Nom : Prénom :
Numéro de matricule : Année d'étude :
Groupe :

Examen SESP 1224 : Introduction à l'Informatique et la Programmation (année académique 2003-2004; examen de septembre)

Professeur : Marco Saerens
Assistants : Bouillon, Coyette, Florins, Fouss, Hoang, Mariage, Nguyen & Souchon
Adresse : Université catholique de Louvain
Institut d'Administration et de Gestion
Information Systems Research Unit (ISYS)
Place des Doyens 1
B-1348 Louvain-la-Neuve, Belgique
Téléphone : 010 47.92.46. (Saerens)
Fax : 010 47.83.24.
Courriel : saerens@isys.ucl.ac.be
Web : http://www.isys.ucl.ac.be/etudes/cours/index.htm

Enoncés des problèmes

Vous avez **3 heures** pour résoudre les 3 problèmes suivants. Lorsque vous aurez terminé, ou bien lorsqu'on vous le demandera, vous remettrez vos feuilles. Lisez attentivement les énoncés !
Attention:

- Vérifiez que vous avez huit feuilles au total: deux pages d'énoncé, trois feuilles à compléter (feuilles réponses) et trois feuilles de brouillon. Vous devez impérativement utiliser le papier brouillon fourni.
- Il y a donc une feuille réponse à compléter par problème à résoudre (3 problèmes); vous pouvez écrire sur le recto et le verso.
- Vous devez compléter vos nom et prénom pour chaque feuille réponse ainsi que l'énoncé.
- Il faut rendre toutes les feuilles réponses, même celles qui sont vides.
- N'oubliez pas d'inclure des **commentaires** dans vos programmes, expliquant ce que vous faites.

Voici les énoncés des trois problèmes (de difficulté croissante) à résoudre. Chaque problème est coté sur 5 points (cote maximale : 15); la note finale sera bien entendu ramenée à 20.

Problème 1 (45 min; 5 points).
Ecrire une méthode, appelée *oddOrEven*, ayant comme paramètre d'entrée un vecteur *e* (tableau à une dimension) d'entiers, et retournant en sortie un deuxième vecteur (tableau à une dimension renvoyé par la méthode) contenant uniquement deux valeurs. La première valeur sera la somme des nombres pairs présents dans le vecteur e, et la seconde valeur sera la somme des nombres impairs présents dans ce même vecteur.

Par exemple si nous avons $e = \{1, 2, 3, 4\}$, la méthode doit renvoyer le vecteur d'entiers $\{6, 4\}$.

Bien entendu, cette méthode doit être suffisamment générale que pour fonctionner à partir de vecteurs de toute taille (pas seulement des vecteurs de 4 éléments !):

```
public int[] oddOrEven(int[] e)
{
    // à implémenter
}
```

Notez qu'un nombre *x* est pair si et seulement si le reste de la division de *x* par 2 est nul: $x\%2 == 0$.

Problème 2 (45 min; 5 points).
Ecrire une méthode, *additionMatrice*, qui, au départ de deux matrices de nombres entiers, renvoie (instruction `return`) la matrice qui représente l'addition de ces deux matrices de départ, si cette addition est possible; sinon elle affiche un message d'erreur et renvoie `null`. Ces deux matrices ainsi que le nombre de lignes (*l*) et de colonnes (*c*) de chaque matrice sont passés en argument de la méthode:

```
public int[][] additionMatrice(int[][] Matrice1, int l1, int c1,
       int[][] Matrice2, int l2, int c2)
{
    // à implémenter
}
```

Rappel: pour effectuer l'addition de deux matrices, la condition préalable est que ces deux matrices aient la même taille (le même nombre de lignes et le même nombre de colonnes). Si cette condition est respectée, l'addition est effectuée comme suit: `Resultat[i][j] = Matrice1[i][j] + Matrice2[i][j]` pour tout *i*, *j*, où *i* et *j* sont les indices de ligne et de colonne.

Par exemple :

1) **Matrice1** :

1	3	5	7
2	8	5	7
1	4	9	6

Matrice2 :

5	7	9
10	12	6
8	4	3

La méthode affichera : *Erreur*

2) **Matrice1** :

1	5	9
4	8	6
2	5	3

Matrice2 :

3	2	5
4	6	8
1	2	7

La méthode retourne la matrice:

4	7	14
8	14	14
3	7	10

Problème 3 (60 min; 5 points).
La matrice 'Results' contient le résultat (le meilleur saut) de chacun des athlètes participant à l'épreuve de qualification (avec un maximum de 50 athlètes) pour la finale olympique du triple saut. La limite de qualification pour la finale a été fixée par les organisateurs à 1695 cm = 16,95 m. Le principe est le suivant : tout athlète ayant atteint cette distance est automatiquement qualifié pour la finale (et ce, quelque soit le nombre d'athlètes ayant atteint cette distance). Par contre, si moins de 12 athlètes ont atteint la limite de qualification, une procédure particulière doit être lancée.

Ecrire une méthode, *qualification*, qui, au départ d'une matrice d'entiers contenant les résultats obtenus par les différents athlètes (la première ligne de la matrice contient les numéros de dossard des athlètes et la deuxième ligne le résultat de l'athlète correspondant, en cm), renvoie une nouvelle matrice (de taille maximale 50) contenant uniquement les athlètes qualifiés pour la finale. Si le nombre de qualifiés est inférieur à 12, la méthode devra afficher à l'écran un message précisant le nombre de qualifiés. On fournit également comme argument à la méthode le nombre de participants aux qualifications, *n*.

```
public int[ ][ ] qualification(int[ ][ ] Results, int n)
{
    // à implémenter
}
```

Par exemple :

Results =

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1698	1705	1540	0	1792	1706	1632	1693	1652	1700	1695	1725	1766	0	1538	1699	1722	1601	1658	1799

Résultat : **Q** =

1	2	5	6	10	11	12	13	16	17	20
1698	1705	1792	1706	1700	1695	1725	1766	1699	1722	1799

La matrice **Q** contient les numéros de dossard et les résultats des 11 participants qualifiés. Comme il y a moins de 12 qualifiés, voici le message affiché à l'écran : "Attention, il n'y a que 11 qualifiés".

Il ne nous reste plus qu'à vous souhaiter bonne chance.

02/09/2004

Feuille de réponse au problème 1

Nom :
Prénom :
Numéro de matricule :
Année d'étude :
Groupe :

02/09/2004

Feuille de réponse au problème 2

Nom :
Prénom :
Numéro de matricule :
Année d'étude :
Groupe :

02/09/2004

Feuille de réponse au problème 3

Nom : Prénom :

Numéro de matricule : Année d'étude :

Groupe :