Université Abdelmalek Essaidi

Ecole Nationale des Sciences Appliquées de Tétouan

RAPPORT DE PROJET

Mini-Supermarché

Filière : GI 1
Module : Structures de données en C

Groupe N° 5 :

- | Garhani | Ilyas
- | Farhan | Othmane
- | Rahel |Salma
- | Sabaoui | Siham

2025 - 2026

ENSA TETOUAN
PR. HACHCHANE Imane

Résumé / Abstract

Ce projet consiste à concevoir un **mini-système de gestion de supermarché** permettant de gérer les produits, les clients, le passage en caisse et l'historique des transactions. La solution est développée en **langage C** en s'appuyant sur les structures de données du module : **liste chaînée + hachage** (produits), **ABR** (clients), **file FIFO** (caisse) et **pile LIFO** (annulation). L'application propose le CRUD, la recherche, le tri, la file d'attente, le ticket, l'historique et l'annulation, avec persistance via fichiers. Elle met en évidence la maîtrise de la modélisation, de la complexité algorithmique et de la gestion mémoire en C.

Fonctionnalités du Système

Gestion des produits

• Ajout / modification / suppression (CRUD) dans la liste chaînée
• Recherche par **ID** via table de hachage et par **nom**
• Affichage et tri des produits (nom / prix / quantité)
• Sauvegarde / chargement des produits depuis fichier

Gestion des clients (ABR)

• Insertion, recherche (par nom / par ID) et suppression de clients
• Affichage des clients triés (parcours infixe) avec total dépensé
• Sauvegarde / chargement des clients depuis fichier

Passage en caisse

• Mise en file d'attente (enfiler / défiler) et service du prochain client
• Constitution du panier : sélection produit + quantité, contrôle stock
• Mise à jour du stock et du total dépensé du client
• Affichage / génération de ticket

Historique & annulation

• Enregistrement des transactions dans une pile (historique)
• Affichage de l'historique (par client et date)
• Annulation de la dernière transaction (dépilement) et restauration stock

Introduction :

L'informatisation des systèmes de gestion est devenue indispensable dans le domaine du commerce, même à petite échelle. Les mini-supermarchés doivent assurer une gestion efficace de leurs produits, de leurs clients et de leurs transactions afin de garantir un bon fonctionnement quotidien.

Ce projet a pour objectif d'implémenter un système de gestion d'un mini-supermarché en langage C, en utilisant les structures de données étudiées en cours Le programme permet de gérer le stock, les informations des clients, le passage en caisse et l'historique des transactions.

Pour assurer l'efficacité du système, plusieurs structures de données sont utilisées, notamment les listes chaînées, les tables de hachage, les arbres binaires de recherche, les files et les piles. Chaque structure est choisie selon les besoins de l'application. Le projet inclut également la gestion des fichiers pour assurer la sauvegarde des données et leur persistance.

Architecture Générale du Système

L’application est pilotée par un menu principal qui redirige vers quatre modules : **Produits**, **Clients**, **Caisse** et **Historique**. Chaque module manipule les structures en mémoire (liste chaînée + hachage, ABR, file, pile) puis assure la **persistance** en sauvegardant/chargeant les données via fichiers. Ce découpage clarifie le flux : navigation → action → mise à jour des structures → enregistrement éventuel.

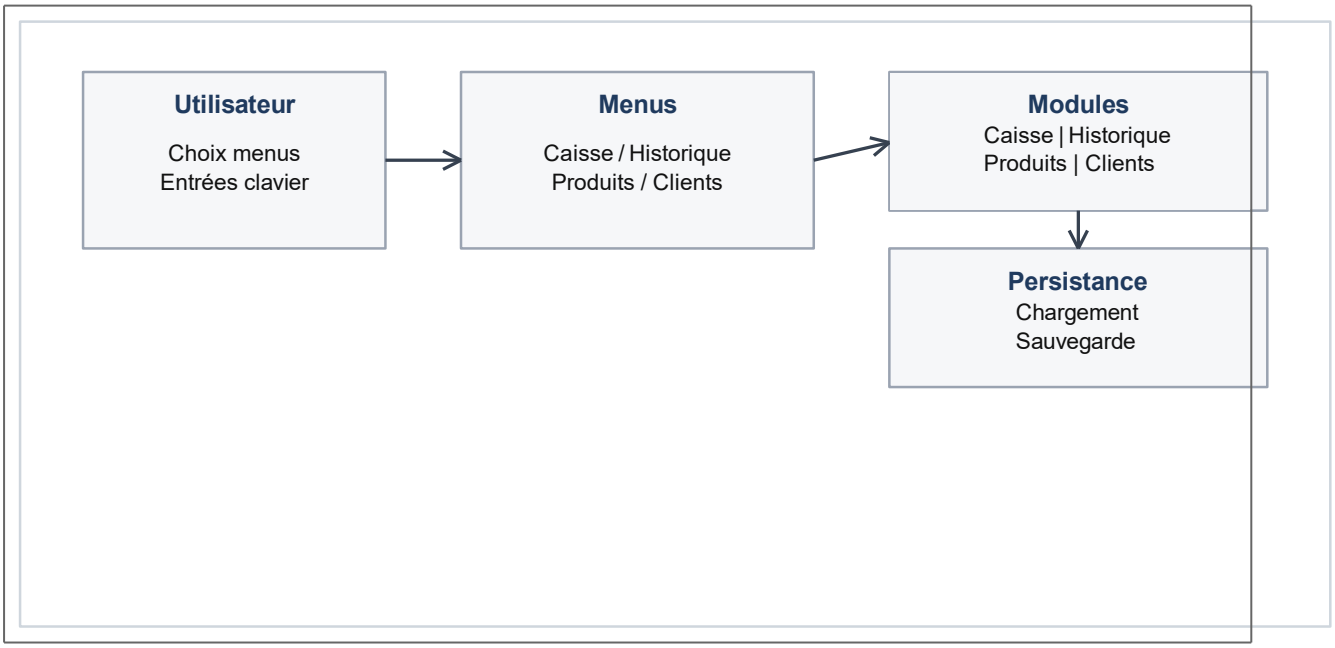

Figure A — Flux simplifié : Utilisateur → Menus → Modules → Fichiers.

Exécution et Résultats

Les figures suivantes présentent des extraits d'exécution console dans un ordre logique (menu → action → résultat). Chaque capture est accompagnée d'une explication courte et vérifiable.

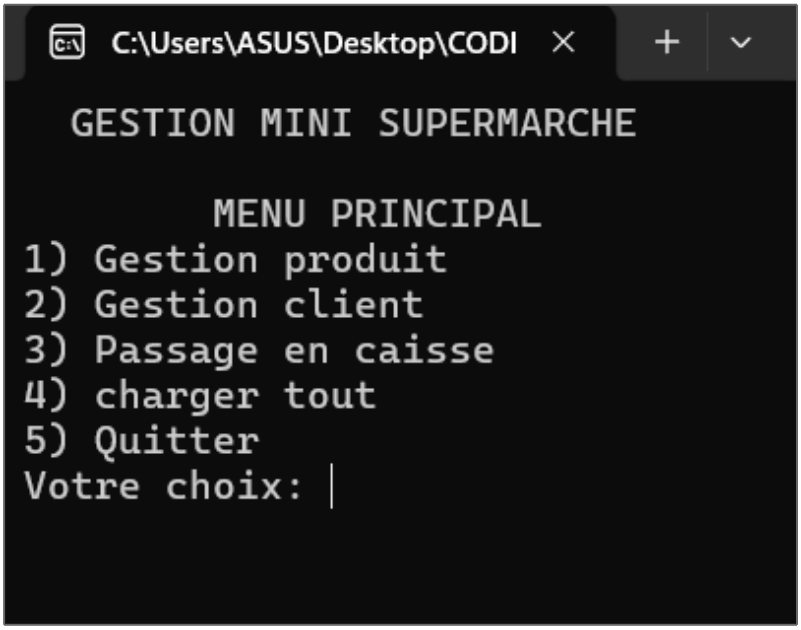

Figure 1 — Menu principal du système

• Affichage du menu principal de l'application.
• Choix d'un module : Produits, Clients, Passage en caisse.
• Navigation centralisée et structurée.

Figure 2 — Affichage de la liste des produits

• Catalogue affiché sous forme : ID | Nom | Prix | Quantité.
• Contrôle visuel de l'état du stock.
• Base de validation avant opérations (recherche/tri/caisse).

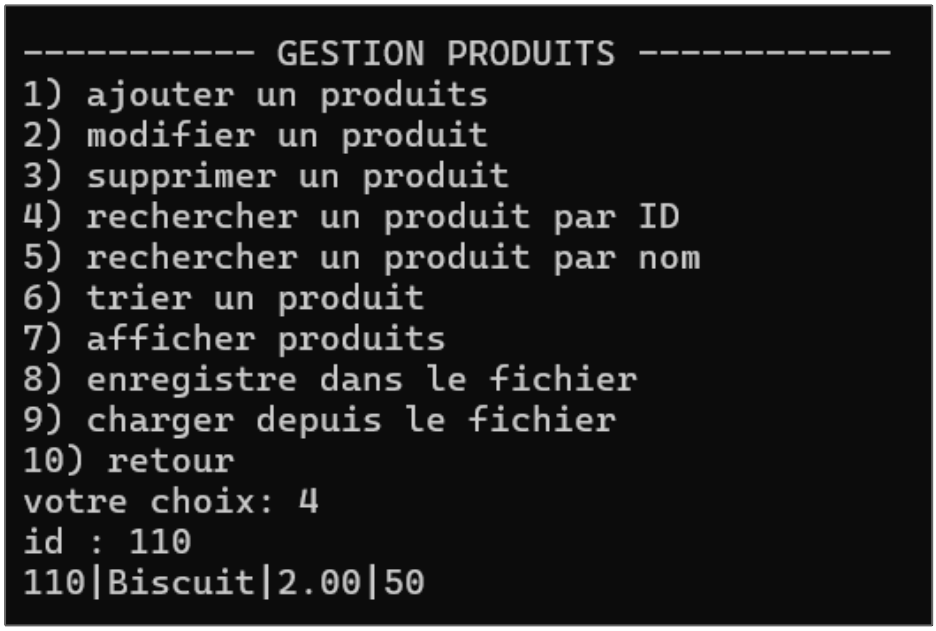

Figure 3 — Recherche d'un produit par ID (hachage)

• Recherche par identifiant via la table de hachage.
• Le produit est retrouvé et affiché.
• Illustration de l'accès O(1) en moyenne (hors collisions).

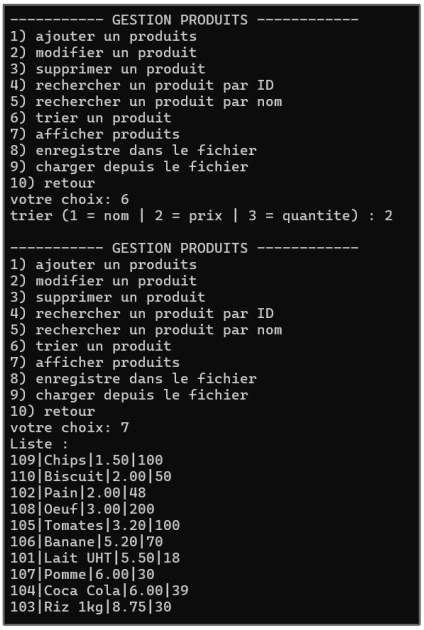

Figure 4 — Tri des produits (ex. par prix)

• Tri selon un critère (nom / prix / quantité).
• Affichage de la liste réordonnée.
• Valide la logique de parcours + permutation sur liste chaînée.

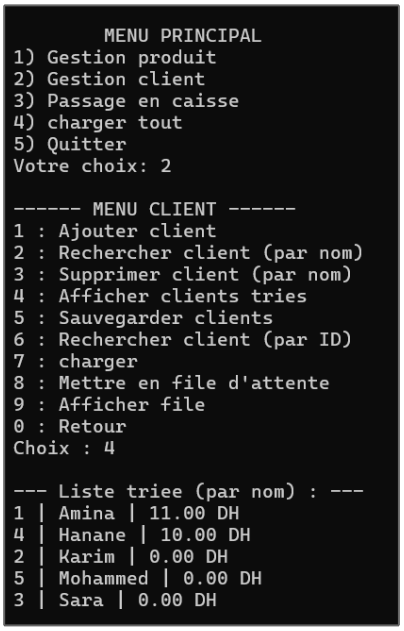

Figure 5 — Clients triés (ABR, parcours infixe)

• Affichage des clients de l'ABR en ordre alphabétique.
• Chaque client : ID, nom, total dépensé.
• Le tri est obtenu naturellement par parcours infixe.

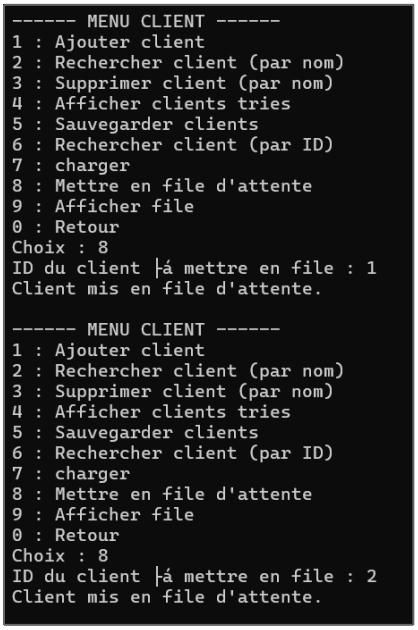

Figure 6 — Mise en file d'attente (FIFO)

• Ajout de clients dans la file d'attente.
• Mise à jour tête/queue de la file.
• Prépare le service en caisse dans l'ordre d'arrivée.

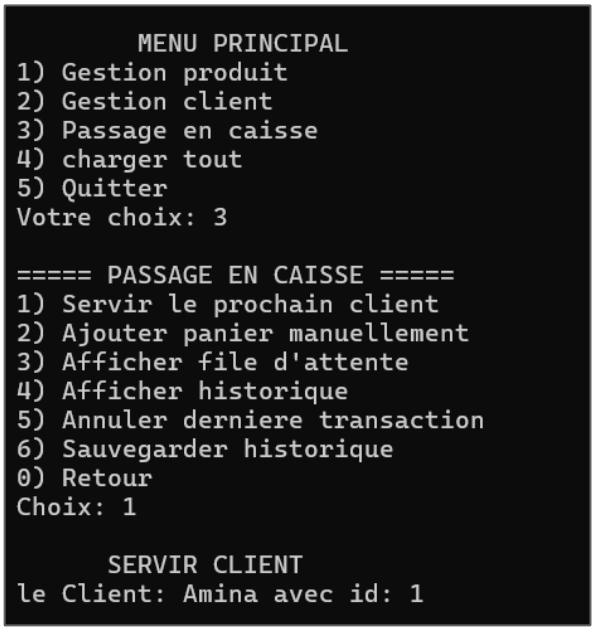

Figure 7 — Service du prochain client

• Défilement (FIFO) du client suivant depuis la file.
• Le client servi est affiché (ID et nom).
• Déclenche la phase panier / transaction.

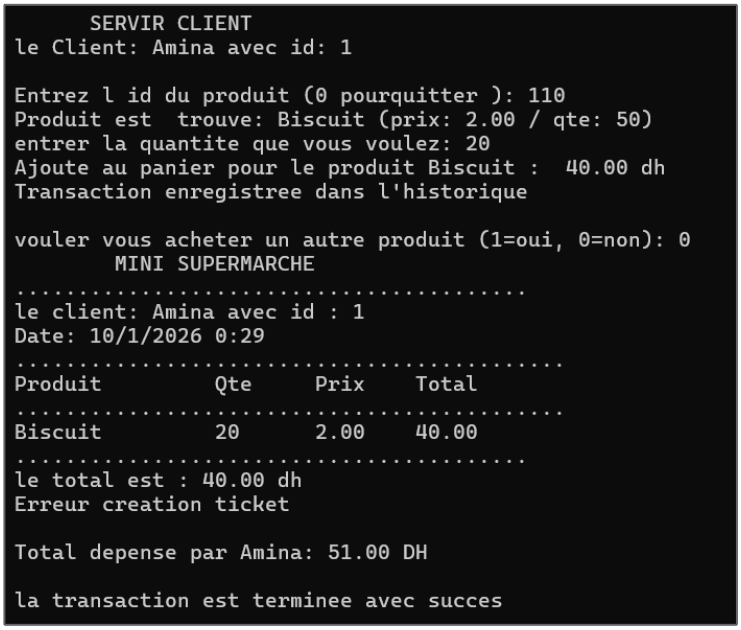

Figure 8 — Panier, transaction et ticket

• Sélection produit (ID), saisie quantité, calcul du total.
• Enregistrement dans l’historique et mise à jour du stock.
• Ticket affiché à l’écran (génération fichier dépend de l’environnement).

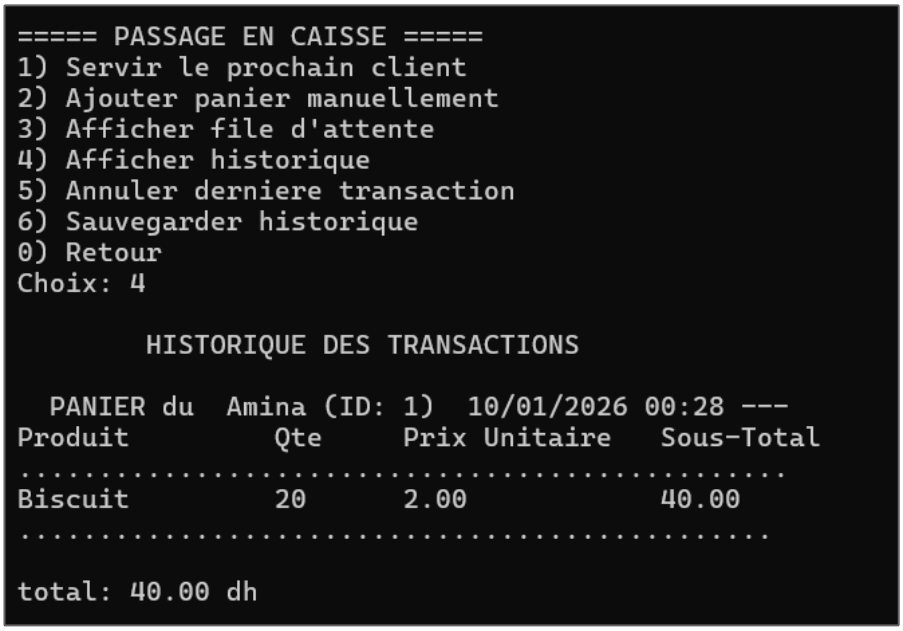

Figure 9 — Historique des transactions (pile)

• Affichage de l'historique par client et date.
• Détail : produit, quantité, prix unitaire, sous-total.
• Traçabilité des paniers enregistrés.

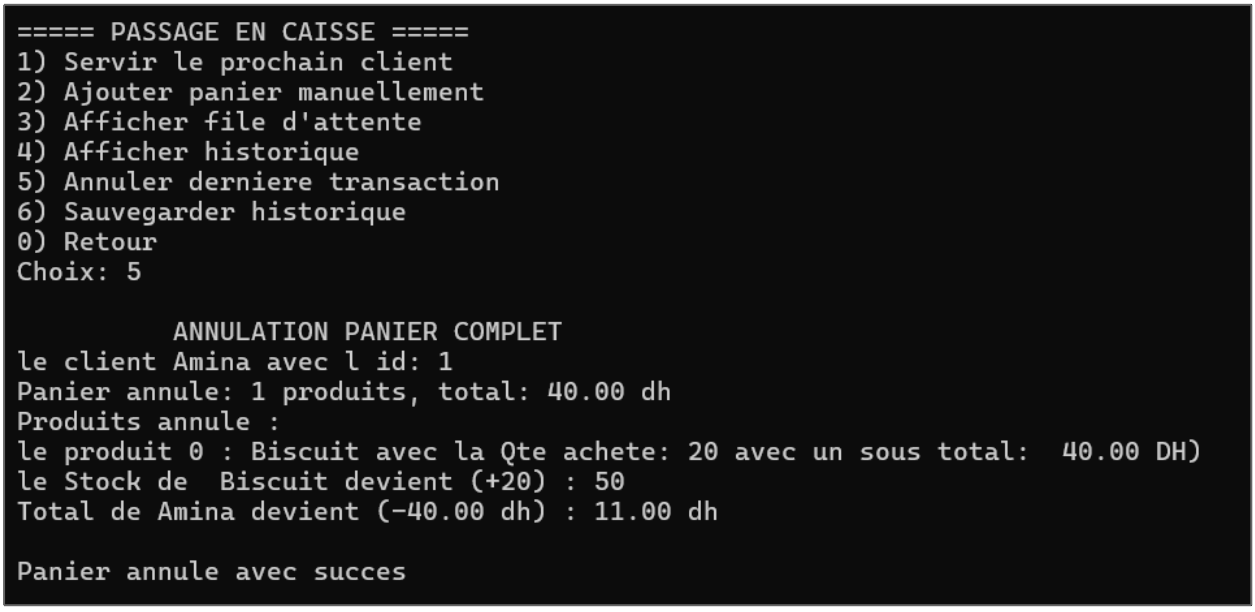

Figure 10 — Annulation dernière transaction (LIFO)

• Annulation du dernier panier via dépilement (LIFO).
• Restauration du stock et correction du total client.
• Valide la fonctionnalité d'annulation conforme au modèle pile.

Tests et Scénarios de Validation

Test	Résultat attendu
Ajout produit (valide)	Saisir un produit avec ID unique, prix>0, quantité≥0 → Ajout confirmé, table de hachage cohérente.
Ajout produit (invalide)	ID existant ou valeurs invalides → Rejet avec message, aucune modification de la liste.
Recherche hachage (collision)	Deux IDs mappant le même index → Parcours de la chaîne du bucket, produit correct retrouvé.
Clients ABR	Insertion + recherche + suppression par nom → Arbre maintenu, affichage infixe toujours trié.
File d'attente	Enfiler plusieurs clients puis servir → Ordre FIFO respecté (premier arrivé, premier servi).
Ticket	Après constitution du panier → Ticket affiché avec détail des achats et total calculé.
Historique / Annulation	Afficher historique puis annuler dernière transaction → Restauration stock + total client corrigé.

Compilation et Exécution

Compilateur : gcc (MinGW / Linux).
Compilation : **gcc** main.c -o mini_supermarche
Exécution : **./mini_supermarche**
Remarque : adapter *main.c* au nom réel du fichier source principal. Si des fichiers de sauvegarde sont utilisés, les placer dans le même dossier que l'exécutable.

Structures de Données Utilisées

A)- Structure des Produits : Liste Chaînée et Table de Hachage

La structure de données liste chaînée est utilisée pour représenter le catalogue des produits, car elle permet une gestion dynamique (ajout, suppression et modification) sans contrainte de taille. En complément, une table de hachage permet un accès rapide aux produits par leur identifiant, optimisant ainsi les opérations de recherche lors du passage en caisse.

```c
typedef struct Produit {
    int id;
    char nom[30];
    float prix;
    int quantite;
    struct Produit *suivant;
} Produit;

typedef struct Liste {
    Produit *tete;
    Produit *queue;
    int taille;
} Liste;

typedef struct {
    Produit *table[TAILLE_TABLE];
} TableHachage;
```

B)- Structure des Clients : Arbre Binaire de Recherche (ABR)

Les clients sont stockés dans une **structure de données de type arbre binaire de recherche**, qui maintient automatiquement les éléments triés par ordre alphabétique selon

le nom, tout en permettant des opérations efficaces de recherche, d'insertion et de suppression.

```c
typedef struct ClientNode {
    int id;
    char nom[30];
    float total;
    struct ClientNode *filsg;
    struct ClientNode *filsd;
} ClientNode;

typedef struct {
    ClientNode *racine;
} ArbreClients;
```

C)- Structure de la File d'Attente : Liste Chaînée FIFO

La file d'attente est implémentée à l'aide d'une **structure de données FIFO basée sur une liste chaînée**, garantissant un traitement des clients dans l'ordre d'arrivée selon le principe "premier arrivé, premier servi". avec des opérations simples et rapides.

```c
typedef struct ClientFile {
    int id;
    char nom[30];
    float total;
    struct ClientFile *suivant;
} ClientFile;

typedef struct File {
    ClientFile *tete;
    ClientFile *queue;
    int nef;
} File;
```

D)- Structure des Transactions : Pile LIFO

```c
typedef struct Transaction {
    int idClient;
    char nomClient[30];
    int idProduit;
    char nomProduit[30];
    int quantite;
    float total;
    char date[20];
    struct Transaction *suivant;
} Transaction;
typedef struct Pile {
    Transaction *tete;
    int nef;
} Pile;
```

L'historique des transactions est géré à l'aide d'une **structure de données de type pile (LIFO)**, adaptée aux besoins d'annulation, car la dernière transaction enregistrée est la première pouvant être retirée.

Fonctions Principales du Programme

1.1 Fonctions Produits

- **ajouterProduit() :**
  La fonction AjouterProduitFin ajoute un produit à la fin d'une liste chaînée après avoir vérifié la validité des données et l'absence de doublons. Elle met à jour les pointeurs de tête et de queue, incrémente la taille de la liste et reconstruit la table de hachage pour maintenir la cohérence des données.
- **supprimerProduit() :**
  La fonction supprimerProduit sert à retirer un produit d'une liste chaînée en utilisant son identifiant. Elle commence par vérifier si la liste est vide, puis cherche le produit dans la liste. S'il est trouvé, elle le supprime en mettant à jour les pointeurs (tête et queue si nécessaire), libère la mémoire occupée, diminue la taille de la liste et reconstruit la table de hachage pour garder des données cohérentes.
- **modifierProduit() :**
  La fonction modifierProduit permet de mettre à jour le nom, le prix ou la quantité d'un produit à partir de son identifiant. Si le produit est trouvé, l'utilisateur peut ajuster ses informations, puis la table de hachage est rafraîchie pour maintenir la cohérence des données
- **rechercherProduitParID()** (avec la table de hachage) :
  rechercherProduitHachage retrouve un produit dans la table de hachage à partir de son ID en parcourant la liste chaînée à l'index calculé, et renvoie le produit ou NULL.
  afficherProduits() + tri.
  Ces fonctions permettent d'afficher une liste de produits et de trier les produits par prix, quantité ou nom en parcourant la liste chaînée et en échangeant les informations des nœuds si nécessaire.

1.2 Fonctions Clients (ABR)

- **insererClient() :**
  La fonction insererClient ajoute un client dans un arbre binaire trié par nom. Elle place le client dans le sous-arbre gauche ou droit selon l'ordre alphabétique et affiche un message si le nom existe déjà.
- **rechercherClient() :**
  Les fonctions rechercherClientNom et rechercherClientID permettent de retrouver un client dans un arbre binaire. La première cherche par nom en suivant l'ordre alphabétique, la seconde cherche par identifiant en parcourant récursivement les sous-arbres gauche et droit, et retournent le client trouvé ou NULL si absent

- **supprimerClient() :**
  La fonction supprimerClientNom supprime un client dans un arbre binaire par son nom, en gérant les cas de feuille, un seul fils ou deux fils (en remplaçant par le successeur minimal). La fonction minNode trouve le nœud avec le plus petit nom dans le sous-arbre droit.
- **afficherClientsInfixe() :**
  La fonction afficherClients parcourt l'arbre en ordre croissant et affiche l'ID, le nom et le total de chaque client.

1.3 Passage en Caisse (File + Table de Hachage)

- **enfilerClient() :**
  La fonction enfilerClient ajoute un client à la fin d'une file. Elle met à jour les pointeurs de tête et de queue et incrémente le nombre d'éléments.
- **defilerClient() :**
  La fonction defilerClient supprime et retourne le premier client d'une file, en mettant à jour les pointeurs et le nombre d'éléments.
- **traiterAchat() :**
  La fonction servirClient défiler le client suivant dans la file, lui permet de choisir des produits en vérifiant la disponibilité et les quantités, ajoute les articles au panier et enregistre les transactions. Elle affiche et génère le ticket, met à jour le stock, reconstruit la table de hachage et actualise le total dépensé par le client dans l'arbre
- **genererTicket() :**
  La fonction afficherTicketEcran affiche à l'écran le ticket d'un client avec son ID, son nom, la date, la liste des produits achetés (quantité, prix, total) et le montant total.

1.4 Historique des Transactions (Pile)

- **pushTransaction() :**
  La fonction empiler ajoute une transaction en haut de la pile, met à jour le nombre d'éléments et enregistre l'opération dans l'historique
- **popTransaction() :**
  La fonction depiler retire et retourne la transaction en haut de la pile, en mettant à jour le nombre d'éléments, ou retourne NULL si la pile est vide.
- **afficherHistorique() :**
  La fonction afficherHistorique parcourt la pile des transactions et affiche, par client et par date, les produits achetés avec leurs quantités, prix, sous-totaux et le total du panier

Analyse de Complexité

Recherche par hachage : rechercherProduitHachage() | moyenne : O(1) Complexité pire cas : O(n)

Explication : La fonction de hachage calcule directement l'indice (id % 13), permettant un accès instantané à la case correspondante. En cas de collision, il faut parcourir la liste chaînée de cette case. Avec une bonne distribution, le nombre de collisions reste faible.

- **Affichage : afficherListeProduits() | Complexité : O(n)**

Explication : Il faut visiter chaque nœud de la liste un par un, du début à la fin, donc proportionnel au nombre de produits.

- **Trie Fonctions : TrierListeProduitsPrix(), TrierListeProduitsQuantite(), TrierListeProduitsNom() = Complexité : O(n²)**

Explication : Utilisation du tri par sélection avec deux boucles imbriquées. Chaque élément est comparé avec tous les autres, ce qui génère n × n comparaisons.

- **Insertion d'un client : insererClient() | Complexité moyenne : O(log n) | pire cas : O(n)**

Explication : À chaque étape, on élimine la moitié de l'arbre (gauche ou droite) selon l'ordre alphabétique. Dans le pire cas (arbre dégénéré en liste), on doit parcourir tous les nœuds.

- **Recherche par nom : rechercherClientNom() | moyenne : O(log n) | pire cas : O(n)**

Explication : Même principe que l'insertion. La structure équilibrée de l'arbre permet de diviser l'espace de recherche à chaque niveau.

- **Suppression client: supprimerClientNom() | moyenne : O(log n) | pire cas : O(n)**

Explication : Recherche du nœud O(log n) + réorganisation locale O(1) = O(log n) au total.

- **Enfiler un client : enfilerClient() | Complexité : O(1)**

Explication : Ajout direct en fin de file grâce au pointeur queue. Aucun parcours nécessaire.

- **Défiler un client : defilerClient() | Complexité : O(1)**

Explication : Retrait direct du premier élément via le pointeur tete. Opération instantanée.

- **Empiler une transaction: empiler() | Complexité : O(1)**

Explication : Insertion en tête de pile. Opération immédiate sans parcours.

- **Dépiler une transaction: depiler() |Complexité : O(1)**

Explication : Retrait du premier élément (sommet de la pile). Accès direct.

Conclusion

Ce projet nous a permis d'appliquer les notions fondamentales des structures de données à travers une application concrète. L'utilisation des listes chaînées, des tables de hachage, des arbres binaires, des files et des piles nous a aidés à mieux comprendre leur rôle et leur fonctionnement.

Nous avons renforcé nos compétences en programmation en langage C, notamment la manipulation des structures, la gestion de la mémoire dynamique, l'utilisation des pointeurs et la modularité du code. Toutefois, le manque de temps a constitué une difficulté importante, ainsi que certains problèmes techniques liés à la suppression des données et à la gestion des fichiers.

Avec plus de temps, des améliorations pourraient être apportées, comme l'amélioration de l'affichage, la détection des produits en rupture de stock, le calcul du chiffre d'affaires journalier et la génération d'un rapport de fin de journée.