

# Gestion de stock : intégration en C

## Objectif du Projet

Le projet vise à développer une application console en C pour la gestion de stock, permettant aux utilisateurs de gérer un inventaire de produits via une interface utilisateur en ligne de commande. L'application supporte la saisie, modification, suppression, recherche, tri, et enregistrement des données produit dans un fichier CSV.

## Conception du Système

L'application utilise une structure de données de type liste chaînée pour stocker les informations des produits dynamiquement. Chaque produit est représenté par un nœud dans la liste contenant divers attributs comme l'ID du produit, nom, description, utilisateur, prix, quantité, seuil d'alerte, et les dates d'entrée et de sortie de stock.

# Fonctions

## Fonctions principales :

**Struct Produit** : Structure de données représentant un produit avec divers champs (nom, description, prix, etc.).

**ajouterProduit**: Ajoute un nouveau produit à la liste.

**afficherProduits**: Affiche tous les produits de la liste.

**supprimerProduit**: Supprime un produit par ID.

**modifierProduit**: Modifie les détails d'un produit par ID.

**rechercherProduit**: Recherche un produit par ID.

**trierProduits**: Trie les produits par nom ou prix unitaire.

**libererListe**: Libère la mémoire allouée pour la liste des produits.

## Fonctions auxiliaires :

**printProductDetails**: Affiche les détails d'un produit.

**rechercherParNom**: Recherche des produits par nom.

**rechercherParUtilisateur**: Recherche des produits par utilisateur.



# Algorithme du code

## 1.Ajout de produit :

**Allocation de mémoire pour un nouveau produit.**

**Lecture des détails du produit depuis l'utilisateur.**

**Insertion du nouveau produit au début de la liste.**

## 2.Affichage des produits :

**Parcourir la liste et afficher les informations de chaque produit.**

## 3.Suppression de produit :

**Parcourir la liste pour trouver le produit par ID.**

**Ajuster les pointeurs pour supprimer le produit de la liste et libérer la mémoire.**

## 4.Modification de produit :

**Parcourir la liste pour trouver le produit par ID.**

**Lecture des nouvelles informations depuis l'utilisateur et mise à jour du produit.**

## 5.Recherche de produit :

**Parcourir la liste pour trouver le produit par ID, nom ou utilisateur et afficher les informations.**

## 6.Tri des produits :

**Utiliser un algorithme de tri (ici, une variante du tri à bulles) pour trier la liste en fonction du nom ou du prix unitaire.**

## 7.Libération de la mémoire :

**Parcourir la liste et libérer la mémoire allouée pour chaque produit.**

# Problèmes résolus

**Gestion de la mémoire : Allocation et libération de la mémoire pour les produits.**

**Manipulation de listes chaînées : Ajout, suppression, modification, et recherche de produits dans une liste chaînée.**

**Entrées utilisateur : Gestion des entrées utilisateur pour les différents détails des produits.**

**Tri : Implémentation d'un algorithme de tri pour organiser les produits par nom ou prix.**

**Taches 1 : Implémentation de chargerProduits et sauvegarderProduits**

**1. Implémentation de la fonction chargerProduits pour lire les produits depuis un fichier CSV.**

**2.Implémentation de la fonction sauvegarderProduits pour écrire les produits dans un fichier CSV.**

**Objectifs:**

**1.Assurer que les produits peuvent être chargés et sauvegardés efficacement.**

**2.Maintenir l'intégrité des données lors des opérations de fichier.**

```
// Charge les produits depuis le fichier CSV dans une liste chaînée
Produit *chargerProduits() { ⚠ This old-style function definition is not preceded by a prototype
    FILE *file = fopen(FILENAME, "r");
    if (!file) {
        printf("Fichier non trouvé, démarrage avec une liste vide.\n");
        return NULL;
    }

    Produit *tete = NULL, *courant = NULL;
    while (!feof(file)) {
        Produit *nouveau = malloc(sizeof(Produit));
        if (fscanf(file, "%d,%49[^,],%254[^,],%49[^,],%f,%d,%d,%10[^,],%10[^\\n]",
                    &nouveau->id, nouveau->nom, nouveau->description,
                    nouveau->utilisateur,
                    &nouveau->prix_unitaire, &nouveau->quantite,
                    &nouveau->seuil_alerte,
                    nouveau->date_entree, nouveau->date_sortie) == 9) {
            nouveau->suivant = NULL;
            if (!tete) {
                tete = nouveau;
                courant = nouveau;
            } else {
                courant->suivant = nouveau;
                courant = nouveau;
            }
        } else {
            free(nouveau);
            break; // Si la lecture échoue, arrêtez la boucle
        }
    }
    fclose(file);
    return tete;
}
```

Expli

La for

en mo

Elle li

utilise

cham

Elle a

même

l'ajou

chaîn

Elle n

s'assu

produ

```
163 // Sauvegarde la liste des produits dans un fichier CSV
164 void sauvegarderProduits(Produit *tete) {
165     FILE *file = fopen(FILENAME, "w");
166     if (!file) {
167         printf("Erreur lors de l'ouverture du fichier.\n");
168         return;
169     }
170
171     Produit *courant = tete;
172     while (courant != NULL) {
173         fprintf(file, "%d,%s,%s,%s,.2f,%d,%d,%s,%s\n",
174                 courant->id, courant->nom, courant->description,
175                 courant->utilisateur,
176                 courant->prix_unitaire, courant->quantite, courant->seuil_alerte,
177                 courant->date_entree, courant->date_sortie);
178         courant = courant->suivant;
179     }
180     fclose(file);
181 }
```

# Tache 2 : Gestion des Produits et Recherche

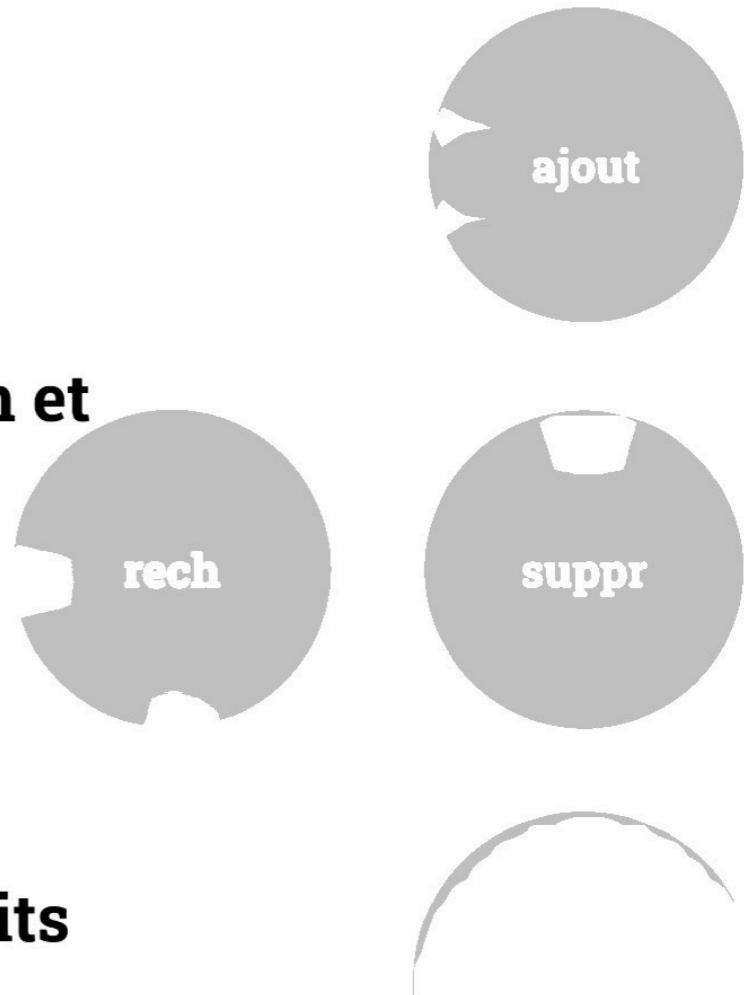
1.Implémentation des fonctions ajouterProduit, supprimerProduit, et modifierProduit.

2.Création des fonctions de recherche rechercherParNom et rechercherParUtilisateur.

**Objectifs :**

1.Permettre une gestion fluide des produits dans l'inventaire.

2.Fournir des méthodes efficaces pour trouver des produits spécifiques.



```

Produit *ajouterProduit(Produit *tete, int *dernierId) {
    Produit *nouveau = malloc(sizeof(Produit));
    if (nouveau == NULL) {
        printf("Erreur d'allocation mémoire.\n");
        return tete;
    }
    nouveau->id = (*dernierId)++;
    printf("Nom du produit: ");
    fgets(nouveau->nom, MAX_NOM, stdin);
    nouveau->nom[strcspn(nouveau->nom, "\n")] = 0; // Remove newline character

    printf("Description: ");
    fgets(nouveau->description, MAX_DESC, stdin);
    nouveau->description[strcspn(nouveau->description, "\n")] = 0;

    printf("Nom d'utilisateur: ");
    fgets(nouveau->utilisateur, MAX_USER, stdin);
    nouveau->utilisateur[strcspn(nouveau->utilisateur, "\n")] = 0;

```

## Explication :

**Alloue de la mémoire pour un nouveau produit.**

**Remplit les champs du produit avec les paramètres fournis.**

**Assigne un ID unique au produit.**

**Insère le produit au début de la liste chaînée.**

```

printf("Prix unitaire: ");
scanf("%f", &nouveau->prix_unitaire);
printf("Quantité en stock: ");
scanf("%d", &nouveau->quantite);
printf("Seuil d'alerte de stock: ");
scanf("%d", &nouveau->seuil_alerte);
getchar(); // Consume the newline character left by scanf

printf("Date d'entrée en stock (YYYY-MM-DD): ");
fgets(nouveau->date_entree, 11, stdin);
nouveau->date_entree[strcspn(nouveau->date_entree, "\n")] = 0;

printf("Date de sortie du stock (YYYY-MM-DD): ");
fgets(nouveau->date_sortie, 11, stdin);
nouveau->date_sortie[strcspn(nouveau->date_sortie, "\n")] = 0;

nouveau->suivant = tete; // Insérer au début de la liste
return nouveau;
}

```

```
|Produit *supprimerProduit(Produit *tete, int id) {  
|    Produit *courant = tete;  
|    Produit *precedent = NULL;  
|    while (courant != NULL) {  
|        if (courant->id == id) {  
|            if (precedent == NULL) {  
|                tete = courant->suivant; // Supprimer le premier élément  
|            } else {  
|                precedent->suivant = courant->suivant;  
|            }  
|            free(courant);  
|            printf("Produit supprimé avec succès.\n");  
|            return tete;  
|        }  
|        precedent = courant;  
|        courant = courant->suivant;  
|    }  
|    printf("Produit avec ID %d non trouvé.\n", id);  
|    return tete;  
|}
```

```
void modifierProduit(Produit *tete, int id) {
    Produit *courant = tete;
    while (courant != NULL) {
        if (courant->id == id) {
            printf("Modification du produit ID %d\n", id);
            printf("Nom actuel: %s, Nouveau nom: ", courant->nom);
            fgets(courant->nom, MAX_NOM, stdin);
            courant->nom[strcspn(courant->nom, "\n")] = 0;

            printf("Description actuelle: %s, Nouvelle description: ",
                   courant->description);
            fgets(courant->description, MAX_DESC, stdin);
            courant->description[strcspn(courant->description, "\n")] = 0;

            printf("Nom d'utilisateur actuel: %s, Nouveau nom d'utilisateur: ",
                   courant->utilisateur);
            fgets(courant->utilisateur, MAX_USER, stdin);
            courant->utilisateur[strcspn(courant->utilisateur, "\n")] = 0;

            printf("Prix unitaire actuel: %.2f, Nouveau prix unitaire: ",
                   courant->prix_unitaire);
            scanf("%f", &courant->prix_unitaire);

            printf("Quantité actuelle: %d, Nouvelle quantité: ",
                   courant->quantite);
            scanf("%d", &courant->quantite);

            printf("Seuil d'alerte actuel: %d, Nouveau seuil: ",
                   courant->seuil_alerte);
            scanf("%d", &courant->seuil_alerte);
            getchar(); // Consume the newline character left by scanf

            printf("Date d'entrée actuelle: %s, Nouvelle date d'entrée
                  (YYYY-MM-DD): ", courant->date_entree);
            fgets(courant->date_entree, 11, stdin);
            courant->date_entree[strcspn(courant->date_entree, "\n")] = 0;

            printf("Date de sortie actuelle: %s, Nouvelle date de sortie
                  (YYYY-MM-DD): ", courant->date_sortie);
            fgets(courant->date_sortie, 11, stdin);
            courant->date_sortie[strcspn(courant->date_sortie, "\n")] = 0;

            printf("Produit modifié avec succès.\n");
            return;
        }
        courant = courant->suivant;
    }
}
```

```
        courant = courant->suivant,
    }
    printf("Produit avec ID %d non trouvé.\n", id);
}
```

## Explication :

**Recherche le produit par ID dans la liste chaînée.**

**Si trouvé, met à jour les champs avec les nouvelles valeurs fournies.**

**Utilise strncpy pour copier les chaînes de caractères de manière sécurisée.**

```
void rechercherParNom(Produit *tete, char *nom) {  
    Produit *courant = tete;  
    int trouve = 0;  
    while (courant != NULL) {  
        if (strcmp(courant->nom, nom) == 0) {  
            printProductDetails(courant);  
            trouve = 1;  
        }  
        courant = courant->suivant;  
    }  
    if (!trouve) {  
        printf("Aucun produit trouvé avec le nom '%s'.\n", nom);  
    }  
}
```

## Explication :

**Parcourt la liste chaînée pour trouver le produit dont le nom correspond à la chaîne de caractères fournie.**  
**Retourne le premier produit trouvé avec ce nom.**

```
void rechercherParUtilisateur(Produit *tete, char *utilisateur) {  
    Produit *courant = tete;  
    int trouve = 0;  
    while (courant != NULL) {  
        if (strcmp(courant->utilisateur, utilisateur) == 0) {  
            printProductDetails(courant);  
            trouve = 1;  
        }  
        courant = courant->suivant;  
    }  
    if (!trouve) {  
        printf("Aucun produit trouvé avec l'utilisateur '%s'.\n", utilisateur);  
    }  
}
```

## Explication :

**Parcourt la liste chaînée pour trouver le produit dont l'utilisateur correspond à la chaîne de caractères fournie.**  
**Retourne le premier produit trouvé avec cet utilisateur**

# **Tache 3 : Interface Utilisateur et Fonctions de Tri**

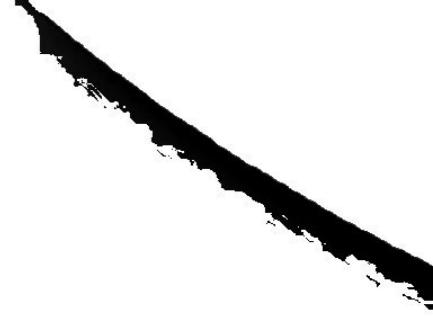
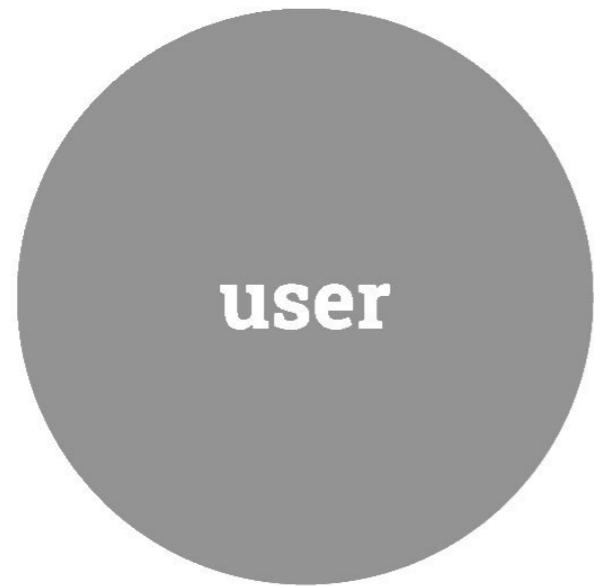
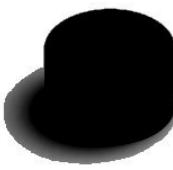
**1.Développement de l'interface utilisateur dans la fonction main.**

**2.Implémentation des fonctions de tri  
trierProduitsParNom et trierProduitsParPrix.**

**Objectifs :**

**1.Créer une interface conviviale pour interagir avec le système.**

**2.Assurer que les produits peuvent être triés correctement selon différents critères.**



```
int main() {
    char utilisateur[MAX_USER];
    printf("Entrez votre nom pour commencer : ");
    fgets(utilisateur, MAX_USER, stdin);
    utilisateur[strcspn(utilisateur, "\n")] = 0; // Remove newline character

    Produit *listeProduits = chargerProduits();
    int dernierId = 1; // Détermine le prochain ID disponible pour un nouveau
                       // produit
    Produit *temp = listeProduits;
    while (temp) {
        if (temp->id >= dernierId) {
            dernierId = temp->id + 1;
        }
        temp = temp->suivant;
    }
}
```

## Explication :

**La fonction main gère l'interface utilisateur en affichant un menu et en traitant les choix de l'utilisateur.**

**Chaque choix est associé à une fonction spécifique qui effectue une opération sur la liste des produits.**

**La boucle continue jusqu'à ce que l'utilisateur choisisse de quitter l'application.**

# "Tri à Bulles"

## Vérification de la Liste Vide :

La fonction commence par vérifier si la liste des produits est vide. Si c'est le cas, il n'y a rien à trier, donc la fonction retourne immédiatement.

## Initialisation des Variables :

Une variable **trie** est utilisée pour indiquer si la liste est déjà triée. Elle est initialement définie à 0, ce qui signifie que la liste n'est pas encore triée.

Une variable **courant** est définie pour parcourir la liste à chaque itération.

## Boucle Principale :

La boucle principale s'exécute tant que la liste n'est pas triée (c'est-à-dire que **trie** est toujours égal à 0). À chaque itération de la boucle principale, on assume que la liste est triée jusqu'à ce qu'on trouve une inversion.

## Parcours de la Liste :

À l'intérieur de la boucle principale, on parcourt la liste chaînée à l'aide d'un pointeur **courant**.

Pour chaque élément de la liste (représenté par **courant**), on compare ses données avec celles de son successeur (**suivant**).

Si les données ne sont pas dans l'ordre selon le mode de tri spécifié (**mode**), alors on échange les données des produits et on ajuste les pointeurs pour corriger l'ordre dans la liste.

### **Échange des Données :**

**Si le mode de tri est 1 (tri par nom) et que le nom du produit actuel est supérieur au nom du produit suivant, ou si le mode de tri est 2 (tri par prix) et que le prix du produit actuel est supérieur au prix du produit suivant, alors les données des deux produits sont échangées.**  
**Cela signifie que les produits sont désordonnés et doivent être échangés pour les trier correctement.**

### **Correction des Pointeurs :**

**Après l'échange des données, les pointeurs courant->suivant et suivant->suivant sont ajustés pour maintenir l'intégrité de la liste chaînée.**

### **Marquage de la Liste comme Triée :**

**Une fois que la boucle interne parcourt toute la liste sans effectuer d'échanges, cela signifie que la liste est triée.**

**La variable `trie` est alors définie sur 1 pour indiquer que la liste est triée.**

### **Fin du Tri :**

**Une fois que la boucle principale a terminé son exécution, la liste est triée et la fonction affiche un message indiquant que les produits ont été triés avec succès.**

## **Difficultés trouvées**

**Gestion des entrées utilisateur : S'assurer de la saisie correcte et de la gestion des caractères de nouvelle ligne.**

**Manipulation des listes chaînées : Gérer les pointeurs pour l'ajout, la suppression et la modification de produits.**

**Tri efficace : Implémenter un algorithme de tri adapté aux listes chaînées.**

**Robustesse : Assurer la gestion correcte des erreurs et la fiabilité du programme.**

# Conclusion

**En conclusion, ce système de gestion d'inventaire pour les produits offre une panoplie de fonctionnalités clés pour une gestion efficace des stocks. En utilisant des structures de données appropriées et des opérations bien définies, il permet à l'utilisateur d'ajouter, de supprimer, de modifier et de rechercher des produits avec facilité. De plus, la capacité de trier les produits selon différents critères et l'intégration d'une interface utilisateur conviviale améliorent l'expérience globale de l'utilisateur. L'utilisation de l'algorithme de tri à bulles garantit un classement précis et une visualisation claire des produits. Enfin, la gestion robuste de la mémoire et la manipulation des fichiers CSV assurent l'intégrité des données tout au long du processus. En combinant ces aspects, ce système offre une solution solide pour la**