UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES

Méthodes informatiques pour physiciens

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

Corrigé 6

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/L6` :
pour les notes du cours, les exercices et les corrigés

6.1 Questions

1. **Comment accédez-vous à l'adresse mémoire d'une variable ?**
   L'adresse mémoire d'une variable `n` est renvoyée par l'*opérateur d'adresse* **&**, par exemple l'instruction
   ```
   cout << &n;
   ```
   affichera l'adresse mémoire de la variable `n`.

2. **Comment accédez-vous au contenu d'une location mémoire dont l'adresse est stockée dans un pointeur ?**
   Pour accéder au contenu de la location mémoire sur laquelle pointe un pointeur, il faut *déréférencer* le pointeur à l'aide de l'opérateur *.
   P. ex. si
   ```
   int i = 3;
   int *p = &i;
   ```
   l'instruction
   ```
   cout << *p;
   ```
   affichera la valeur de `i`.

3. **Quelle est la différence entre : `int &r = n` et `p = &n` ?**
   Dans le premier cas, `r` est une référence, c'est-à-dire un *alias* de la variable `n` : agir sur `r` revient à agir sur `n`. Dans le deuxième cas `p` est un pointeur auquel on affecte l'adresse mémoire de `n` et non pas la valeur de `n`.

4. **Peut-on déréférencer un pointeur NULL ?**
   Non, le déréférencement d'un pointeur NULL résulte en une erreur de segmentation.

5. **Pourquoi vaut-il mieux initialiser un pointeur lors de sa déclaration ?**
   Un pointeur non initialisé ne pointe sur rien en particulier. Ceci peut entraîner des erreurs pendant l'exécution du programme, par exemple en réecrivant sur un espace mémoire déjà alloué à une autre variable.

6.2 Trouvez l'erreur

Dans le premier programme le pointeur n'est pas correctement initialisé ; il faut affecter l'adresse mémoire d'une variable avant d'affecter une valeur au pointeur déréférencie. P. ex. le programme à droite est une version corrigée de celui de gauche.

```
1 int main() {
2   int *pInt;
3   *pInt = 9;
4   cout << *pInt;
5 }
```

```
1 int main() {
2   int j=4;
3   int *pInt = &j;
4   cout << *pInt;    //affiche 4
5   *pInt = 9;
6   cout << *pInt;    //affiche 9
7 }
```

Dans le deuxime programme on affecte au pointeur l'adresse mémoire d'un tableau. Etant donne que le nom du tableau contient l'adresse mémoire du tableau, on n'a plus besoin de l'opérateur adresse mémoire `&` (`double *y = a;` est équivalente à `double *y = &a[0];`) . Le programme à droite est le correct.

```
1 int main() {
2   double x[10];
3   double *y = &a;
4 }
```

```
1 int main() {
2   double x[10];
3   double *y = a;
4 }
```

6.3 Exercises

1. **Utilisez des pointeurs pour passer des paramètres aux fonctions !**
   Dans le programme **SphereP.cpp**, on passe les adresses mémoire des variables `surface` et `volume` à la fonction `sphere` à l'aide de l'opérateur &. La fonction `sphere` enregistre les résultats dans ce deux variables. On peut accéder à ces résultats dans le programme principal en *déréférenciant* les pointeurs correspondants.
   SphereP.cpp

```
1  //passage de variables par pointeur
2  #include <iostream>
3  #include <cmath>
4
5  using namespace std;
6
7  //prototype de la fonction sphere
8  //le premier argument est passe par valeur, les autres par pointeur
9  void sphere(double r, double *surface, double *volume);
10
11 int main() {
12   //saisi des donnees
13   double rayon;
14   cout << "Entrez le rayon de la sphere : ";
15   cin >> rayon;
16
17   //appel de la fonction sphere
18   double surface, volume;
19   sphere(rayon, &surface, &volume);
20
21   cout << "La surface de la sphere est : " << surface << endl;
22   cout << "et le volume est : " << volume << endl;
23
24   return 0;
25 }
26
27 //fonction sphere
28 void sphere(double rayon, double *surface, double *volume) {
29   *surface = 4. * M_PI * pow(rayon,2);
30   *volume = 4./3. * M_PI * pow(rayon,3);
31 }
```

2. **Développez le programme Adresse.cpp !**
   Le programme ci-dessous, aprés l'initialisation d'un pointeur, montre comment l'on accède aux différentes caractéristique de ce pointeur, valeur, adresse, taille.
   Adresse.cpp

```
1 //adresse memoire
2 #include <iostream>
3
4 using namespace std;
5
```

```
6  int main() {
7    int j = 4;
8    cout << "La valeur de j est : " << j
9         << " et l'adresse memoire de j est : " << &j << endl;
10
11   //declaration d'un pointeur
12   int *p = 0;   //le pointeur est initialise a 0 = NULL
13   p = &j;   //l'adresse de j est stockee dans p
14
15   int k = *p;   //la valeur de j est affectee a k
16   cout << "La valeur de j par pointeur est : " << k << "\t" << *p << endl;
17   cout << "Adresse memoire du pointeur : " << &p
18        << " et sa taille : " << sizeof(p) << endl;
19
20   *p = 5;   //la valeur 5 est affectee a j
21   cout << "La valeur de j est modifiee par pointeur : "
22        << *p << "\t"  << j << endl;
23
24   //et maintenant avec des tableau
25   double a[5]={1.,2.,3.,4.,5.};
26   cout << "a est un tableau; l'adresse memoire de a est : " << a
27        << ", \nqui est identique a &a[0] = " << &a[0] << endl;
28
29   return 0;
30 }
```

3. **Répondez aux questions de la page 9 du polycopié !**
   On definit un tableau : `double x[5]` .
   `cout << x;` renvoie l'adresse du tableau
   `cout << &x[0];` renvoie l'adresse du tableau également (`x` et `&x[0]` sont équivalents)
   `cout << x[1];` renvoie la valeur `x[1]`, mais comme celui-ci n'a pas été initialisé, cette valeur est aléatoire.
   On définit ensuite un pointeur sur le tableau :
   `double *y = x;`
   `cout << *y;` renvoie la valeur de `x[0]`
   `cout << y;` renvoie l'adresse du tableau
   On incrément le pointeur `y` de 3 (le pointeur se déplace de 3 variables) :
   `y = y + 3;`
   `cout << *y;` renvoie la valeur de `x[3]`
   `cout << y;` renvoie l'adresse de l'element `x[3]`
   On décrémente le pointeur `y` de 1 :
   `y = y - 1;`
   `cout << *y;` renvoie la valeur de `x[2]`
   `cout << y;` renvoie l'adresse de l'element `x[2]`

4. **Développez le programme Allocation.cpp !**
   Le tableau d'entiers `a` est créé pendant l'exécution du programme grâce à l'allocation dynamique de mémoire. La dimension du tableau n'est pas connue au lancement du programme. Elle est entrée par l'utilisateur et le tableau `a` est crééà l'aide du mot-clé `new`. On peut vérifier, si l'allocation dynamique a réussi : si l'opération a échoué le pointeur sera initialisé à zéro (pointeur NULL). A la fin du programme, on libère la mémoire alloué au tableau `a` pendant l'exécution en utilisant le mot-clé `delete`.

Allocation.cpp

```
1  //allocation dynamique de la memoire
2  #include <iostream>
3  #include <cstdlib>
4
5  using namespace std;
6
7  int main() {
8    int n;
9    cout << "Combien d'elements a le tableau ?";
10   cin >> n;
11
12   //allocation de la memoire pour le tableau
13   int *a = new int[n];
14   if (a==0) {
15     cout << "Erreur pendant l'initialisation du tableau. " << endl;
16     exit(0);
17   }
18
19   //saisis des donnees et remplissage du tableau
20   cout << "Entrez " << n << " nombres entiers : " << endl;
21   for (int i=1; i<=n; i++) {
22     cout << i <<" : ";
23     cin >> a[i-1];
24   }
25
26   cout << "Mentenant ils sont dans l'ordre inverse: " << endl;
27   for (int i=n-1; i>=0; i--)
28     cout << a[i] << endl;
29
30   //on libere la memoire reserve par le tableau
31   delete[] a;
32
33   return 0;
34 }
```

5. **Valeur moyenne des éléments d'un tableau.**
   Le tableau est initialisé de manière dynamique (la dimension du tableau est donnée par l'utilisateur) : le pointeur *tab est déclaré de type double, puis initialisé à l'aide du mot-clé `new`. Un fois créé, le tableau se comporte comme un tableau ordinaire. Les valeurs du tableau sont entrées par l'utilisateur. Ensuite le tableau est passé à la fonction `moyenne` qui calcule la moyenne des valeurs enregistrées dans le tableau `a`. En fait, lorsque l'on passe un tableau à une fonction, on passe un pointeur sur le premier élément du tableau (nom du tableau). Il nous faut alors aussi passer la dimension du tableau à la fonction `moyenne` afin de pouvoir parcourir le tableau dans la fonction. En fin de programme on libèrera la mémoire allouée dynamiquement au tableau pendant l'exécution à l'aide du mot-clé `delete`.

   Moyenne.cpp

```
1  //moyenne des valeurs enregistree dans un tableau
2  //le tableau est cree pendant l'execution du programme
3  //par allocation dynamique de la memoire
4  #include <iostream>
5
6  using namespace std;
7
8  //declaration de la fonction qui calcule la moyenne d'un tableau
```

```
9  double moyenne(double tab[], int n);
10
11 int main() {
12   int dim;
13   cout << "Quelle est la taille du tableau ? " << endl;
14   cin >> dim;
15
16   //tableau dynamique
17   double *tab = new double[dim];
18   if (tab==0) exit(0);    //la creation du tableau a echoue
19
20   //seasi des valeurs du tableau
21   for (int i=0; i<dim; i++) {
22     cout << "Element " << i+1 << " : ";    cin >> tab[i];
23   }
24
25   //calcul de la moyenne avec la fonction moyenne
26   double m = moyenne(tab,dim);
27   cout << "La moyenne du tableau est : " << m << endl;
28
29   delete[] tab;
30
31   return 0;
32 }
33
34 //definition de la fonction moyenne
35 double moyenne(double tab[], int n) {
36   double moyenne=0.;
37   for (int i=0; i<n; i++)
38     moyenne += tab[i];
39   moyenne = moyenne / n;
40
41   return moyenne;
42 }
```

6. **Ecrivez un programme pour ordonner les éléments d'un tableau de dimension N en ordre croissant ! (Utilisez l'allocation dynamique pour créer le tableau. Utilisez les pointeurs pour parcourir le tableau.)**
   Pour ranger le tableau on compare chaque élément avec tous les éléments suivants : si on trouve un élément dont la valeur est plus petite, on échange leur position. On a besoin de deux boucles : la première pour parcourir le tableau, la deuxième pour parcourir tous les éléments qui suivent l'élément en question. Le tableau a ranger est passé à la fonction `rangeTab` par pointeur, tandis que sa dimension est passée par valeur. Dans la fonction le tableau est parcouru à l'aide d'un second pointeur, qui pointe à l'adresse mémoire de l'élément suivant du tableau. Dans une boucle sur la dimension du tableau, les deux pointeurs (nom du tableau + pointeur sur élément suivant) sont incrémentés et les valeurs stockées aux emplacements mémoire correspondants sont comparées.
   RangeTab.cpp

```
1  //ce programme range un tableau de n variables double
2  //le tableau est cree par allocation dynamique de la memoire
3  #include <iostream>
4
5  using namespace std;
6
7  //declaration de la fonction pour ranger le tableau
8  //le tableau est passe a la fonction par pointeur
```

```
9  void rangeTab(double *tab, int n);
10
11 int main() {
12   int n;
13   cout << "Quelle est la taille du tableau ? ";
14   cin >> n;
15
16   //creation dynamique du tableau
17   double *v = new double[n];
18
19   //remplissage du tableau
20   cout << "Entrez les " << n << " valeurs !" << endl;
21   for (int i=0; i<n; i++) {
22     cout << "Element " << i+1 << " : ";
23     cin >> v[i];
24   }
25
26   //appel de la fonction rangeTab qui range le tableau
27   //le tableau est passe par pointeur,
28   //donc il faut passer l'adresse memoire du premier element
29   rangeTab(&v[0],n);
30
31   cout << "Maintenant les elements du tableau sont en ordre croissant :\n";
32   for (int i=0; i<n; i++)
33     cout << v[i] << endl;
34
35   delete[] v;
36
37   return 0;
38 }
39
40 //pour ranger le tableau on compare chaque element avec tous
41 //les elements suivants : si on trouve un element dont la valeur
42 //est plus petite on echange leur position
43 void rangeTab(double *tab, int n) {
44   double temp = 0.;
45   for (int i=0; i<n; i++) {
46     //on initialise le pointeur pour parcourir la partie suivante du
       tableau
47     double *v2 = tab+1;
48     for (int j=i+1; j<n; j++) {
49       if ((*tab)>(*v2)) {     //comparaison de deux valeurs !
50       temp = *v2;
51       *v2 = *tab;
52       *tab = temp;
53       }
54       v2++;
55     }
56     tab++;
57   }
58 }
```

7. **Des mesures sont stockées dans un fichier (ou entrées par le clavier). Ecrivez un programme pour lire ces données. Enregistrez les mesures dans un tableau de dimension appropriée. La premire donnée dans le fichier donne le nombre N de mesures. Le tableau est créé de facon dynamique pendant l'exécution du programme. Calculez la moyenne μ et lécart quadratique moyen σ des données. Affichez le résultat sur l'écran.**

Des mesures sont stockées dans le fichier `mesures.dat`. Pour lire les données à partir du fichier vous pouvez prendre exemple sur l'exercice 10 du corrigé 5. Dans le programme proposé on utilise l'allocation dynamique pour créer un tableau dont la dimension n'est pas connue au moment de la compilation du programme. Pour faire cela on utilise l'instruction `double *data= new double[dim]`. Avec cette instruction on réserve `n` éléments consécutifs de type double et on assigne l'adresse du premier au pointer `data`.
Puis on lit les données avec une boucle. La variable de contrôle `i` est définie localement dans la boucle, un fois sortie de la boucle la variable est effacée. Il faut donc déclarer une deuxième variable externe à la boucle pour compter les données (nombre d'itérations).
Ensuite on utilise une boucle pour parcourir le tableau `data` et calculer la moyenne :

$$\mu = \frac{\sum_{i=0}^{N-1} \text{data[i]}}{N} \tag{1}$$

et un deuxième boucle pour calculer l' écart quadratique moyenne

$$\sigma = \sqrt{\frac{\sum_{i=0}^{N-1} (\mu - \text{data[i]})^2}{N-1}} \, . \tag{2}$$

Puis on affiche les résultats sur l' écran. A la fin du programme on vide la mémoire occupée par le tableau `data` avec l'instruction `delete[] data;` .

Mesures.cpp

```
1  //calcule de la moyenne et de sigma d'un ensamble de mesures
2  //les mesures sont enregistees dans un fichier
3  #include <iostream>
4  #include <cmath>
5  #include <fstream>
6
7  using namespace std;
8
9  int main() {
10   //ouverture du fichier mesures.dat
11   ifstream fin("mesures.dat");
12   if (!fin) {
13     cout << "Erreur: le fichier n'existe pas ! STOP" << endl;
14     system("PAUSE");
15     return -1;
16   }
17
18   //lecture de la premiere donnee -> nombre total de mesures
19   int nData = 0;
20   fin >> nData;
21   if (!fin || nData==0) {
22     cout << "Il n'y a pas des donnees ! STOP" << endl;
23     system("PAUSE");
24     return -2;
25   }
26
27   //creation dynamique du tableau data de dimension nData
28   double *data = new double [nData];
29   if (data==0) {      //memoire insuffisante !
30     cout << "L'allocatione dynamique du tableau a echoue ! STOP" << endl;
31     system("PAUSE");
32     return -3;
33   }
34
```

```
35    //lecture du fichier et remplissage du tableau
36    /*la variable de controle i est locale a la boucle,
37      un fois sorti de la boucle la variable est effacee ;
38      donc il faut declarer une deuxieme variable externe a
39      la boucle pour compter les donnees (nombre d'iterations) */
40    int nRead = 0;
41    for (int i=0; i<nData; i++) {
42      fin >> data[i];
43      if (!fin) {
44        cout << "Les donnees sont finies !" << endl;
45        break;
46      }
47      nRead++;
48    }
49    cout << "Il y a " << nRead << " mesures" << endl;
50    if (nRead != nData) {
51      cout << "Le nobmre de mesures " << nData << " et les mesures lues "
52           << nRead << " ne correspondent pas ! STOP" << endl;
53      system("PAUSE");
54      return -4;
55    }
56
57    //calcul de la moyenne
58    double moyenne = 0.;
59    for (int i=0; i<nData; i++)
60      moyenne += data[i];
61    moyenne = moyenne / double(nData);
62
63    //calcul de l'ecart quadratique moyen
64    double sigma = 0.;
65    if (nData > 1) {
66      for (int i=0; i<nData; i++) {
67        sigma += pow((data[i]-moyenne),2);
68      }
69      sigma = sqrt(sigma / double(nData-1.));
70    }
71    else
72      sigma = 0.;
73
74    cout << "\nLa moyenne des mesures est : " << moyenne << endl
75         << "et l'ecart quadratique moyen est : " << sigma << endl << endl;
76
77    return 0;
78  }
```

8. **Ecrivez des fonctions pour addtionner, multiplier, etc., deux matrices de dimension quelconque ! (Dans les fonctions, les matrices seront parcourues avec des pointeurs.**
   Pour simplifier le problème, la solution proposée utilise des matrices carrées. Le programme peut être généralisé facilement à des matrices de dimension quelconque. Le deux matrices sont allouées dynamiquement. Une matrice bidimensionnelle est stockée dans la mémoire ligne après ligne de manière séquentielle. Les matrices sont crées comme des tableaux unidimensionnels de dimension `n*n` avec l'instructions `double *mat1 = new double[n*n];` et `double *mat2 = new double[n*n];`. `mat1` et `mat2` sont deux pointeurs de type double. La dimension de les matrices et ses valeurs sont fournis par l'utilisateur. Les deux matrices sont affichées avec la fonction `void imprimeMat(double *mat, int n)`. On calcule ensuite la transposée de la matrice avec la fonction `double`

`*transMat(double *mat, int n)`. La fonction est de type `*double`, càd elle renvoie un pointeur de type double. La matrice est passée à la fonction par pointeur. Dans la fonction une nouvelle matrice est crée dynamiquement pour enregistrer le résultat. La fonction renvoie le pointeur qui point sur l'adresse mémoire de la matrice résultat. La matrice transposée est accessible dans le programme principal en utilisant ce pointeur. Le produit est calculé dans la fonction `double *prodMat(double *mat1, double *mat2, int n)`. Comme pour la transposée, le résultat est stocké dans une nouvelle matrice crée dynamiquement dans la fonction. La matrice produit est accessible dans le programme principal en utilisant le pointeur renvoyé par cette fonction. Avant de sortir de la fonction, les matrices temporaires, crées pour effectuer le calcul, sont effacées avec l'instruction `delete`. A noter que la fonction `prodMat` calcule le produit en utilisant la transposée de la deuxième matrice : le produit ligne par colonne $C_{i,j} = \sum_{k=0}^{n} A_{i,k} B_{k,j}$ peut être calculé aussi comme produit ligne par ligne selon la formule $C_{i,j} = \sum_{k=0}^{n} A_{i,k} B^T_{k,j}$.

MatricesPointeurs.cpp.

```
1  //manipulation des matrices carres
2  //on utilise des pointeurs pour parcourrir les matrices
3  //et l'allocation dynamique de la memoire pour creer les matrices
4  #include <iostream>
5  #include <cmath>
6  #include <iomanip>
7
8  using namespace std;
9
10 //la fonction calcule la trsnsposee d'une matrice carre
11 //le resultat est enregistre dans un nouvelle matrice
12 //la fonction renvoie un pointeur qui "point" sur l'adress memoire
13 //du premier element de la matrice resultat
14 double *transMat(double *mat, int n);
15 //la fonction calcule le produit de deux matrices
16 double *prodMat(double *mat1, double *mat2, int n);
17 //la fonction affiche sur l'ecran une matrice n*n
18 void imprimeMat(double *mat, int n);
19
20 int main() {
21   int n;
22   cout << "Quelle est la taille de la matrice carree ? ";
23   cin >> n;
24
25   //creation dynamique de la matrice carree n*n
26   //un tableau bidimensionnel n*n est un tableau unidimensionnel
27   //compose de n tableaux unidimensionnels de dimension n
28   //donc il faut alluer de l'espace memoire suffisant pour stocker n*n
       elements
29   double *mat1 = new double[n*n];
30   double *mat2 = new double[n*n];
31
32   //remplissage de les matrices
33   cout << "Entrez les " << n*n
34        << " elementes de la premiere matrice ligne par colonne :" << endl;
35   for (int i=0; i<n; i++)
36     for (int j=0; j<n; j++) {
37       cout << "ligne " << i+1 << " colonne " << j+1 << " : ";
38       cin >> mat1[i*n+j];
39     }
40   cout << "et de la deuxieme matrice ligne par colonne :" << endl;
41   for (int i=0; i<n; i++)
42     for (int j=0; j<n; j++) {
```

```
43        cout << "ligne " << i+1 << " colonne " << j+1 << " : ";
44        cin >> mat2[i*n+j];
45      }
46
47    cout << "La premiere matrice est :" << endl;
48    imprimeMat(mat1,n);
49    cout << "et sa trasposee" << endl;
50    double *matT1 = transMat(mat1,n);
51    imprimeMat(matT1,n);
52
53    cout << "La deuxieme matrice est :" << endl;
54    imprimeMat(mat2,n);
55    cout << "et sa trasposee" << endl;
56    double *matT2 = transMat(mat2,n);
57    imprimeMat(matT2,n);
58
59    cout << "Le produit M1*M2 est :" << endl;
60    double *matR = prodMat(mat1,mat2,n);
61    imprimeMat(matR,n);
62
63    //on libere la memoire allouee dynamiquement,
64    //c'est a dire on detruit les pointeurs crees avec "new"
65    delete[] mat1;
66    delete[] mat2;
67    delete[] matT1;
68    delete[] matT2;
69    delete[] matR;
70
71    return 0;
72  }
73
74  //calcul du produit de deux matrices
75  //le deux matrices sont passees par pointeur et
76  //la fonction renvoie un pointuer sur la matrice - produit
77  /*le produit ligne par colonne peut etre aussi calcule comme
78    produit ligne par ligne en utilisant la transposee de la
79    deuxieme matrice */
80  double *prodMat(double *mat1, double *mat2, int n) {
81    double *resultat = new double[n*n];
82    double *auxMat1 = 0;
83    double *newMat2 = transMat(mat2,n);
84    double *auxMat2 = 0;
85    for (int i=0; i<n; i++) {
86      for (int j=0; j<n; j++) {
87        resultat[i*n+j] = 0.0;
88        auxMat2 = newMat2+n*j;
89        auxMat1 = mat1+n*i;
90        for(int k=0; k<n; k++) {
91        resultat[i*n+j] += (*auxMat1)*(*auxMat2);
92        auxMat1++;
93        auxMat2++;
94        }
95      }
96    }
97    delete[] auxMat1;
98    delete[] auxMat2;
99    delete[] auxMat2;
100
101   return resultat;
102 }
```

```
103
104 //la matrice a imprimer est passee par pointeur
105 void imprimeMat(double *mat, int n) {
106   cout << setw(5) << "/" << setw(n*10+5) << "\\" << endl;
107   for (int i=0; i<n; i++) {
108     cout << setw(5) << "|";
109     for (int j=0; j<n; j++) {
110       cout << setw(10) << setprecision(5) << *mat;
111       mat++;
112     }
113     cout << setw(5) << "|" << endl;
114   }
115   cout << setw(5) << "\\" << setw(n*10+5) << "/" << endl;
116   cout << endl;
117 }
118
119 //calcul de la transposee d'une matrice
120 //la matrice est passee par pointeur;
121 //la fonction renvoie un pointuer sur la matrice transposee
122 double *transMat(double *mat, int n) {
123   double *transpose = new double[n*n];
124   for (int i=0; i<n; i++)
125     for (int j=0; j<n; j++)
126       transpose[i*n+j] = mat[j*n+i];
127
128   return transpose;
129 }
```