Licence d'informatique
Module de C/Unix

Examen – Partie C

Jean-Luc LEVAIRE

Février 2002

Les documents de cours et TD sont autorisés.
On rendra deux copies séparées pour la partie Unix et pour la partie langage. La partie C est à rendre sur une *copie de couleur blanche.*

1 Questions de cours

Exercice 1
Répondez le plus précisément possible à ces questions.

1. La déclaration suivante est-elle valide ? `int n = 10; char *t[n];`
2. Qu'est-ce qu'un descripteur de fichiers sous Unix ?
3. Quel est le contenu des blocs de données d'un répertoire ?

Exercice 2
Que signifient les déclarations suivantes ?

1. `int (*f(int))(int);`
2. `void map(int *, void (*)(int *));`

2 Bourse

Le but du problème est de réaliser la cotation des valeurs des actions cotées sur une place boursière. Le principe d'une place boursière est de réaliser les transactions entre acheteurs et vendeurs d'actions. Pour une action A, il existe un ensemble d'acheteurs et un ensemble de vendeurs potentiels qui passent des *ordres*. Les ordres d'achat proposent un cours *maximal* d'achat : « je suis prêt à acheter 10 actions A à *moins de* 100 euros l'action ». Les vendeurs proposent eux un cours *minimal* de vente : « je suis prêt à vendre 10 actions A à *au moins* 100 euros l'action ».

Le rôle de la place boursière est de faire correspondre ordres d'achat et ordres de vente, et de réaliser une transaction pour ceux qui sont compatibles. Un ordre d'achat est compatible avec un ordre de vente si le prix proposé à l'achat est *supérieur ou égal* au prix de vente minimal. Si plusieurs ordres d'achat sont compatibles avec un ordre de vente, celui qui sera retenu pour la transaction est celui dont le prix d'achat proposé est le plus grand. Ceci est symétrique pour ordre de vente et ordre d'achat. Un ordre de vente est compatible avec un ordre d'achat si le prix proposé à la vente est *inférieur ou égal* au prix d'achat maximal. Si plusieurs ordres de vente sont compatibles avec un ordre d'achat, celui qui sera retenu pour la transaction est celui dont le prix de vente proposé est le plus petit. Dans les deux cas, la cotation de l'action correspond au prix de la dernière transaction effectuée sur cette action.

Nous ne considérerons dans la suite qu'une seule action A. Un ordre d'achat (resp. de vente) est caractérisé par un identifiant entier unique, un nombre d'actions à acheter (resp. vendre) et un cours maximal d'achat (resp. minimal de vente).

Id: xamc02.tex,v 1.1 2002/02/07 18:01:52 marquet Exp

Exercice 3
On ne distingue pas ordre d'achat et ordre de vente au niveau du typage C, cette distinction se fera quand nécessaire par un type `TypeOrdre`. Proposer un type `Ordre` pour représenter un ordre d'achat ou de vente, et un type `TypeOrdre` pour pouvoir spécifier si un ordre est un ordre d'achat ou de vente. Proposer un type `ListeOrdre` pour représenter une liste d'ordres d'achat ou de vente, ainsi que le type `NoeudOrdre` représentant un nœud de cette liste.

Exercice 4
Écrire les fonctions

```
unsigned int TotalNb(ListeOrdre l);
float Avg(ListeOrdre l);
```

qui retournent respectivement le nombre total d'actions (à l'achat ou à la vente) d'une liste d'ordres `l`, et le cours moyen (d'achat ou de vente) des ordres de cette liste.

Exercice 5
Écrire une procédure

```
void InsertOrdre(Ordre o, TypeOrdre t, ListeOrdre *lp);
```

qui insère un ordre `o` de type `t` dans une liste d'ordres pointée par `lp`. On supposera évidemment que la liste pointée par `lp` ne contient que des ordres de type `t`. *Noter que la méthode d'insertion aura une influence sur les fonctions suivantes, et qu'elle peut différer suivant le type de l'ordre (achat ou vente).*

Exercice 6
Écrire une fonction

```
void SupprimerOrdre(int ordreid, ListeOrdre *lp);
```

qui supprime l'ordre d'identifiant `ordreid` de la liste d'ordres pointée par `lp`. Si l'ordre n'existe pas, la liste ne sera pas modifiée.

Pendant une période de pré-ouverture de 7h30 à 9h00, les ordres arrivent de la part des intermédiaires et sont stockés dans une liste d'ordres de vente et une liste d'ordres d'achat suivant leur type. À l'ouverture, il faut réaliser toutes les transactions possibles à partir des deux listes remplies pendant la période de pré-ouverture: c'est le fixing. Le principe est de parcourir la liste d'ordres de vente dans l'ordre croissant de leurs cours de vente et d'essayer d'exécuter chaque ordre de vente. Pour exécuter un ordre de vente, on prend l'ordre d'achat offrant le plus haut cours, et si il est supérieur au cours de vente, on réalise la transaction. Si l'ordre d'achat offre plus d'actions à l'achat que l'ordre de vente, la transaction se termine en modifiant l'ordre d'achat en conséquence et en supprimant l'ordre de vente. Si l'ordre d'achat offre moins d'actions à l'achat que l'ordre de vente, la transaction est partiellement réalisée: l'ordre d'achat est supprimé et l'ordre de vente est modifié en conséquence. On essaie alors de réaliser complètement la transaction en passant à l'ordre d'achat suivant. Il se peut donc que des transactions restent incomplètement réalisées.

Exercice 7
Écrire une fonction

```
int ExecuteVente(Ordre *o, ListeOrdre *lap);
```

qui exécute l'ordre de vente pointé par `o` avec une liste d'ordres d'achat pointée par `lap`. Le passage se fait ici par pointeur car ordre de vente et liste d'ordres d'achat peuvent être modifiés pendant la transaction. La fonction renvoie 0 si la transaction est partiellement réalisée, et 1 si elle est complètement réalisée. On explicitera l'algorithme en fonction de la méthode d'insertion choisie à l'exercice 5.

Exercice 8
Écrire une fonction

```
float Fixing(ListeOrdre *lvp, ListeOrdre *lap);
```

qui réalise le fixing pour une liste d'ordres de vente pointée par `lvp` et une liste d'ordres d'achat pointée par `lap`. Les deux listes sont modifiées par les transactions réalisées, et la fonction retourne le cours de la dernière transaction effectuée (dans l'ordre de parcours des ordres de vente), qui est le cours à l'ouverture.

Fin du sujet