UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES

Méthodes informatiques pour physiciens

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

Corrigé 9

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/L9` :
pour les notes du cours, les exercices et les corrigés

9.1 Exercices

1. **Implémentation de la classe `Point`**
   Un point est caractérisé par ses coordonnées cartésiennes x, y et z. Trois constructeurs sont définis, le constructeur par défaut, un constructeur qui prend en paramètre d'entrée les coordonnées et un constructeur de copie. Les coordonnées du point sont déclarées comme `private` mais toutes les méthodes sont déclarées comme `public`.
   Pour la symétrie par rapport à l'origine, deux méthodes sont proposées. La première (`Negate1()`) modifie les coordonnées du point qui peut ensuite être affecté à un autre point ou utilisé tel quel. Toutefois, les coordonnées originelles sont perdues et il faut à nouveau appliquer la méthode pour revenir au point d'origine. La deuxième méthode prend le point en entrée et retourne un autre point correspondant au point symétrique.
   Pour additionner deux vecteurs définis respectivement entre l'origine et deux points, on a simplement surchargé l'opérateur d'addition `+`.
   Point.cpp.

```
1  //developpement de la classe Point
2  #include <iostream>
3  #include <cmath>
4
5  using namespace std;
6
7  class Point {
8    //membres publiques
9    public:
10     //constructeurs
11     Point();
12     Point(double x, double y , double z);
13     Point(const Point &p);   //constructeur de copie
14
15     //destructeur
16     ~Point() {};
17
18     //methode pour afficher le point
19     void Print() const;
20     //methode pour calculer la norme du vecteur (P-O)
21     double Norm() const;
22     //methode pour appliquer une symmetrie d'origine
23     void Negate1();
24     //methode qui renvoye -q sans modifier le point
25     Point Negate2() const;
26     //surcharge de l'operateur pour additionner deux points
27     Point operator+(const Point &p) const;
28
29   //membres prives
30   private:
31     //donnees membres
32     double x;
33     double y;
34     double z;
35 };
36
37 //constructeurs
38 Point::Point() {
39   x = 0.;
40   y = 0.;
41   z = 0.;
42   cout << "Un point a ete cree sans initialisation." << endl;
```

```
43 }
44 Point::Point(double x, double y, double z): x(x), y(y), z(z) {
45   cout << "Un point a ete cree." << endl;
46 }
47 Point::Point(const Point &p) {
48   x = p.x;
49   y = p.y;
50   z = p.z;
51   cout << "Un point a ete copie." << endl;
52 }
53
54 void Point::Print() const {
55   cout << "(" << x << " , " << y << " , " << z << ")" << endl;
56 }
57
58 double Point::Norm() const {
59   double norme = sqrt(pow(x,2)+pow(y,2)+pow(z,2));
60   return norme;
61 }
62
63 void Point::Negate1() {
64   x = -x;
65   y = -y;
66   z = -z;
67 }
68
69 //cette fonction renvoit une "variable" de type Point
70 Point Point::Negate2() const {
71   Point p(-x, -y, -z);
72   return p;
73 }
74
75 //surcharge de l'operateur d'addition +
76 Point Point::operator+(const Point &q) const {
77   Point res(x+q.x, y+q.y, z+q.z);
78   return res;
79 }
80
81 int main() {
82   Point p(2,2,3);
83   cout << "p = ";   p.Print();
84   cout << "norme de p = " << p.Norm() << endl;
85   Point q(3,3,0);
86   cout << "q = ";   q.Print();
87   cout << "norme de q = " << q.Norm() << endl;
88
89   p.Negate1();
90   cout << "Apres Negate1(p), p = ";   p.Print();
91   Point r = p;      //copie de p
92   cout << "r = ";   r.Print();
93
94   Point s;
95   s = q.Negate2();
96   cout << "Apres Negate2(q), q = ";
97   q.Print();
98   cout << "s = ";
99   s.Print();
100
101  Point t = q + s;
102  cout << "Somme des vecteurs definis par q et s depuis l'origine : \n";
```

```
103     cout << "t = ";
104     t.Print();
105     cout << "norme de t = " << t.Norm() << endl;
106
107     Point u = p + s;
108     cout << "Somme des vecteurs definis par p et s depuis l'origine : \n";
109     cout << "u = ";
110     u.Print();
111     cout << "norme de u = " << u.Norm() << endl;
112
113     return 0;
114 }
```

2. **Implémentation de la classe `Cercle`**
   Un cercle est défini par les coordonnée x, y de son centre et par la valeur de son rayon. Tous ces paramètres sont declarés comme `private`, tout comme les fonctions qui calculent la surface et le périmètre du cercle. Nous avons défini comme `public` les méthodes pour accéder aux paramètres du cercle et pour modifier les coordonées du centre et le rayon. Dans le programme, si l'on change le rayon du cercle, la surface et le périmètre sont recalculés.
   Cercle.cpp.

```
1  #include <iostream>
2  #include <iomanip>
3  #include <cmath>
4
5  using namespace std;
6
7  class Cercle {
8    public:
9      //constructeurs
10     Cercle(double xCen, double yCen, double rayon);
11
12     //fonctions d'acces
13     double area() const;
14     double circumference() const;
15     double getXCentre() const;
16     double getYCentre() const;
17     double getRadius() const;
18
19     //methodes publiques: fonctions pour modifier le cercle
20     void setXCentre(double nX);
21     void setYCentre(double nY);
22     void setRadius(double nR);
23
24     //fonction pour imprimer l'information du cercle
25     void imprimerInfo() const;
26
27   private:
28     //donnees privees
29     double radius, xCen, yCen;
30     double surf, perim;
31     //methodes privess
32     void setArea();
33     void setCircumf();
34 };
35
36 //constructeur
```

```
37 Cercle::Cercle(double x, double y, double r): radius(r), xCen(x), yCen(y) {
38   setArea();
39   setCircumf();
40 }
41
42 //fonctions d'acces
43 double Cercle::area() const {
44   return surf;
45 }
46 double Cercle::circumference() const {
47   return perim;
48 }
49 double Cercle::getXCentre() const {
50   return xCen;
51 }
52 double Cercle::getYCentre() const {
53   return yCen;
54 }
55 double Cercle::getRadius() const {
56   return radius;
57 }
58
59 //methode publique pour changer le centre du cercle en X
60 void Cercle::setXCentre(double nX) {
61   cout << "INFO : modification coordonnee X du centre du cercle" << endl;
62   xCen = nX;
63 }
64
65 //methode publique pour changer le centre du cercle en Y
66 void Cercle::setYCentre(double nY) {
67   cout << "INFO : modification coordonnee Y du centre du cercle" << endl;
68   yCen = nY;
69 }
70
71 //methode publique pour changer le rayon du cercle
72 void Cercle::setRadius(double nR) {
73   cout << "INFO : modification du rayon du cercle" << endl;
74   radius = nR;
75   setArea();
76   setCircumf();
77 }
78
79 //methode privee pour calculer la surface du cercle,
80 //on l'appelle aussi quand le rayon du cercle change
81 void Cercle::setArea() {
82   cout << "INFO : (re)calcule de l'aire du cercle" << endl;
83   surf = M_PI*radius*radius;
84 }
85
86 //methode privee pour calculer la circonference du cercle,
87 //on l'appelle aussi quand le rayon du cercle change
88 void Cercle::setCircumf() {
89   cout << "INFO : (re)calcule de la circonference du cercle" << endl;
90   perim = 2.*M_PI*radius;
91 }
92
93 void Cercle::imprimerInfo() const {
94   cout << "Le centre du cercle est dans "
95        << "(x = " << xCen << ", y = " << yCen << "),"<< endl;
96   cout << "et le rayon du cercle est r = " << radius << endl;
```

```
97  }
98
99  int main() {
100   double x, y, r;
101   cout << "Coordonee x du centre du cercle : ";
102   cin >> x;
103   cout << "Coordonee y du centre du cercle : ";
104   cin >> y;
105   cout << "Rayon du cercle : ";
106   cin >> r;
107
108   Cercle lune(x,y,r);
109   lune.imprimerInfo();
110
111   cout << "L'aire du cercle est A = " << lune.area() << endl
112        << "et sa circonference est P = " << lune.circumference() << endl;
113
114   cout << "Entrez une nouvelle coordonee x pour le cercle : ";
115   cin >> x;
116   lune.setXCentre(x);
117   cout << "et une nouvelle coordonee y pour le cercle : ";
118   cin >> y;
119   lune.setYCentre(y);
120   cout << "et un nouveau rayon pour le cercle : ";
121   cin >> r;
122   lune.setRadius(r);
123
124   cout << "La nouvelle aire du cercle est A' = " << lune.area() << endl
125        << "et sa circonference est P' = " << lune.circumference() << endl;
126   lune.imprimerInfo();
127   cout << endl;
128
129   double a;
130   cout << "Dernier example" << endl;
131   cout << "Multiplication des coordonees du centre et du rayon par : ";
132   cin >> a;
133   x = lune.getXCentre()*a;
134   y = lune.getYCentre()*a;
135   r = lune.getRadius()*a;
136   lune.setXCentre(y);
137   lune.setYCentre(x);
138   lune.setRadius(r);
139   lune.imprimerInfo();
140
141   cout << "\nLa nouvelle surface A = " << lune.area() << endl;
142   cout << "et la circonference P = " << lune.circumference() << endl;
143
144   return 0;
145 }
```

3. **Surcharge des opérateurs dans la classe** `Rational`
   Voir et completer le programme **RationalX.cpp**.
   RationalX.cpp.

```
1  //developpement de la classe Rational
2  #include <iostream>
3  #include <cmath>
4
5  using namespace std;
```

```
6
7  class Rational {
8    //membres publiques
9    public:
10
11     //constructeurs
12     Rational();      //constructeur par defaut (0 parametres)
13     Rational(int num);    //constructeur avec 1 parametres
14     Rational(int num, int den);    //constructeur avec 2 parametres
15     Rational(const Rational &r);      //constructeur de copie
16
17     //destructeur
18     ~Rational();
19
20     //fonctions d'access
21     int GetNum() const;
22     int GetDen() const;
23     void SetNum(int num);
24     void SetDen(int den);
25
26     //methodes publiques
27     double Convert() const;
28     void Invert();
29     void Print() const;
30     void Reduce();
31
32     //operateurs surchages
33     Rational operator+(const Rational &) const;
34     Rational operator-() const;
35     Rational operator-(const Rational &) const;
36     Rational operator*(const Rational &) const;
37     Rational operator/(const Rational &) const;
38     bool operator==(const Rational &) const;
39     bool operator<(const Rational &) const;
40     bool operator<=(const Rational &) const;
41     bool operator>(const Rational &) const;
42     bool operator>=(const Rational &) const;
43
44   //membres privees
45   private:
46     //donnees privees
47     int num;
48     int den;
49
50     //methodes privees
51
52 };   //n'oubliez pas le ;
53
54 //constructeur par defaut
55 Rational::Rational() {
56   num = 0;
57   den = 1;
58   cout << "Un nombre Rational a ete cree par defaut." << endl;
59 }
60 //constructeur avec 1 parametre
61 //les deux constructeurs (fonctions) suivants sont equivalents
62 /*Rational::Rational(int n) {
63   num = n;
64   den = 1;
65   cout << "Un nombre Rational a ete cree; denom. = 1." << endl;
```

```
66 }
67 */
68 Rational::Rational(int n): num(n), den(1) {
69   cout << "Un nombre Rational a ete cree; denom. = 1." << endl;
70 }
71 //constructeur avec 2 parametres
72 //les deux constructeurs (fonctions) suivants sont equivalents
73 /*Rational::Rational(int n, int d) {
74   num = n;
75   den = d;
76   Reduce();
77   cout << "Un nombre Rational a ete cree." << endl;
78 }
79 */
80 Rational::Rational(int n, int d): num(n), den(d) {
81   Reduce();
82   cout << "Un nombre Rational a ete cree." << endl;
83 }
84
85 //constructeur de copie
86 //les deux constructeurs (fonctions) de copie suivants sont equivalents
87 /*Rational::Rational(const Rational &r) {
88   num = r.num;
89   den = r.den;
90   cout << "Un nombre Rational a ete copie." << endl;
91 }
92 */
93 Rational::Rational(const Rational &r): num(r.num), den(r.den) {
94   cout << "Un nombre Rational a ete copie." << endl;
95 }
96
97 //destructeur
98 Rational::~Rational() {
99   cout << "Un nombre Rational a ete detruit." << endl;
100 }
101
102 //fonctions d'acces
103 int Rational::GetNum() const {
104   return num;
105 }
106
107 int Rational::GetDen() const {
108   return den;
109 }
110
111 void Rational::SetNum(int n) {
112   num = n;
113 }
114
115 void Rational::SetDen(int d) {
116   den = d;
117 }
118
119 //algorithme d'Euclide pour trouver le plus grand diviseur commun
120 void Rational::Reduce() {
121   int m = abs(num);
122   int n = abs(den);
123   if (m<n) {
124     int temp = m;
125     m = n;
```

```
126       n = temp;
127     }
128     int r;
129     while (n>0) {
130       r = m % n;
131       m = n;
132       n = r;
133     }
134     int gdc = m;
135 
136     num = abs(num) / gdc;
137     if (num*den<0) num = -num;
138     den = abs(den) / gdc;
139 }
140 
141 double Rational::Convert() const {
142     return double(num) / den;
143 }
144 
145 void Rational::Invert() {
146     int temp = num;
147     num = den;
148     den = temp;
149 }
150 
151 void Rational::Print() const {
152     cout << num << '/' << den << endl;
153 }
154 
155 //surcharge des operateurs
156 //N.B. : declaration differente pour les methodes
157 //qui ne sont pas membre de la classe !
158 Rational Rational::operator+(const Rational &q) const {
159     int n, d;
160     n = num*q.den + den*q.num;
161     d = den*q.den;
162     Rational res = Rational(n,d);
163     return res;
164 }
165 Rational Rational::operator-() const {
166     int n, d;
167     n = -num;
168     d = den;
169     Rational res = Rational(n,d);
170     return res;
171 }
172 Rational Rational::operator-(const Rational &q) const {
173     int n, d;
174     n = num*q.den - den*q.num;
175     d = den*q.den;
176     Rational res = Rational(n,d);
177     return res;
178 }
179 Rational Rational::operator*(const Rational &q) const {
180     int n, d;
181     n = num * q.num;
182     d = den * q.den;
183     Rational res = Rational(n,d);
184     return res;
185 }
```

```
186 Rational Rational::operator/(const Rational &q) const {
187   int n, d;
188   n = num*q.den;
189   d = den*q.num;
190   Rational res = Rational(n,d);
191   return res;
192 }
193 bool Rational::operator==(const Rational &q) const {
194   return (num*q.den == den*q.num);
195 }
196 bool Rational::operator<(const Rational &q) const {
197   return (num*q.den < den*q.num);
198 }
199 bool Rational::operator<=(const Rational &q) const {
200   return (num*q.den <= den*q.num);
201 }
202 bool Rational::operator>(const Rational &q) const {
203   return (num*q.den > den*q.num);
204 }
205 bool Rational::operator>=(const Rational &q) const {
206   return (num*q.den >= den*q.num);
207 }
208
209 int main() {
210   Rational q;
211   int num, den;
212   cout << "Entrez un nombre Rational (fraction) ! " << endl;
213   cout << "numerateur : ";
214   cin >> num;    q.SetNum(num);
215   cout << "denominateur : ";
216   cin >> den;  q.SetDen(den);
217   q.Reduce();
218   cout << "La fraction est : ";
219   q.Print();
220   cout << "et sa valeur decimal : " << q.Convert() << endl;
221
222   cout << endl;
223   cout << "Operations arithmetiques avec Rational" << endl;
224   Rational a(21, 9);
225   cout << "a = ";
226   a.Print();
227   Rational b(6, 36);
228   cout << "b = ";
229   b.Print();
230   Rational s = a+b;
231   cout << "a + b = ";
232   s.Print();
233   Rational d = a-b;
234   cout << "a - b = ";
235   d.Print();
236   Rational p = a*b;
237   cout << "a * b = ";
238   p.Print();
239   Rational r = a/b;
240   cout << "a / b = ";
241   r.Print();
242   cout << "a / b (decimal) = " << r.Convert() << endl;
243   Rational t = -a;
244   cout << "-a = ";
245   t.Print();
```

```
246   Rational u = a;      //copie
247   cout << "u = ";
248   u.Print();
249
250   cout << endl;
251   cout << "Operations logiques avec Rational" << endl;
252   if (a==b) cout << "a et b sont egales !" << endl;
253   else cout << "a et b sont differentes !" << endl;
254   if (a<b) cout << "a est plus petite !" << endl;
255   if (a>b) cout << "a est plus grande !" << endl;
256
257   //avec un pointeur
258   cout << endl;
259   cout << "Rational avec un pointeur" << endl;
260   Rational *x;
261   x = new Rational;
262   x->SetNum(5);
263   x->SetDen(13);
264   cout << "x = ";
265   x->Print();
266   cout << "x (decimal) = " << x->Convert() << endl;
267   delete x;
268
269   return 0;
270 }
```

4. **Implémentation de la classe des nombres complexes** `Complexe`
   Un nombre complexe est caractérisé par sa partie réelle et imaginaire. Trois constructeurs sont définis, le constructeur par défaut, un constructeur de copie et un constructeur qui prend en paramètre d'entrée les parties réelle et imaginaire et initialise les données privées qui sont le module et la phase. Nous avons surchargé les opérateurs de somme `+`, soustraction `-`, multiplication `*` et division `/` entre nombres complexes, et nous avons ajouté une méthode pour prendre le conjugué, et pour accéder à la norme.
   Complexe.cpp.

```
1  //developement d'une classe pour les nombres complexes
2  #include <iostream>
3  #include <iomanip>
4  #include <cmath>
5
6  using namespace std;
7
8  class Complexe {
9    public:
10     //constructeurs
11     Complexe();
12     Complexe(double re_z, double im_z);
13     Complexe(const Complexe &r);      //constructeur de copie
14
15     //fonctions d'acces
16     double getModZ() const;
17     double getPhaseZ() const;
18
19     //fonctions surchargees
20     Complexe operator+(const Complexe &) const;
21     Complexe operator-() const;
22     Complexe operator-(const Complexe &) const;
23     Complexe operator*(const Complexe &) const;
```

```
24      Complexe operator/(const Complexe &) const;
25
26      //autres methodes publiques
27      void print() const;
28      Complexe conjugue() const;
29      double norm() const;
30
31    private:
32      double reZ, imZ;
33      double modZ, phaseZ;
34  };
35
36  //constructeurs
37  Complexe::Complexe() {
38  }
39  Complexe::Complexe(double re_Z, double im_Z): reZ(re_Z), imZ(im_Z) {
40    //les autres donnees privees sont initialisees
41    //lors de l'initialisation de la classe
42    modZ = sqrt(reZ*reZ+imZ*imZ);
43    //double tang = imZ/reZ;
44    phaseZ = atan(imZ/reZ);
45  }
46  Complexe::Complexe(const Complexe &z): reZ(z.reZ), imZ(z.imZ) {
47  }
48
49  //fonctions d'acces
50  double Complexe::getModZ() const {
51    return modZ;
52  }
53  double Complexe::getPhaseZ() const {
54    return phaseZ;
55  }
56
57  //operateurs surchargees
58  Complexe Complexe::operator+(const Complexe &v) const {
59    Complexe res(reZ+v.reZ,imZ+v.imZ);
60    return res;
61  }
62  Complexe Complexe::operator-() const {
63    Complexe res(-reZ,-imZ);
64    return res;
65  }
66  Complexe Complexe::operator-(const Complexe &v) const {
67    Complexe res(reZ-v.reZ,imZ-v.imZ);
68    return res;
69  }
70  Complexe Complexe::operator*(const Complexe &v) const {
71    //pour la multiplication des complexes on peut utilizer aussi
72    //la notation polaire, mais la precision peut etre pire
73    //  float reAux = modZ*v.modZ *cos(phaseZ+v.phaseZ);
74    //  float imAux = modZ*v.modZ *sin(phaseZ+v.phaseZ);
75    double reAux = reZ*v.reZ - imZ*v.imZ;
76    double imAux = reZ*v.imZ + imZ*v.reZ;
77
78    Complexe res(reAux,imAux);
79    return res;
80  }
81  Complexe Complexe::operator/(const Complexe &v) const {
82    //la division entre complexes z/v est en fait z*v.conjugue()/v.norm()^2
83    //on pourrait aussi definir la division exactement de cette maniere,
```

```
84    //mais il faudrait prevoir la division d'un nombre complexe par un reel
85    if (v.modZ==0)
86      cout << "Attention !, division par zero !" << endl;
87    double reAux = (reZ*v.reZ + imZ*v.imZ) / (v.modZ*v.modZ);
88    double imAux = (imZ*v.reZ - reZ*v.imZ) / (v.modZ*v.modZ);
89
90    Complexe res(reAux,imAux);
91    return res;
92  }
93
94  //methodes publiques
95  //nous n'imprimons pas la partie reelle si cette est zero
96  //meme pour la partie imaginaire
97  //si les deux partie sont egaux a zero, on imprime 0
98  void Complexe::print() const {
99    if (reZ!=0) {
100     cout << setprecision(5) << setw(7) << reZ;
101     if (imZ!=0) {
102       if (imZ>0)
103         cout<<" + ";
104       else
105         cout<<" - ";
106       cout << setprecision(5) << setw(7) << fabs(imZ) << " i";
107     }
108   }
109   else
110     if (imZ!=0)
111       cout << setprecision(5) << setw(7) << imZ << " i";
112     else
113       cout << setprecision(5) << setw(7) << 0;
114   cout << endl;
115 }
116
117 Complexe Complexe::conjugue() const {
118   Complexe res(reZ,-imZ);
119   return res;
120 }
121
122 double Complexe::norm() const {
123   return modZ;
124 }
125
126 int main(){
127   double re,im;
128   cout << "Entrez la partie reelle d'un nombre complexe : ";
129   cin >> re;
130   cout <<"et la partie imaginaire : ";
131   cin >> im;
132   Complexe a(re,im);
133   cout << "a = ";   a.print();
134   cout << "mod a: " << a.getModZ() << endl;
135   cout << "phase a: " << a.getPhaseZ()/M_PI*180.0 << endl;
136
137   Complexe b(0.,-0.9899887);
138   cout << "b = ";
139   b.print();
140
141   Complexe c = a-b;
142   cout << "c = a - b = ";
143   c.print();
```

```
144
145     Complexe d = a*b;
146     cout << "d = a * b = ";
147     d.print();
148
149     Complexe f = d.conjugue();
150     cout << "f = d* = ";
151     f.print();
152
153     Complexe g = c/a;
154     cout << "g = c / a = ";
155     g.print();
156
157     cout << "mod a = " << a.norm() << ", mod b = " << b.norm()
158          << ", mod c = " << c.norm() << "," << endl;
159     cout << "mod d = " << d.norm() << ", mod f = " << f.norm()
160          << ", mod g = " << g.norm() << endl;
161
162     return 0;
163 }
```