UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES

Méthodes informatiques pour physiciens

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

Corrigé 1

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/L1` :
pour les notes du cours, les exercices et les corrigés

1.1 Questions

1. Le plus petit programme qui ne fait rien et que l'on peut écrire en C++ est le suivant :
   Rien.cpp

```
1 int main() {return 0;}
```

2. On peut soustraire `1` d'un nombre `n` de plusieures manières :
   q2.cpp

```
1  //soustractions
2  #include <iostream>
3
4  using namespace std;
5
6  int main() {
7    int n = 3;
8    cout << "n vaut " << n << endl;
9    n = n - 1;
10   cout << "n = n-1 -> n vaut maintenant " << n << endl;
11   n -= 1;
12   cout << "n -= 1 -> n vaut maintenant " << n << endl;
13   n--;
14   cout << "n-- -> n vaut maintenant " << n << endl;
15   return 0;
16 }
```

3. L'éditeur de liens lie les fichiers objets (fichiers simplement compilés) avec les fichiers pré-compilés d'une ou plusieures bibliothèques.

4. Le commentaire le plus commun en C++ est introduit avec la double barre oblique `//` au début du commentaire ; le texte à droite de `//` est ignoré par le compilateur. Le deuxième type de commentaire est introduit avec les symboles `/*` qui ouvre le commentaire et `*/` qui ferme le commentaire (tout le texte entre ces symobles est ignoré).

5. La différence principale est le fait que la représentation en virgule flottante est une approximation des nombres réels. Suivant le type de la variable (`float` ou `double`) cette représentation est codée sur une plage différente qui agit directement sur la précision du résultat (simple ou double).

6. Voir et exécuter le programme :
   q6.cpp

```
1 #include <iostream>
2
3 int main() {
4   std::cout << char(100) << std::endl;
5   return 0;
6 }
```

7. Dans le programme **q7.cpp**, on initialise d'abord la variable `c` de type caractère (`char`) avec la valeur `65` pius on l'imprime. Ce caractère correspond à la lettre `A` selon le code ASCII (`http://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange`). L'autre façon d'initialiser `c` à la même valeur est d'affecter la variable `c` explicitement avec le caractère '`A`'. La dernière instruction affiche la valeur numérique du caractère `c` (i.e. la variable `c` est transtypée en entier).
   q7.cpp

```
1 #include <iostream>
```

```
2
3  int main() {
4    char c;
5    c = 65;
6    std::cout << c << std::endl;
7
8    c = 'A';    //le code ASCII de A est 65
9    std::cout << c << " est aussi " << int(c) << std::endl;
10
11   return 0;
12 }
```

1.2 Trouvez l'erreur !

Les erreurs sont les suivantes :

programme 1 :

l. 5 : il manque la double barre oblique `//` au début du commentaire (ou les symboles `/*` au debut et `*/` à la fin du commentaire).

l. 8 : il manque le point-virgule `;` après l'instruction `return 0`.

l. 9 : il manque une accolade } à la fin du programme.

Version correcte :

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main () {
6   //imprime " Hello World "
7   cout << "Hello World !\n";
8   return 0;
9 }
```

programme 2 :

l. 3 : la fonction de sortie `cout` est utilisée sans indiquer l'espace de noms `std::`. Il faut définir l'espace de noms standard en en-tête du programme avec l'instruction `using namespace std;` ou utiliser la syntaxe complète `std::cout`.

l. 8 : l'instruction `return` est écrite en lettres majuscules.

Version correcte :

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main () {
6   int n = 22;
7   cout << n << endl;
8   return 0;
9 }
```

programme 3 :

l. 3 : la fonction de sortie `cout` est utilisée sans indiquer l'espace des noms `std::`.

l. 6 : il manque le point-virgule `;` à la fin de l'instruction de déclaration et initialisation de la variable `n`.

l. 8 : il manque l'instruction `return 0;` à la fin du programme.

l. 9 : il ne devrait pas y avoir le point-virgule ; après la dernière accolade.
Version correcte : **voir programme2.**

programme 4 :
l. 3 : l'instruction `using namespace std;` doit être entièrement en minuscules.
l. 7 : la variable `n` n'a pas été déclarée.
l. 9 : l'instruction `return` doit être écrite entièrement en minuscules. `return` ne retourne pas 0. : Ça ne donne pas une erreur de compilation et ne cause pas un échec pendant l'exécution du programme non plus. Cependant le standard en C++ c'est d'indiquer un problème générique d'exécution avec `return 1;` ou si tout va bien l'indiquer avec `return 0;`. D'autres valeurs sont admises, mais dans ce cas là le programmeur doit fournir une table avec les codes de retour et leur signification.
Version correcte :

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main () {
6    int m = 3;
7    int n = 22;
8    cout << m+n << endl;
9    return 0;
10 }
```

1.3 Exercices

Ici vous trouverez les solutions des exercices 4, 6, 7, 9, 10. Pour les autres, consultez les notes du cours.

Exercice 4 : Dans la fonction principale `main`, les variables `a` et `b`, de type entier, sont d'abord définies et des valeurs leur sont ensuite attribuées (affectées). Grâce à la fonction `cout` le programme affiche sur l'écran le résultat de la somme, de la différence, du produit et de la division avec reste (modulo) de ces deux entiers. Etant donné que les variables sont des nombres entiers, le résultat de la division est aussi un nombre entier. L'affichage est composé de trois parties, chacune séparée par l'*operateur* "`<<`" : après une phrase introductive ("la somme de a ..."), les résultats des opérations apparaissent et la fonction `endl` est appelée pour aller à la ligne. La fonction de sortie `cout` est utilisée sans l'opérateur d'extraction `::` de l'espace des noms `std` parce que celui-ci est choisi au début du programme avec l'instruction `using namespace std;`.

Opera.cpp

```
1  //exercice avec les operateurs arithmetiques
2  #include<iostream>
3
4  using namespace std;
5
6  int main() {
7    int a, b;
8    a = 60;
9    b = 7;
10   cout << "Operations avec entiers" << endl;
11   cout << "La somme de a et b est " << a+b << endl;
12   cout << "La difference de a et b est " << a-b << endl;
13   cout << "Le produit de a par b est " << a*b << endl;
14   cout << "La division de a par b est " << a/b;
```

```
15    cout << " avec reste " << a%b << endl;
16    return 0;
17  }
```

Exercice 6 : On propose ici deux solutions. Dans la première la valeur de l'angle est définie par l'utilisateur : le programme assigne à la variable `angle` la valeur entrée au clavier par l'utilisateur. Pour conserver une bonne précision, la constante `deg2rad` est définie de type `double`. Dans la deuxième, la valeur de la constante π est introduite par la varialbe `M_PI` définie dans la bibliothèque `cmath`. Pour accéder à cette variable il faut inclure la bibliothèque `cmath` en en-tête du programme avec l'instruction `#include <cmath>`.

deg2radv1.cpp

```
1  //conversion de degres en radians
2  #include <iostream>
3
4  using namespace std;
5
6  int main() {
7    //saisi de l'angle
8    double angle;
9    cout << "Quel est l'angle en degres ?\n";
10   cin >> angle;
11
12   //conversion
13   const double deg2rad = 3.14159 / 180.;      //facteur de conversion
14   angle = angle * deg2rad;
15   cout << "L'angle en radians est " << angle << endl;
16
17   return 0;
18 }
```

deg2radv2.cpp

```
1  //conversion de degres en radians
2  #include <iostream>
3  #include <cmath>
4
5  using namespace std;
6
7  int main() {
8    //saisi de l'angle
9    double angle;
10   cout << "Quel est l'angle en degres ?\n";
11   cin >> angle;
12
13   //conversion sur la meme ligne de sortie
14   cout << "L'angle en radians est : " << angle * (M_PI / 180.) << endl;
15
16   return 0;
17 }
```

Exercice 7 : Le programme suivant convertit les livres en kg.

lb2kg.cpp

```
1  //conversion de livres en kg
2  #include <iostream>
3
```

```
4  using namespace std;
5
6  int main() {
7    //saisi du poids
8    double poids;
9    cout << "Quel est le poids en livres ? ";
10   cin >> poids;
11
12   //conversion : 1 lb = 0.453 kg
13   const double lb2kg = 0.453;
14   poids = poids * lb2kg;
15   cout << "Le poids en kg est " << poids << endl;
16
17   return 0;
18 }
```

Exercice 9 : Ce programme calcule la surface et le volume d'un cube à partir de la valeur de son côté. Le valeur du côté est rentrée au clavier par l'utilisateur. Le programme récupère cette valeur grâce à la fonction `cin` et l'assigne à la variable côté (`cote`) définie précedemment. Le programmme calcule ensuite la surface (une variable de type double afin de conserver une bonne précision) et le volume. On calcule ce dernier avec la fonction `pow(double base, int exposant)`. Il est nécessaire d'inclure la bibliothèque `cmath`, avec l'instruction `#include <cmath>`, afin de pouvoir utiliser cette fonction `pow`.

Cube.cpp

```
1  //calcul de la surface et du volume d'un cube
2  #include <iostream>
3  #include <cmath>      //bibliotheque mathematique
4
5  using namespace std;
6
7  int main() {
8    //saisi des donnees
9    double cote;
10   cout << "Quel est le cote du cube ? ";
11   cin >> cote;
12
13   //calcul de la surface du cube
14   double surface;
15   surface = 6. * cote * cote;
16   cout << "La surface est " << surface << endl;
17
18   //calcul du volume du cube
19   double volume = pow(cote, 3.);
20   cout << "Le volume est " << volume << endl;
21
22   return 0;
23 }
```

Exercice 10 : Les quatre premiers termes de la série de Taylor des fonctions cosinus et sinus sont :

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + ... \quad (1)$$

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + ... \quad (2)$$

Le programme assigne à la variable **x** la valeur pour laquelle nous voulons calculer le sinus et le cosinus (l. 8, qui vaut $\pi/4$ dans cet exemple). Le programme calcule ensuite la valeur des cosinus et sinus avec les fonctions standard de C++ et affiche à l'écran les valeurs correspondantes. Pour utiliser les fonctions `sin(x)` et `cos(x)` il faut inclure en ête du programme (l. 3) la bibliothèque `cmath` avec l'instruction `#include <cmath>`. Le programme calcule également la valeur du cosinus et du sinus en utilisant la série de Taylor.

Taylor.cpp

```
1  //calcul du sinus et du cosinus
2  #include <iostream>
3  #include <cmath>
4
5  using namespace std;
6
7  int main() {
8    double x = M_PI/4.;    //calcul pour x = 45 degres
9    double c, s;
10
11   //fonctions cos et sin
12   cout << "Pour pi/4 les fonctions mathematiques donnent :" << endl;
13   cout << "cos(x) = " << cos(x) << " et sin(x) = " << sin(x) << endl;
14
15   //premieres 4 termes dans l'expansion de Taylor
16   c = 1 - pow(x,2.)/2. + pow(x,4.)/24. - pow(x,6.)/720.;
17   s = x - pow(x,3.)/6. + pow(x,5.)/120. - pow(x,7.)/5040.;
18   cout << "L'expansion de Taylor (premieres 4 termes) donne :" << endl;
19   cout << "cos(x) = " << c << " et sin(x) = " << s << endl;
20
21   return 0;
22 }
```

1.4 Problèmes

Cinématique : Ecrivez un programme qui calcule la distance parcourue par un projectile en fonction de sa vitesse initiale v_0 et angle de jetée ϑ.
Pour résoudre le problème il faut d'abord établir les équations du mouvement du projectile. Les coordonnées et vitesse initiales sont :

$$\begin{aligned} x_0 = 0 \quad &\text{et} \quad v_{x,0} = v_0 \cdot \cos(\vartheta) \\ y_0 = 0 \quad &\text{et} \quad v_{y,0} = v_0 \cdot \sin(\vartheta) \\ z_0 = 0 \quad &\text{et} \quad v_{z,0} = 0 \end{aligned} \tag{3}$$

Dans la mesure où le projectile n'est soumise qu'à son poids ($m \cdot \vec{a} = m \cdot \vec{g}$) et connaissant les conditions initiales, on a donc :

$$\begin{aligned} a_x = 0 \quad &\text{et} \quad v_x = v_{x,0} \\ a_y = -g \quad &\text{et} \quad v_y = -g \cdot t + v_{y,0} \\ a_z = 0 \quad &\text{et} \quad v_z = 0 \end{aligned} \tag{4}$$

L'on peut ensuite établir les équations du mouvement du projectile :

$$\begin{aligned} x(t) &= v_x \cdot t \\ y(t) &= -\frac{1}{2} \cdot g \cdot t^2 + v_y \cdot t \\ z(t) &= 0 \end{aligned} \tag{5}$$

La jetée du projectile correspond à la valeur de x pour $y = 0$ (i.e. quand le projectile touche le sol), soit :

$$-\frac{1}{2} \times g \cdot t^2 + v_y \cdot t = 0 \tag{6}$$

d'où $t = 0$ ou $t = 2 \cdot v_y/g$.
Pour $t = 2 \cdot v_y/g$ on a donc :

$$x = 2 \cdot v_x \cdot v_y/g \ . \tag{7}$$

La hauteur maximale est atteinte quand v_y est nulle, soit :

$$-g \cdot t + v_y = 0 \tag{8}$$

d'où $t = v_y/g$. On calcule ensuite y pour $t = v_y/g$ et on obtient

$$y_{max} = v_y^2/2 \ . \tag{9}$$

Le programme **Balle.cpp** prend en paramètre d'entrée la vitesse initiale et l'angle de jetée du projectile. Il renvoie la jetée, le temps de vol et la hauteur maximale. Pour un temps t rentré par l'utilisateur, il renvoie la position du projectile. Pour compléter cette partie, il faudrait vérifier que le temps rentré soit compatible avec la trajectoire du projectile (prochaine leçon).

Balle.cpp

```
1  //Etude de la trajectoire d'un projectile
2  #include <iostream>
3  #include <cmath>
4
5  using namespace std;
6
7  int main() {
8    //saisi des conditions initiales
9    double v0, theta;
10   cout << "Entrez la vitesse initiale (m/s) : ";
11   cin >> v0;
12   cout << "Entrez l'angle initial (degres) : ";
13   cin >> theta;
14
15   //vecteur vitesse initial
16   double vx, vy;
17   vx = v0*cos(theta*M_PI/180.);
18   vy = v0*sin(theta*M_PI/180.);
19
20   //parametres physiques
21   const double G = 9.81;        //acceleration gravitationnelle g
22
23   //calcule de la cinematique
24   double portee = 2. * vx * vy / G;      //jetee maximale
25   double hauteur = (1./2.) * pow(vy,2) / G;      //hauteur maximale
26   double tof = 2. * vy / G;      //temps de vol
27
```

```
28    cout << "Distance calculee " << portee << " metres." << endl;
29    cout << "Hauteur calculee " << hauteur << " metres." << endl;
30    cout << "Temps de vol calcule "  << tof << " secondes." << endl;
31
32    //calcul de la position pour un temps donne
33    double t;
34    cout << "Entrez un temps t entre 0 et " << tof << " sec : ";
35    cin >> t;
36
37    //position au moment donne t
38    double x = vx * t;
39    double y = vy * t - (1./2.) * G * pow(t,2);
40    cout << "Au temps t = " << t << " sec," <<
41      " le projectile se trouve en :" << endl;
42    cout << "x = " << x << " metres" << endl;
43    cout << "y = " << y << " metres" << endl;
44
45    return 0;
46  }
```

Optique : Ecrivez un programme pour calculer l'angle de réflexion α d'un rayon laser réfléchi par une sphère de rayon R avec paramètre d'impact b.
Pour calculer l'angle de réflexion on introduit le plan tangente à la surface de la sphère dans le point d'impact du rayon laser et on calcule l'angle d'incidence ϑ_{in} puis l'ange de réflexion $\vartheta_{out} = \vartheta_{in}$ du rayon par rapport à la normale à ce plan, normale qui passe aussi par le centre de la sphère. Il s'agit donc d'un simple problème de trigonométrie :

$$\alpha = 2 \cdot \arcsin \frac{b - R}{R} \ . \tag{10}$$

Pour compléter le programme, il faudrait vérifier que $0 < b < 2R$ (prochaine leçon).

Laser.cpp

```
1   //Reflexion d'un rayon laser
2   #include <iostream>
3   #include <cmath>
4
5   using namespace std;
6
7   int main() {
8     //saisi des parametres initiales
9     double rayon, b;
10    cout << "Entez le rayon de la sphere en m : ";
11    cin >> rayon;
12    cout << "Entrez le parametre d'impact du rayon laser en m "
13         << "(" << 0 << " < b < " << rayon << ") : ";
14    cin >> b;
15
16    //angle de reflexion
17    double alpha;
18    alpha = asin((b-rayon) / rayon);
19    alpha = 2. * alpha * 180. / M_PI;
20    cout << "L'angle de reflextion est " << alpha << endl;
21
22    return 0;
23  }
```