Chapitre 3 : Structures de contrôle

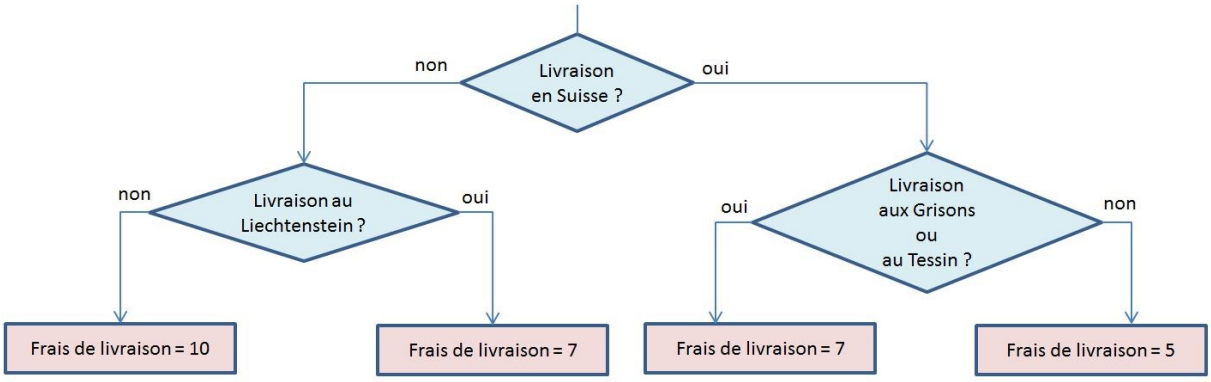
Exercice 3.1 Frais de livraison

Une société suisse tarife ses frais de livraison comme suit :

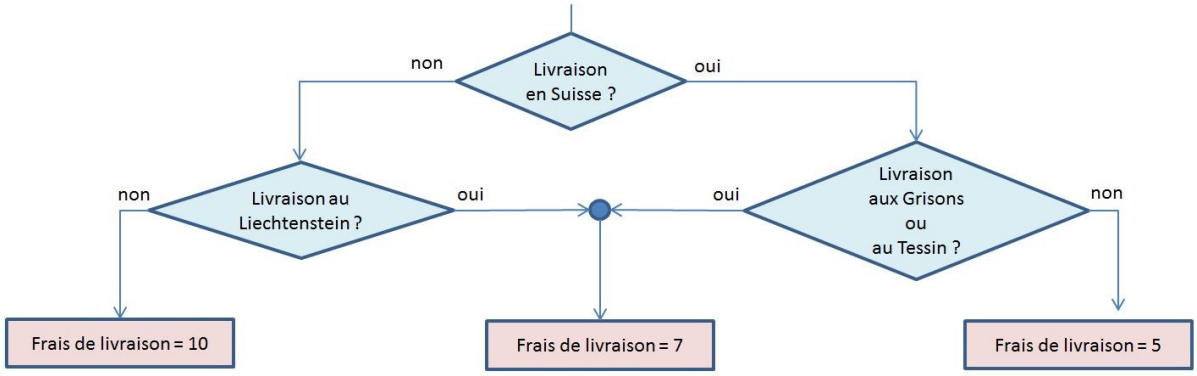
- 5 CHF si livraison en Suisse, à l'exception des cantons des Grisons et du Tessin où les frais de livraison se monte à 7 CHF
- 7 CHF si livraison au Liechtenstein
- 10 CHF partout ailleurs dans le monde

Représenter sous forme d'organigramme les diverses situations décrites ci-dessus.

Solution exercice 3.1



A ne pas faire, par contre :



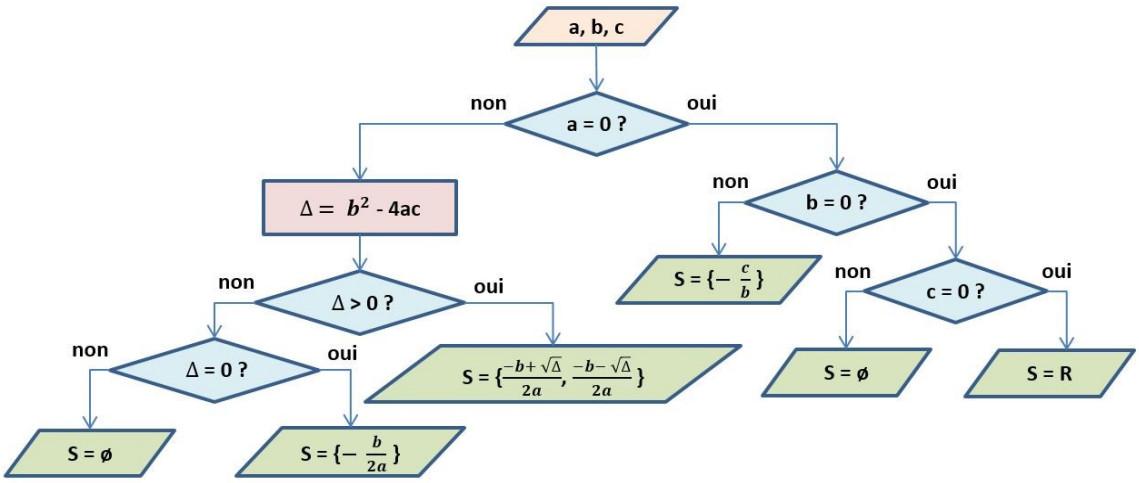
Exercice 3.2 Algorithme de résolution de $ax^2 + bx + c = 0$

Représenter sous forme d'organigramme l'algorithme de résolution de l'équation :

$ax^2 + bx + c = 0; a, b, c \; et \; x \in R$

Solution exercice 3.2

S = ensemble-solution



Exercice 3.3 If… else équivalents ou pas ?

Les deux extraits de code suivants sont-ils équivalents ? Justifier votre réponse.
Dans les 2 cas ci-dessous, on suppose bien sûr que prixActuel est tel que prixActuel – le rabais accordé est > 0.

1)
```
if (prixActuel > 100) {
   nouveauPrix = prixActuel - 20;
} else {
   nouveauPrix = prixActuel - 10;
}
```

2)
```
if (prixActuel < 100) {
   nouveauPrix = prixActuel - 10;
} else {
   nouveauPrix = prixActuel - 20;
}
```

Solution exercice 3.3

Non.

Si prixActuel vaut 100 :
1) => nouveauPrix = prixActuel – 10 = 90
2) => nouveauPrix = prixActuel – 20 = 80

Exercice 3.4 If…else dans tous leurs états

Quelle(s) remarque(s) vous inspirent les extraits de code suivants ?
NB Dans chacun des cas, proposez un correctif si vous estimez que l'extrait de code est mal écrit ou qu'il présente une erreur.
Dans tous les cas ci-dessous, on suppose bien sûr que prixActuel est tel que prixActuel – le rabais accordé est > 0.

1)
```
if (prixActuel >= 100) {
   nouveauPrix = prixActuel - 20;
   cout << "Nouveau prix = " << nouveauPrix << endl;
} else {
   nouveauPrix = prixActuel - 10;
   cout << "Nouveau prix = " << nouveauPrix << endl;
}
```

2)
```
if (prixActuel < 100) {
   nouveauPrix = prixActuel - 10;
} else if (prixActuel = 100) {
   nouveauPrix = prixActuel;
} else {
   nouveauPrix = prixActuel + 10;
}
```

3)
```
if (prixActuel < 100) {
   nouveauPrix = prixActuel - 10;
} else if (prixActuel >= 100 && prixActuel <= 150) {
   nouveauPrix = prixActuel - 15;
} else {
   nouveauPrix = prixActuel - 20;
}
```

4)
```
if (prixActuel < 100) {
   nouveauPrix = prixActuel - 10;
} else if (prixActuel >= 100) {
   nouveauPrix = prixActuel - 20;
}
```

5)
```
if (prixActuel >= 100) {
   droitAuRabais = true;
} else {
   droitAuRabais = false;
}
```

6)
```
if (prixActuel >= 100) {
   pourcentRabais = 5;
} else if (prixActuel > 500) {
   pourcentRabais = 10;
} else if (prixActuel > 1000) {
   pourcentRabais = 15;
} else {
   pourcentRabais = 0;
}
```

Solution exercice 3.4

1) Code maladroit car redondant.
   Correctif :

```
if (prixActuel >= 100) {
   nouveauPrix = prixActuel - 20;
} else {
   nouveauPrix = prixActuel - 10;
}
cout << "Nouveau prix = " << nouveauPrix << endl;
```

   ou

```
nouveauPrix = prixActuel >= 100 ? prixActuel - 20 : prixActuel - 10;
cout << "Nouveau prix = " << nouveauPrix << endl;
```

2) Code faux.
   La branche else ne sera jamais exécutée… car la condition précédente (else if) est toujours vraie (affectation et non comparaison!).
   Corollaire : si, par ex, prixActuel = 110, on obtiendra nouveauPrix = 100 au lieu de 120 !
   Correctif :

```
if (prixActuel < 100) {
   nouveauPrix = prixActuel - 10;
} else if (prixActuel == 100) {
   nouveauPrix = prixActuel;
} else {
   nouveauPrix = prixActuel + 10;
}
```

   A noter que l'on pourrait aussi écrire :

```
nouveauPrix = prixActuel;
if (prixActuel < 100) {
   nouveauPrix -= 10;
} else if (prixActuel > 100) {
   nouveauPrix += 10;
}
```

3) Code maladroit.
La condition prixActuel >= 100 dans le else if est inutile car forcément vraie d'après ce qui précède.
Correctif :

```
if (prixActuel < 100) {
   nouveauPrix = prixActuel - 10;
} else if (prixActuel <= 150) {
   nouveauPrix = prixActuel - 15;
} else {
   nouveauPrix = prixActuel - 20;
}
```

4) Code maladroit.
Mieux de remplacer le else if par un else car permet ainsi de mieux mettre en évidence que la branche else est l'unique alternative.
Correctif :

```
if (prixActuel < 100) {
   nouveauPrix = prixActuel - 10;
} else {
   nouveauPrix = prixActuel - 20;
}
```

5) Code maladroit.
Correctif :

```
droitAuRabais = prixActuel >= 100;
```

6) Code faux.
Les branches else if ne seront jamais exécutées.

Correctif :

```
if (prixActuel > 1000) {
   pourcentRabais = 15;
} else if (prixActuel > 500) {
   pourcentRabais = 10;
} else if (prixActuel >= 100) {
   pourcentRabais = 5;
} else {
   pourcentRabais = 0;
}
```

Exercice 3.5 Qui vaut 0 ?

On suppose disposer de deux entiers *x* et *y*.

Ecrire (de plusieurs manières différentes) la condition permettant de tester :

a) que nos deux entiers valent 0
b) qu'au moins l'un de nos deux entiers vaut 0
c) qu'un seul de nos deux entiers vaut 0

Solution exercice 3.5

`&&` ou and, `||` ou or

a)
```
x == 0 && y == 0                    !x && !y
!(x != 0 || y != 0)                 !(x || y)
```

b)
```
x == 0 || y == 0                    !x || !y
!(x != 0 && y != 0)                 !(x && y)
x * y == 0                          !(x * y)
```

c)
```
(x == 0 && y != 0) || (x != 0 && y == 0)      (!x && y) || (x && !y)
!(x != 0 || y == 0) || !(x == 0 || y != 0)    !(x || !y) || !(!x || y)
!( (x != 0 || y == 0) && (x == 0 || y != 0) ) !( (x || !y) && (!x || y) )
x * y == 0 && x + y != 0                      !(x * y) && x + y
x != y && x * y == 0                          x != y && !(x * y)
```

Exercice 3.6 Evaluations d'expressions

Que va afficher le programme C++ suivant ?

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {

   int i, j, k;

   i = j = k = 1;
   i += j += k;
   cout << "A : i = " << i << " j = " << j << " k = " << k << endl;

   i = 3; j = 2;
   k = i++ > j || j++ != 3;
   cout << "B : i = " << i << " j = " << j << " k = " << k << endl;

   i = 3; j = 2;
   k = i++ < j || j++ != 3;
   cout << "C : i = " << i << " j = " << j << " k = " << k << endl;

   i = 3; j = 2;
   k = ++i == 3 && ++j == 3;
   cout << "D : i = " << i << " j = " << j << " k = " << k << endl;

   i = 3; j = 2;
   k = ++i > 3 && ++j == 3;
   cout << "E : i = " << i << " j = " << j << " k = " << k << endl;

   return EXIT_SUCCESS;
}
```

Solution exercice 3.6

Point important dans cet exercice : Se rappeler que les opérateurs && et || n'évaluent leur second opérande que lorsque c'est nécessaire.

```
A : i = 3 j = 2 k = 1
B : i = 4 j = 2 k = 1
C : i = 4 j = 3 k = 1
D : i = 4 j = 2 k = 0
E : i = 4 j = 3 k = 1
```

Exercice 3.7 Pourquoi faire simple quand … ?

Trouver une formulation équivalente, **la plus courte et simple possible**, aux deux extraits de code ci-dessous :
(i, j et k sont supposés de type int et contenir une valeur parfaitement définie; b est supposé du type bool)

1)
```
if (j == 0) {
   b = true;
} else {
   if (i / j < k) {
      b = false;
   } else {
      b = true;
   }
}
```

2)
```
if (i < 1) {
   b = true;
} else {
   b = i > 2;
}
```

Solution exercice 3.7

1) `b = !j || i / j >= k;`
2) `b = i < 1 || i > 2;`

Exercice 3.8 Nombre de jours dans un mois donné (1)

Ecrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir un no de mois (1 pour janvier, 2 pour février, etc), affiche combien le mois choisi compte de jour.

IMPORTANT

- On suppose la saisie utilisateur correcte
- Le programme ne doit utiliser ni *if* ni *switch* mais doit exploiter les opérateurs logiques
- Si l'utilisateur entre la valeur 2, le programme doit répondre :
  "Ce mois comporte 28 ou 29 jours"

Exemple d'exécution

```
Entrez un no de mois (1-12) : 5
Ce mois comporte 31 jours.
```

Solution exercice 3.8

Variante 1

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {

   unsigned noMois;

   cout << "Entrez un no de mois (1-12) : ";
   cin >> noMois;

   cout << "Ce mois comporte "
        << ( noMois == 2 ? "28 ou 29" :
             noMois == 4 ||
             noMois == 6 ||
             noMois == 9 ||
             noMois == 11 ? "30" : "31"
           )
        << " jours." << endl;

   return EXIT_SUCCESS;
}
```

Variante 2

```
   cout << "Ce mois comporte "
        << ( noMois == 2 ? "28 ou 29" :
             (noMois <= 6 && noMois % 2 == 0) ||
             (noMois >= 9 && noMois % 2 != 0) ? "30" : "31"
           )
        << " jours." << endl;
```

Exercice 3.9 Minimum de 3 entiers

Ecrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir (sur une même ligne) 3 entiers (de type *int*) détermine / affiche le minimum de ces 3 entiers.

IMPORTANT

- On suppose la saisie utilisateur correcte
- Le problème doit être résolu sans utiliser d'autres librairies que *cstdlib* et *iostream*

Solution exercice 3.9

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {

   int n1, n2, n3;

   cout << "Entrez 3 entiers : ";
   cin >> n1 >> n2 >> n3;

   { // Variante 1
      int min = n1;

      if (n2 < min) {
         min = n2;
      }

      if (n3 < min) {
         min = n3;
      }

      cout << "La plus petite valeur est " << min << endl;
   }

   { // Variante 2
      int min = n1 < n2 ? n1 : n2;
      min = n3 < min ? n3 : min;
      cout << "La plus petite valeur est " << min << endl;
   }

   { // Variante 3
      int min = n1 < n2 ? (n1 < n3 ? n1 : n3) : (n2 < n3 ? n2 : n3);
      cout << "La plus petite valeur est " << min << endl;
   }

   return EXIT_SUCCESS;
}
```

Exercice 3.10 Tri croissant de 3 entiers

Ecrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir (sur une même ligne) 3 entiers (de type *int*) affiche la valeur de ces derniers dans l'ordre croissant.

IMPORTANT

- On suppose la saisie utilisateur correcte
- Le problème doit être résolu sans utiliser la fonction prédéfinie *swap*

Solution exercice 3.10

Variante 1

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {

   int n1, n2, n3,
       tmp;

   cout << "Entrez 3 entiers : ";
   cin >> n1 >> n2 >> n3;

   if (n1 > n2) {
      tmp = n1;
      n1 = n2;
      n2 = tmp;
   }

   if (n1 > n3) {
      tmp = n1;
      n1 = n3;
      n3 = tmp;
   }

   if (n2 > n3) {
      tmp = n2;
      n2 = n3;
      n3 = tmp;
   }

   cout << "Les 3 entiers tries par ordre croissant : "
        << n1 << " " << n2 << " " << n3 << endl;

   return EXIT_SUCCESS;
}
```

Variante 2 (plus efficace que variante 1)

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {

   int n1, n2, n3,
       tmp;

   cout << "Entrez 3 entiers : ";
   cin >> n1 >> n2 >> n3;

   if (n1 > n2) {
      tmp = n1;
      n1 = n2;
      n2 = tmp;
   }

   if (n2 > n3) {
      tmp = n2;
      n2 = n3;
      n3 = tmp;
      if (n1 > n2) {
         tmp = n1;
         n1 = n2;
         n2 = tmp;
      }
   }

   cout << "Les 3 entiers tries par ordre croissant : "
        << n1 << " " << n2 << " " << n3 << endl;

   return EXIT_SUCCESS;
}
```

Exemple n1 = 2, n2 = 1, n3 = 3

- Variante 1 : 3 tests + 1 permutation
- Variante 2 : 2 tests + 1 permutation

Exercice 3.11 Multiple de 3 et/ou de 5

Ecrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir un entier $n \geq 0$, affiche, en fonction de la valeur de n, l'une des 4 réponses ci-dessous :

- <*n*> est un multiple de 3
- <*n*> est un multiple de 5
- <*n*> est un multiple de 3 et de 5
- <*n*> n'est ni un multiple de 3 ni un multiple de 5

IMPORTANT

- On suppose la saisie utilisateur correcte

Solution exercice 3.11

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {

   unsigned n;

   cout << "Entrez un entier >= 0 : ";
   cin >> n;

   cout << n;

   if (n % 3 == 0 && n % 5 == 0) { // ou if (n % 15 == 0) {
      cout << " est un multiple de 3 et de 5";
   } else if (n % 3 == 0) {
      cout << " est un multiple de 3";
   } else if (n % 5 == 0) {
      cout << " est un multiple de 5";
   } else {
      cout << " n'est ni un multiple de 3 ni un multiple de 5";
   }

   cout << endl;

   return EXIT_SUCCESS;
}
```

Exercice 3.12 Moyenne de notes et appréciation

Ecrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir (sur une même ligne) 4 notes (de type *double*) données au dixième de point affiche :

1°) la moyenne (au dixième de point) de ces 4 notes
2°) une appréciation dépendant de cette moyenne selon le barème suivant :

- **Insuffisant** si moyenne < 4.0
- **Moyen** si moyenne comprise dans l'intervalle [4.0; 4.5]
- **Bien** si moyenne comprise dans l'intervalle (4.5; 5]
- **Très bien** si moyenne comprise dans l'intervalle (5.0; 5.5]
- **Excellent** si moyenne > 5.5

Exemple d'exécution

```
Entrez 4 notes : 5.7 4.3 5.2 4.4
Moyenne = 4.9 - Bien
```

IMPORTANT

- On suppose travailler avec des notes comprises (bornes incluses) entre 1 et 6
- On suppose la saisie utilisateur correcte
- **Rappel**
  Pour afficher une valeur de type *double* avec 1 chiffre après la virgule, il faut :
  - faire un `#include <iomanip>` (nécessaire pour pouvoir utiliser le manipulateur *setprecision*)
  - écrire : `cout << fixed << setprecision(1) << valeur_de_type_double`
- Le problème doit être résolu sans utiliser d'autres librairies que *cstdlib, iomanip* et *iostream*

Solution exercice 3.12

```
#include <cstdlib>
#include <iomanip>
#include <iostream>
using namespace std;

int main() {

   const unsigned NB_NOTES = 4;

   double n1, n2, n3, n4;

   // Saisie des notes
   cout << "Entrez " << NB_NOTES << " notes : ";
   cin >> n1 >> n2 >> n3 >> n4;

   // Calcul de la moyenne
   double moyenne = (n1 + n2 + n3 + n4) / NB_NOTES;

   // Affichage de la moyenne
   cout << fixed << setprecision(1) // ou cout << setprecision(2)
        << "Moyenne = " << moyenne << " - ";

   // et de l'évaluation correspondante à cette moyenne
   if (moyenne > 5.5) {
      cout << "Excellent";
   } else if (moyenne > 5) {
      cout << "Tres bien";
   } else if (moyenne > 4.5) {
      cout << "Bien";
   } else if (moyenne >= 4) {
      cout << "Moyen";
   } else {
      cout << "Insuffisant";
   }

   // Autre formulation possible
//    if (moyenne < 4) {
//       cout << "Insuffisant";
//    } else if (moyenne <= 4.5) {
//       cout << "Moyen";
//    } else if (moyenne <= 5) {
//       cout << "Bien";
//    } else if (moyenne <= 5.5) {
//       cout << "Tres bien";
//    } else {
//       cout << "Excellent";
//    }

   cout << endl;

   return EXIT_SUCCESS;
}
```

Exercice 3.13 Instruction switch (1)

Indiquer si les affirmations suivantes sont justes ou fausses.

Proposer un correctif si vous jugez l'affirmation fausse.

1) Après le mot réservé **case** on peut donner une liste de valeurs séparées par des virgules :

   ……

2) Après le mot réservé **case** on peut donner un intervalle de valeurs :

   ……

3) La branche "**default**" peut venir n'importe où dans la liste des cas :

   ……

4) Les cas dans les différentes branches doivent être donnés dans un ordre croissant des valeurs :

   ……

5) Les valeurs de cas peuvent être données par des variables :

   ……

6) Si une branche comporte plusieurs instructions, il faut les mettre entre { } :

   ……

7) Toutes les valeurs possibles de l'expression doivent être prévues dans les différentes branches du **switch** :

   ……

8) Sans instruction particulière, après le traitement d'une branche, on passe à la branche suivante :

   ……

9) Deux branches différentes peuvent comporter la même valeur de cas :

……

10) Une instruction **switch** peut toujours remplacer une instruction **if** :

……

11) Une instruction **if** peut toujours remplacer une instruction **switch** :

……

Solution exercice 3.13

1) Après le mot réservé **case** on peut donner une liste de valeurs séparées par des virgules :

Faux. On ne peut donner qu'une seule valeur. Si l'on veut que plusieurs valeurs mènent à la même branche, chacune doit avoir son propre ***case.***

2) Après le mot réservé **case** on peut donner un intervalle de valeurs :

Faux selon la norme C++... mais certains compilateurs le permettent sous la forme a ... b

3) La branche "**default**" peut venir n'importe où dans la liste des cas :

Juste.

4) Les cas dans les différentes branches doivent être donnés dans un ordre croissant des valeurs :

Faux. L'ordre est sans importance.

5) Les valeurs de cas peuvent être données par des variables :

Faux. Il ne peut s'agir que de valeurs statiques.

6) Si une branche comporte plusieurs instructions, il faut les mettre entre { } :

En principe, pas nécessaire (bien que possible). Devient toutefois obligatoire si des déclarations figurent dans les instructions.

7) Toutes les valeurs possibles de l'expression doivent être prévues dans les différentes branches du **switch** :

   Faux. Si la valeur n'existe pas, le programme se poursuit en séquence.

8) Sans instruction particulière, après le traitement d'une branche, on passe à la branche suivante :

   Juste.

9) Deux branches différentes peuvent comporter la même valeur de cas :

   Faux. Jamais possible.

10) Une instruction **switch** peut toujours remplacer une instruction **if** :

   Faux… car dans switch (expression), expression ne peut être que de type entier (ou char) ou enum. Par conséquent switch (s) avec s de type string, par exemple, n'est pas possible.

11) Une instruction **if** peut toujours remplacer une instruction **switch** :

   Juste… mais pas toujours la façon de faire la plus appropriée.

Exercice 3.14 Instruction switch (2)

Soit le programme suivant :

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {

   int n;

   cout << "Donnez un entier : ";
   cin >> n;

   switch (n) {
      case 0:  cout << "A\n";
      case 1:
      case 2:  cout << "B\n";
               break;
      case 3:
      case 4:
      case 5:  cout << "C\n";
      default: cout << "D\n";
   }

   return EXIT_SUCCESS;
}
```

Que va afficher le programme ci-dessus lorsque l'utilisateur entre comme valeur :

1) 0
2) 1
3) 4
4) 6
5) -1

Solution exercice 3.14

1) A
   B
2) B
3) C
   D
4) D
5) D

Exercice 3.15 Nombre de jours dans un mois donné (2)

Même énoncé que l'exercice 3.8, mais à résoudre cette fois en utilisant l'instruction *switch* et en définissant une énumération fortement typée pour représenter les mois.

Solution exercice 3.15

```
#include <cstdlib>
#include <iostream>
using namespace std;

enum class Mois {JANVIER = 1, FEVRIER, MARS, AVRIL, MAI, JUIN,
                 JUILLET, AOUT, SEPTEMBRE, OCTOBRE, NOVEMBRE, DECEMBRE};

int main() {

   unsigned noMois;

   cout << "Entrez un no de mois (1-12) : ";
   cin >> noMois;

   cout << "Ce mois comporte ";

   switch ((Mois) noMois) {
      case Mois::FEVRIER:
         cout << "28 ou 29";
         break;
      case Mois::AVRIL:
      case Mois::JUIN:
      case Mois::SEPTEMBRE:
      case Mois::NOVEMBRE:
         cout << "30";
         break;
      default:
         cout << "31";
         break;
   }

   cout << " jours." << endl;

   return EXIT_SUCCESS;
}
```

Exercice 3.16 Tracing manuel de variables

Soit l'extrait de code suivant :

```
int n = 1789;
int somme = 0;
int chiffre;
while (n > 0) {
   chiffre = n % 10;
   somme = somme + chiffre;
   n = n / 10;
}
cout << somme << endl;
```

Reporter dans la colonne correspondante du tableau ci-dessous, chaque changement de valeur des diverses variables.

n	somme	chiffre	output

Solution exercice 3.16

n	somme	chiffre	output
~~1789~~	~~0~~	~~?~~	
~~178~~	~~9~~	~~9~~	
~~17~~	~~17~~	~~8~~	
~~1~~	~~24~~	~~7~~	
0	25	1	**25**

Exercice 3.17 Boucle while (1)

Que va afficher à l'exécution chacun des groupes d'instructions ci-dessous ?

1)
```
int i = 0;
while (i - 10) {
   i += 2; cout << i << " ";
}
```

2)
```
int i = 0;
while (i - 10)
   i += 2; cout << i << " ";
```

3)
```
int i = 0;
while (i < 11) {
   i += 2; cout << i << " ";
}
```

4)
```
int i = 11;
while (i--) {
   cout << i-- << " ";
}
```

5)
```
int i = 12;
while (i--) {
   cout << --i << " ";
}
```

6)
```
int i = 0;
while (i++ < 10) {
   cout << i-- << " ";
}
```

7)
```
int i = 1;
while (i <= 5) {
   cout << 2 * i++ << " ";
}
```

8)
```
int i = 1;
while (i != 9) {
   cout << (i = i + 2) << " ";
}
```

Solution exercice 3.17

1) `2 4 6 8 10`
2) `10` (à noter que cout ne fait pas partie de la boucle)
3) `2 4 6 8 10 12`
4) `Boucle infinie : 10 8 6 …`
5) `10 8 6 4 2 0`
6) `Affiche perpétuellement 1`
7) `2 4 6 8 10`
8) `3 5 7 9`

Exercice 3.18 Boucle while (2)

Que va afficher le programme suivant ?

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {

   int i, j;

   i = 0;
   while (i <= 5) i++;
   cout << "A : i = " << i << endl;

   i = j = 0;
   while (i <= 5) i += j++;
   cout << "B : i = " << i << " j = " << j << endl;

   i = j = 0;
   while (i <= 5) i += ++j;
   cout << "C : i = " << i << " j = " << j << endl;

   i = j = 0;
   while (j <= 5) i += j++;
   cout << "D : i = " << i << " j = " << j << endl;

   i = j = 0;
   while (j <= 5) i += ++j;
   cout << "E : i = " << i << " j = " << j << endl;

   i = j = 0;
   while (i <= 5) i += 2; j++;
   cout << "F : i = " << i << " j = " << j << endl;

   return EXIT_SUCCESS;
}
```

Solution exercice 3.18

```
A : i = 6
B : i = 6 j = 4
C : i = 6 j = 3
D : i = 15 j = 6
E : i = 21 j = 6
F : i = 6 j = 1
```

Exercice 3.19 Doublement d'un avoir bancaire

Ecrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir un montant et un taux d'intérêt annuel (en %), détermine / affiche combien d'années (entières) sont nécessaires pour doubler ledit montant.

Résoudre le problème ci-dessus :
1) En utilisant une boucle while
2) Sans utiliser de boucle

IMPORTANT
- Les saisies utilisateur sont supposées correctes
- Le taux d'intérêt annuel est supposé rester constant au cours du temps

Solution exercice 3.19

Variante avec boucle while

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {

   double montantInitial,
          tauxInteretAnnuel; // en %

   cout << "Entrez le montant initial > ";
   cin >> montantInitial;

   cout << "Entrez le taux d'interet annuel en % > ";
   cin >> tauxInteretAnnuel;

   double montantCourant = montantInitial;

   const double MONTANT_CIBLE = 2 * montantInitial;

   unsigned nbAnnees = 0;

   // NB  while et non pas do while car la condition peut être d'emblée fausse
   // (cas si montantInitial = 0)
   while (montantCourant < MONTANT_CIBLE) {
      nbAnnees++;
      montantCourant = montantCourant * (1 + tauxInteretAnnuel / 100);
   }

   cout << "Le montant aura double apres "
        << nbAnnees << " an" << (nbAnnees > 1 ? "s" : "") << "." << endl;

   return EXIT_SUCCESS;
}
```

Variante sans boucle (à relever que la notion de montant n'apparaît plus !)

```
#include <cmath>
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {

   double tauxInteretAnnuel; // en %

   cout << "Entrez le taux d'interet annuel en % > ";
   cin >> tauxInteretAnnuel;

   const unsigned NB_ANNEES =
      (unsigned) ceil(log10(2) / log10(1 + tauxInteretAnnuel / 100));

   cout << "Le montant aura double apres "
        << NB_ANNEES << " an" << (NB_ANNEES > 1 ? "s" : "") << "." << endl;

   return EXIT_SUCCESS;
}
```

Exercice 3.20 Affichage de nombres par lots

Ecrire un programme C++ qui affiche les nombres de 20 à 1 par lots de trois comme suit :

```
20 19 18
17 16 15
14 13 12
11 10  9
 8  7  6
 5  4  3
 2  1
```

Résoudre le problème de deux manières différentes :
1) En utilisant une boucle *while*
2) En utilisant une boucle *for*

Indication
- Utiliser le manipulateur *setw* défini dans la librairie *iomanip* pour réaliser l'affichage demandé

Solution exercice 3.20

```
#include <cmath>
#include <cstdlib>
#include <iomanip>
#include <iostream>
using namespace std;

int main() {

   const unsigned NB_COLONNES = 3;
   const unsigned N_DEPART = 20; // Doit être > 0
   const unsigned N_FIN = 1; // Doit être > 0 et <= N_DEPART
   const unsigned W = (unsigned) log10(N_DEPART) + 1; // Nb de chiffres
                                                      // dans N_DEPART

   unsigned n = N_DEPART, noColonne = 1;
   while (n >= N_FIN) {
      cout << setw(W) << n; // affiche le nombre
      if (noColonne == NB_COLONNES) {
         cout << endl; // change de ligne
         noColonne = 0;
      } else {
         if (n != N_FIN) {
            cout << " ";
         }
      }
      noColonne++;
      n--;
   }
   cout << endl << endl;

   for (unsigned n = N_DEPART, noColonne = 1; n >= N_FIN; ++noColonne, --n) {
      cout << setw(W) << n; // affiche le nombre
      if (noColonne == NB_COLONNES) {
         cout << endl; // change de ligne
         noColonne = 0; // ++noColonne dans le for ajoutera 1
      } else {
         if (n != N_FIN) {
            cout << " ";
         }
      }
   }
   cout << endl;

   return EXIT_SUCCESS;
}
```

Exercice 3.21 Boucles for (1)

Que va afficher à l'exécution chacun des groupes d'instructions ci-dessous ?

1) `for (int i = 1; i < 10; ++i) {cout << i << " ";}`

2) `for (int i = 1; i < 10; i += 2) {cout << i << " ";}`

3) `for (int i = 10; i > 1; --i) {cout << i << " ";}`

4) `for (int i = 0; i < 10; ++i) {cout << i << " ";}`

5) `for (int i = 1; i < 10; i = i * 2) {cout << i << " ";}`

6) `for (int i = 1; i < 10; ++i) {if (i % 2 == 0) {cout << i << " ";}}`

Solution exercice 3.21

1) `1 2 3 4 5 6 7 8 9`
2) `1 3 5 7 9`
3) `10 9 8 7 6 5 4 3 2`
4) `0 1 2 3 4 5 6 7 8 9`
5) `1 2 4 8`
6) `2 4 6 8`

Exercice 3.22 Boucles for (2)

Pour chacun des cas ci-dessous, indiquer combien de fois la boucle est exécutée :
(**NB** Dans chacun des cas, la variable de boucle *i* est supposée être non modifiée dans le corps de la boucle)

1) `for (int i = 1; i <= 10; ++i)…`

2) `for (int i = 0; i < 10; ++i)…`

3) `for (int i = 10; i > 0; --i)…`

4) `for (int i = -10; i <= 10; ++i)…`

5) `for (int i = 10; i >= 0; ++i)…`

6) `for (int i = -10; i <= 10; i = i + 2)…`

7) `for (int i = -10; i <= 10; i = i + 3)…`

Solution exercice 3.22

1) `10`
2) `10`
3) `10`
4) `21`
5) `numeric_limits<int>::max() - 10 + 1 (= 2147483638)`
6) `11`
7) `7`

Exercice 3.23 Boucles for (3)

Que va afficher à l'exécution chacun des groupes d'instructions ci-dessous ?

1)
```
int s = 1;
for (int n = 1; n <= 5; ++n) {
   s = s + n;
   cout << s << " ";
}
```

2)
```
int s = 1;
for (int n = 1; n <= 10; cout << s << " ") {
   n = n + 2;
   s = s + n;
}
```

3)
```
int s = 1;
int n;
for (n = 1; n <= 5; ++n) {
   s = s + n;
   n++;
}
cout << s << " " << n;
```

Solution exercice 3.23

1) `2 4 7 11 16`
2) `4 9 16 25 36`
3) `10 7`

Exercice 3.24 Balle qui rebondit

Lorsqu'une balle tombe d'une hauteur initiale $h0$, sa vitesse d'impact au sol est $v0 = \sqrt{2gh0}$ (où g = 9.81 m/s^2 = constante de gravité terrestre). Immédiatement après le rebond, sa vitesse est $v1 = \varepsilon \cdot v0$ (où ε est le coefficient de rebond de la balle). Elle remonte alors à la hauteur $h = v1^2/2g$.

Ecrire un programme C++ qui calcule et affiche la hauteur à laquelle la balle remonte après un nombre donné de rebonds.

Les données suivantes seront entrées par l'utilisateur et le programme devra vérifier la validité partielle de ces données (partielle… car on supposera que l'utilisateur entre bien une valeur numérique) :

- ε, le coefficient de rebond, tel que $0 \le \varepsilon < 1$;
- $h0$, la hauteur initiale, telle que $h0 \ge 0$;
- n, le nombre de rebonds, tel que $n \ge 0$.

IMPORTANT

- Pour la vérification partielle des saisies utilisateur, utiliser des boucles *do…while*.
- Cherchez à résoudre le problème, non pas du point de vue formel (équations), mais par simulation du système physique (balle).
- Afficher le résultat avec 2 chiffres après la virgule.

Solution exercice 3.24

```
#include <cstdlib>
#include <iomanip>
#include <iostream>
#include <cmath>
using namespace std;

const double g = 9.81; // constante de gravité terrestre

int main() {

   int n;          // nombre de rebonds à simuler
   double eps,     // coefficient de rebond de la balle
          h0, h1, // hauteur avant et après le rebond [m]
          v0, v1; // vitesse avant et après le rebond [m/s]

   // Saisie des valeurs avec test de validité partiel
   do {
      cout << "Coefficient de rebond (0 <= coeff < 1) : ";
      cin >> eps;
   } while (eps < 0.0 || eps >= 1.0);

   do {
      cout << "Hauteur initiale [m]  (h0 >= 0)        : ";
      cin >> h0;
   } while (h0 < 0.0);

   do {
      cout << "Nombre de rebonds     (n >= 0)         : ";
      cin >> n;
   } while (n < 0);

   // Calculs proprement dits
   // Chaque itération correspond à un rebond
   for (int rebonds = 1; rebonds <= n; ++rebonds) {
      v0 = sqrt(2 * g * h0);     // vitesse avant rebond
      v1 = eps * v0;             // vitesse après rebond
      h1 = (v1 * v1) / (2 * g); // hauteur après rebond
      h0 = h1;                   // ... qui devient la nlle hauteur
                                 // avant rebond suivant
   }

   // Affichage du résultat
   cout << fixed << setprecision(2)
        << "La hauteur atteinte apres " << n
        << " rebond" << (n > 1 ? "s = " : " = ") << h0 << " [m]" << endl;

   return EXIT_SUCCESS;
}

// Coefficient de rebond (0 <= coeff < 1) : 0.9
// Hauteur initiale [m]  (h0 >= 0)        : 100
// Nombre de rebonds     (n >= 0)         : 10
// La hauteur atteinte apres 10 rebonds = 12.16 [m]
```

Exercice 3.25 Boucles for imbriquées

Que va afficher à l'exécution chacun des groupes d'instructions ci-dessous ?

1)
```
for (int i = 1; i <= 3; ++i) {
   for (int j = 1; j <= 4; ++j) {
      cout << "*";
   }
   cout << endl;
}
```

2)
```
for (int i = 0; i < 4; ++i) {
   for (int j = 0; j < 3; ++j) {
      cout << "*";
   }
   cout << endl;
}
```

3)
```
for (int i = 1; i <= 4; ++i) {
   for (int j = 1; j <= i; ++j) {
      cout << "*";
   }
   cout << endl;
}
```

4)
```
for (int i = 1; i <= 3; ++i) {
   for (int j = 1; j <= 5; ++j) {
      if (j % 2 == 0) {
         cout << "*";
      } else {
         cout << "_";
      }
   }
   cout << endl;
}
```

5)
```
for (int i = 1; i <= 3; ++i) {
   for (int j = 1; j <= 5; ++j) {
      if ( (i + j) % 2 == 0) {
         cout << "*";
      } else {
         cout << " ";
      }
   }
   cout << endl;
}
```

Solution exercice 3.25

1)
```
****
****
****
```

2)
```
***
***
***
***
```

3)
```
*
**
***
****
```

4)
```
_*_*_
 *_*_
_*_*_
 *_*_
```

5)
```
* * *
 * *
* * *
```

Exercice 3.26 Triangle d'étoiles

Ecrire un programme C++ qui affiche un triangle comme dans l'exemple ci-dessous.

La hauteur du triangle (càd le nombre de lignes) est fixée par l'utilisateur.

Faire en sorte :

- que l'utilisateur soit invité à refaire sa saisie s'il entre une hauteur négative
- que la dernière ligne du triangle s'affiche sur le bord gauche de l'écran

Exemple d'exécution

```
Hauteur du triangle (h >= 0) : 10

          *
         ***
        *****
       *******
      *********
     ***********
    *************
   ***************
  *****************
*******************
```

Solution exercice 3.26

Variante avec saisie utilisateur partiellement contrôlée

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {

   const char MOTIF = '*';

   int hauteur; // hauteur du triangle

   // Saisie de la hauteur du triangle, avec test de validité partiel
   do {
      cout << "Hauteur du triangle (h >= 0) : ";
      cin >> hauteur;
   } while (hauteur < 0);

   // Dessin du triangle
   cout << endl;
   for (int noLigne = 1; noLigne <= hauteur; ++noLigne) {
      // Affichage des espaces blancs
      for (int i = 1; i <= hauteur - noLigne; ++i)
         cout << ' ';
      // ... puis du motif, autant de fois que nécessaire
      for (int i = 1; i <= 2 * noLigne - 1; ++i)
         cout << MOTIF;
      cout << endl;
   }
   cout << endl;

   return EXIT_SUCCESS;
}
```

Variante avec saisie utilisateur entièrement contrôlée et setw

```
#include <cstdlib>
#include <iomanip>
#include <iostream>
#include <limits>
using namespace std;

int main() {

   const char MOTIF = '*';

   int hauteur; // hauteur du triangle

   // Saisie totalement contrôlée de la hauteur
   bool saisie_ok;
   do {
      cout << "Hauteur du triangle (h >= 0) : ";
      if (!(saisie_ok = cin >> hauteur && hauteur >= 0)) {
         cin.clear(); // reset des bits d'erreur
         cout << "Saisie incorrecte.\n";
      }
      cin.ignore(numeric_limits<streamsize>::max(), '\n'); // vider le buffer
   } while (!saisie_ok);

   // Dessin du triangle
   cout << endl;
   for (int noLigne = 1; noLigne <= hauteur; ++noLigne) {
      // Affichage des espaces blancs
      cout << setw(hauteur - noLigne + 1);
      // ... puis du motif, autant de fois que nécessaire
      for (int i = 1; i <= 2 * noLigne - 1; ++i)
         cout << MOTIF;
      cout << endl;
   }
   cout << endl;

   return EXIT_SUCCESS;
}
```

Exercice 3.27 Le code est-il simplifiable ?

Soit le code suivant :

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {

   const int N = 5;

   int compteur = 0;

   for (int i = 0; i < N; ++i) {
      for (int j = i; j < N; ++j) {
         compteur++;
      }
   }
   cout << "compteur = " << compteur << endl;


   return EXIT_SUCCESS;
}
```

1) Que va afficher le code ci-dessus ?
2) Le code ci-dessus est-il simplifiable ? Si oui, proposer un (des) correctif(s).

Solution exercice 3.27

1) compteur = 15

2) En notant que le code ci-dessus revient à faire : 1 + 2 + … + N, on peut le simplifier en remplaçant les deux boucles for par une seule :

```
for (int i = 1; i <= N; ++i) {
   compteur = compteur + i;
}
```

… ou mieux encore (en exploitant les mathématiques !), en écrivant directement :

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {
   const int N = 5;
   cout << "compteur = " << N * (N + 1) / 2 << endl;
   return EXIT_SUCCESS;
}
```

Exercice 3.28 Série harmonique

Ecrire un programme C++ permettant de calculer la somme des n premiers termes de la série harmonique, c'est-à-dire :

$$s = \sum_{k=1}^{n} \frac{1}{k}$$

La valeur de n sera fixée par l'utilisateur.

Faire en sorte que l'utilisateur soit invité à refaire sa saisie s'il entre une valeur négative ou nulle (on suppose, par contre, que l'utilisateur a bien saisi un nombre et pas autre chose).

Exemple d'exécution

```
Combien de termes voulez-vous ? 0
Erreur. La valeur saisie doit etre > 0
Combien de termes voulez-vous ? 3
La somme des 3 premiers termes de la serie vaut 1.83333
```

Solution exercice 3.28

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {

   int n;              // nombre de termes de la série
   double somme = 0.0; // somme des termes de la série

   do {
      cout << "Combien de termes voulez-vous ? ";
      cin >> n;
   } while (n < 1 && cout << "Erreur. La valeur saisie doit etre > 0\n");

   for (int i = 1; i <= n; ++i) {
      somme = somme + 1.0 / i;
   }

   if (n == 1) {
      cout << "Le premier terme de la serie vaut " << somme << endl;
   } else {
      cout << "La somme des " << n << " premiers termes de la serie vaut "
           << somme << endl;
   }

   return EXIT_SUCCESS;
}
```

Exercice 3.29 ppmc

Ecrire un programme C++ permettant de calculer le ppmc (plus petit multiple commun) de deux nombres entiers (> 0) *m* et *n* saisis (sur la même ligne) par l'utilisateur.

Faire en sorte que la saisie utilisateur soit vérifiée. On suppose toutefois que l'utilisateur entre des valeurs du bon type et dans l'intervalle de définition du type.

Solution exercice 3.29

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {

   int n, m,      // les 2 nombres saisis par l'utilisateur
       min, max, // le plus petit, resp le plus grand, des 2 nombres
                 // saisis par l'utilisateur
       ppmc;     // ppmc(n,m)

   // Saisie des 2 entiers, avec test de validité partiel
   do {
      cout << "Donnez 2 nombres entiers positifs : ";
      cin >> n >> m;
   } while (n < 1 || m < 1);

   // Recherche du ppmc de n et m
   if (n < m) {
      min = n;
      max = m;
   } else {
      min = m;
      max = n;
   }

   ppmc = max;
   while (ppmc % min)
      ppmc += max;

   // Affichage du résultat
   cout << "ppmc(" << n << "," << m << ") = " << ppmc << endl;

   return EXIT_SUCCESS;
}

// Donnez 2 nombres entiers positifs : 6 8
// ppmc(6,8) = 24
```

Remarques

- *n* et *m* sont déclarés de type *int* et non *unsigned int*… sinon problème (cyclicité) lorsque l'utilisateur entre une valeur < 0.
- Plutôt que d'utiliser *if*... *else* dans le code ci-dessus, on aurait pu écrire :
  - `n < m ? (min = n, max = m) : (min = m, max = n);`

    ou (en faisant un #include <cmath>)
  - ```
    const int MIN = (int) fmin(n, m),
              MAX = (int) fmax(n, m);
    ```

    ou (en faisant un #include <algorithm>)
  - ```
    const int MIN = min(n, m),
              MAX = max(n, m);
    ```

Exercice 3.30 Simulations de boucles

1) Soit la boucle `for` suivante :

```
for (; i < 10; ++i) {cout << i << endl;}
```

Récrire la boucle `for` ci-dessus, en utilisant :

a) une boucle `while`

b) une boucle `do… while`

2) Soit la boucle `while` suivante :

```
while (i-- > 10) {cout << i << endl;}
```

Récrire la boucle `while` ci-dessus, en utilisant :

a) une boucle `for`

b) une boucle `do… while`

Solution exercice 3.30

1)

a)
```
while (i < 10) {cout << i++ << endl;}
```

b)
```
if (i < 10) {
   do {
      cout << i++ << endl;
   } while (i < 10);
}
```

2)

a)
```
for (; i-- > 10; cout << i << endl);
```

b)
```
if (i > 10) {
   do {
      cout << --i << endl;
   } while (i > 10);
}
--i; // nécessaire pour qu'au sortir de la boucle, i ait la même
     // valeur que dans le cas de la boucle while et de la
     // boucle for
```

Exercice 3.31 continue et break dans boucle do… while

Que va afficher le programme suivant ?

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {

   int n = 0;

   do {
      if (n % 2 == 0) {cout << n << " est pair\n";
                       n += 3;
                       continue;
                      }
      if (n % 3 == 0) {cout << n << " est multiple de 3\n";
                       n += 5;
                      }
      if (n % 5 == 0) {cout << n << " est multiple de 5\n";
                       break;
                      }
      n++;
   } while (true);

   return EXIT_SUCCESS;
}
```

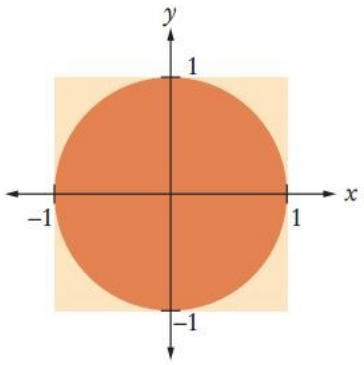
Solution exercice 3.31

```
0 est pair
3 est multiple de 3
9 est multiple de 3
15 est multiple de 3
20 est multiple de 5
```

Exercice 3.32 Approximation de π par la méthode Monte-Carlo

Wikipédia *Le terme **méthode de Monte-Carlo**, ou **méthode Monte-Carlo**, désigne une famille de méthodes algorithmiques visant à calculer une valeur numérique approchée en utilisant des procédés aléatoires, c'est-à-dire des techniques probabilistes. Le nom de ces méthodes, qui fait allusion aux jeux de hasard pratiqués à Monte-Carlo, a été inventé en 1947 par Nichols Metropolis, et publié pour la première fois en 1949 dans un article coécrit avec Stanislaw Ulam.*

Le but de cet exercice est de réussir à établir une bonne approximation de π en réalisant une simulation basée sur la méthode Monte-Carlo.
Pour ce faire, on considère une cible carrée dans laquelle est inscrit un cercle de rayon R = 1 :



L'idée consiste à simuler le tir aléatoire (au hasard) de *n* fléchettes sur une telle cible. On admet que toute fléchette tirée atteint la cible. Pour simuler aléatoirement le point d'impact d'une fléchette, c'est simple : il suffit de générer aléatoirement une coordonnée x_{impact} et une coordonnée y_{impact}, comprises toutes deux dans l'intervalle [-1.0; 1.0].
Désignons par *m* le nombre de fléchettes tirées dont le point d'impact figure dans ou sur le bord du cercle.

Etant donné le caractère totalement aléatoire de nos tirs, on s'attend à ce que, pour un nombre de tirs suffisamment grand :

$$\frac{surface\ cercle}{surface\ carrée} \approx \frac{m}{n}$$

et ainsi de pouvoir en déduire une bonne approximation de π.

A vous de jouer maintenant !

Indication / prescription

- Pour réaliser des tirages aléatoires, utiliser ici la fonction *rand()* de *cstdlib*.
- Afficher le résultat avec 2 chiffres après la virgule

Solution exercice 3.32

```
#include <cstdlib>
#include <ctime>
#include <iomanip>
#include <iostream>
using namespace std;

int main() {

   const unsigned NB_TIRS = 100000;

   double r,             // Nombre aléatoire dans [0; 1]
          x, y,          // Coordonnées aléatoires du point d'impact
          estimation_pi; // Rappel : pi = 3.14159...

   unsigned nb_succes = 0; // Nb de fléchettes tirées et telles que leur point
                           // d'impact se trouve dans ou sur le bord du cercle

   // Initialisation du générateur aléatoire
   srand((unsigned)time(NULL));

   for (unsigned i = 1; i <= NB_TIRS; ++i) {
      r = rand() * 1.0 / RAND_MAX; // entre 0 et 1
      x = -1 + 2 * r;              // entre -1 et 1
      r = rand() * 1.0 / RAND_MAX;
      y = -1 + 2 * r;
      if (x * x + y * y <= 1) { // tir dans le cercle
         nb_succes++;
      }
   }

   estimation_pi = 4.0 * nb_succes / NB_TIRS; // 4.0 = surface carré

   cout << fixed << setprecision(2)
        << "Estimation de pi : " << estimation_pi << endl;

   return EXIT_SUCCESS;
}
```

Autre variante possible (plus C++, meilleure distribution… mais hors de portée des étudiants)

```
#include <chrono>
#include <cstdlib>
#include <functional>
#include <iomanip>
#include <iostream>
#include <random>
using namespace std;

int main() {

   const unsigned NB_TIRS = 100000;

   double x, y,          // Coordonnées aléatoires du point d'impact
          estimation_pi; // Rappel : pi = 3.14159...

   unsigned nb_succes = 0; // Nb de fléchettes tirées et telles que leur point
                           // d'impact se trouve dans ou sur le bord du cercle

   // Obtenir une racine à partir de l'heure du système
   auto racine = chrono::system_clock::now().time_since_epoch().count();

   uniform_real_distribution<double> distribution(-1.0,1.0);
   auto tirage_aleatoire = bind(distribution, default_random_engine(racine));

   for (unsigned i = 1; i <= NB_TIRS; ++i) {
      x = tirage_aleatoire();
      y = tirage_aleatoire();
      if (x * x + y * y <= 1) { // tir dans le cercle
         nb_succes++;
      }
   }

   estimation_pi = 4.0 * nb_succes / NB_TIRS;

   cout << fixed << setprecision(2)
        << "Estimation de pi : " << estimation_pi << endl;

   return EXIT_SUCCESS;
}
```

Exercice 3.33 Le problème des 3 portes

Wikipédia ***Le problème des 3 portes ou problème de Monty Hall*** *est un casse-tête probabiliste librement inspiré du jeu télévisé américain Let's Make a Deal. Il est simple dans son énoncé mais non intuitif dans sa résolution et c'est pourquoi on parle parfois à son sujet de* ***paradoxe de Monty Hall****. Il porte le nom de celui qui a présenté ce jeu aux États-Unis pendant treize ans, Monty Hall.*

Enoncé du problème

Le jeu oppose un présentateur à un candidat (le joueur). Le joueur est placé devant trois portes fermées. Derrière l'une d'elles se trouve une voiture (ou tout autre prix magnifique) et derrière chacune des deux autres se trouve une chèvre (ou tout autre prix sans importance). Le joueur doit tout d'abord désigner une porte. Puis le présentateur doit ouvrir une porte qui n'est ni celle choisie par le candidat, ni celle cachant la voiture (le présentateur sait quelle est la bonne porte dès le début). Le candidat a alors le droit ou bien d'ouvrir la porte qu'il a choisie initialement, ou bien d'ouvrir la troisième porte.

Les questions qui se posent au candidat sont :

1. *Que doit-il faire ?*
2. *Quelles sont ses chances de gagner la voiture en agissant au mieux ?*

Tentez de répondre aux deux questions ci-dessus, en imaginant une simulation Monte-Carlo du jeu.

Solution exercice 3.33

```
#include <cstdlib>
#include <iomanip>
#include <iostream>
#include <ctime>
using namespace std;

int main() {

   const unsigned
      NB_PORTES = 3,
      NB_EXPERIENCES = 1'000'000; // (C++14)

   unsigned
      porte_voiture, // no de la porte derrière laquelle se cache la voiture
      porte_joueur;  // no de la porte choisie initialement par le joueur

   srand((unsigned)time(NULL));

   // On part du principe que le joueur maintient toujours son choix initial
   // et l'on comptabilise combien de fois il gagne
   unsigned compteur = 0;

   for (unsigned i = 1; i <= NB_EXPERIENCES; ++i) {
      porte_voiture = (unsigned) rand() % NB_PORTES + 1;
      porte_joueur  = (unsigned) rand() % NB_PORTES + 1;
      compteur += porte_voiture == porte_joueur;
   }

   cout << fixed << setprecision(3)
        << "Probabilite de succes si maintien du choix initial   : "
        << (double) compteur / NB_EXPERIENCES << endl
        << "Probabilite de succes si changement du choix initial : "
        << (double) (NB_EXPERIENCES - compteur) / NB_EXPERIENCES << endl;

   return EXIT_SUCCESS;
}

// Probabilite de succes si maintien du choix initial   : 0.333
// Probabilite de succes si changement du choix initial : 0.667 #include <cstdlib>
```

Autre variante possible (plus C++, meilleure distribution… mais hors de portée des étudiants)

```
#include <chrono>
#include <cstdlib>
#include <functional>
#include <iomanip>
#include <iostream>
#include <random>
using namespace std;

int main() {

   const unsigned
      NB_PORTES = 3,
      NB_EXPERIENCES = 1'000'000; // (C++14)

   unsigned
      porte_voiture, // no de la porte derrière laquelle se cache la voiture
      porte_joueur;  // no de la porte choisie initialement par le joueur

   // Obtenir une racine à partir de l'heure du système
   auto racine = chrono::system_clock::now().time_since_epoch().count();

   uniform_int_distribution<unsigned> distribution(1, NB_PORTES);
   auto tirage_aleatoire = bind(distribution, default_random_engine(racine));

   // On part du principe que le joueur maintient toujours son choix initial
   // et l'on comptabilise combien de fois il gagne
   unsigned compteur = 0;

   for (unsigned i = 1; i <= NB_EXPERIENCES; ++i) {
      porte_voiture = tirage_aleatoire();
      porte_joueur = tirage_aleatoire();
      compteur += porte_voiture == porte_joueur;
   }

   cout << fixed << setprecision(3)
        << "Probabilite de succes si maintien du choix initial   : "
        << (double) compteur / NB_EXPERIENCES << endl
        << "Probabilite de succes si changement du choix initial : "
        << (double) (NB_EXPERIENCES - compteur) / NB_EXPERIENCES << endl;

   return EXIT_SUCCESS;
}

// Probabilite de succes si maintien du choix initial   : 0.333
// Probabilite de succes si changement du choix initial : 0.667
```