UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES

Méthodes informatiques pour physiciens

introduction à C++ et
résolution de problèmes de physique par ordinateur

Corrigé 4

Professeur : Alessandro Bravar
Alessandro.Bravar@unige.ch

Université de Genève
Section de Physique

Semestre de printemps 2015

Références :

M. Michelou et M. Rieder
Programmation orientée objets en C++

J.C. Chappelier et F. Seydoux
C++ par la pratique

B. Stroustrup
Programmation *Principes et pratique avec C++*

`http://dpnc.unige.ch/~bravar/C++2015/L4` :
pour les notes du cours, les exercices et les corrigés

4.1 Questions

1. **Combien de types différents peuvent avoir les éléments d'un tableau ?**
   Seulement un, tous les éléments d'un tableau sont du même type.
2. **De quel type doit être un indice de tableau et dans quel intervalle doit-il se situer ?**
   Un indice de tableau doit être une variable entière (`short`, `unsigned short`, `int`, `unsigned int`, `long`, `unsigned long`). L'indice doit se situer entre 0 et n-1, si la dimension du tableau est `n`.
3. **a. Quelles seront les valeurs des éléments d'un tableau si sa déclaration n'inclut pas d'initialisateur ?**
   S'il n'y a pas d'initialisateur dans la déclaration d'un tableau les valeurs des éléments sont imprévisibles, comme pour les variables non initialisées.
   b. Quelles seront les valeurs des éléments d'un tableau si sa déclaration inclut un initialisateur comportant un nombre inférieur d'éléments du tableau ?
   Si le tableau a `n` éléments et l'initialisetur en inclut seulement `k`, les `k` premiers éléments seront initialisés avec les valeurs choisies dans la déclaration, le reste prendra la valeur 0 (ou caractère vide, dans le cas d'un tableau de caractères).
4. **Que se passe-t-il si l'initialisateur d'un tableau contient d'avantage de valeurs que la taille du tableau ?**
   Dans ce cas le compilateur générera un message d'erreur. Par exemple, avec le compilateur GNU sur Linux et MAC l'erreur est la suivante :
   ```
   > g++ test.cpp -o test
   test.cpp: In function 'int main()':
   test.cpp:17:
   error: too many initializers for 'int [7]
   ```
5. **Lorsqu'un tableau multidimensionnel est passé à une fonction, pourquoi C++ exige que toutes les dimensions soient spécifiées sauf la première ?**
   Un tableau multidimensionnel est stocké dans la mémoire séquentiellement, pour ranger correctement le tableau la fonction doit connaitre toutes les dimensions sauf le nombre de lignes qui est obtenu en parcourant les éléments jusqu'au dernier.
6. **Citez le premier et le dernier élément du `tableau[25]`.**
   Ces éléments sont `tableau[0]` et `tableau[24]`.
7. **Qu'est-ce que donne `a[-5]` ?**
   Le résultat de l'instruction `a[-5]` est la valeur stockée dans l'adresse mémoire 5 places **avant** le premier élément du tableau, l'adresse exacte depend du type de tableau, et n'aura probablement aucun sens.

4.2 Trouvez l'erreur

Les programmes corrigés sont donnés ci-dessous. Les erreurs sont les suivantes :

1. Dans le premier programme le deuxième indice du tableau `j` dépasse la taille du tableau, dont la dernière ligne est `tab[i][3]`. Le programme essaye d'accéder à la colonne `tab[i][4]`, qui peut contenir n'importe quoi. De plus, les valeurs de la dernière ligne ne sont pas remplies, parce que l'indice `i` va jusqu'à 3.
   Une version correcte est montrée ci-dessous.

```
1 int tab[5][4];
2 for(int i=0; i<5; i++)
```

```
3       for(int j=0; j<4; j++)
4           tab[i][j] = i+j;
```

2. Dans le deuxième programme l'erreur est à nouveau liée aux indices hors rang du tableau. Cette fois pour les lignes et les colonnes, car la boucle `for` inclut `i=5` et `j=5`. La bonne version de ce code est montré ci-dessous. Notez que pour les nombres entiers la comparaison `i < n` est équivalente à `i <= n-1` .

```
1  int tab[5][4];
2  for(int i=0; i<=4; i++)
3      for(int j=0; j<=3; j++)
4          tab[i][j] = 0;
```

4.3 Exercices

Exercice 1 : Initialisez un tableau !

Dans le programme **Ex_tab1.cpp** l'on définit la constante (`SIZE`) au début du programme qui indique la taille du tableau. Le programme demande ensuite à l'utilisateur d'entrer les valeurs de chaque élément. Enfin le programme affiche les valeurs des éléments du tableau dans l'ordre inverse.

Ex_tab.cpp

```
1  //exemple tableau unidimensionel
2  #include <iostream>
3
4  using namespace std;
5
6  int main() {
7    const int SIZE = 6;
8    int a[SIZE];  //la dimension du tableau ne peut pas changer
9
10   cout << "Entrez " << SIZE << " nombres entiers : " << endl;
11   for (int i=0; i<SIZE; i++) {
12     cout << i+1 <<": ";
13     cin >> a[i];
14   }
15
16   cout << "Maintenant ils sont dans l'ordre inverse: " << endl;
17   for (int i=SIZE-1; i>=0; i--)
18     cout << a[i] << endl;
19
20   return 0;
21 }
```

Le programme **Ex_tab2.cpp** produit le même résultat. Dans ce programme la dimension du tableau n'est pas constante, mais elle est défini par l'utilisateur après le lancement du programme. Cette procédure n'est pas acceptée par le normes ANSI du C++, mais plusieurs compilateurs acceptent cette procédure. Dans ce cours, nous ne l'acceptons pas. La taille du tableau doit être connue avant le lancement du programme.

Ex_tab2.cpp

```
1  //exemple de tableau unidimensionel
2  #include <iostream>
3
4  using namespace std;
5
```

```
6  int main() {
7    int dim;
8    cout << "Quelle est la dimension du tableau ? ";
9    cin >> dim;
10   //selon la norme ANSI cette procedure est interdit
11   //la dimension du tableau doit etre connue avant l'execution du programme
12   //mais elle est toleree par la majorite des compilateurs inclus le notre
13   int a[dim];
14
15   cout << "Entrez " << dim << " nombres entiers: " << endl;
16   for (int i=0; i<dim; i++) {
17     cout << i+1 <<": ";
18     cin >> a[i];
19   }
20
21   cout << "Maintenant ils sont dans l'ordre inverse: " << endl;
22   for (int i=dim-1; i>=0; i--)
23     cout << a[i] << endl;
24
25   return 0;
26 }
```

Exercice 2 : Calculez le produit scalaire et vectoriel de deux vecteurs !

Le programme **Prod_scal.cpp** calcule le produit scalaire de deux vecteurs selon la formule

$$a \cdot b = \sum_{i=0}^{n} a_i b_i \ .$$

On utilise une boucle `for` pour parcourir les deux vecteurs (deux tableaux uni-dimensionnels).

Prod_scal.cpp

```
1  //produit scalaire de deux vecteurs
2  #include <iostream>
3
4  using namespace std;
5
6  int main() {
7    const int SIZE = 3;
8    double a[SIZE];
9    double b[SIZE];
10   cout << "Entrez le premier vecteur :" << endl;
11   cin >> a[0] >> a[1] >> a[2];
12   cout << "Entrez le deuxieme vecteur :" << endl;
13   cin >> b[0] >> b[1] >> b[2];
14
15   double prodScal = 0.;
16   for (int i=0; i<=2; i++)
17     prodScal += a[i]*b[i];
18
19   cout << "Le produit scalaire de a et b est " << prodScal << endl;
20
21   return 0;
22 }
```

Le programme **Prod_scalF.cpp** produit le même résultat en utilisant la fonction `prodScal`. Comme les vecteurs sont uni-dimensionnels il ne faut pas définir leur dimension, elles sont passées par valeur à la fonction. Si la dimension ces deux tableaux est

omise dans l'appel de la fonction `prodScal`, le programme utilisera la valeur par défaut, qui est ici fixée à 3.

Prod_scalF.cpp

```
1  //produit scalaire de deux vecteurs avec une fonction
2  #include <iostream>
3
4  using namespace std;
5
6  //declaraction de la fonction pour le produit scalaire
7  double prodScal(double a[], double b[], int n = 3);
8
9  int main() {
10   const int SIZE = 3;
11   double a[SIZE];
12   double b[SIZE];
13   cout << "Entrez le premier vecteur :" << endl;
14   cin >> a[0] >> a[1] >> a[2];
15   cout << "Entrez le deuxieme vecteur :" << endl;
16   cin >> b[0] >> b[1] >> b[2];
17
18   cout << "Le produit scalaire de a et b est " << prodScal(a,b,SIZE) <<
         endl;
19
20   return 0;
21 }
22
23 //definition de la fonction
24 double prodScal(double a[], double b[], int n) {
25   double scalaire = 0.;
26   for (int i=0; i<n; i++)
27     scalaire += a[i]*b[i];
28   return scalaire;
29 }
```

Le programme **Prod_vect.cpp** calcule le produit vectoriel de deux vecteurs $\vec{a} = (a_1, a_2, a_3)$ et $\vec{b} = (b_1, b_2, b_3)$ en utilisant la formule

$$\vec{c} = \vec{a} \times \vec{b} = \mathrm{Det} \begin{pmatrix} \hat{i} & \hat{j} & \hat{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{pmatrix}$$

où $\hat{i}$, $\hat{j}$, $\hat{k}$ sont des vecteurs unitaires dans les directions x, y, z. Dans le deuxième programme le produit vectoriel est calculé avec une fonction.

Prod_vect.cpp

```
1  //produit vectoriel de deux vecteurs
2  #include <iostream>
3
4  using namespace std;
5
6  int main() {
7    const int SIZE = 3;
8    double a[SIZE];
9    double b[SIZE];
10   cout << "Entrez le premier vecteur :" << endl;
11   cin >> a[0] >> a[1] >> a[2];
12   cout << "Entrez le deuxiem vecteur :" << endl;
```

```
13    cin >> b[0] >> b[1] >> b[2];
14
15    double c[SIZE];
16    c[0] = a[1]*b[2] - a[2]*b[1];
17    c[1] = a[2]*b[0] - a[0]*b[2];
18    c[2] = a[0]*b[1] - a[1]*b[0];
19
20    cout << "Le produit vectoriel de a et b est "
21         << c[0] << " " << c[1] << " " << c[2] << endl;
22
23    return 0;
24  }
```

Prod_vectF.cpp

```
1   //produit vectoriel de deux vecteurs avec une fonction
2   #include <iostream>
3
4   using namespace std;
5
6   //declaration de la fonction pour le produit vecoriel
7   void prodVect(const double a[], const double b[], double c[], int n = 3);
8
9   int main() {
10    const int SIZE = 3;
11    double a[SIZE];
12    double b[SIZE];
13    double c[SIZE];
14    cout << "Entrez le premier vecteur :" << endl;
15    cin >> a[0] >> a[1] >> a[2];
16    cout << "Entrez le deuxiem vecteur :" << endl;
17    cin >> b[0] >> b[1] >> b[2];
18
19    prodVect(a,b,c,SIZE);
20
21    cout << "Le produit vectoriel de a et b est "
22         << c[0] << " " << c[1] << " " << c[2] << endl;
23
24    return 0;
25  }
26
27  void prodVect(const double a[], const double b[], double c[], int n) {
28    c[0] = a[1]*b[2] - a[2]*b[1];
29    c[1] = a[2]*b[0] - a[0]*b[2];
30    c[2] = a[0]*b[1] - a[1]*b[0];
31  }
```

Exercice 3 : Rangez un tableau de 10 (puis n) éléments en ordre ascendant !

Voir le programme **RangeVect.cpp**. Dans ce programme le rangement se fait par la comparaison de chaque élément `i` avec les `n-i` suivants : si la valeur de l'élément d'indice plus petit est plus grande, on échange les valeurs dans le tableau. On utilise une variable temporaire pour l'échange des valeurs.

RangeVect.cpp

```
1   //ce programme range les elements d'un tableau
2   #include <iostream>
3   #include <fstream>
4   #include <iomanip>
```

```
5
6  using namespace std;
7
8  int main() {
9    const int MAX=10;
10   double v[MAX];
11
12   //saisi des donnees
13   cout << "Entrez " << MAX << " donnees ! \n";
14   for (int i=0; i<MAX; i++) {
15     cout << "Element " << i+1 << " : ";    cin >> v[i];
16   }
17
18   //ici on range le vecteur
19   double temp;
20   for (int i=0; i<MAX; i++)
21     for (int j=i+1; j<MAX; j++)
22       if (v[i]>v[j]) {
23         temp = v[j];
24         v[j] = v[i];
25         v[i] = temp;
26       }
27
28   cout << "Maintenant ils sont dans l'ordre ascendant : " << endl;
29   for (int i=0; i<MAX; i++)
30     cout << v[i] << endl;
31
32   return 0;
33 }
```

Exercice 4 : Passez des tableaux unidimensionnels aux fonctions !
Voir p. ex. les programmes **Prod_scalF.cpp** et **Prod_vectF.cpp** (Exercice 2).

Exercice 5 : Imprimez la transposée d'une matrice !
Pour transposer une matrice representée comme un tableau bi-dimensionnel on doit juste échanger les indices entre les lignes et les colonnes, c'est à dire `T(M[i][j]) = M[j][i]`. Le programme proposé affiche seulement la matrice transposée sans produire un nouveau tableau (matrice).

TransposeeMatrice.cpp

```
1  //transpose d'une matrice
2  #include <iostream>
3  #include <iomanip>
4
5  using namespace std;
6
7  int main() {
8    const int N = 100;
9
10   int m, n;
11   cout << "Entrez les domensions de la matrice (max. 100 x 100) ?\n";
12   cin >> m >> n;
13   if(m > N || n > N) {
14     cout << "Les dimensions de la matrice sont trop grands !" << endl;
15     system("PAUSE");
16     return 1;
17   }
18
```

```
19    double matrice[N][N];
20    cout << "Entrez le " << m << " x " << n << " elements (ligne apres ligne)
          :" << endl;
21    for (int i=0; i<m; i++)
22      for (int j=0; j<n; j++)
23        cin >> matrice[i][j];
24
25    cout << "\nLa matrice de depart est : " << endl;
26    cout << setw(5) << "/" << setw(n*10+5) << "\\" << endl;
27    for (int i=0; i<m; i++) {
28      cout << setw(5) << "|";
29      for (int j=0; j<n; j++)
30        cout << setw(10) << setprecision(5) << matrice[i][j];
31      cout << setw(5) << "|" << endl;
32    }
33    cout << setw(5) << "\\" <<setw(n*10+5) << "/" << endl;
34
35    cout << endl << "et sa transposee est :" << endl;
36    cout << setw(5) << "/" << setw(m*10+5) << "\\" << endl;
37    for (int i=0; i<n; i++) {
38      cout << setw(5) <<"|";
39      for (int j=0; j<m; j++)
40        cout << setw(10) << setprecision(5) << matrice[j][i];
41      cout << setw(5) << "|" << endl;
42    }
43    cout << setw(5) << "\\" << setw(m*10+5) << "/" << endl << endl;
44
45    return 0;
46  }
```

Exercice 6 : Ecrivez un programme pour effectuer une rotation d'un vecteur en 3 dimensions !

On utilise les angles d'Euler pour effectuer la rotation. Leur définition est donnée dans la figure 1. L'ordre de la rotation est important, la première rotation correspond à la matrice la plus à droite qui multiplie le vecteur colonne :

$$\vec{v'} \;=\; R\,\vec{v} = R_B R_C R_D\,\vec{v}$$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Le morceau de code qui fait la multiplication entre la matrice de rotation complète (correspondant à la multiplication des trois matrices dans l'ordre correct) et le vecteur colonne est donné ci dessous.

```
1  for (int i=0; i<3; i++)
2    for (int j=0; j<3; j++)
3      vecteurR[i] += matRot[i][j] * vecteur[j];
```

Notez qu'il n'y a pas d'accolades et un seul ; . Il s'agit d'une seule instruction.
Au début le programme demande à l'utilisateur d'entrer les tableaux et les valeurs des angles de rotation. Ensuite, le programme affiche les tableaux de rotation et calcule le produit des trois tableaux en utilisant le morceau du code donné ci dessus. Enfin le programme calcule le produit du tableau de rotation par le vecteur initial.
RotationVecteur.cpp

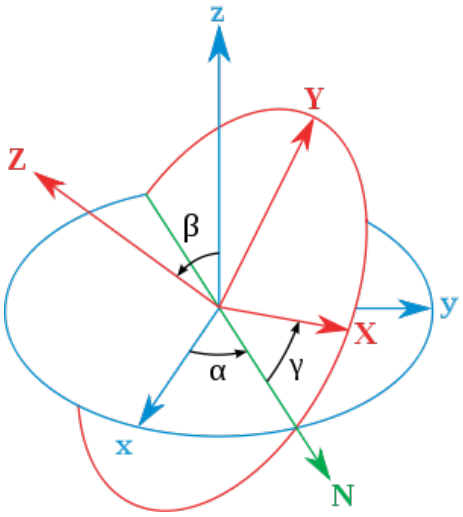

FIGURE 1 – Définition des angles d'Euler décrivant une rotation en trois dimensions. La première rotation s'effectue autour de l'axe z, la deuxième autour de l'axe x', et la troisième autour de l'axe z'.

```
1  //rotation d'un vecteur
2  #include <iostream>
3  #include <iomanip>
4  #include <cmath>
5
6  using namespace std;
7
8  int main() {
9    double vecteur[3] = {0.};
10   cout << "Entrez les 3 coordonees du vecteur :" << endl;
11   for (int k=0; k<3 ; k++)
12     cin >> vecteur[k];
13
14   double angles[3] = {0.};
15   cout << "Entrez les 3 angles d'Euler de la rotation (en degrees) :\n";
16   for (int k=0; k<3 ; k++) {
17     cin >> angles[k];
18     angles[k] *= M_PI/180.0;        //angles en radians
19   }
20
21   //d'abord il faut calculer les matrices de rotation:
22   //MatD pour l'axe z, MatC pour l'axe x', et MatB pour l'axe z'
23   double matD[3][3] = {0.};
24   double matC[3][3] = {0.};
25   double matB[3][3] = {0.};
26
27   matD[0][0] = matD[1][1] = cos(angles[0]);
28   matD[0][1] = -sin(angles[0]);
29   matD[2][2] = 1.0;
30   matD[1][0] = -1.0 * matD[0][1];
31
32   matC[0][0] = 1.0;
33   matC[1][2] = -sin(angles[1]);
34   matC[2][1] = -1.0 * matC[1][2];
35   matC[2][2] = matC[1][1] = cos(angles[1]);
36
37   matB[0][0] = matB[1][1] = cos(angles[2]);
```

```
38    matB[0][1] = -sin(angles[2]);
39    matB[1][0] = -1.0 * matB[0][1];
40    matB[2][2] = 1.0;
41
42    //la matrice de rotation est donne par matRot = matB*matC*matD
43    //on calcule le produit matriciel de deux matrices a chaque fois
44    double matM[3][3] = {0.};
45    double matRot[3][3] = {0.};
46
47    //matC*matD -> matM
48    for (int i=0; i<3; i++)
49      for (int j=0; j<3; j++)
50        for (int k=0; k<3; k++)
51          matM[i][j] += matC[i][k] * matD[k][j];
52
53    //matB*matM -> matRot
54    for (int i=0; i<3; i++)
55      for (int j=0; j<3; j++)
56        for (int k=0; k<3; k++)
57          matRot[i][j] += matB[i][k] * matM[k][j];
58
59    cout << "\nLa matrice de rotation est :" << endl;
60    cout << setw(5) << "/" << setw(3*12+5) << "\\" << endl;
61    for (int i=0; i<3; i++) {
62      cout << setw(5) << "|";
63      for (int j=0; j<3; j++)
64        cout << setw(12) << setprecision(5) << matRot[i][j];
65      cout << setw(5) << "|" << endl;
66    }
67    cout << setw(5) << "\\" << setw(3*12+5) << "/" << endl;
68
69    cout << "Le vecteur apres la rotation est :" << endl;
70    double vecteurR[3]={0.0};
71    for (int i=0; i<3; i++) {
72      for (int j=0; j<3; j++)
73        vecteurR[i] += matRot[i][j]*vecteur[j];
74      cout << vecteurR[i] << endl;
75    }
76
77    return 0;
78  }
```

Exercice 7 : Calculez le produit de deux matrices !

Dans l'exercice précédent (rotation d'un vecteur) on a déjà calculé le produit des trois matrices, qu'on doit faire par paires. L'algorithme pour la multiplication de deux matrices 3×3 est montré dans le cadre ci dessous :

```
1  double matriceC[N][N] = {0.};
2  for (int i=0; i<3; i++)
3    for (int j=0; j<3; j++)
4      for (int k=0; k<3; k++)
5        matriceC[i][j] += matriceA[i][k] * matriceB[k][j];
```

Le programme complet qui multiplie deux matrices et affiche le résultat sur l'écran est montré ici.

ProdMatrices.cpp

```
1  //produit de deux matrices
2  #include <iostream>
```

```
3  #include <iomanip>
4  #include <cmath>
5
6  using namespace std;
7
8  int main() {
9    const int N = 100;
10
11   int l, m;
12   cout << "Entrez les dimensions de la 1ere matrice (max. 100 x 100) !\n";
13   cout << "Nombre de lignes : ";
14   cin >> l;
15   cout << "Nombre de colones : ";
16   cin >> m;
17   if (l > N || m > N) {
18     cout << "Les dimensions de la matrice sont trop grands !" << endl;
19     return 1;
20   }
21
22   double matriceA[N][N] = {0.};
23   cout << "Entrez le " << l << " x " << m
24        << " elements de la 1ere matrice (ligne apres ligne) :" << endl;
25   for (int i=0; i<l; i++)
26     for (int j=0; j<m; j++)
27       cin >> matriceA[i][j];
28
29   int m2, n;
30   cout << "Entrez les dimensions de la 2eme matrice (max. 100 x 100) !\n";
31   cout << "Nombre de lignes : ";
32   cin >> m2;
33   cout << "Nombre de colones : ";
34   cin >> n;
35   if (m2 > N || n > N) {
36     cout << "Les dimensions de la matrice sont trop grands !" << endl;
37     return 1;
38   }
39   if (m2 != m) {
40     cout << "Le nombre de lignes de la 2eme matrice ne correspond pas au
         nombre de colonnes de la 1ere !" << endl;
41     return 1;
42   }
43
44   double matriceB[N][N] = {0.};
45   cout << "Entrez le " << m << " x " << n
46        << " elements de la 2eme matrice (ligne apres ligne) :" << endl;
47   for (int i=0; i<m; i++)
48     for (int j=0; j<n; j++)
49       cin >> matriceB[i][j];
50
51   //multiplication ligne par colonne
52   double matriceC[N][N] = {0.};
53   for (int i=0; i<l; i++)
54     for (int k=0; k<n; k++)
55       for (int j=0; j<m; j++)
56       matriceC[i][k] += matriceA[i][j] * matriceB[j][k];
57
58   //affichage du produit
59   cout << endl << "La matrice produit est :" << endl;
60   cout << setw(5) << "/" << setw(l*12+5) << "\\" << endl;
61   for (int i=0; i<l; i++) {
```

```
62      cout << setw(5) << "|";
63      for (int j=0; j<n; j++)
64        cout << setw(12) << setprecision(5) << matriceC[i][j];
65      cout << setw(5) <<"|" << endl;
66    }
67    cout << setw(5) << "\\" << setw(l*12+5) << "/" <<endl;
68
69    return 0;
70  }
```

Exercice 8 : Calculez le déterminant d'une matrice !

Le programme suivant calcule explicitement le déterminant d'une matrice 3×3 donnée.

Matrice3x3Det.cpp

```
1  //calcul du determinant d'une matrice 3 * 3
2  #include <iostream>
3  #include <iomanip>
4
5  using namespace std;
6
7  int main() {
8    double matrice[3][3];
9    cout << "Entrez les 3 x 3 elements de la matrice :" << endl;
10   for (int i=0; i<3; i++)
11     for (int j=0; j<3; j++)
12       cin >> matrice[i][j];
13
14   //calcul du determinant:
15   //on applique la formule du determinant pour une matrice 3 * 3
16   double det;
17   det = matrice[0][0]*matrice[1][1]*matrice[2][2] -
18         matrice[2][0]*matrice[1][1]*matrice[0][2]+
19         matrice[0][1]*matrice[1][2]*matrice[2][0] -
20         matrice[2][1]*matrice[1][2]*matrice[0][0]+
21         matrice[0][2]*matrice[1][0]*matrice[2][1] -
22         matrice[2][2]*matrice[1][0]*matrice[0][1];
23   cout << "Le determinant de la matrice est :" << det << endl;
24
25   return 0;
26 }
```

Dans le cas géneral la solution est donnée par l'algorithme dans le programme **MatriceDet.cpp**. Nous avons utilisé la recursivité pour calculer le déterminant de n'importe quelle matrice carrée. On utilise une fonction qui s'appelle elle-même, montrée dans le cadre ci dessous. Le programme calcul le déterminant du tableau

$$M1 = \begin{pmatrix} a_{0,0} & a_{0,1} & .. & a_{0,n-1} \\ a_{1,0} & a_{1,1} & .. & a_{1,n-1} \\ .. & .. & .. & .. \\ a_{n-1,0} & a_{n-1,1} & .. & a_{n-1,n-1} \end{pmatrix}$$

avec la fonction

$$\mathrm{Det}(M1) = \sum_{j=0}^{j=n-1} (-1)^j a_{0,j} \mathrm{Det}(M1, 0, j)$$

où

$$\mathrm{Det}(M1,0,j) = \begin{pmatrix} a_{1,0} & a_{1,1} & .. & a_{1,j-1} & a_{1,j+1} & .. & a_{1,n-1} \\ .. & .. & .. & a_{2,j-1} & a_{2,j+1} & .. & .. \\ a_{n-1,0} & a_{n-1,1} & .. & a_{n-1,j-1} & a_{n-1,j+1} & .. & a_{n-1,n-1} \end{pmatrix}$$

Pour faire ça nous allons écrire chaque fois le sous-tableau :

```
1  //calcule du determinant par recursivite
2  double detMat(double mat[][N], int n) {
3    double det = 0.;
4    double k = 1.;
5    if (n==1)   //n==1 il n'y a pas de sousmatrice
6      det = mat[0][0];
7    else
8      for (int j=0; j<n; j++) {   //sousmatrices (n-1)*(n-1)
9        double auxM[N][N];
10       for (int ai=0; ai<n-1; ai++) {
11         for(int aj=0; aj<n-1; aj++) {
12           auxM[ai][aj]=0.;
13         if (aj<j) auxM[ai][aj] += mat[ai+1][aj];
14         else if (aj>=j) auxM[ai][aj] += mat[ai+1][aj+1];
15         }
16       }
17       //on prenne la premiere ligne de la matrice
18       //pour chaque element on demande le determinant de la suosmatrice
19       det = det + k*mat[0][j] * detMat(auxM,n-1);
20       k *= -1.; // k=(-1)^j
21     }
22
23   return det;
24 }
```

Nous avons aussi inclus une fonction pour afficher la matrice.

Exercice 9 : Ecrivez une fonction pour additionner, multiplier, ... 2 matrices !
Dans le programme **MatricesCalculateur.cpp** vous trouverez des fonctions pour multiplier, additionner et soustraire deux matrices. P. ex., dans le cadre suivant la fonction qui soustrait deux matrices est montrée. La fonction est de type `void` parce qu'elle ne retourne aucune valeur, mais le résultat est stocké dans la troisième matrice `MatR`.

```
1  void diffMat(double Mat1[][N], double Mat2[][N], double MatR[][N], int n) {
2    for (int i=0; i<n; i++)
3      for (int j=0; j<n; j++)
4        MatR[i][j] = Mat1[i][j] - Mat2[i][j];
5  }
```

Exercice 11 : Reprenez l'exercice 3. Lisez les données depuis un fichier, rangez les, écrivez le résultat dans un autre fichier.
La solution pour cet exercice est donnée au point 3. Pour lire les données à partir du ficher on utilise la bibliothèque `<fstream>`. Après avoir associé le fichier d'entré à l'objet `fin` avec la fonction `ifstream` on lit les données jusqu'à la fin du fichier en remplissant le tableau. Pour écrire les données du vecteur rangé dans un fichier on associe l'objet `fout` au fichier de sortie avec la fonction `ofstream`, et ensuite on écrit dans le ficher avec l'instruction

`fout << setw(15) << setprecision(9) << v[i] << endl;`.

RangeVect_2.cpp

```
1  //ce programme range les elements d'un tableau avec une fonction;
2  //les elements du tableau sont lus depuis un fichier
3  #include <iostream>
4  #include <fstream>
5  #include <iomanip>
6
7  using namespace std;
8
9  //declaration de la fonction
10 void rangeVect(double vect[], int n);
11
12 int main() {
13   const int MAX=10000;
14   double v[MAX];
15
16   char nomFichierEntree[256];
17   cout << "Quel est le fichier d'entree ? ";
18   cin >> nomFichierEntree;
19
20   //lecture depuis le fichier
21   ifstream fin(nomFichierEntree);
22   int n=0;
23   while (1) {
24     fin >> v[n];
25     if (!fin) {
26       cout << "Les donnees sont finies !" << endl;
27       break;
28     }
29     if (n>=MAX) {
30       cout << "Il y a trop des elements ! On rangera les premiers "
31            << MAX << " elements" << endl;
32       break;
33     }
34     n++;
35   }
36   cout << "Il y a " << n << " elements a ranger."<<endl;
37
38   //ici on appele la foncion qui fait le rangement
39   rangeVect(v,n);
40
41   char nomFichierSortie[256];
42   cout << "Quel est le fichier de sortie ? ";
43   cin >> nomFichierSortie;
44
45   //apres avoir rangee les elements, on les imprime sur l'ecran et
46   //dans le fichier de sortie
47   ofstream fout(nomFichierSortie);
48   for (int i=0; i<n; i++) {
49     cout << setw(10) << setprecision(4) << v[i] << endl;
50     fout << setw(15) << setprecision(9) << v[i] << endl;
51   }
52
53   return 0;
54 }
55
56 //definition de la fonction qui range le tableau
57 void rangeVect(double  vect[], int n) {
58   double temp;
59   for(int i=0; i<n; i++)
```

```
60      for(int j=i+1; j<n; j++)
61        if (vect[i]>vect[j]) {
62          temp = vect[j];
63          vect[j] = vect[i];
64          vect[i] = temp;
65        }
66  }
```

Exercice 12 : Reprenez le problème 1 de la 2ème leçon : à chaque rebond, écrivez dans un fichier la vitesse juste après le rebond et la hauteur maximale atteignable après le rebond. La masse de la balle est 100 g.

Le problème a déjà été étudié dans le corrigé 2. Ici on utilise l'objet `fout` de type `ofstream` pour écrire dans le fichier (n'oubliez pas d'introduire la bibliothèque `fstream`!). Le programme écrit les valeurs de la vitesse à chaque rebond avec l'instruction

`fout << "rebond : " << compt+1 << vitesse << h << endl;` .

Rebond_file.cpp

```
1   //reonds d'une balle avec perte d'energie a chaque rebond
2   //les donnees sont enregistrees dans un fichier
3   #include <iostream>
4   #include <cmath>
5   #include <fstream>
6   #include <iomanip>
7
8   using namespace std;
9
10  //on utilise des unite MKS
11  const double GN = 9.81;
12  const double masse = 0.1;    //masse de la balle
13
14  int main() {
15    double h;
16    cout << "Entrez la hauteur initiale h en metres : ";
17    cin >> h;
18
19    double eps;
20    cout << "Entrez la perte de energie a chaque rebond, en % : ";
21    cin >> eps;
22    eps = eps/100.;
23
24    int n;
25    cout << "Combien de rebonds voulez-vous evaluer ? ";
26    cin >> n;
27
28    //creation d'un fichier ou enregistrer les donnees
29    ofstream fout("rebond.dat");
30    if (!fout) {
31      cout << "Je ne peux pas ouvrir le fichier ! Je m'arrete !" << endl;
32      system("PAUSE");
33      return -1;
34    }
35
36    double energie, vitesse;
37    for (int compt=0;  compt<n; ++compt) {
38      energie = masse * GN * h;      //energie potentielle
39      vitesse = sqrt(2.*energie/masse);
40      h = h*(1.0-eps);
41      cout << "rebond : " << setw(5) << compt+1 << setw(15) << vitesse
```

```
42          << setw(15) << h << endl;
43      fout << "rebond : " << setw(5) << compt+1 << setw(15) << vitesse
44          << setw(15) << h << endl;
45
46      if (h <= 0.) {
47        cout << "La balle a perdu toute son energie apres "
48            << compt+1 << " rebonds" << endl;
49        break;
50      }
51    }
52    cout << "La hauteur finale est " << h << " metres." << endl;
53
54    return 0;
55  }
```