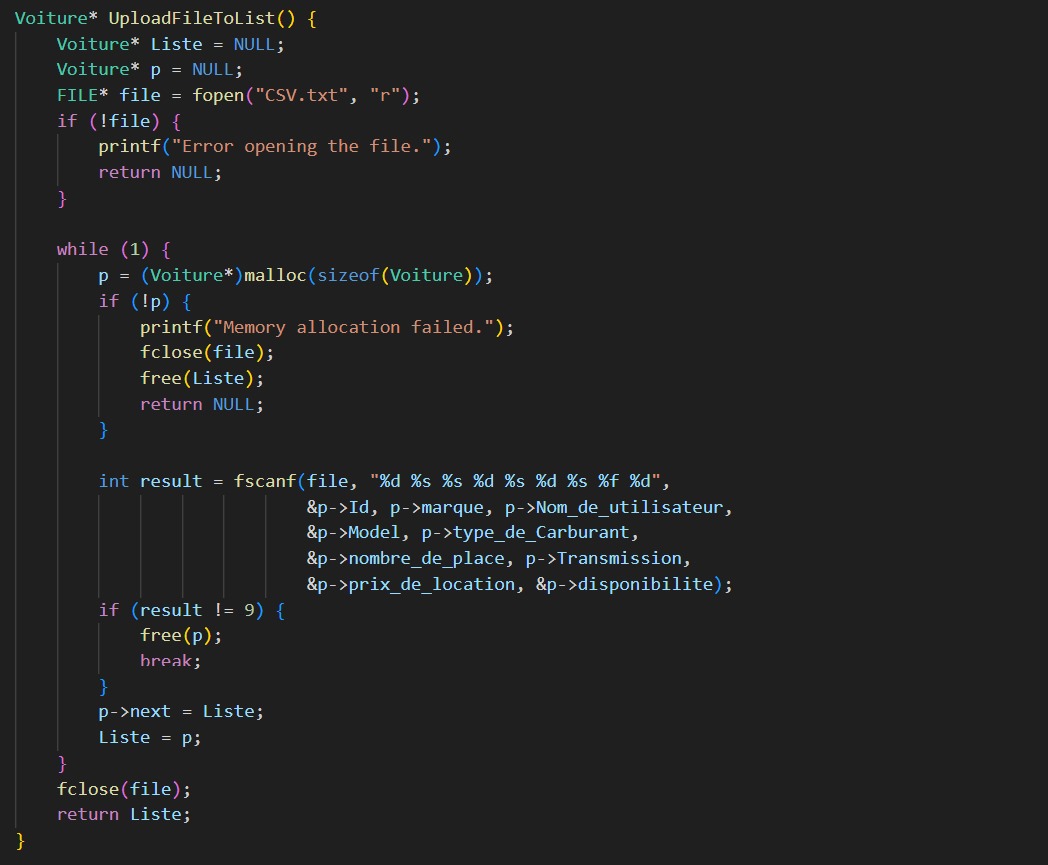
**4. `while (clock() < start\_time + milli\_seconds);` : C'est une boucle qui continue à s'exécuter jusqu'à ce que le temps écoulé depuis le début de la pause soit égal au nombre de millisecondes spécifié par `milli\_seconds`. Cela crée l'effet de pause.**

1. **Fonction InterfaceApp :**

****

**Cette fonction, nommée `InterfaceApp`, affiche un message de bienvenue à l'utilisateur, puis met en pause l'exécution du programme en attendant que l'utilisateur appuie sur une touche pour continuer. Ensuite, elle efface l'écran en exécutant la commande `system("cls")`, ce qui nettoie la console. Son objectif est de fournir une interface utilisateur conviviale au démarrage de l'application.**

1. **Fonction UploadFileToList :**

****

**Cette fonction, lit les données des voitures à partir de fichier texte "CSV.txt" et les stocke dans une liste chaînée de structures `Voiture`. Voici ce qu'elle fait :**

**1. Elle initialise un pointeur `Liste` à `NULL`, qui servira à pointer vers la tête de la liste chaînée.**

**2. Elle ouvre le fichier "CSV.txt" en mode lecture.**

**3. Si l'ouverture du fichier échoue, elle affiche un message d'erreur et retourne la liste vide.**

**4. À l'intérieur d'une boucle infinie, elle alloue dynamiquement de la mémoire pour une nouvelle structure `Voiture` avec `malloc`.**

**5. Elle lit les données de la voiture à partir du fichier à l'aide de `fscanf`, en supposant que chaque ligne du fichier contient les informations suivantes, séparées par des espaces :**

**- ID**

**- Marque**

**- Nom d'utilisateur**

**- Modèle**

**- Type de carburant**

**- Nombre de places**

**- Transmission**

**- Prix de location**

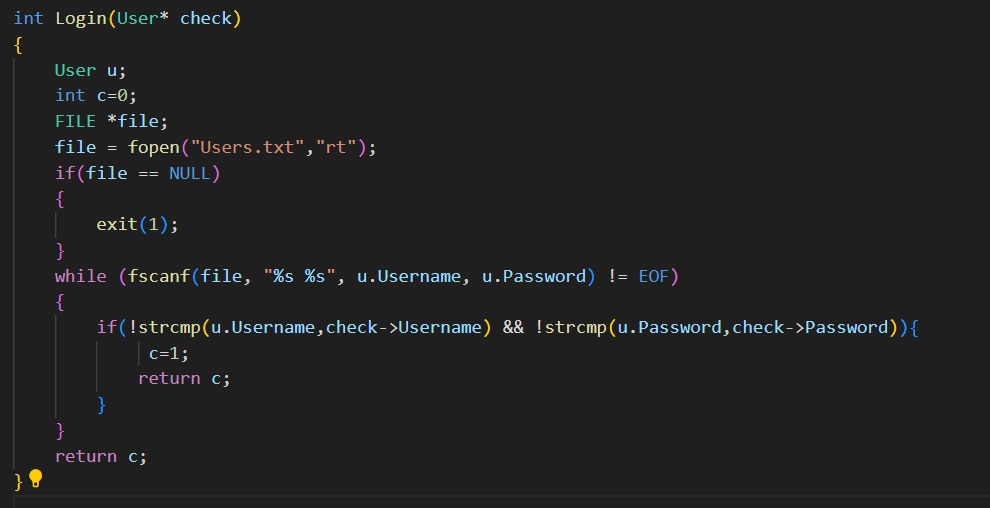
**- Disponibilité**

**6. Si le nombre d'éléments lus n'est pas égal à 9, cela signifie que la fin du fichier a été atteinte ou une erreur s'est produite lors de la lecture. Dans ce cas, elle libère la mémoire allouée pour la voiture en utilisant `free` et sort de la boucle.**

**7. Sinon, elle ajoute la voiture à la tête de la liste chaînée.**

**8. Une fois la lecture terminée, elle ferme le fichier et retourne la liste chaînée des voitures lues à partir du fichier.**

1. **Fonction Login :**

****

**Cette fonction `Login` permet de vérifier les informations d'identification d'un utilisateur. Voici comment elle fonctionne :**

**1. Elle ouvre le fichier "Users.txt" en mode lecture (`"rt"` signifie "read text").**

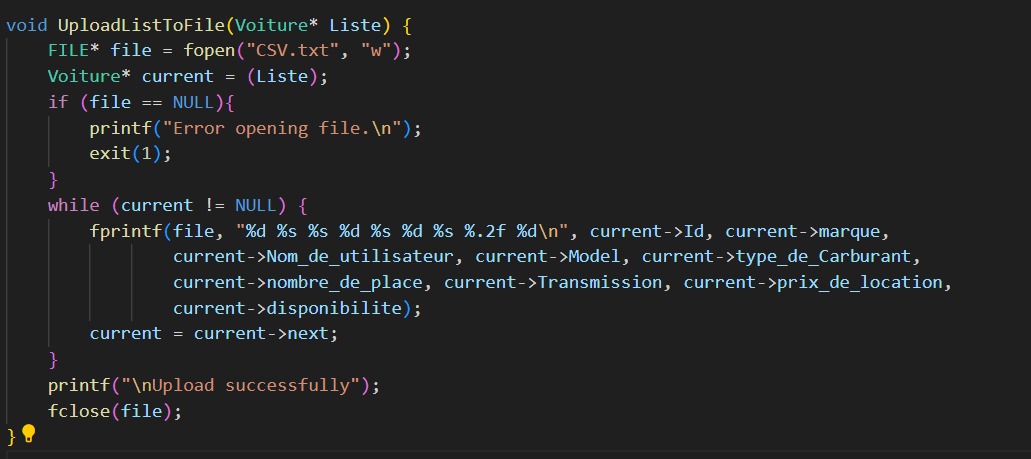
**2. Si l'ouverture du fichier échoue (c'est-à-dire si `file` est `NULL`), elle quitte le programme en appelant `exit(1)` pour indiquer une erreur.**

**3. Elle lit les informations d'identification utilisateur (nom d'utilisateur et mot de passe) à partir du fichier à l'aide de `fscanf` dans une boucle `while`.**

**4. À chaque itération, elle compare les informations d'identification lues avec celles fournies en argument (`check`). Si les informations d'identification correspondent, elle met `c` à 1 et retourne immédiatement, indiquant que l'authentification est réussie.**

**5. Si la fin du fichier est atteinte (c'est-à-dire que `fscanf` retourne `EOF`, End-Of-File), cela signifie que les informations d'identification fournies ne correspondent à aucun utilisateur dans le fichier. Dans ce cas, la fonction retourne 0 pour indiquer que l'authentification a échoué.**

1. **Fonction UploadListToFile :**



**Cette fonction `UploadListToFile` enregistre la liste des voitures dans un fichier texte "CSV.txt". Voici comment elle fonctionne :**

**1. Elle ouvre le fichier "CSV.txt" en mode écriture (`"w"`) pour effacer le contenu existant et écrire de nouvelles données. Si l'ouverture du fichier échoue, elle affiche un message d'erreur et quitte le programme en appelant `exit(1)`.**

**2. Elle parcourt la liste des voitures à l'aide d'une boucle `while`.**

**3. À chaque itération, elle écrit les informations de la voiture actuelle dans le fichier sous forme de ligne CSV (Comma-Separated Values) en utilisant `fprintf`.**

**4. Une fois toutes les voitures écrites dans le fichier, elle affiche un message pour indiquer que l'enregistrement a réussi.**

**5. Enfin, elle ferme le fichier avec `fclose`.**

1. **Fonction ModifyMenu :**

****

**La fonction `ModifyMenu` affiche un menu permettant à l'utilisateur de choisir la méthode de modification des données d'une voiture. Voici comment elle fonctionne :**

**1. Elle efface l'écran à l'aide de la fonction `system("cls")` pour une meilleure présentation du menu.**

**2. Ensuite, elle affiche le menu avec différentes options de modification :**

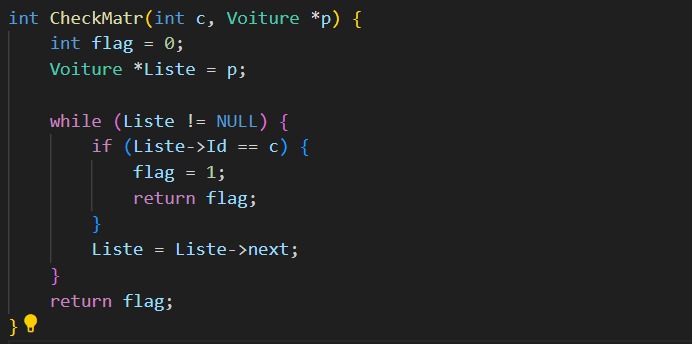
**- Modifier par matricule.**

**- Retourner au menu précédent.**

**3. Elle demande à l'utilisateur de saisir son choix à l'aide de `scanf`.**

**4. Elle retourne le choix de l'utilisateur, qui sera utilisé par la suite pour diriger le programme vers la fonction de modification appropriée.**

1. **Fonction CheckMatr :**

****

**La fonction `CheckMatr` vérifie si un matricule donné existe déjà dans la liste des voitures. Voici comment elle fonctionne :**

**1. Elle prend en entrée un entier `c` représentant le matricule à vérifier et un pointeur `p` vers la liste des voitures.**

**2. Elle initialise un drapeau `flag` à 0 pour indiquer que le matricule n'a pas été trouvé.**

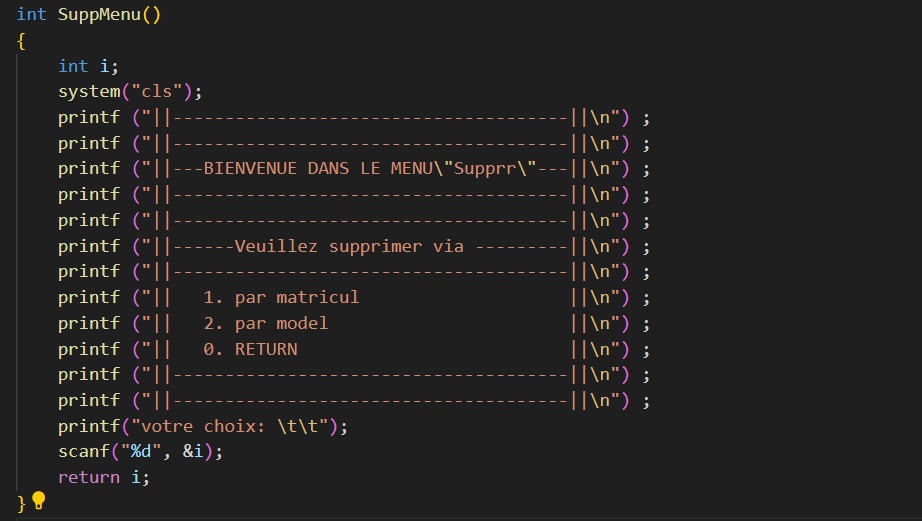
**3. Elle parcourt la liste des voitures à l'aide d'une boucle `while`.**

**4. À chaque itération, elle vérifie si le matricule de la voiture actuelle correspond à celui recherché (`c`).**

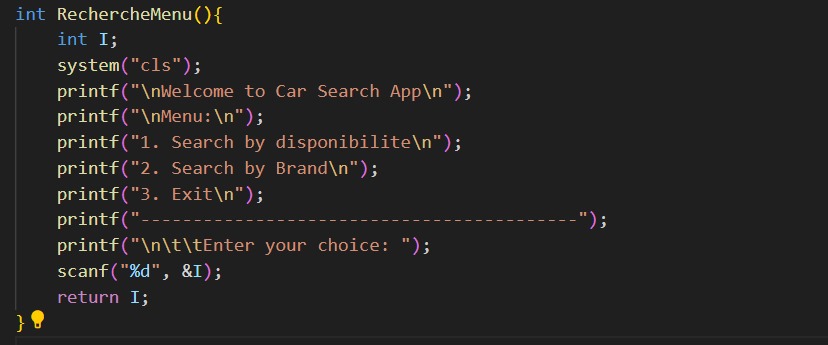
**5. Si elle trouve une correspondance, elle met à jour le drapeau `flag` à 1 et retourne immédiatement le drapeau, indiquant ainsi que le matricule existe.**

**6. Si elle termine la boucle sans trouver de correspondance, elle renvoie le drapeau `flag` inchangé, indiquant que le matricule n'existe pas dans la liste.**

1. **Fonction SuppMenu :**

****

1. **Fonction RechercheMenu :**

****

**La fonction `RechercheMenu` affiche un menu permettant à l'utilisateur de choisir le critère de recherche pour les voitures. Voici comment elle fonctionne :**

**1. Elle ne prend aucun paramètre en entrée.**

**2. Elle utilise la fonction `system("cls")` pour effacer l'écran de la console et afficher un menu clair à l'utilisateur.**

**3. Elle affiche les options de recherche disponibles, à savoir la recherche par disponibilité et la recherche par marque, ainsi qu'une option pour quitter.**

**4. Elle demande à l'utilisateur de saisir son choix à l'aide de `scanf`.**

**5. Elle retourne l'entier représentant le choix de l'utilisateur.**

1. **Fonction SortListByPrice :**

****

**La fonction `SortListByPrice` trie une liste de voitures en fonction du prix de location croissant. Voici comment elle fonctionne :**

**1. Elle prend en entrée un pointeur vers le début de la liste `Liste`.**

**2. Elle commence par initialiser une variable `swapped` à 0. Cette variable sera utilisée pour déterminer si des échanges ont été effectués pendant le parcours de la liste.**

**3. Elle utilise deux pointeurs, `ptr1` et `lptr`. `ptr1` est utilisé pour parcourir la liste et `lptr` pointe vers le dernier élément de la partie de la liste déjà triée.**

**4. Elle commence une boucle `do-while` pour parcourir la liste et effectuer les échanges si nécessaire.**

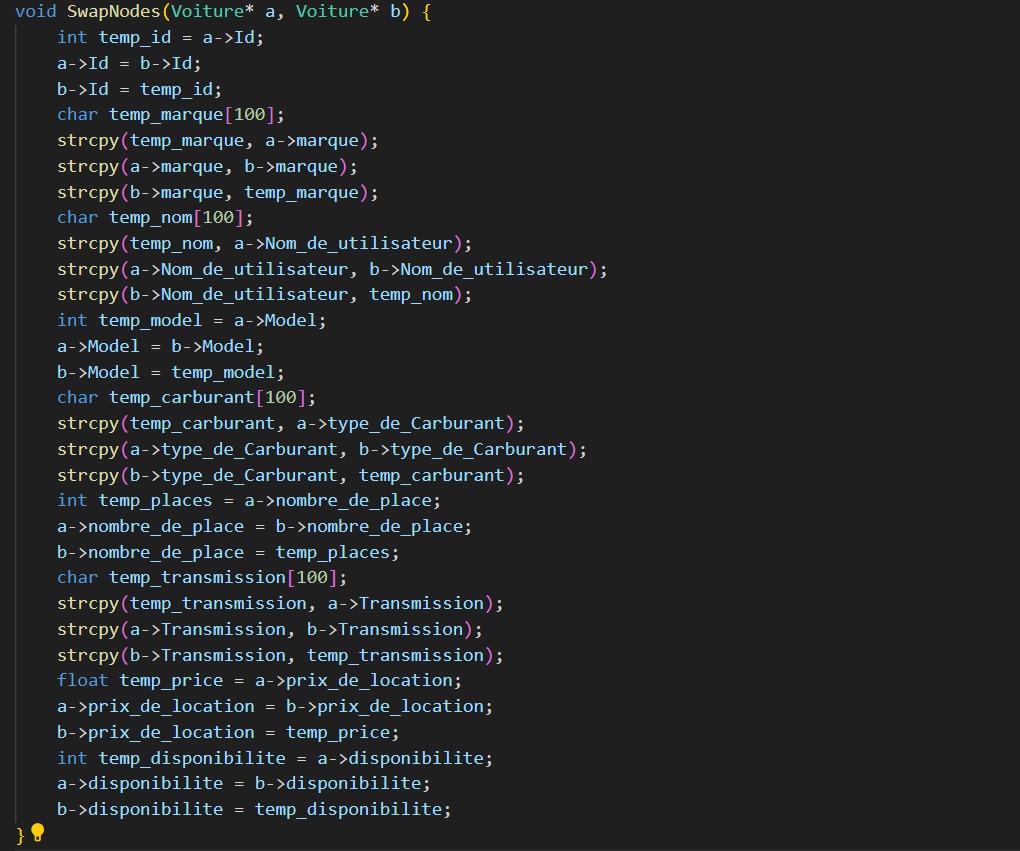
**5. À chaque itération de la boucle externe, elle réinitialise `swapped` à 0, puis commence à parcourir la liste à l'aide du pointeur `ptr1`.**

**6. À chaque itération de la boucle interne, elle compare les prix de location de deux voitures adjacentes et les échange si nécessaire en appelant une fonction `SwapNodes`. Si un échange est effectué, elle met à jour la variable `swapped` à 1.**

**7. Après avoir parcouru toute la liste, elle met à jour `lptr` pour pointer vers le dernier élément trié.**

**8. Elle répète cette opération tant que des échanges ont été effectués pendant le parcours de la liste.**

1. **Fonction SwapNodes :**

****

**Cette fonction `SwapNodes` échange les données de deux nœuds d'une liste chaînée de voitures. Elle prend en paramètres deux pointeurs vers des structures `Voiture`, représentant les nœuds à échanger.**

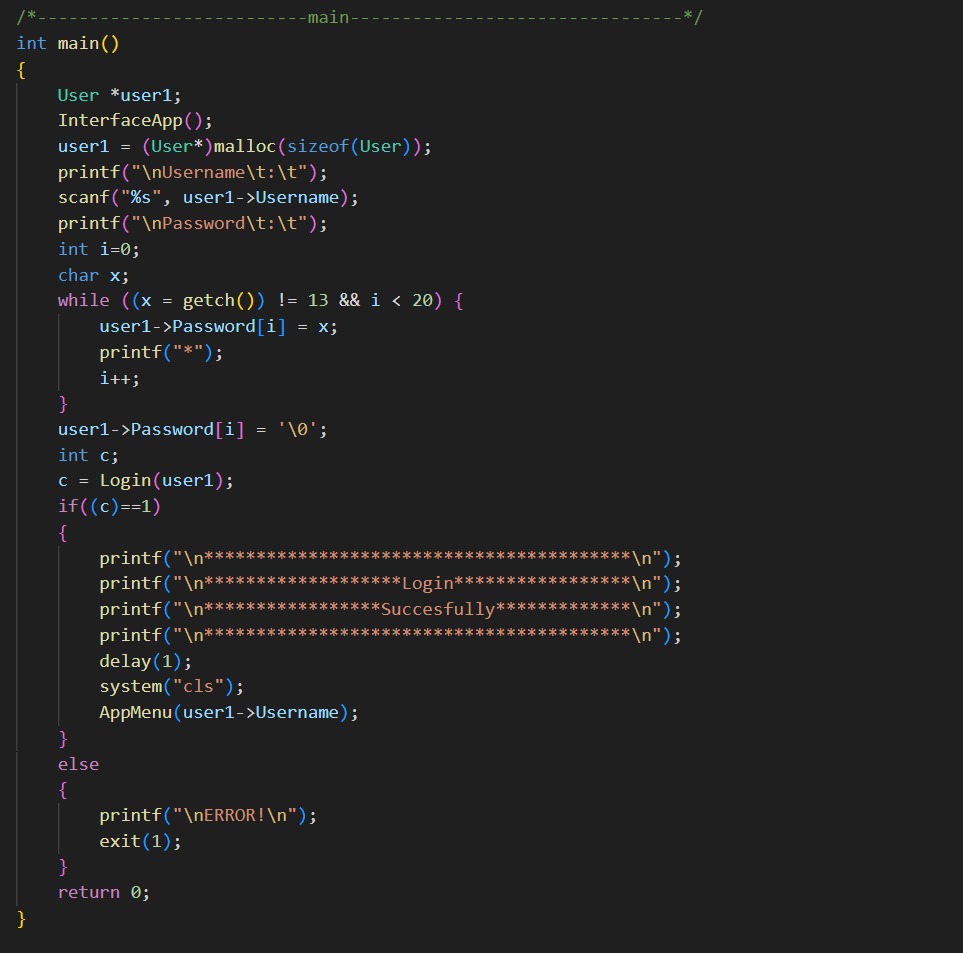
**Voici comment elle fonctionne :**

**- Elle utilise des variables temporaires pour stocker les valeurs des champs de la première voiture (`a`) afin de pouvoir les échanger avec les valeurs des champs de la deuxième voiture (`b`), et vice versa.**

**- Pour échanger les chaînes de caractères (comme `marque`, `Nom\_de\_utilisateur`, `type\_de\_Carburant`, `Transmission`), elle utilise la fonction `strcpy` qui copie le contenu d'une chaîne dans une autre.**

**- Après l'échange des valeurs, les données des deux nœuds `a` et `b` sont correctement échangées.**

1. **Fonction principale (Main) :**

****

**Ce bloc de code représente la fonction `main` du programme. Voici ce qu'il fait :**

**1. \*\*Initialisation d'un utilisateur (`User\* user1`)\*\* : Un pointeur vers une structure `User` est initialisé pour stocker les informations de connexion de l'utilisateur.**

**2. \*\*Interface utilisateur\*\* : La fonction `InterfaceApp` est appelée pour afficher l'écran d'accueil de l'application.**

**3. \*\*Saisie des informations de connexion\*\* : L'utilisateur est invité à saisir son nom d'utilisateur et son mot de passe. Le mot de passe est saisi de manière masquée à l'aide de l'appel à `getch` qui lit un caractère sans afficher sur l'écran, et les caractères saisis sont affichés comme des étoiles `\*`.**

**4. \*\*Authentification\*\* : Les informations de connexion saisies sont vérifiées en appelant la fonction `Login`. Si l'authentification réussit (c'est-à-dire si `Login` renvoie 1), l'utilisateur est connecté et dirigé vers le menu principal de l'application (`AppMenu`). Sinon, le programme se termine avec un message d'erreur.**

**5. \*\*Retour\*\* : La fonction `main` se termine normalement avec `return 0`.**