

🔍 Rapport du Projet : Gestion d'un Hôpital

****

**Table des Matières**

1. Introduction
2. Objectifs du Projet
3. Méthodologie 3.1. Structures de Données 3.2. Gestion de la Mémoire Dynamique 3.3. Gestion des Entrées/Sorties Utilisateur 3.4. Gestion des Buffers de Saisie (main) 3.5. Opérations sur la Liste des Maladies 3.5.1. Création de Nœuds (creermaladie) 3.5.2. Ajout dans la Liste (ajoutD, ajoutF, ajout\_A) 3.5.3. Suppression de Nœuds (supp\_D, supp\_F, supp\_pos) 3.5.4. Recherche dans la Liste (recherche\_mal\_nom, etc.) 3.5.5. Affichage de la Liste (affichage, affichage\_cat) 3.5.6. Modification de Nœuds (modif\_mal) 3.6. Opérations sur la Liste des Patients 3.6.1. Création de Nœuds et Liaison Maladie Simplifiée (creerPatient et logique dans main) 3.6.2. Ajout dans la Liste (ajoutDebutPatient, etc.) 3.6.3. Suppression de Nœuds (suppPatientSpecifique, etc.) 3.6.4. Recherche dans la Liste (recherchePatientParNom, etc.) 3.6.5. Affichage de la Liste (affichagePatients, etc.) 3.7. Persistance des Données (Sauvegarde Fichier) (savePatientToFile)
4. Résultats Obtenus
5. Analyse des Résultats
6. Conclusion
7. Domaines d'Application Potentiels

**1. Introduction**

Ce rapport présente le développement d'un système de gestion rudimentaire pour des données médicales basiques, centré sur la gestion de maladies et de patients en langage C. Le projet s'appuie sur l'implémentation de listes doublement chaînées pour permettre une gestion dynamique de la mémoire pour les collections de données. L'interface utilisateur est en mode console. Cette version spécifique du projet intègre la possibilité de sauvegarder les informations des patients dans un fichier externe pour une persistance simple, tout en permettant de lier interactivement une maladie existante à un patient au moment de sa création.

**2. Objectifs du Projet**

Les objectifs principaux poursuivis dans ce projet étaient de :

* Définir et implémenter des structures de données adaptées (listes doublement chaînées) pour modéliser les maladies et les patients ainsi que leurs relations.
* Développer des fonctions pour gérer de manière sécurisée et dynamique l'allocation et la libération de la mémoire (évitant les fuites).
* Implémenter les opérations de gestion de base (Création, Ajout, Suppression, Recherche, Affichage, Modification pour les maladies) pour les maladies et les patients.
* Permettre la liaison d'un patient à une maladie enregistrée dans le système.
* Gérer les entrées utilisateur de manière robuste en évitant les erreurs courantes (gestion du buffer, dépassements de tampon).
* Ajouter une fonctionnalité de sauvegarde des données patients dans un fichier texte pour assurer une forme de persistance entre les exécutions du programme.
* Fournir une interface utilisateur textuelle simple pour interagir avec le système.

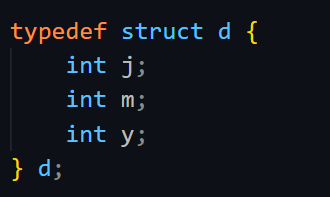
**3. Méthodologie**

Le système a été développé en langage C, en s'appuyant sur les pointeurs et les structures pour construire et manipuler des listes doublement chaînées. L'interaction avec l'utilisateur se fait via l'entrée/sortie standard (console).

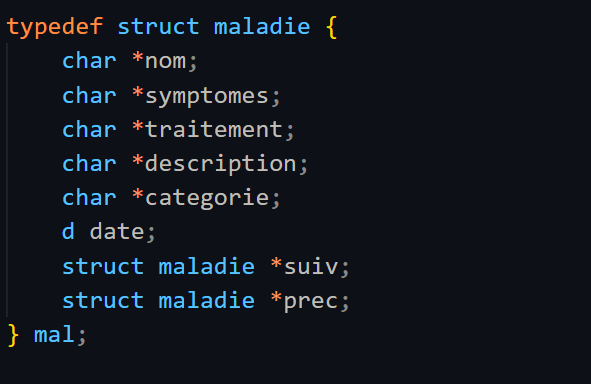
**3.1. Structures de Données**

Trois structures fondamentales sont utilisées :

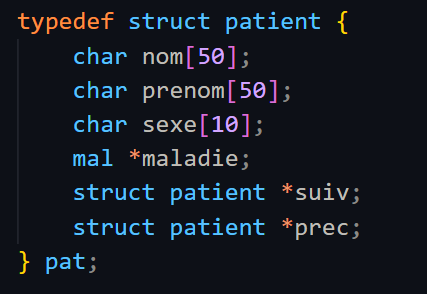
* struct d: Pour représenter une date avec des champs entiers pour le jour, le mois et l'année.



* struct maladie: Pour stocker les informations d'une maladie. Elle contient des pointeurs char\* pour les champs textuels (nom, symptômes, traitement, description, catégorie) afin de gérer des chaînes de caractères de tailles variables. Elle inclut également une date et des pointeurs suiv et prec pour la navigation dans la liste doublement chaînée des maladies.



* struct patient: Pour stocker les informations d'un patient. Elle utilise des tableaux de caractères de taille fixe pour le nom, le prénom et le sexe. Elle inclut un pointeur mal \*maladie pour établir un lien vers une maladie dans la liste des maladies (un patient peut être associé à une maladie). Elle contient également des pointeurs suiv et prec pour la liste doublement chaînée des patients.



**3.2. Gestion de la Mémoire Dynamique**

L'allocation de mémoire pour les nœuds de listes (mal, pat) et les champs char\* de la structure mal est effectuée explicitement avec la fonction malloc() (et vérifiée pour les erreurs). La libération de cette mémoire se fait à l'aide de la fonction free() lorsque les nœuds sont supprimés individuellement ou lors de la sortie du programme via les fonctions freeMaladieList et freePatientList. Il est crucial de libérer la mémoire de chaque champ char\* d'un nœud mal avant de libérer le nœud mal lui-même. Pour la structure pat, seul le nœud lui-même est libéré, car les tableaux de caractères sont inclus dans la structure et le pointeur maladie référence une mémoire gérée par la liste des maladies.

**3.3. Gestion des Entrées/Sorties Utilisateur**

L'interaction se fait via la console standard en utilisant printf() pour les messages et scanf() combiné à fgets() pour la lecture des entrées utilisateur. La gestion des caractères de fin de ligne laissés par scanf est assurée par getchar(), et la suppression du saut de ligne lu par fgets est gérée avec strcspn.

**3.4. Gestion des Buffers de Saisie (main)**

Pour améliorer la sécurité et éviter les dépassements de tampon lors de la lecture de chaînes de caractères de différentes tailles (noms, descriptions, catégories, etc.) dans la fonction main, des tableaux de caractères distincts sont déclarés avec des tailles correspondant aux tailles attendues par les structures ou allouées dans les fonctions de création. L'utilisation de sizeof() comme argument de taille dans fgets() garantit que la lecture ne dépasse jamais la capacité du buffer.

Exemple de déclaration et utilisation :

C

// Variables de saisie avec des tailles appropriées

char input\_maladie\_nom[256]; // Pour les noms de maladies et traitements

char input\_maladie\_description[512]; // Pour les descriptions

// ... autres buffers ...

// Lors de la saisie d'une description:

printf("Saisir la description de la maladie à rechercher : ");

fgets(input\_maladie\_description, sizeof(input\_maladie\_description), stdin);

input\_maladie\_description[strcspn(input\_maladie\_description, "\n")] = 0;

// ... appel à recherche\_mal\_description(..., input\_maladie\_description);

Cette technique prévient les avertissements de compilation liés aux tailles de buffer (-Wstringop-overflow=) et renforce la robustesse du programme.

**3.5. Opérations sur la Liste des Maladies**

Les fonctions suivantes sont implémentées pour manipuler la liste des maladies :

**3.5.1. Création de Nœuds (creermaladie)**

Alloue mémoire pour un nouveau nœud mal et ses champs char\*, invite l'utilisateur à saisir les détails de la maladie, et retourne le pointeur vers le nœud ou NULL en cas d'échec d'allocation.

**3.5.2. Ajout dans la Liste (ajoutD, ajoutF, ajout\_A)**

Fonctions pour ajouter un nouveau nœud mal respectivement au début, à la fin, ou après un nœud spécifique identifié par son nom. Elles gèrent la mise à jour des pointeurs suiv et prec de la liste doublement chaînée.

**3.5.3. Suppression de Nœuds (supp\_D, supp\_F, supp\_pos)**

Fonctions pour supprimer un nœud mal au début, à la fin, ou le premier nœud correspondant à un nom spécifique. Elles gèrent la mise à jour des pointeurs des nœuds voisins et libèrent la mémoire du nœud supprimé et de ses champs char\*.

**3.5.4. Recherche dans la Liste (recherche\_mal\_nom, etc.)**

Fonctions pour parcourir la liste des maladies et trouver un nœud correspondant à un critère (nom, date, description, traitement). Elles retournent un pointeur vers le nœud trouvé ou NULL.

**3.5.5. Affichage de la Liste (affichage, affichage\_cat)**

Fonctions pour parcourir la liste et imprimer les détails de tous les nœuds ou des nœuds filtrés par catégorie.

**3.5.6. Modification de Nœuds (modif\_mal)**

Recherche une maladie par nom et permet à l'utilisateur de modifier interactivement les champs (symptômes, traitement, description) du nœud trouvé en mémoire.

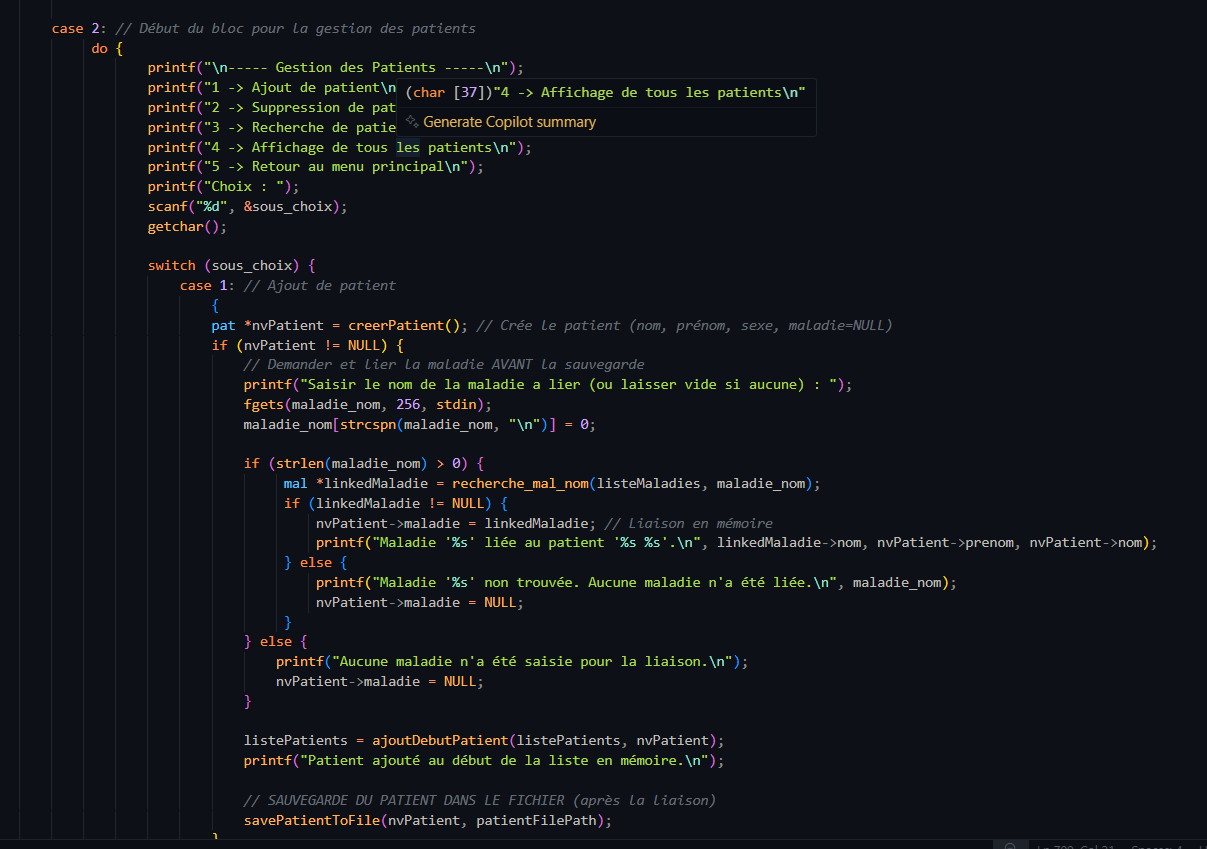
**3.6. Opérations sur la Liste des Patients**

Les fonctions suivantes sont implémentées pour manipuler la liste des patients :

**3.6.1. Création de Nœuds et Liaison Maladie Simplifiée (creerPatient et logique dans main)**

La fonction creerPatient() alloue un nœud pat, invite à saisir nom, prénom, sexe, et initialise maladie à NULL. La logique dans main, après la création et l'ajout à la liste en mémoire, demande directement à l'utilisateur le nom de la maladie à lier. Si un nom est saisi et qu'une maladie correspondante est trouvée avec recherche\_mal\_nom, le pointeur nvPatient->maladie est mis à jour vers cette maladie.

Extrait de main montrant la logique simplifiée :



**3.6.2. Ajout dans la Liste (ajoutDebutPatient, etc.)**

Fonctions pour ajouter un nouveau nœud pat au début, à la fin, ou après un nœud spécifique.

**3.6.3. Suppression de Nœuds (suppPatientSpecifique, etc.)**

Fonctions pour supprimer un nœud pat de la liste (début, fin, ou par nom). Elles libèrent la mémoire du nœud pat supprimé. Notez que cela ne libère pas la mémoire de la maladie pointée par patient->maladie.

**3.6.4. Recherche dans la Liste (recherchePatientParNom, etc.)**

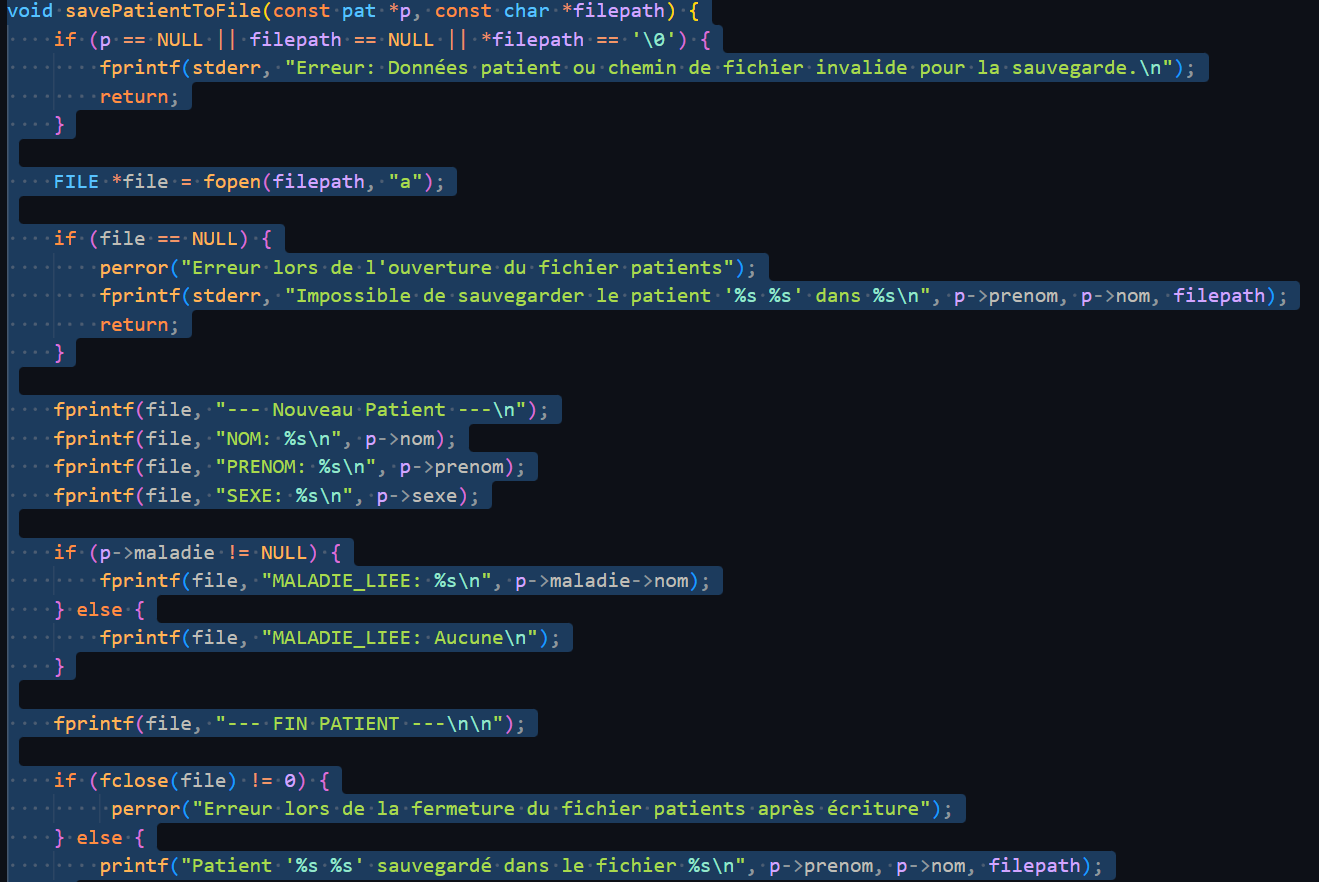
Fonctions pour trouver des patients par nom ou par la maladie qui leur est associée (en comparant le nom de la maladie pointée).

**3.6.5. Affichage de la Liste (affichagePatients, etc.)**

Fonctions pour afficher les détails des patients, y compris le nom de la maladie associée si le pointeur maladie n'est pas NULL.

**3.7. Persistance des Données (Sauvegarde Fichier) (savePatientToFile)**

Cette version inclut une fonctionnalité de sauvegarde des données patients dans un fichier texte. La fonction savePatientToFile prend un pointeur vers un nœud patient et le chemin du fichier comme arguments.



fait en mode "ajout" ("a"), ce qui permet d'ajouter de nouveaux enregistrements patients à la fin du fichier sans écraser le contenu existant. La fonction est appelée dans main juste après la création/liaison d'un patient, s'assurant que l'état actuel du nœud en mémoire est enregistré. Le chemin du fichier est demandé à l'utilisateur au début de l'exécution du programme.

**4. Résultats Obtenus**

Le programme permet une gestion fonctionnelle des maladies et des patients en mémoire. L'utilisateur peut interagir via un système de menus pour ajouter, supprimer, rechercher et afficher ces données. La modification est possible pour les maladies. Lors de l'ajout d'un patient, il est possible de le lier à une maladie existante par son nom. Cette liaison en mémoire est ensuite enregistrée dans un fichier spécifié par l'utilisateur au début du programme, sous forme d'enregistrements texte ajoutés au fichier. La gestion améliorée des buffers de saisie assure une meilleure stabilité du programme lors des interactions. La mémoire allouée est correctement libérée lors de la sortie.

**5. Analyse des Résultats**

**Points Forts :**

* **Flexibilité des Listes Chaînées :** Permet de gérer dynamiquement les collections de données sans limites de taille fixes.
* **Opérations de Gestion de Base Complètes (Maladies) :** Ajout, suppression, recherche, affichage et modification sont bien implémentés pour les maladies.
* **Opérations de Base (Patients) :** Ajout, suppression, recherche, affichage sont implémentés pour les patients.
* **Liaison Patient-Maladie Intégrée :** La possibilité de lier une maladie lors de l'ajout de patient est directement accessible.
* **Persistance des Patients :** La sauvegarde des patients dans un fichier est une fonctionnalité importante pour conserver un historique des données entre les exécutions.
* **Robustesse de la Saisie :** La gestion des buffers de tailles adaptées résout les problèmes de dépassement de tampon et rend les entrées utilisateur plus sûres.
* **Gestion de la Mémoire :** L'allocation et la libération manuelle de la mémoire sont gérées aux points clés, prévenant les fuites.

**Points Faibles et Axes d'Amélioration :**

* **Persistance Unidirectionnelle :** Le programme peut sauvegarder des patients, mais il ne peut pas charger les données (maladies ou patients) depuis les fichiers au démarrage. Toutes les données, y compris la liste des maladies et les patients déjà sauvegardés dans le fichier lors d'exécutions précédentes, doivent être recréées manuellement à chaque lancement.
* **Liaison Patient-Maladie Limitée :** Un patient ne peut être associé qu'à *une seule* maladie dans cette structure.
* **Absence de Modification Avancée des Patients :** Il manque une option de modification pour les patients (changer nom/prénom/sexe) et surtout pour modifier ou ajouter une maladie *après* la création initiale.
* **Gestion des Pointeurs Danglants :** Si une maladie est supprimée alors que des patients y sont liés, les pointeurs maladie dans les nœuds patients correspondants deviennent invalides. Accéder à patient->maladie après une telle suppression peut causer un crash. Une gestion plus avancée (par exemple, mettre le pointeur à NULL chez les patients concernés lors de la suppression d'une maladie) serait nécessaire.
* **Validation d'Entrée :** La validation reste basique (principalement format et taille). Des contrôles plus poussés sur la validité des données (dates, choix numériques) seraient bénéfiques.
* **Interface :** L'interface en ligne de commande est simple et limitée en termes de convivialité par rapport à une interface graphique.

**6. Conclusion**

Ce projet démontre une implémentation solide des listes doublement chaînées en C pour la gestion dynamique de données. Les objectifs de base pour la manipulation des entités maladies et patients sont atteints, avec une gestion attentive de la mémoire et des entrées/sorties. L'ajout d'une fonctionnalité de sauvegarde des patients dans un fichier représente un pas important vers la persistance des données, bien qu'elle soit limitée par l'absence de fonction de chargement. Le système actuel est une base fonctionnelle mais temporaire, offrant une bonne compréhension des mécanismes de bas niveau en C pour la gestion de données liées en mémoire et une introduction simple à la persistance sur fichier.

**7. Domaines d'Application Potentiels**

Bien que limitée par l'absence de chargement des données, cette version du système pourrait servir de :

* **Outil Pédagogique :** Excellent exemple pratique pour l'apprentissage des structures de données dynamiques, de la gestion de la mémoire et des I/O en C, y compris l'interaction entre différentes listes et la persistance simple.
* **Base pour Développement Ultérieur :** Peut être étendu pour inclure le chargement des données depuis des fichiers (pour les maladies et les patients), la gestion de plusieurs maladies par patient, des options de modification complète des patients, et potentiellement une interface utilisateur plus avancée.
* **Prototype Simple :** Pour modéliser des scénarios de gestion de données en mémoire avant de passer à des systèmes plus complexes basés sur des bases de données externes.