UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES

Méthodes informatiques pour physiciens

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

Corrigé 2

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/L2` :
pour les notes du cours, les exercices et les corrigés

2.1 Questions

1. **Déterminer si chacune des expressions suivantes est vraie ou fausse :**

   (a) !(p||q) *est identique à* !p||!q et est **fausse**. !(p||q) est vraie seulement si p||q est fausse. Cela arrive quand `p` et `q` sont fausses, c'est-à-dire !p&&!q (attention à la priorité des opérateurs). !p||!q est vraie aussi lorsqu'une des deux est fausse et l'autre vraie, donc les résultats ne sont pas équivalents. Voir la table de vérité ci dessous.

p	v	v	f	f
q	v	f	v	f
$(p\|\|q)$	v	v	v	f
$!(p\|\|q)$	f	f	f	v
$!p$	f	f	v	v
$!q$	f	v	f	v
$!p\|\|!q$	f	v	v	v
$!p\&\&!q$	f	f	f	v

TABLE 1 – Table de vérité pour déterminer le résultat de !(p||q). Il y a quatre possibilité : `p` et `q` sont vraies, `p` est vraie et `q` est fausse, `p` est fausse et `q` vraie, les deux sont fausses. Les résultats des opérations logiques sont montrés dans les colonnes. Par exemple, si `p` et `q` sont vraies, on voit dans la première colonne que p||q est aussi vraie et !(p||q) fausse, aussi comme !p||!q et !p&&!q. L'expression qui donne le même résultat que !(p||q) est !p&&!q .

   (b) !!!p *est identique à* !p est **vraie**, parce que deux négations consécutives " `!`" s'annulent entre elles. Un nombre impair d'opérateurs logiques " `!`" est équivalent à un seul opérateur logique " `!`", tandis que un nombre pair d'opérateurs logiques " `!`" est equivalent à n'en avoir aucun.

   (c) p&&q||r *est identique à* p&&(q||r) est **fausse**, parce que l'opérateur logique && à la priorité sur l'opérateur ||. L'ordinateur exécutera d'abord l'opération logique && et après, avec le résultat de cette opération, l'operation || (ça veut dire que p&&q||r est équivalente à (p&&q)||r).

2. **Quel est le nombre minimal d'itérations que peut faire une boucle `while` ? et une boucle `do - while` ?**
   Le nombre minimal d'iterations qu'une boucle `while` peut faire est zéro. Une boucle `do - while` fera au moins une itération.

3. **Quel est le problème dans la boucle**
   ```
   while (n<100)
     sum += n*n;  ?
   ```
   Il n'y a pas d'incrémentation (ou de décrémentation) de la variable `n`, qui joue aussi le rôle de variable de contrôle. La valeur initiale de `n` ne change pas dans la boucle et la boucle sera exéecutée une infinité de fois si `n < 100` .

4. **Comment peut-on écrire une boucle de sorte qu'elle se termine avec une instruction située au milieu du bloc correspondant ?**
   Nous pouvons sortir d'une boucle avec une instruction `break` située au milieu du bloc d'instructions.

5. **Pourquoi utiliser des parenthèses superflues lorsque la priorité des opérateurs est évidente ?**
   Afin d'éviter toute ambiguït, soit pour les autres programmeurs qui vont utiliser vos

programmes, soit pour vous même et pour d'être sûr(e) que vos programmes font ce que vous voulez. Faites attention á la portée de variables définies dans des blocs d'instructions.

6. **Soit `inst` une instruction et `exp` une expression. Comment peut-on écrire les boucles suivantes de manière équivalente à `while` ?**

(a)
```
for ( ; exp; )
    inst;
```

(b)
```
for ( ; ; exp)
    inst;
```

La boucle dans le cadre (a) est équivalente à celle dans le cadre (a') ci-dessous et la boucle dans le cadre (b) est équivalente à celle dans le cadre (b').

(a')
```
while (exp)
    inst;
```

(b')
```
while (1) {
    inst;
    exp; }
```

7. **De même, quel est l'équivalent de :**
   `for (exp1; exp2; exp3) inst;` **?**
   Que se passe-t-il si `inst` contient une instruction `continue` ?
   L'équivalent `while` de cette boucle `for` est montré dans le cadre (c). Si `inst` contient une instruction `continue` au milieu, `inst` sera exćutée jusqu'à cette instruction e le programme reviendra au début de la boucle. Dans le cas de la boucle `for` les variables des contrôle seront mises au jour (`exp3`) avant de reprendre l'exécution de la boucle. Dans le cas de la boucle `while`, l'instruction `exp3` ne sera pas exécutée et donc il n'y aura pas la mise au jour des variables de contrôle. On cour le risque de construire une boucle infinie.

(c)
```
exp1;
  while (exp2) {
    inst;
    exp3;
}
```

8. **Si les opérateurs relationnels renvoient `true` ou `false`, pourquoi une valeur différente de zéro est-elle considérée comme vraie ? Les nombres négatifs peuvent-ils avoir les valeurs `true` ou `false` ?**
   Le variable logique de type `bool` est représenté par un octet (*byte*). La valeur `false` est représentée avec tous les bits mis à zéro. Quand cette variable booléenne est convertie en une variable de type entière cella sera le zéro. Si n'importe quel bit de la variable booléenne est égal à 1, la valeur de ce-ci est `true`. Quand convertie en une variable de type entière sa valeur sera différente de zéro. Le premier bit des nombres négatifs est égal à 1 ; donc les valeurs sont toujours équivalents à `true`.

2.2 Trouvez l'erreur !

Tous ces morceaux de code (sauf le deuxième) sont correct du point de vue de la syntaxe **C++**. Le compilateur ne générera pas de messages d'erreur. Seulement pendant ou après l'exécution du programme on s'apercevra du mal fonctionnement du programme, parce qu'il produira des résultats inattendus. Dans la doute, il faut toujours écrire un petit programme pour tester le fonctionnement du **C++**.

1. L'expression dans la condition `if (x=0)` est une affectation, pas une comparaison. L'expression correcte est `if (x==0)`. En général une affectation donne toujours une réponse affirmative sauf que avec le zéro, donc l'affectation `x = 0` donne une réponse fausse. Dans cet exemple la première instruction ne sera jamais exécutée, tandis que la deuxième le sera toujours.
2. La condition `x < y < z`, bien qu'intuitive, n'est pas compréhensible pour l'ordinateur, parce qu'il y a deux comparaisons. L'expression correcte est `if (x<y && y<z)`.
3. L'instruction `else` est executée lorsque `x` et `y` sont égaux à zéro. Cette instruction est alternative à l'instruction qui suive la condition `if (y==0)`. Pour que le programme fasse ce que nous voulons, nous devons mettre des accolades autour du deuxième `if` :

```
if (x==0) {
  if (y==0) cout << "x et y sont egaux a zero" << endl;
}
else cout << "x n'est pas egal a zero" << endl;
```

   ou combiner les deux conditions en une avec des opérateurs logiques :

```
if (x==0 && y==0) {
  cout << "x et y sont egaux a zero" << endl;
}
else {
  if (x!=0) cout << "x n'est pas egal a zero" << endl;
  if (y!=0) cout << "y n'est pas egal a zero" << endl;
}
```

4. Il n'y a pas d'incrémentation ou de décrémentation de la variable de contrôle `compteur` dans la boucle. La boucle devient donc infinie, parce que la condition `while` sera toujours vraie.
5. Il y a un point virgule de trop, à la fin de la ligne de l'instruction `for`.
6. La variable `compteur` est initialisée avec une valeur supérieure à `10`, donc le programme n'entrera pas dans la boucle. Par contre si nous initialisons la variable avec n'importe quelle valeur inférieure à `10`, le programme entrera dans la boucle et ne s'arrêtera jamais, parce que la valeur de `compteur` est décrémentée de `1` à chaque itération. P. ex. on peut changer la condition dans l'instruction `while (compteur<10)` avec `while (compteur>10)`. Dans ce cas nous avons un compte à rebours depuis `100` jusqu'à `11`.

2.3 Exercises avec `if`

Exercice 2 : Deux solutions sont proposées. Dans le programme **Minmax_v1.cpp** on utilise seulement les opérateurs relationnels, tandis que dans le programme **Minmax_v2.cpp** on utilise aussi l'opérateur logique `&&`. La conditions composée est vrai (`true`) si les deux conditions sont vrais (écriture plus compacte et facile à lire).

Minmax_v1.cpp

```
1 //Le programme trouve le min et la max parmi 3 nombres
2 #include <iostream>
3
4 using namespace std;
```

```
5
6  int main() {
7    int a, b, c;
8    cout << "Entrez les 3 nombres a comparer :" << endl;
9    cin >> a >> b >> c;
10
11   int min, max;
12   if (a > b)
13     if (a > c) {
14       max = a;
15       if (b > c)
16       min = c;
17       else
18       min = b;
19     }
20     else {
21       max = c;
22       min = b;
23     }
24   else
25     if (b > c) {
26       max = b;
27       if (a > c)
28       min = c ;
29       else
30       min = a;
31     }
32     else {
33       max = c;
34       min = a;
35     }
36
37   cout << "Le minimum est " << min << " et le maximum est " << max << endl;
38
39   return 0;
40 }
```

Minmax_v2.cpp

```
1  //le programme trouve le min et la max parmi 3 nombres
2  #include <iostream>
3
4  using namespace std;
5
6  int main() {
7    double a, b, c;
8    cout << "Entrez les 3 nombres a comparer: " << endl;
9    cin>> a >> b >> c;
10
11   double min, max;
12   if (a > b && a > c) {      //la variable a est le maximum
13     max = a;
14     if (b > c)
15       min = c;
16     else
17       min = b;
18   }
19   else
20     if (a < b && a < c) {    //la variable a est le minimum
21       min = a;
```

```
22        if (c > b)
23          max = c;
24        else
25          max = b;
26      }
27      else                        //la variable a est au milieu
28        if (c > b) {
29          max = c;
30          min = b;
31        }
32        else {
33          max = b;
34          min = c;
35        }
36
37    cout << "Le minimum est " << min << " et le maximum est " << max << endl;
38
39    return 0;
40  }
```

Exercise 3 : Voir et exécuter le programme **Ex3_if.cpp**. Dans cette version le code est plus lisible et on peut voir, par exemple, que les instructions `a = a - 1;` et `c = c - 1;` ne seront jamais exécutées. Si la condition `b > 0` est fausse, la condition `b < 4` est vrai et donc seulement l'instruction `c = c + 1;` sera exécutée. En ce qui concerne l'instruction `c = c - 1;`, si la condition `c > 0` est fausse la condition `c < 4` est vrai et donc aussi l'instruction `c = c - 1;` ne sera jamais exécutée.

Ex3_if.cpp

```
1   //programme obscure
2   #include <iostream>
3
4   using namespace std;
5
6   int main() {
7     int a,b,c;
8     cout << "Entrez 3 nombres :" << endl;
9     cin >> a >> b >> c;
10
11    if (a > 0) {
12      if (b > 0)
13        a = a + 1;
14      else if (c > 0) {
15        if (a < 4)
16          b = b + 1;
17        else if (b < 4)
18          c = c + 1;
19        else
20          a = a - 1;
21      }
22      else if (c < 4)
23        b = b - 1;
24      else
25        c = c - 1;
26    }
27    else
28      a = 0;
29
30    cout << "a = " << a << " b = " << b << " c = " << c << endl;
31
```

```
32    return 0;
33  }
```

Exercice 4 : Deux solutions équivalentes sont proposées. Dans le programme **Somme_v1.cpp** on calcule la somme des chiffres d'un nombre en additionnant chiffre après chiffre. La valeur de chaque chiffre est obtenue à partir du modulo de la division de ce nombre par une puissance de 10. Par exemple la valeur des unités est donnée par le reste (modulo) de la division par 10, etc. Dans le programme **Somme_v2.cpp** on utilise une boucle `for` : à chaque itération on extrait une chiffre en commençant par la dernière. Chaque chiffre est additionnée à la variable `somme`.

Somme_v1.cpp

```
1  //ce programme imprime la somme des chiffres d'un entier donne
2  #include <iostream>
3
4  using namespace std;
5
6  int main() {
7    int n;
8    cout << "Entrez un entier a six chiffres : ";
9    cin >> n;
10
11   int somme = 0;
12   somme = n%10 + (n/10)%10 + (n/100)%10 + (n/1000)%10
13           + (n/10000)%10 + (n/100000)%10;
14   cout << "La somme des chiffres de " << n << " est " << somme << endl;
15
16   return 0;
17 }
```

Somme_v2.cpp

```
1  //ce programme imprime la somme des chiffres d'un entier donne
2  #include <iostream>
3
4  using namespace std;
5
6  int main() {
7    int a = 0;
8    cout << "Entrez un numero entier < 1000000 (max 6 chiffres) : ";
9    cin >> a;
10
11   int somme = 0;
12   int chiffre = 0;
13   for(int i=1; i<1000000; i*=10) {
14     chiffre = (a/i)%10;
15     somme += chiffre;
16   }
17   cout << "La somme des chiffres est " << somme << endl;
18
19   return 0;
20 }
```

2.4 Exercices avec les boucles

Exercice 1 : Etudiez, compilez et exécutez **Ex1_Serie_for.cpp**, **Ex1_Serie_while.cpp** et **Ex1_Serie_do.cpp**. Dans le programme **Ex1_Serie_while.cpp** il faut faire attention à

l'opérateur de pré-incrémentation `++` : on a écrit `++i` au lieu de `i++`. D'abord on incrémente `i` de `1` (`i = i + 1`), puis on ajoute le terme à la série. Par contre, si on utilise la post-incrémentation `i++`, d'abord on ajoute le terme à la série puis on incrémente `i` de `1`. Le programme sera correctement compilé (il n'y a pas d'erreurs de syntaxe), mais pendant la première itération de la boucle `while` le programme effectuera une division par zéro dont le résultat est `1.#INF`. Essayez !

Ex1_Serie_while.cpp

```
1  //somme des n premiers reciproques
2  #include <iostream>
3
4  using namespace std;
5
6  int main() {
7    int n;
8    cout << "Entrez un entier positif : ";
9    cin >> n;
10
11   double somme = 0.;
12   int i = 0;
13   while (i <= n)
14     somme += 1./++i;  //difficile a dechifrer
15   /*code equivalant, plus simple a dechifrer
16   { i = i + 1;
17     somme = somme + 1./i;
18   }
19   */
20
21   cout << "La somme des " << n << " premiers reciproques est : " << somme
        << endl;
22
23   return 0;
24 }
```

Exercice 5 : Les trois programmes **Factorielle_for.cpp**, **Factorielle_do.cpp** et **Factorielle_while.cpp** sont équivalents. La solution la plus compacte est celle avec la boucle `for`. La factorielle est calcule dans le sens $n! = 1 \times 2 \times 3 \times \ ... \times (n-1) \times n$. Dans les deuxièmes versions de ces programmes la factorielle est calcule dans le sens inverse $n! = n \times (n-1) \times \ ... \times 2 \times 1$. Dans **Factorielle_do_v2.cpp** on a du ajouter une instruction à la fin de la boucle pour tenir compte du cas `n = 0`, parce que le programme entre dans la boucle `do - while` avant de vérifier la condition `i <= n`.

Factorielle_for.cpp

```
1  //calcul de la factorielle avec for : 1...(n-1)n
2  #include <iostream>
3
4  using namespace std;
5
6  int main() {
7    int n;
8    cout << "Calcul de la factorielle, entrez n >= 0: ";
9    cin >> n;
10
11   int fact = 1;
12   for (int i=1; i <= n; i++)
13     fact = fact * i;
14
15   cout << "La factorielle de " << n << " est : " << fact << endl;
```

```
16
17    return 0;
18  }
```

Factorielle_while.cpp

```
1  //calcul de la factorielle avec while : 1...(n-1)n
2  #include <iostream>
3
4  using namespace std;
5
6  int main() {
7    int n;
8    cout << "Calcul de la factorielle, entrez n >= 0 : ";
9    cin >> n;
10
11   int fact = 1;
12   int i = 1;
13   while (i <= n) {
14     fact = fact * i;
15     i++;  //i = i + 1
16   }
17
18   cout << "La factorielle de " << n << " est : " << fact << endl;
19
20   return 0;
21 }
```

Factorielle_do.cpp

```
1  //calcul de la factorielle avec do - while : 1...(n-1)n
2  #include <iostream>
3
4  using namespace std;
5
6  int main() {
7    int n;
8    cout << "Calcul de la factorielle, entrez n >= 0 : ";
9    cin >> n;
10
11   int fact = 1;
12   int i = 1;
13   do {
14     fact = fact * i;
15     i++;  //i = i + 1
16   } while (i <= n);
17
18   cout << "La factorielle de " << n << " est : " << fact << endl;
19
20   return 0;
21 }
```

Exercice 6 : Etudiez, compilez et exécutez les programmes **Ex6_Serie_for.cpp**, **Ex6_Serie_while.cpp** et **Ex6_Serie_do.cpp**. Les trois programmes sont équivalents et trés similaires aux programmes donnés dans l'**Exercice 1**.

Ex6_Serie_for.cpp

```
1  //somme des n premiers termes d'une serie
2  #include <iostream>
```

```
3  #include <cmath>
4
5  using namespace std;
6
7  int main(){
8    int n;
9    cout << "Entrez un entier positif : ";
10   cin >> n;
11
12   double somme = 0.;
13   for (int i=1; i<=n; i++)
14     somme += pow(-1.,i)/(i*i);
15
16   cout << "La somme des " << n << " premiers termes est : " << somme <<
       endl;
17
18   return 0;
19 }
```

Ex6_Serie_while.cpp

```
1  //somme des n premiers termes d'une serie
2  #include <iostream>
3  #include <cmath>
4
5  using namespace std;
6
7  int main() {
8    int n;
9    cout << "Entrez un entier positif : ";
10   cin >> n;
11
12   double somme = 0.;
13   int i = 1;
14   while (i <= n) {
15     somme += pow(-1.,i)/(i*i);
16     i++;
17   }
18
19   cout << "La somme des " << n << " premiers reciproques est : " << somme
       << endl;
20
21   return 0;
22 }
```

Ex6_Serie_do.cpp

```
1  //somme des n premiers termes d'une serie
2  #include <iostream>
3  #include <cmath>
4
5  using namespace std;
6
7  int main() {
8    int n;
9    cout << "Entrez un entier positif : ";
10   cin >> n;
11
12   double somme = 0.;
13   int i = 1;
```

```
14    do {
15      somme += pow(-1.,i)/(i*i);
16      i++;
17    } while (i <= n);
18
19    cout << "La somme des " << n << " premiers termes est : " << somme <<
          endl;
20
21    return 0;
22  }
```

2.5 Problèmes

N'oubliez pas d'analyser le problème et de dessiner le diagramme de flux avant d'écrire le programme.

Rebond d'une balle : Le même programme repond à deux questions. Après le saisi de la hauteur initiale et de la perte d'énergie fractionnaire à chaque rebond, le programme demande à l'utilisateur de choisir à quelle question répondre. Le choix plus naturel pour gérer les deux réponses est d'utiliser l'instruction `switch` (voir **Rebond_v2.cpp**). La solution donnée ci-dessous utilise la construction `if - else`.

Pour répondre aux questions nous avons besoin des boucles. La solution à la première question utilise une boucle `for` et calcule l'énergie de la balle après chaque rebond et sort de la boucle après `n` cycles (`n` est donné par l'utilisateur). La figure 1 montre le diagramme de flux pour ce problème.

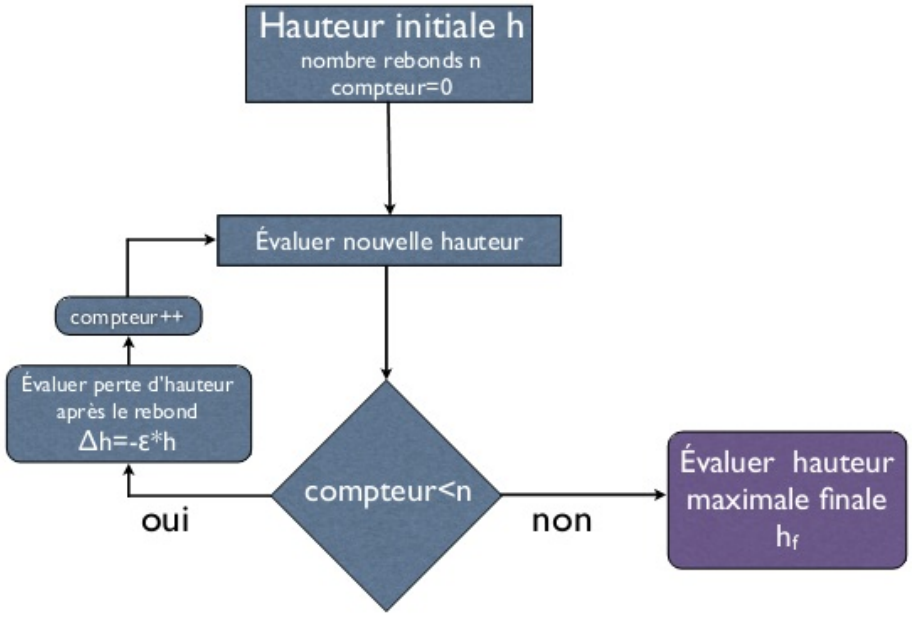

FIGURE 1 – Diagramme de flux pour le calcul de la hauteur maximale après n rebonds.

La solution à la deuxième question utilise une boucle `do - while`. Le programme calcule l'énergie de la balle aprés chaque rebond jusqu'á elle descend á 1% de l'énergie initiale. Une variable compte le nombre des rebonds et les imprime à la sortie de la boucle. La figure 2 montre le diagramme de flux pour ce problème.

Rebond.cpp

```
1  //reonds d'une balle avec perte d'energie a chaque rebond
2  #include <iostream>
3
```

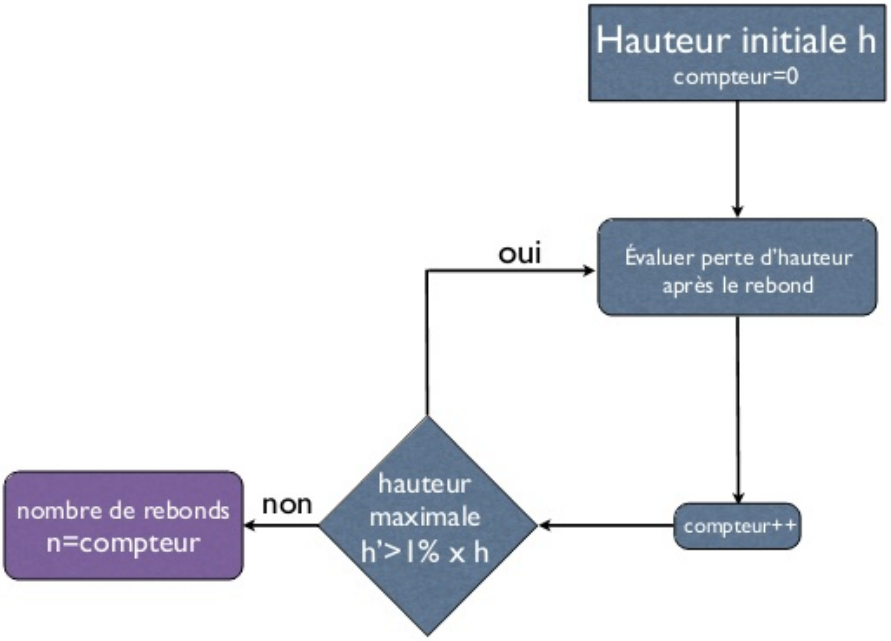

FIGURE 2 – Diagramme de flux pour le calcul du nombre de rebonds nécessaires pour perdre 99% de l'energie mécanique de la balle.

```
4  using namespace std;
5
6  int main() {
7    double h;
8    cout << "Quelle est la hauteur initiale h en metres ? ";
9    cin >> h;
10
11   double eps;
12   cout << "Quelle est la perte de energie a chaque rebond, en % ? ";
13   cin >> eps;
14   eps = eps/100.;
15
16   char opt;
17   cout << "A quelle question voulez-vous repondre (a ou b) ? ";
18   cin >> opt;
19
20   if (opt=='a') {
21     int n;
22     cout << "Combien de rebonds voulez-vous evaluer ? ";
23     cin >> n;
24
25     for (int compt=0; compt<n; ++compt) {
26       h = h*(1.0-eps);
27       if (h <= 0.) {
28       cout << "La balle a perdu toute son energie apres " << compt+1 << "
             rebonds" << endl;
29       break;
30       }
31     }
32     cout << "La hauteur finale est " << h << " metres." << endl;
33   }
34   else if (opt=='b') {
35     int compt = 0;
36     double hfinale = h*0.01;
37     do {
```

```
38          h = h*(1.0-eps);
39          compt++;
40        } while (h > hfinale);
41        cout << "Le nombre de rebonds pour atteindre 1% de la hauteur initiale
              est " << compt << endl;
42    }
43    else
44      cout << "pas une bonne choix ! " << endl;
45
46    return 0;
47  }
```

Estimez π : La figure 3 montre le principe du calcul de π. D'abord il faut inscrire un cercle dans un carré de coté `a`, puis répartir le carré en n × n petits carrés de coté `a/n` et enfin compter tous les carrés dans le cercle. Si le centre du petit carré est dans le cercle on considère le petit carré inscrit dans le cercle. Dans la figure 3, les petits carrés de couleur verte sont ceux dont le centre est dans le cercle. Si Σ est le nombre des petits carrés dans le cercle, π est donné par $\pi = \Sigma \times 4/n^2$. Ce résultat ne dépend pas de `a`. Pour augmenter la précision du calcul il faut augmenter le nombre de divisions n.

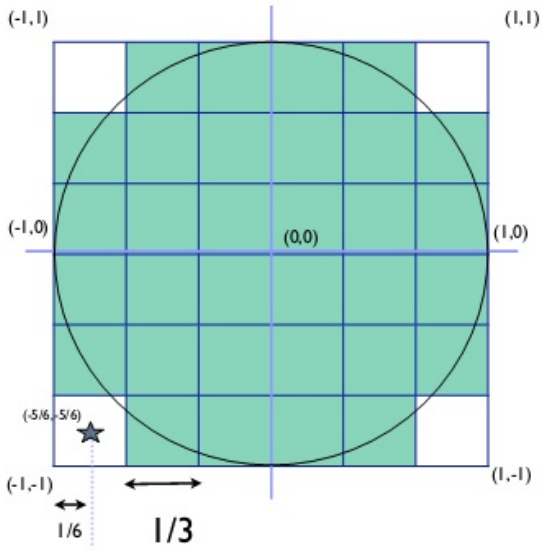

FIGURE 3 – Principe du calcul de π avec $n = 6$.

Dans la figure 3, `a = 2`. La position du centre du premier petit carré (en-bas à gauche) est $x = -1 + 1/n$, parce que le coté du carré est $2/n$. La position du centre du petit carré suivant est décalé de $2/n$ par rapport au premier carré. Ici, nous avons choisi d'avancer d'abord en horizontal et après en vertical, une fois que le rang horizontal est fini. Pour parcourir tous les petits carrés nous avons besoin des deux boucles imbriques, une boucle pour parcourir le carré en horizontal (boucle interne) et une boucle pour parcourir le carré en vertical (boucle externe). La boucle `for` est le choix le plus naturel pour ce calcul. Le diagramme de flux du programme est illustré dans la figure 4.
Les fonctions `setw()` et `setprecision()` sont utilisées pour la mis en page de la sortie. Elle sont définies dans la bibliothèque `iomanip`, qu'il faut inclure en tête du programme. La fonction `setw()` détermine le nombre de caractères que l'ordinateur réservera pour écrire la variable qui suive la fonction `setw()`. La fonction `setprecision()` détermine le nombre de chiffres décimaux popur la même variable.

Pi.cpp

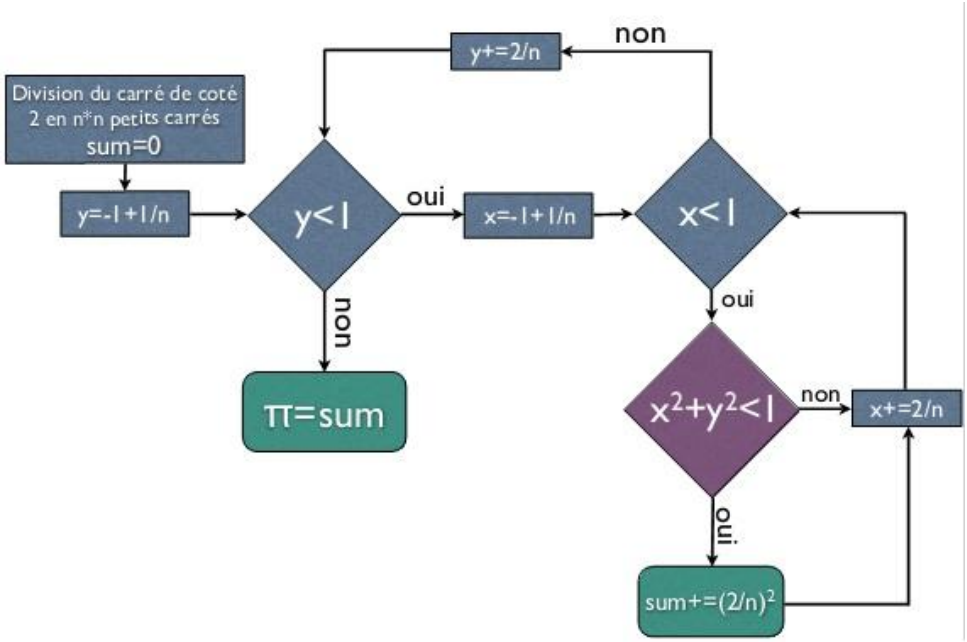

FIGURE 4 – Diagramme de flux pour le calcul de π

```
1  //calcul de pi par "integration numerique"
2  #include <iostream>
3  #include <iomanip>
4  #include <cmath>
5
6  using namespace std;
7
8  int main() {
9    int n;
10   cout << "Combien de divisions pour le carre ? ";
11   cin >> n;
12
13   int somme = 0;
14   double x, y, r;
15   double div = 2./n;
16   double x0 = 1./n - 1.;
17   for (int ix=0; ix<n; ix++) {
18     for (int iy=0; iy<n; iy++) {
19       x = ix*div + x0;
20       y = iy*div + x0;
21       r = x*x + y*y;
22       if (r < 1.) somme++;
23     }
24   }
25   double pi = somme * (div * div);
26   cout << "Le resultat de l'integration numerique pour pi est "
27        << setw(15) << setprecision(10) << pi << endl;
28   cout << "tandis que la valeur de pi est "
29        << setw(15) << setprecision(10) << M_PI << endl;
30
31   return 0;
32 }
```