Projet: Développement d'un Matching Engine en C++

28 mai 2025

Projet à réaliser en groupe de 3 à 4 étudiants
Date de rendu : Dimanche 8 Juin 2025
Présentations : Semaine du 9 au 13 Juin 2025 (en visioconférence)

Description du Projet

Contexte

Un *matching engine* est le cœur d'un système de trading financier, permettant l'exécution des ordres d'achat et de vente. Il joue un rôle crucial dans les marchés financiers modernes où la performance et la précision sont essentielles.

Objectif

L'objectif de ce projet est de développer un *matching engine* performant en C++ capable de traiter des ordres depuis un fichier CSV, d'exécuter les ordres selon les règles de priorité standards, et de produire un fichier CSV contenant les résultats des opérations.

1 Spécifications Techniques

1.1 Fonctionnalités Requises

- Lecture et désérialization d'ordres à partir d'un fichier CSV d'entrée
- Implémentation d'un carnet d'ordres (*order book*) pour chaque instrument financier (configurable afin s'adapter a l'input)
- Exécution des ordres selon les règles de priorité : prix, puis temps
- Gestion des différents types d'actions : ajout, modification, suppression, exécution
- Gestion des types d'ordres (Market et Limit obligatoire)
- Sérialization des résultats dans un fichier CSV de sortie

1.2 Format du Fichier d'Entrée (CSV)

Le fichier CSV d'entrée contiendra les colonnes suivantes :

```
timestamp,order_id,instrument,side,type,quantity,price,action
```

Où :

- `timestamp` : Horodatage de l'ordre (en nanosecondes depuis l'époque Unix)

— `order_id` : Identifiant unique de l'ordre

— `instrument` : Code de l'instrument financier (ex : "AAPL", "EURUSD")

— `side` : Côté de l'ordre (BUY ou SELL)

— `type` : Type d'ordre (LIMIT, MARKET, etc.)

— `quantity` : Quantité de l'instrument à acheter/vendre

— `price` : Prix limite (pour les ordres de type LIMIT)

— `action` : Action à effectuer (NEW, MODIFY, CANCEL)

1.3 Format du Fichier de Sortie (CSV)

Le fichier CSV de sortie devra contenir les colonnes suivantes :

```
timestamp,order_id,instrument,side,type,quantity,price,action,
status,executed_quantity,execution_price,counterparty_id
```

Les colonnes supplémentaires sont :

— `status` : Statut de l'ordre (EXECUTED, PARTIALLY_EXECUTED, PENDING, CANCELED, REJECTED)

— `executed_quantity` : Quantité exécutée (si applicable)

— `execution_price` : Prix d'exécution (si applicable)

— `counterparty_id` : Identifiant de l'ordre contrepartie (si applicable)

2 Exigences Techniques

2.1 Langage et Bibliothèques

— Le projet doit être implémenté en C++ (C++17 ou supérieur)

— Vous pouvez utiliser les bibliothèques standards C++

— Aucune bibliothèque externe de trading ou de matching engine n'est autorisée

2.2 Performance

— Le matching engine doit être optimisé pour minimiser la latence

— La complexité temporelle des opérations doit être comprises

— Des tests de performance doivent être fournis

2.3 Structure du Projet

— Code source clair et bien organisé

— Séparation des préoccupations (parsing, logique métier, sortie)

— Tests unitaires pour chaque composant

3 Organisation et Livrables

3.1 Organisation du Travail

— Le projet sera réalisé par groupe de 3 à 4 étudiants

— Date de rendu : **Dimanche 8 Juin 2025**

— Les présentations auront lieu la semaine suivant le rendu en visioconférence

3.2 Livrables

1. Dépôt Git contenant :
   — Code source complet
   — Tests unitaires et de performance
   — Scripts de compilation (Makefile)
   — Fichiers README
2. Présentation de 15 minutes suivie de 10 minutes de questions

4 Critères d'Évaluation

— **Fonctionnalité (45%)**
  — Correctitude des exécutions d'ordres
  — Conformité aux spécifications
  — Gestion des cas particuliers et erreurs
— **Performance (25%)**
  — Temps d'exécution
  — Utilisation de la mémoire
  — Passage à l'échelle avec un grand nombre d'ordres
— **Qualité du code (20%)**
  — Les logs apporté
  — Clarté et lisibilité
  — Organisation et architecture
  — Respect des bonnes pratiques C++
— **Tests (10%)**
  — Couverture des tests
  — Pertinence des scénarios de test
  — Robustesse face aux entrées invalides

5 Conseils

— Commencez par une implémentation simple et fonctionnelle avant d'optimiser
— Utilisez des structures de données appropriées pour le carnet d'ordres (map, list, vector, etc.)
— Portez une attention particulière à la gestion de la mémoire et aux allocations dynamiques
— Documentez votre code au fur et à mesure
— Utilisez des benchmarks pour mesurer les améliorations de performance

6 Annexe : Exemple de Cas de Test

6.1 Exemple d'entrée (input.csv)

```
timestamp,order_id,instrument,side,type,quantity,price,action
1617278400000000000,1,AAPL,BUY,LIMIT,100,150.25,NEW
1617278400000000100,2,AAPL,SELL,LIMIT,50,150.25,NEW
1617278400000000200,3,AAPL,SELL,LIMIT,60,150.30,NEW
1617278400000000300,4,AAPL,BUY,LIMIT,40,150.20,NEW
1617278400000000400,1,AAPL,BUY,LIMIT,100,150.30,MODIFY
1617278400000000500,3,AAPL,SELL,LIMIT,60,0,CANCEL
```

6.2 Exemple de sortie attendue (output.csv)

```
timestamp,order_id,instrument,side,type,quantity,price,action,status,executed_quanti
1617278400000000000,1,AAPL,BUY,LIMIT,100,150.25,NEW,PENDING,0,0,0
1617278400000000100,2,AAPL,SELL,LIMIT,0,150.25,NEW,EXECUTED,50,150.25,1
1617278400000000100,1,AAPL,BUY,LIMIT,50,150.25,NEW,PARTIALLY_EXECUTED,50,150.25,2
1617278400000000200,3,AAPL,SELL,LIMIT,60,150.30,NEW,PENDING,0,0,0
1617278400000000300,4,AAPL,BUY,LIMIT,40,150.20,NEW,PENDING,0,0,0
1617278400000000400,1,AAPL,BUY,LIMIT,0,150.30,MODIFY,EXECUTED,50,150.30,3
1617278400000000400,3,AAPL,SELL,LIMIT,10,150.30,NEW,PARTIALLY_EXECUTED,50,150.30,1
1617278400000000500,3,AAPL,SELL,LIMIT,0,0,CANCEL,CANCELED,0,0,0
```

Note Importante

Ce projet vise à vous familiariser avec les concepts fondamentaux des systèmes de trading et vous permettre d'appliquer vos connaissances en programmation C++ dans un contexte réaliste. La précision et la performance sont des aspects cruciaux dans l'industrie financière, et ce projet vous donne l'opportunité de développer ces compétences essentielles.

Bonne chance !