Correction des exercices

Document rédigé par Raphaël et André

Table des matières

1 Syntaxe, variables et I/O

1.1 Explications 1

Pour chaque programme, lister les variables, quel type de données elles peuvent contenir (ex : dire "nombre entier" pour "int") et expliquer brievement ce qu'il fait.

Correction :

a. Variable : age (nombre entier)
Fonctionnement : le programme demande l'âge à l'utilisateur et affiche l'âge rentré.

b. Variable : nombre (nombre réel / flottant)
Fonctionnement : le programme demande un nombre réel à l'utilisateur en entrée et affiche son carré.

c. Variable : diviseur, dividande (nombres réels / flottants)
Fonctionnement : le programme demande deux nombres réels à l'utilisateur et affiche le quotient du premier par le deuxième.

1.2 Erreurs 1

Trouver la ou les erreur(s) dans chaque programme. Donner une correction possible.

Correction :

a. L'opérateur "»" (ligne 6) n'est pas celui qui doit être placé après *cout*. Il faut le remplacer par "«".

b. Il manque un ";" à la ligne 13.

c. La variable *a* ne peut pas être modifié (ligne 8) car il s'agit d'une constante. Également, il manque un "}" à la fin du programme.

1.3 Complétions 1

Compléter la ou les lignes manquantes dans chaque programme [1].

Correction :

a.
```
#include <iostream>
using namespace std;

// On cherche à calculer et afficher le périmètre d'un cercle
int main()
{
    const float pi = 3.14;
    float rayon;  // Il faut déclarer la variable rayon

    cout << "Donner le rayon du cercle : \n";
    cin >> rayon;

    cout << "Le périmètre du cercle est " << 2 * pi * rayon << ".";

    return 0;
}
```

1. Chercher au besoin les formules sur Internet.

b.

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

// On cherche à calculer et afficher l'aire d'un triangle
int main()
{
    float base, hauteur;

    cout << "Donner la base et la hauteur du triangle : \n";
    cin >> base >> hauteur; // Il faut permettre àl'utilisateur de rentrer les valeurs

    cout << "L'aire du triangle est : ";
    cout << base * hauteur / 2; // Chercher au besoin la formule sur Internet

    return 0;
}
```

1.4 Alice et Bob 1

Bob a écrit le programme suivant :

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

int main()
{
    int a, b;

    cout << "Donner deux nombres : \n";
    cin >> a >> b;

    cout << "Le quotient de " << a << " par " << b << " est " << a / b;

    return 0;
}
```

Lorsqu'Alice teste le programme de Bob avec les valeurs $a = 1$ et $b = 3$ elle n'obtient pas la valeur attendue. Trouver le problème.

Correction :

Alice ne trouve pas la valeur attendue car les variables utilisées sont de type *int*. En les changeant pour le type *float*, la bonne valeur est donnée.

1.5 Triple d'un nombre

Écrire un programme qui renvoie le triple d'un nombre.

Correction :

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

int main()
```

```
{
    float nombre;

    cout << "Quel est le nombre ?\n >>> ";
    cin >> nombre;
    cout << "Le triple du nombre est " << nombre * 3;

    return 0;
}
```

1.6 Aire d'un cercle

Écrire un programme qui calcule l'aire d'un cercle.

Correction :

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

int main()
{
    float rayon;
    const float pi = 3.1415;

    cout << "Quel est le rayon ?\n >>> ";
    cin >> rayon;
    cout << "L'aire du cercle est " << rayon * rayon * pi;

    return 0;
}
```

1.7 Une expression... compliquée !

Voici une expression un peu compliquée :

$$\frac{(a+b+c)(a+c)(a+b)(b+c)}{2}$$

Écrire un programme qui calcule cette expression pour n'importe quel a, b et c. Attention aux priorités opératoires !

Correction :

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

int main()
{
    float a, b, c;

    cout << "Quels sont les nombres ?\n >>> ";
    cin >> a >> b >> c;
    cout << "L'expression compliquée est " << (a + b + c) * (a + c) * (a + b) * (b + c) / 2;
```

```
    return 0;
}
```

1.8 Racines carrées

Il est possible de calculer la racine carrée d'une expression en C++. Pour ce faire, il faut en premier temps rajouter la ligne suivante au début du programme :

```
#include <cmath>
```

Supposons que l'on veuille calculer $\sqrt{2}$. La ligne suivante permet de stocker cette valeur dans une variable :

```
float racinecarree = sqrt(2);
```

Écrire un programme calculant la racine carrée de l'expression de l'exercice 1.7.

Correction :

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

int main()
{
    float a, b, c;

    cout << "Quels sont les nombres ?\n >>> ";
    cin >> a >> b >> c;
    cout << "La racine carrée de l'expression compliquée est " << sqrt((a + b + c) * (a + c)
        * (a + b) * (b + c) / 2);

    return 0;
}
```

1.9 Puissances

Cet exercice nécessite le programme du 1.8. Il est possible de calculer la puissance x-ième [2] d'un nombre en C++. Si l'on veut calculer la puissance 0.3-ième de 2 et garder le résultat dans une variable, on peut écrire :

```
float puissance = pow(2, 0.3);
```

Calculer la puissance 0.5-ième de l'expression de l'exercice 1.7. Que remarquez-vous ?

Correction :

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

2. Je veux dire "nombre puissance x"

```
int main()
{
    float a, b, c;

    cout << "Quels sont les nombres ?\n >>> ";
    cin >> a >> b >> c;
    cout << "La puissance 0.5-ième de l'expression compliquée est " << pow((a + b + c) * (a +
        c) * (a + b) * (b + c) / 2, 0.5);

    return 0;
}
```

Remarque : la valeur est la même que celle de l'exercice 1.8.

1.10 Un peu de binaire

Un bit [3] est une variable ne pouvant prendre que les valeurs 0 ou 1. Il est possible d'encoder les nombres entiers positifs avec des bits en utilisant les puissances de 2. Pour n bits, les exposants vont de $n-1$ à 0 de gauche à droite. Ainsi, le n-ième bit est relié à la valeur 2^{n-1}, le n-1-ième est relié à la valeur 2^{n-2}... et le premier bit (tout à droite) est relié à la valeur 2^0. Un bit valant 0 indique que cette valeur n'est pas prise en compte.
Pour décoder le groupe 1011 on fait :

Bit	1	0	1	1	Total
Puissance de 2	2^3	2^2	2^1	2^0	15
Valeur effective	8	0	2	1	11

Écrire un programme permettant de décoder n'importe quel groupe de 4 bits.

Correction :

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

int main()
{
    int bit1, bit2, bit3, bit4;

    cout << "Quels sont les bits ?\n >>> ";
    cin >> bit1 >> bit2 >> bit3 >> bit4;
    cout << "En décodant le groupe de 4 bits, on obtient la valeur " << 8 * bit1 + 4 * bit2 +
        2 * bit3 + 1 * bit4;

    return 0;
}
```

3. De l'anglais *binary digit*

2 Structures conditionnelles et boucles

2.1 Explications 2

Pour chaque programme lister tous les blocs, décrire les conditions d'arrêt des boucles ainsi que les conditions à satisfairent pour rentrer dans les structures conditionnelles. Enfin, expliquer brievement ce que fait le programme.

a.
```cpp
#include <iostream>
using namespace std;

int main()
{
    int age;

    cout << "Donnez votre âge :\n"
    cin >> age;

    if (age >= 18)
    {
        cout << "Vous êtes majeur";
    }

    else if (age >= 0)
    {
        cout << "Vous êtes mineur";
    }

    else
    {
        cout << "Vous êtes un menteur";
    }

    return 0;
}
```

b.
```cpp
#include <iostream>
using namespace std;

int main()
{
    int age = -1;

    while (age < 0)
    {
        cout << "Donnez votre âge :\n"
        cin >> age;
    }

    if (age % 3 == 0 && (age - 2) % 5 == 1)
    {
        cout << "Votre âge satisfait la condition spéciale";
    }

    return 0;
```

```
}
```

c.

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

int main()
{
    int base = 0;
    int a = 1;
    int b = 1;
    int temp = 0;

    for (int i = 0; i < 10; i++)
    {
        temp = b;
        b = a;
        a += temp;

        base += a * i;

        if (base > 100)
        {
            cout << "La suite commence à exploser";
        }
    }

    return 0;
}
```

Correction :

a. (1) Bloc : « *if*$(age >= 18)$ »
Condition d'entrée : il faut que *age* soit plus grand ou égal à 18 pour entrer dans la structure conditionnelle.

(2) Bloc : « *else if* $(age >= 0)$ »
Condition d'entrée : il faut que la condition (1) soit fausse et que *age* soit plus grand ou égal à 0 pour entrer dans la structure conditionnelle.

(3) Bloc : « *else* »
Condition d'entrée : il faut que les conditions (1) et (2) soit fausses pour entrer dans la structure conditionnelle.

Ce programme vérifie l'âge entré par l'utilisateur et différencie les réponses en fonction des cas possibles :

— L'utilisateur est majeur
— L'utilisateur est mineur
— L'utilisateur est un menteur

b. (1) Bloc : « *while* $(age < 0)$ »
Condition d'arrêt : Il faut que *age* soit plus grand ou égal à 0.

(2) Bloc : « *if* $(age \,\%\, 3 == 0 \;\&\&\; (age - 2) \,\%\, 5 == 1)$ »
Condition d'entrée : il faut que *age* soit divisible par 3 et que le reste de la division euclidienne de « *age-2* » par 5 soit égal à 1 pour entrer dans la structure conditionnelle.

Ce programme demande un âge à l'utilisateur. Si celui-ci est négatif, le programme le redemande jusqu'à avoir une réponse plus grande ou égale à 0. Il vérifie ensuite une condition spéciale.

c. (1) Bloc : « *for* $(int\ i = 0; i < 10; i++)$ »
Condition d'arrêt : Pour que la boucle s'arrête, il faut que i soit plus grand ou égal à 10. La boucle doit donc tourner 10 fois.

(2) Bloc : « *if* $(base > 100)$ »
Condition d'entrée : il faut que *base* soit plus grand que 100 pour entrer dans la structure conditionnelle.

Ce programme prend des valeurs dans des paramètres, il entre dans une boucle qui répétera 10 fois des calculs entre les paramètres et si le résultat des calculs (« *base* » dans le code) est au-dessus de 100, alors le code nous le fait savoir en l'affichant.

2.2 Erreurs 2

Trouver la ou les erreur(s) dans chaque programme ainsi que les possibles boucles infinies. Donner une correction possible.

a.
```cpp
#include <iostream>
using namespace std;

// On veut afficher 10 fois "bonjour".
int main()
{
    for (int i = 0; i <= 10; i++)
    {
        cout << "bonjour";
    }

    return 0;
}
```

b.
```cpp
#include <iostream>
using namespace std;

int main()
{
    int nombre;

    cout << "Donner un nombre :\n";
    cin >> nombre;

    while (true)
    {
        nombre *= 2;
        cout << nombre << endl;

        if (nombre > 100)
        {
            break;
        }
    }

    return 0;
}
```

c.

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

int main()
{
    int de;

    cout << "Donner le résultat du dé :\n";

    cin >> de;

    switch (de)
    {
        case 1:
            cout << "C'est 1 !";
        case 2:
            cout << "C'est 2 !";
        case 3:
            cout << "C'est 3 !";
        case 4:
            cout << "C'est 4 !";
        case 5:
            cout << "C'est 5 !";
        case 6:
            cout << "C'est 6 !";
        default:
            break;
    }

    return 0;
}
```

Correction :

a. Dans la boucle *for*, la condition de sortie est une inégalité large, ce qui va faire tourner la boucle 1 fois de trop et donc nous aurons écrit 11 fois « bonjour » au lieu de 10 fois. Pour résoudre cette erreur il suffit que changer l'inégalité large en inégalité stricte.

b. Dans la boucle *while*(*true*), la condition peut causer de gros problème car celle-co peut être infinie : sa seul condition de sortie est : « *if* (*nombre* $>$ 100) ». Cependant, si *nombre* n'arrive jamais au-dessus de 100, la boucle ne finira jamais. Ce cas est possible lorsque *nombre* est négatif. On peut alors :

— Filtrer ce que l'utilisateur rentre comme valeur dans *nombre* en vérifiant que c'est bien un entier positif

— Mettre une deuxième condition de sortie, notamment si la boucle tourne depuis trop longtemps. En mettant une variable qui compte le nombre de passages dans la boucle, on peut considérer que il faut arrêter et afficher un message d'erreur au bout d'un certain temps (ex : 1000 passages).

c. Dans le *switch*, il n'y a pas de « *break*; » à la fin de chaque *case*, ce qui fait si un des *case* est exécuté, le programme considère aussi les suivants. Pour régler ce problème il suffit de rajouter des *break* après chaque *cout*.

2.3 Complétions 2

Compléter la ou les lignes manquantes dans chaque programme.

Correction :

a.

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

// On cherche à déterminer si le nombre est premier (divisible seulement par lui-même
// et 1)
int main()
{
  int nombre;
  bool premier = true;

  cout << "Donner un nombre : ";
  cin >> nombre;

  // Il faut préciser le contenu de la boucle
  for (int i = 2 ; i < nombre ; i++)
  {
    if (nombre % i == 0)
    {
      premier = false;
      // On veut quitter la boucle dans ce cas
      break;
    }
  }

  // Cette partie est plutôt facile
  if (premier)
  {
    cout << "Le nombre est premier";
  }

  else
  {
    cout << "Le nombre n'est pas premier";
  }

  return 0;
}
```

b.

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

// On cherche à déterminer le nombre entier n tel que n^3 = 4n^2
int main()
{
  int n = 1;

  // n * n * n pour le cube de n et n * n pour le carré
  // Tant que les valeurs sont différentes, on continue
  while (n * n * n != 4 * n * n)
```

```
  {
    n++;
  }

  cout << n;

  return 0;
}
```

2.4 Du tableau à la boucle

Écrire une boucle et son contenu en se basant sur le tableau suivant :

# de passage	$a < 0.125$	a
0	false	1
1	false	0.5
2	false	0.25
3	false	0.125
4	true	0.0625

Correction :

```
while ( a >= 0.125 )
{
    a /= 2;
}
```

2.5 Parité

Écrire un programme qui détermine si un nombre est pair [4] ou non.

Correction :

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

int main()
{
  int x;

  cout << "Entrez un nombre : ";
  cin >> x;

  if (x % 2 == 0)
  {
    cout << "Ce nombre est pair";
  }

  else
  {
    cout << "Ce nombre n'est pas pair";
```

4. Un nombre pair est divisible par 2. Gardez cela en tête !

```
  }

  return 0;
}
```

2.6 Les premiers nombres premiers

Écrire un programme qui affiche tous les nombres premiers entre 2 et 100 (inclus). [5]

Correction :

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

int main()
{
    int indice = 2;
    int x;

    bool pasPremier;

    while (indice <= 100)
    {
        x = 2;
        pasPremier = false;

        while (x < indice)
        {
            if (indice % x == 0)
            {
                pasPremier = true;
                break;
            }

            x++;
        }

        if (!pasPremier)
        {
            cout << indice << endl;
        }

        indice++;
    }

    return 0;
}
```

2.7 Stan Robotix 1

Lors de la programmation du robot, il sera régulièrement utile de raisonner avec des systèmes qui peuvent avoir plus que deux états distincts. Il également plus agréable de travailler avec des valeurs

5. Astuce : utilisez une boucle dans une autre.

plus précises que "1", "2" ou "3". Une structure existe pour nous permettre de créer ces systèmes avec différents états : l'énumération. Supposons que le robot puisse rouler avec trois vitesses définies à l'avance. On peut écrire :

```
enum Vitesse
{
    lente,
    moyenne,
    rapide
};
```

Un nouveau type de variable, le type "Vitesse" est alors créé et utilisable comme n'importe quel type. Plus généralement, on a :

```
enum Système
{
    État1,
    État2,
    ...,
    ÉtatN
};
```

Les types venant d'énumération sont utilisables dans les structures *switch-case*. Écrire un programme qui prend un nombre entier en entrée. Afficher le type de vitesse en utilisant une structure *switch-case*.

— Si le nombre est positif, la vitesse est rapide.

— Si le nombre est négatif, la vitesse est lente.

— Si le nombre est nul, la vitesse est moyenne.

Correction :

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

enum Vitesse
{
    lente,
    moyenne,
    rapide
};

int main()
{

    Vitesse vitesse;

    int nb;
    cout << "Entrer un nombre entier : " << endl;
    cin >> nb;

    if (nb > 0)
    {
        vitesse = rapide;
    }

    else if (nb == 0)
```

```
    {
        vitesse = moyenne;
    }

    else
    {
        vitesse = lente;
    }

    switch (vitesse)
    {
    case lente:
        cout << "La vitesse est lente" ;
        break;
    case moyenne:
        cout << "La vitesse est moyenne" ;
        break;

    case rapide:
        cout << "La vitesse est rapide" ;
        break;

    default:
        break;
    }

    return 0;
}
```

2.8 La meilleure des structures conditionnelles

Dans le roman *Le meilleur des mondes*[6] la société est divisée en cinq castes : *Alpha*, *Beta*, *Gamma*, *Delta*, *Epsilon*. Les *Alpha* sont les plus grands. À l'opposé, les *Epsilon* sont les plus petits. Bien que peu de détails ne soient donnés quant à la correspondance entre taille et caste, on peut raisonnablement la faire ainsi :

- *Alpha* : 1m80 et plus
- *Beta* : 1m75-1m80
- *Gamma* : 1m70-1m75
- *Delta* : 1m65-1m70
- *Elpha* : moins de 1m65

En supposant que la taille soit le seul élément déterminant la caste, écrire un programme qui la détermine en fonction de la taille indiquée en entrée. L'exercice précédent peut être utilisé.

Correction :

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

enum Castes
{
    Alpha,
```

6. Aldous Huxley, 1932

```
    Beta,
    Gamma,
    Delta,
    Elpha
};

int main()
{

    Castes caste;

    float taille;
    cout << "Entrer la taille : " << endl;
    cin >> taille;

    if (taille >= 1.80)
    {
        caste = Alpha;
    }

    else if (taille >= 1.75)
    {
        caste = Beta;
    }

    else if (taille >= 1.70)
    {
        caste = Gamma;
    }

    else if (taille >= 1.65)
    {
        caste = Delta;
    }

    else
    {
        caste = Elpha;
    }

    switch (caste)
    {
    case Alpha:
        cout << "La caste est Alpha" ;
        break;

    case Beta:
        cout << "La caste est Beta" ;
        break;

    case Gamma:
        cout << "La caste est Gamma" ;
        break;

    case Delta:
        cout << "La caste est Delta" ;
```

```
        break;

    case Elpha:
        cout << "La caste est Elpha" ;
        break;

    default:
        break;
    }

    return 0;
}
```

2.9 Nombres aléatoires 1

Il existe plusieurs manières de générer des nombres aléatoires en C++. Il faut d'abord rajouter en début de programme la ligne :

```
#include <random>
```

Dans cet exercice, nous allons nous intéresser à un générateur à "distribution normale". Pour générer un nombre aléatoire de ce type, il faut rajouter les lignes suivantes :

```
random_device systeme{};
mt19937 generateur{systeme()};
normal_distribution<float> distribution{0.0, 1.0};

float nombre = distribution(generateur);
```

Écrire un programme générant 200 nombres de cette manière. Le programme devra séparer les nombres en trois catégories [7] :

- $x < -1$
- $x > 1$
- $-1 \leq x \leq 1$

Afficher combien il y a de nombres dans chaque catégorie. Y-a t-il une catégorie plus représentée que les autres ?

Correction : Il y a nettement plus de nombres entre -1 et 1 que dans les autres catégories.

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

int main()
{
    int plusGrand = 0, plusPetit = 0, entre = 0;

    random_device systeme{};
    mt19937 generateur{systeme()};
    normal_distribution<float> distribution{0.0, 1.0};
```

7. Certains pourraient être tentés d'utiliser les énumérations. Ce n'est pas utile dans ce cas.

```
    for (int i=0; i < 200; i++)
    {
        float nombre = distribution(generateur);

        if (nombre > 1)
        {
            plusGrand++;
        }
        else if (nombre < -1)
        {
            plusPetit++;
        }

        else
        {
            entre++;
        }
    }

    cout << "Il y a " << plusGrand << " nombre plus grand que 1" << endl
        << "Il y a " << plusPetit << " nombre plus petit que -1"<< endl
        << "Il y a " << entre   << " nombre entre 1 et -1";

    return 0;
}
```

2.10 Nombres aléatoires 2

Cet exercice se base sur le 2.9. Ajoutons deux nouvelles catégories de cette manière :

- $x < -2$
- $-2 \leq x < -1$
- $-1 \leq x \leq 1$
- $1 < x \leq 2$
- $x > 2$

Compter le nombre d'occurences dans chaque catégorie. Afficher par des étoiles (*) la distribution dans l'ordre indiqué des catégories. [8]

Correction :

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

int main()
{
    int moins2 = 0, moins1 = 0, entre1 = 0, plus1 = 0, plus2 = 0;
    random_device systeme{};
    mt19937 generateur{systeme()};
    normal_distribution<float> distribution{0.0, 1.0};

    for (int i=0; i < 200; i++)
```

8. Astuce : afficher la moitié du nombre d'occurence pour chaque catégorie.

```
{
    float nombre = distribution(generateur);

    if (nombre > 2)
    {
        plus2++ ;
    }

    else if (nombre > 1)
    {
        plus1++ ;
    }

    else if (nombre >= -1)
    {
        entre1++ ;
    }

    else if (nombre >= -2)
    {
        moins1++ ;
    }

    else
    {
        moins2++ ;
    }
}

cout << "\nx < -2     : ";

for (int k = 0; k < moins2 / 3; k++)
{
    cout << "*";
}

cout << "\nx < -1     : ";

for (int k = 0; k < moins1 / 3; k++)
{
    cout << "*";
}

cout << "\n-1 <= x <= 1 : ";

for (int k = 0; k < entre1 / 3; k++)
{
    cout << "*";
}

cout << "\nx > 1      : ";

for (int k = 0; k < plus1 / 3; k++)
{
    cout << "*";
}
```

```
    cout << "\nx > 2      : ";

    for (int k = 0; k < plus2 / 3; k++)
    {
        cout << "*";
    }

    cout << endl;

    return 0;
}
```

---

3 Vecteurs, tableaux et pointeurs

3.1 Tableau ou vecteur ?

Choisir, entre un tableau et un vecteur, le moyen le plus approprié de stocker les données suivantes :

a. Tous les nombres premiers entre 1 et 100
b. Les 10 premiers nombres premiers entre 1 et 100
c. Le mode de vitesse du robot durant les phases de la période autonome
d. L'âge des répondants d'un sondage

Correction :

a. Vecteur (on ne connait pas forcément le nombre de nombres premiers entre 1 et 100)
b. Tableau (la taille est fixée à 10)
c. Tableau (le nombre de phases et connu à l'avance)
d. Vecteur (on ne connait pas forcément la taille de l'échantillon)

3.2 À couvert !

On a rangé les précipitations moyennes de pluie et de neige à Montréal entre 1981 et 2010 [9] dans des tableaux. Compléter le programme suivant pour obtenir la moyenne et le total annuel pour chaque tableau.

Correction :

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

int main()
{
    float pluie[12] = {27.3, 20.9, 29.7, 67.7, 81.2, 87.0, 89.3, 94.1, 83.0, 89.1, 76.7,
        38.8};
    float neige[12] = {49.5, 41.2, 36.2, 12.9, 0, 0, 0, 0, 0, 1.8, 19.0, 48.9};

    float moyennePluie = 0, moyenneNeige = 0, totalPluie = 0, totalNeige = 0;

    for (int i = 0; i < 12; i++)
    {
        totalPluie += pluie[i];
        totalNeige += neige[i];
    }

    moyennePluie = totalPluie / 12;
    moyenneNeige = totalNeige / 12;

    cout << "Pluie :\n Total : " << totalPluie << "\nMoyenne : " << moyennePluie;
    cout << "Neige :\n Total : " << totalNeige << "\nMoyenne : " << moyenneNeige;

    return 0;
}
```

9. Statistiques de la ville de Montréal : *https* : *//ville.montreal.qc.ca/pls/portal/docs/*1/85762076.*JPG*

3.3 Une racine carrée... compliquée !

Se référer à l'exercice 1.8 pour l'utilisation de la racine carrée. Pour calculer la distance entre deux points A et B de coordonnées respectives (x_a, y_a) et (x_b, y_b), on utilise la formule suivante :

$$d = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$$

Par ailleurs, on a rangé les coordonnées de deux points dans des tableaux :

```
float posX[2] = {0.3, -1.2};
float posY[2] = {0.6, 2};
```

Donner la ligne permettant de calculer la distance entre les deux points.

Correction :

```
float distance = sqrt((posX[0] - posX[1]) * (posX[0] - posX[1]) + (posY[0] - posY[1]) *
    (posY[0] - posY[1]))
```

3.4 Prédictions 2

Pour chaque programme, dire les valeurs de toutes les variables à la fin de l'exécution de celui-ci.

a.
```
#include <iostream>
using namespace std;
int main()
{
    int a = 2;
    int* b = new int;
    b = &a;
    a = 5;
    return 0;
}
```

b.
```
#include <iostream>
using namespace std;
int main()
{
    int a = 3;
    int* b = new int;
    *b = a;
    *b = 4;
    return 0;
}
```

Correction :

a. a = 5, b = 5

b. a = 3, b = 4

3.5 Alice et Bob 2

Bob a créé un programme permettant de ranger n'importe quelle quantité de nombres entiers dans un tableau.

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

int main()
{
    int taille;
    cin >> taille;

    int tableau[taille];
    int nombre;

    for (int i = 0; i < taille; i++)
    {
        cin >> nombre;
        tableau[i] = nombre;
    }

    return 0;
}
```

Lorsqu'Alice teste son programme, celui-ci ne fonctionne pas. Trouver le problème et le corriger.

Correction :
Le tableau n'est pas initialisé avec une taille fixée à l'avance. Il faut donc l'allouer dynamiquement.

3.6 Un vecteur bien long

Écrire un programme rangeant les $2n + 1$ premiers carrés dans un vecteur, où n est un entier donné par l'utilisateur. Afficher la taille du vecteur et le nombre en son milieu.

Correction :

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int n;
    cin >> n;

    vector<int> carres;

    for (int i = 1; i <= 2 * n + 1; i++)
    {
        carres.push_back(i * i);
    }

    cout << "La taille est : " << carres.size() << "\nLe nombre au milieu est : " <<
        carres.at(n + 1);

    return 0;
}
```

3.7 Stan Robotix 2

Écrire un programme prenant un entier en entrée et ayant un vecteur de type Vitesse. Le programme fait les actions suivantes :

- Le nombre est 1 : on ajoute au vecteur un élément "lente" de type vitesse
- Le nombre est 2 : on ajoute au vecteur un élément "moyenne" de type vitesse
- Le nombre est 3 : on ajoute au vecteur un élément "rapide" de type vitesse
- Le nombre est autre chose : le programme s'arrête.

Si le nombre est 1, 2 ou 3, le programme doit continuer à demander un nombre.

Correction :

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

enum Vitesse
{
    lente,
    moyenne,
    rapide
};

int main()
{
    int typeVitesse;
    bool continuerBoucle = true;
    vector<Vitesse> vitesses;

    while (continuerBoucle)
    {
        cout << "Entrez la vitesse : \n >>> ";
        cin >> typeVitesse;

        switch(typeVitesse)
        {
            case 1:
                vitesses.push_back(Vitesse::lente);
                break;

            case 2:
                vitesses.push_back(Vitesse::moyenne);
                break;

            case 3:
                vitesses.push_back(Vitesse::rapide);
                break;

            default:
                continuerBoucle = false;
                break;
        }
    }

    return 0;
```

```
}
```

3.8 Céline et les statistiques

Céline a écrit un programme permettant de prédire [10] des mesures à partir d'observations passées. Son programme contient deux vecteurs comme suit :

```
vector<int> occurences = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
    19, 20};
vector<int> observations = {0.17, 0.26, 0.35, 0.51, 0.55, 0.45, 0.6, 0.63, 0.68, 0.72, 0.79,
    0.86, 0.82, 0.9, 0.88, 0.93, 0.97, 1.0, 1.1, 1.11, 1.15};
```

Céline a également calculé une valeur précise qu'elle stocke dans une variable :

```
float a = 0.4435;
```

Mais Céline a oublié comment calculer la deuxième ! Cette valeur se calcule ainsi :

$$b = \bar{y} - a\bar{x}$$

Où $\bar{x}$ et $\bar{y}$ représentent respectivement la moyenne des occurences et la moyenne des observations. Écrire un programme calculant la valeur manquante. Prédire la mesure pour 21 occurences avec la formule suivante :

$$m = ao + b$$

Avec m la mesure et o le nombre d'occurences.

Correction :

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

int main()
{
    vector<int> occurences = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
        18, 19, 20};
    vector<float> observations = {0.17, 0.26, 0.35, 0.51, 0.55, 0.45, 0.6, 0.63, 0.68, 0.72,
        0.79, 0.86, 0.82, 0.9, 0.88, 0.93, 0.97, 1.0, 1.1, 1.11, 1.15};

    float totalOccurences = 0, totalObservations = 0;
    float a = 0.4435;

    for (int i = 0; i < occurences.size(); i++)
    {
        totalOccurences += occurences[i];
        totalObservations += observations[i];
    }

    float b = totalObservations / observations.size() - a * totalOccurences /
        occurences.size();

    cout << "La valeur attendue pour la 21e occurence est : " << a * 21 + b;
```

10. La méthode utilisée est une régression linéaire

```
    return 0;
}
```

3.9 Un vecteur bien long... mais sans pairs !

Pour effacer le premier élément d'un vecteur, on peut utiliser la ligne suivante :

```
vecteur.erase(vecteur.begin());
```

Si l'on souhaite effacer l'élément à la position n [11], on peut modifier la ligne en :

```
vecteur.erase(vecteur.begin() + n);
```

Ajouter au programme de l'exercice 3.6 un bout de code supprimant tous les nombres pairs du vecteur avant d'afficher sa taille et son premier élément.

Correction :

```
// Ici on n'efface pas les nombres pairs, mais le résultat est le même
vector<int> carresImpairs;

for (int i = 0; i < carres.size(); i++)
{
    if (carres.at(i) % 2 == 1)
    {
        carresImpairs.push_back(carres.at(i));
    }
}

carres = carresImpairs;
```

11. Rappel : la position dans un vecteur commence à 0

4 Fonctions

4.1 Le meilleur des types de retour

Pour chaque proposition, indiquer le type de retour le plus judicieux (plusieurs réponses sont parfois possibles).

a. Une fonction déterminant la parité d'un nombre

b. Une fonction calculant l'aire d'un cercle en fonction du rayon

c. Une fonction déterminant si le robot est à moins d'un mètre du mur

d. Une fonction réinitialisant les paramètres du robot

e. Une fonction qui affiche un message en fonction de l'heure

Correction :

a. *bool* : il n'y a que deux états possibles

b. *float* ou *double* : en fonction de la précision du rayon

c. *bool* : il n'y a que deux états possibles

d. *void* ou *bool* : la fonction peut ne rien renvoyer, ou renvoyer un état indiquant si la réinitialisation s'est bien faite

e. *void* : on ne veut rien renvoyer

4.2 Une fonction... spéciale !

Voici une fonction un peu spéciale :

```
float fonctionSpeciale(float x)
{
    return sqrt((x - 2) / x);
}
```

Cette fonction renvoie des valeurs aberrantes pour $0 \leq x < 2$. Proposer un ajout permettant d'éviter ces valeurs aberrantes.

Correction : On peut modifier la fonction de deux manières différentes :

a. On renvoie une valeur arbitraire pour $0 \leq x < 2$

```
float fonctionSpeciale(float x)
{
    if (x >= 0 && x < 2)
    {
        return 0;
    }

    return sqrt((x - 2) / x);
}
```

b. On peut prolonger la fonction f en utilisant la valeur absolue *abs* du module *cmath* :

```
float fonctionSpeciale(float x)
{
    return sqrt(abs((x - 2) / x));
}
```

4.3 Le plus petit des multiples

Écrire et implémenter une fonction à un paramètre (entier) qui détermine le plus petit entier divisible par 17 supérieur au paramètre.

Correction :

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

int plusPetitMultiple(int nombre)
{
    if (nombre < 17)
    {
        return 17;
    }

    nombre++;

    while (nombre % 17 != 0 || nombre == 17)
    {
        nombre++;
    }

    return nombre;
}

int main()
{
    int entree = 0;

    cout << "Veuillez rentrer un nombre :\n>>> ";
    cin >> entree;
    cout << plusPetitMultiple(entree);

    return 0;
}
```

4.4 Mes carrés !

Écrire et implémenter une fonction qui renvoie les 10 premiers carrés sous forme d'un vecteur.

Correction :

```cpp
#include <iostream>
#include <vector>
using namespace std;

vector<int> premiersCarres()
{
    vector<int> carres;

    for (int i = 0; i < 10; i++)
    {
        carres.push_back(i * i);
```

```
    }

    return carres;
}

int main()
{
    vector<int> carres = premiersCarres();

    for (int i = 0; i < 10; i++)
    {
        cout << "Le carré #" << i << " est : " << carres[i] << endl;
    }

    return 0;
}
```

4.5 Céline et le jeu

Céline joue à un jeu qui a les spécificités suivantes :

Gains	0	5	-10
Probabilité	$0,2$	$0,3$	$0,5$

Actuellement, le gain moyen espéré lors d'une partie est de :

$$0 \times 0,2 + 5 \times 0,3 - 10 \times 0,5 = -3,5$$

Céline joue donc à un jeu qui est défavorable. Écrire une fonction à deux paramètres calculant le gain moyen espéré par joueur qui joue à un jeu ayant les spécificités suivantes :

Gains	0	5	g
Probabilité	$0,2$	$0,3$	p

Essayer quelques valeurs de g (négatives) et de p (entre 0 et 1) pour voir quelles spécificités seraient favorables.

Correction :

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

float esperance(float g, float p)
{
    return 5 * 0.3 + g * p;
}

int main()
{
    cout << esperance(-4, 0.5);

    return 0;
}
```

4.6 In french please !

Un mile vaut environ $1,609$ kilomètres. De même, un kilomètre vaut environ $0,621$ mile. Écrire une fonction permettant de faire la conversion dans les deux sens pour une mesure donnée. [12]

Correction :

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

float conversion(float valeur, bool mile)
{
    if (mile)
    {
        return valeur * 1.609;
    }

    return valeur * 0.621;
}

int main()
{
    float x;
    bool mode;

    cin >> x >> mode;

    cout << "La conversion vaut : " << conversion(x, mode);

    return 0;
}
```

4.7 Prédictions 2

Pour chaque programme, prédire les valeurs de chaque variables à la fin de l'exécution de celui-ci :

a.
```cpp
#include <iostream>
using namespace std;

void cube(int nombre)
{
    nombre = nombre * nombre * nombre;
}

int main()
{
    int* nombre = new int;
    *nombre = 3;
    cube(*nombre);

    return 0;
}
```

12. Vous pouvez utiliser un deuxième paramètre pour spécifier le sens de conversion.

b.
```
#include <iostream>
using namespace std;

void cube(int* nombre)
{
    *nombre = *nombre * *nombre * *nombre;
}

int main()
{
    int nombre = 3;
    cube(&nombre);

    return 0;
}
```

Correction :

a. Dans la fonction *cube*, *nombre* vaut 27. Dans la fonction *main*, *nombre* vaut 3 (transmission par valeur).

b. Dans la fonction *cube*, *nombre* vaut 27. Dans la fonction *main*, *nombre* vaut aussi 27 (transmission par adresse).

4.8 Nombres aléatoires 3

L'exercice 2.9 présentait un générateur de nombres aléatoires à distribution normale. Les paramètres utilisés étaient $0,0$ et $1,0$, mais il est possible de les changer. En effet, la ligne suivante est tout à fait valide :

```
normal_distribution<float> distribution{esperance, ecartType};
```

Il suffit que les variables *esperance* et *ecartType* soit deux *float* bien définis. Écrire et implémenter une fonction permettant de générer des nombres aléatoires à distribution normale mais à paramètres variables.

Correction :

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

float aleatoireNormale(float esperance, float ecartType)
{
    random_device systeme{};
    mt19937 generateur{systeme()};
    normal_distribution<float> distribution{esperance, ecartType};

    return distribution(generateur);
}

int main()
{
    float esperance, ecartType;
```

```
    cout << "Veuillez rentrer l'espérance et l'écart-type :\n>>> ";
    cin >> esperance;
    cout << ">>> ";
    cin >> ecartType;

    return 0;
}
```

4.9 SohCahToa !

Il est possible d'utiliser les fonctions trigonométriques en C++. Pour ce faire, il est nécessaire d'inclure le module *cmath* comme pour la racine carrée. Supposons que l'on veuille calculer le cosinus d'un angle, on peut procéder ainsi :

```
float resultat = cos(angle);
```

Notez bien que les fonctions trigonométriques de ce module fonctionnent avec des radians et non pas des degrés classiques.
La fonction suivante fait intervenir la fonction sinus mais n'est pas définie en 0 :

```
float f(float x)
{
    return sin(x) / x;
}
```

Tester la fonction avec de petites valeurs (inférieures à 0,5 mais supérieures à 0) et proposer un ajout permettant d'éviter un résultat abérrant si $x = 0$. Implémenter cet ajout.

Correction :

```
float f(float x)
{
    if (x == 0)
    {
        return 1;
    }

    return sin(x) / x;
}
```

4.10 Centaines, dizaines et unités

L'opérateur % permet de séparer un nombre et chaque chiffre qui le compose. En effet, on peut remarquer que :

$$639\%10 = 9$$
$$\frac{639-9}{10}\%10 = 63\%10 = 3$$
$$\frac{63-3}{10}\%10 = 6\%10 = 6$$

Écrire et implémenter une fonction séparant de cette manière un nombre à 3 chiffres. La fonction peut afficher les chiffres séparés par un espace.

Correction :

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

void separer(int nombre)
{
    int unite = nombre % 10;
    int dizaine = ((nombre - unite) / 10) % 10;
    int centaine = ((nombre - unite - 10 * dizaine) / 100) % 10;

    cout << "La décomposition de " << nombre << " en centaine / dizaine / unité est " <<
        centaine << "/" << dizaine << "/" << unite;
}

int main()
{
    int nbr;

    cout << "Choisir un nombre entre 1 et 999 :\n>>> ";
    cin >> nbr;

    separer(nbr);

    return 0;
}
```

4.11 Stan Robotix 3

Il est possible d'obtenir l'heure actuelle en C++. Pour ce faire, il faut rajouter en début de programme la ligne suivante :

```
#include <ctime>
```

Supposons que l'on souhaite obtenir les secondes actuelles. On peut alors écrire :

```
time_t temps = time(0);
tm* heureActuelle = localtime(&temps);
int secondes = heureActuelle->tm_sec;
```

Lors de la période autonome, il peut être utile [13] de séparer ladite période en sections durant un certain temps. Pour simplifier, on peut résumer les déplacements du robots en 4 catégories :

- Avancer
- Reculer
- Tourner à droite
- Tourner à gauche

Voici un exemple de période autonome, avec t le nombre de secondes écoulées :

1. $0 \leq t < 10$: avancer
2. $10 \leq t < 12$: tourner à droite

13. C'est une méthode peu efficace mais simple à mettre en place

3. $12 \leq t < 20$: avancer
4. $20 \leq t < 28$: reculer
5. $28 \leq t < 34$: tourner à droite
6. $34 \leq t < 40$: avancer
7. $40 \leq t < 46$: reculer
8. $46 \leq t < 48$: tourner à droite
9. $48 \leq t < 58$: reculer
10. $58 \leq t < 60$: ne rien faire

En supposant que la période autonome dure 60s [14], écrire et implémenter une fonction renvoyant l'action que doit faire le robot en fonction des secondes actuelles.

Correction :

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

enum Deplacement
{
    Avancer,
    Reculer,
    TournerDroite,
    TournerGauche,
    Rien
};

Deplacement autonome(int secondes)
{
    if (secondes < 10)
    {
        return Avancer;
    }

    if (secondes < 12)
    {
        return TournerDroite;
    }

    if (secondes < 20)
    {
        return Avancer;
    }

    if (secondes < 28)
    {
        return Reculer;
    }

    if (secondes < 34)
    {
        return TournerDroite;
    }
```

14. Elle dure en réalité 15s.

```
    if (secondes < 40)
    {
        return Avancer;
    }

    if (secondes < 46)
    {
        return Reculer;
    }

    if (secondes < 48)
    {
        return TournerDroite;
    }

    if (secondes < 58)
    {
        return Reculer;
    }

    return Rien;
}

int main()
{
    time_t temps = time(0);
    tm* heureActuelle = localtime(&temps);
    int secondes = heureActuelle->tm_sec;

    Deplacement deplacement = autonome(secondes);

    switch(deplacement)
    {
        case Avancer:
            cout << "Le robot avance !";
            break;

        case Reculer:
            cout << "Le robot recule !";
            break;

        case TournerDroite:
            cout << "Le robot tourne à droite !";
            break;

        case TournerGauche:
            cout << "Le robot tourne à gauche !";
            break;

        case Rien:
            cout << "Le robot ne fait rien !";
            break;
    }

    return 0;
```

```
}
```

5 Classes

5.1 Définitions

En utilisant la définition typique des méthodes *get* et *set*, donner une définition possible des méthodes *getX*, *getY*, *setX* et *setY*.

Correction :

```
float Point::getX()
{
    return x;
}

float Point::getY()
{
    return y;
}

void Point::setX(float X)
{
    x = X;
}

void Point::setY(float Y)
{
    y = Y;
}
```

5.2 Objet construit !

Lister, dans l'ordre, **toutes** les différentes informations affichées par la console lors de l'exécution de la fonction *main* suivante :

Correction :

```
int main()
{
    Point pt;
    pt.setX(2);
    pt.setY(3);

    cout << 3 * pt.getX() - 4 * pt.getY();

    return 0;
}
```

1. "Objet construit !"
2. "-6"
3. "Objet détruiut !"

5.3 Le retour de la racine carrée perdue

Cet exercice nécessite la classe du 5.1. Compléter le programme suivant pour obtenir la distance entre les deux points (voir le 3.3 pour la formule).

Correction :

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

class Point
{
    // Ne pas compléter la classe
};

int main()
{
    Point pt1, pt2;
    pt1.setX(2);
    pt1.setY(3);
    pt2.setX(3);
    pt2.setY(5);

    cout << "La distance entre les points est : " << sqrt( (pt1.getX() - pt2.getX()) *
        (pt1.getX() - pt2.getX()) + (pt1.getY() - pt2.getY()) * (pt1.getY() - pt2.getY()));

    return 0;
}
```

5.4 Triangle

On utilise un troisième point de coordonnées $(4; 4)$ pour former un triangle. Écrire un programme en utilisant la classe *Point* pour déterminer la longueur des trois côtés du triangle, respectivement a, b et c.

Correction :

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

class Point
{
    // Déclaration déjà connue
};

// On suppose que les méthodes sont bien définies

int main()
{
    Point pt1, pt2, pt3;
    pt1.setX(2);
    pt1.setY(3);
    pt2.setX(3);
    pt2.setY(5);
```

```
    pt3.setX(4);
    pt3.setY(4);

    cout << "La distance entre les points 1 et 2 est : " << sqrt( (pt1.getX() - pt2.getX()) *
        (pt1.getX() - pt2.getX()) + (pt1.getY() - pt2.getY()) * (pt1.getY() - pt2.getY())) <<
        endl;
    cout << "La distance entre les points 2 et 3 est : " << sqrt( (pt2.getX() - pt3.getX()) *
        (pt2.getX() - pt3.getX()) + (pt2.getY() - pt3.getY()) * (pt2.getY() - pt3.getY())) <<
        endl;
    cout << "La distance entre les points 1 et 3 est : " << sqrt( (pt1.getX() - pt3.getX()) *
        (pt1.getX() - pt3.getX()) + (pt1.getY() - pt3.getY()) * (pt1.getY() - pt3.getY())) <<
        endl;

    return 0;
}
```

5.5 Cercle

Écrire et implémenter une classe cercle ayant les spécificités suivantes :

- Un attribut *rayon*
- Des méthodes *get* et *set* classiques
- Une méthode donnant la circonférence du cercle
- Une méthode donnant l'aire du cercle

Correction :

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

class Cercle
{
    float rayon;
    const float pi = 3.1415;

public:
    Cercle();
    ~Cercle();

    float getRayon();
    float getCircomference();
    float getAire();

    void setRayon(float Rayon);
};

Cercle::Cercle() { }

Cercle::~Cercle() { }

float Cercle::getRayon()
{
    return rayon;
}
```

```
float Cercle::getCircomference()
{
    return 2 * rayon * pi;
}

float Cercle::getAire()
{
    return rayon * rayon * pi;
}

void Cercle::setRayon(float Rayon)
{
    rayon = Rayon;
}

int main()
{
    Cercle cercle;
    cercle.setRayon(3.1415);

    cout << "Le cercle de rayon " << cercle.getRayon() << " a un périmètre de " <<
        cercle.getCircomference() << " et une aire de " << cercle.getAire();

    return 0;
}
```

5.6 Le fameux hérisson

Compléter la classe suivante pour déterminer la distance parcourue par Sonic en 2 heures :

Correction :

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

class Sonic
{
    const float vitesse = 342.88;

public:
    Sonic();
    ~Sonic();

    float getVitesse();
    float distanceParcourue(float secondes);
};

Sonic::Sonic() { }

Sonic::~Sonic() { }

float Sonic::getVitesse()
{
    return vitesse;
```

```
}

float Sonic::distanceParcourue(float secondes)
{
    return vitesse * secondes;
}

int main()
{
    Sonic sonic;

    cout << "Sonic a parcouru " << sonic.distanceParcourue(7200) << " mètres en 2h";

    return 0;
}
```

5.7 Une architecture classique

En utilisant la classe de l'exercice 5.6, créer deux autres fichiers et répartir le programme en utilisant l'architecture classique :

- sonic.h
- sonic.cpp
- main.cpp

Pas de correction pour cet exercice. Poser des questions au besoin.

5.8 Jeu

L'objectif de cet exercice est de créer une classe modélisant l'affrontement entre deux joueurs. Le jeu possède les caractéristiques suivantes :

Joueur 1 / Joueur 2	Coopérer	Tricher
Coopérer	+3 / +3	+5 / −1
Tricher	−1 / +5	+0 / +0

Le joueur 1 joue de manière aléatoire, alors que le joueur 2 commence par coopérer puis copie la dernière action du joueur 1 [15]. Le nombre de tours durant une partie doit être décidé à l'avance.

a. Créer une classe *Jeu* permettant d'encapsuler deux joueurs et le nombre de tours dans une partie.

b. Ajouter des méthodes *get* et *set*.

c. Créer une méthode *jouer* [16] modélisant l'affrontement entre les joueurs. Afficher à la fin de chaque tour le score de chaque joueur. À la fin de la partie, afficher le joueur gagnant.

Correction :

```
// jeu.h
```

15. Astuce : assurez-vous de garder la dernière action dans une variable.
16. La logique de cette fonction pourra faire l'objet d'une discussion au besoin.

```h
#include "joueur.h"

enum TypeVictoire
{
    Joueur1Victorieux,
    Joueur2Victorieux,
    Egalite
};

class Jeu
{
    Joueur joueur1;
    Joueur joueur2;
    int nombreTours;

public:
    Jeu();
    ~Jeu();

    int getNombreTours();

    Joueur getJoueur1();
    Joueur getJoueur2();

    void setNombreTours(int);
    void setJoueur1(Joueur);
    void setJoueur2(Joueur);

    TypeVictoire jouer();
};
```

```cpp
// jeu.cpp

#include "jeu.h"

#include <random>
#include <iostream>
using namespace std;

Jeu::Jeu() { }
Jeu::~Jeu() { }

int Jeu::getNombreTours()
{
    return nombreTours;
}

Joueur Jeu::getJoueur1()
{
    return joueur1;
}

Joueur Jeu::getJoueur2()
{
    return joueur2;
```

```
}

void Jeu::setNombreTours(int NombreTours)
{
    nombreTours = NombreTours;
}

void Jeu::setJoueur1(Joueur Joueur1)
{
    joueur1 = Joueur1;
}

void Jeu::setJoueur2(Joueur Joueur2)
{
    joueur2 = Joueur2;
}

TypeVictoire Jeu::jouer()
{
    Action derniereAction = Action::Cooperer;
    Action actionAleatoire;

    joueur1.setScore(0);
    joueur2.setScore(0);

    for (int i = 0; i < nombreTours; i++)
    {
        actionAleatoire = choixAleatoire();

        switch(actionAleatoire)
        {
            case Action::Cooperer:
                if (derniereAction == actionAleatoire)
                {
                    joueur1.setScore(joueur1.getScore() + 3);
                    joueur2.setScore(joueur2.getScore() + 3);
                }

                else
                {
                    joueur1.setScore(joueur1.getScore() - 1);
                    joueur2.setScore(joueur2.getScore() + 5);
                }

                break;

            case Action::Tricher:
                if (derniereAction != actionAleatoire)
                {
                    joueur1.setScore(joueur1.getScore() + 5);
                    joueur2.setScore(joueur2.getScore() - 1);
                }

                break;
        }
```

```
        derniereAction = actionAleatoire;

        cout << "Joueur 1 : " << joueur1.getScore() << endl << "Joueur 2 : " <<
            joueur2.getScore() << endl;
    }

    if (joueur1.getScore() > joueur2.getScore())
    {
        return TypeVictoire::Joueur1Victorieux;
    }

    else if (joueur1.getScore() < joueur2.getScore())
    {
        return TypeVictoire::Joueur1Victorieux;
    }

    else
    {
        return TypeVictoire::Egalite;
    }
}
```

5.9 Simulation

Créer une classe *Simulation* permettant de simuler un nombre indéfini (c'est-à-dire à préciser par l'utilisateur) de parties de 5 tours. Afficher le nombre de victoires de chaque joueur.

Correction :

```
// simulation.h

#include "jeu.h"

class Simulation
{
    Jeu jeu;

    int nombreVictoiresJoueur1;
    int nombreVictoiresJoueur2;
    int nombreSimulations;

public:
    Simulation();
    ~Simulation();

    int getNombreVictoiresJoueur1();
    int getNombreVictoiresJoueur2();
    int getNombreSimulations();

    Jeu getJeu();

    void setNombreSimulations(int);
    void setJeu(Jeu);

    void simuler();
```

```
};
```

```
// simulation.cpp

#include "simulation.h"

Simulation::Simulation() { }

Simulation::~Simulation() { }

int Simulation::getNombreVictoiresJoueur1()
{
    return nombreVictoiresJoueur1;
}

int Simulation::getNombreVictoiresJoueur2()
{
    return nombreVictoiresJoueur2;
}

int Simulation::getNombreSimulations()
{
    return nombreSimulations;
}

Jeu Simulation::getJeu()
{
    return jeu;
}

void Simulation::setNombreSimulations(int NombreSimulations)
{
    nombreSimulations = NombreSimulations;
}

void Simulation::setJeu(Jeu jeu_)
{
    jeu = jeu_;
}

void Simulation::simuler()
{
    jeu.setNombreTours(5);
    TypeVictoire typeVictoire;
    nombreVictoiresJoueur1 = nombreVictoiresJoueur2 = 0;

    for (int i = 0; i < nombreSimulations; i++)
    {
        typeVictoire = jeu.jouer();

        switch(typeVictoire)
        {
            case TypeVictoire::Joueur1Victorieux:
                nombreVictoiresJoueur1++;
                break;
```

```
            case TypeVictoire::Joueur2Victorieux:
                nombreVictoiresJoueur2++;
                break;

            default:
                break;
        }
    }
}
```

```
// main.cpp

#include <iostream>
using namespace std;

#include "simulation.h"

int main()
{
    Joueur joueur1;
    Joueur joueur2;
    Jeu jeu;

    jeu.setJoueur1(joueur1);
    jeu.setJoueur2(joueur2);

    Simulation simulation;

    simulation.setJeu(jeu);

    simulation.simuler();

    cout << "Le joueur 1 a " << simulation.getNombreVictoiresJoueur1() << " victoires.\n" <<
        endl << "Le joueur 2 a " << simulation.getNombreVictoiresJoueur2() << "victoires.";

    return 0;
}
```

5.10 Prédictions 3

Écrire une classe dans un fichier *.h* qui pourrait avoir comme fichier *.cpp* le fichier suivant :

Correction :

```
#include "sphere.h"
// Le fichier sphere.h ne contient aucune inclusion

Sphere::Sphere() { }
Sphere::~Sphere() { }

float Sphere::getRayon()
{
    return rayon;
```

```
}

void Sphere::setRayon(float Rayon)
{
    rayon = Rayon;
}

float Sphere::getAire()
{
    return 4 * pi * rayon * rayon;
}

float Sphere::getVolume()
{
    return 4 / 3 * pi * rayon * rayon * rayon;
}
```

```
class Sphere
{
    float rayon;
    const float pi = 3.141592;

public:
    Sphere();
    ~Sphere();

    float getRayon();

    void setRayon(float Rayon);

    float getAire();
    float getVolume();
};
```

6 Git et Github

6.1 Des commandes et des explications

Associer chaque commande à son utilité :

a. git pull
b. git push
c. git fetch
d. git commit -m "Ajout d'un fichier"
e. git add fichier.h
f. git checkout MaBranche
g. git status
h. git clone

1. Créer un paquet ayant comme titre "Ajout d'un fichier"
2. Télécharger (sur un serveur) le paquet sur la branche active
3. Obtenir des informations sur la branche et l'espace de travail local
4. Basculer sur la branche "MaBranche"
5. Télécharger (sur son ordinateur) les nouveaux fichiers présents sur la branche
6. Télécharger (sur son ordinateur) les nouvelles informations de l'espace de travail virtuel
7. Cloner l'espace de travail virtuel sur son ordinateur
8. Préparer un paquet en y ajoutant le fichier "fichier.h"

Correction : a. 5., b. 2., c. 6., d. 1., e. 8., f. 4., g. 3., h. 7.

6.2 Entrainement

a. Créer une branche à partir de *master*.
b. Ajouter dans le répertoire Activités/2023/ un exercice [17]. Bien mentionner son nom.
c. Télécharger l'exercice sur l'espace de travail virtuel.
d. Basculer sur la branche d'un autre groupe.
e. Télécharger l'exercice de l'autre groupe sur l'espace de travail local.
f. Ajouter un commentaire dans le fichier *main.cpp*.
g. Télécharger les changements sur l'espace de travail virtuel.
h. Basculer sur sa branche.
i. Télécharger les modifications sur l'espace de travail local.
j. Observer les modications.

Pas de correction pour cet exercice. Poser des questions au besoin.

17. Décider lequel sur le moment

7 Exercices supplémentaires

7.1 Nombres aléatoires 4

Écrire une fonction à un paramètre, que l'on note ici n, et générer n paires de nombres aléatoires entre 0 et 1, notés ici x et y. Après chaque génération de paire, tester la condition suivante :

$$x^2 + y^2 \leq 1$$

Si la condition est remplie, incrémenter un compteur. Sinon, ne rien faire. La fonction devra renvoyer le valeur suivante :

$$4 \times \frac{compteur}{n}$$

Si la fonction renvoie un nombre entier, forcer la conversion de n en *float*. Que remarquez-vous [18] ?

Correction :

```
float monteCarlo(int nbr)
{
  int compteur = 0;
  float x, y;

  random_device random;
  mt19937 generateur(random());
  uniform_real_distribution<> distribution(0, 1);

  for(int i = 0; i < nbr; i++)
    {
      x = distribution(generateur);
      y = distribution(generateur);

      if(x * x + y * y <= float(1))
      {
        compteur++;
      }
    }

  return float(4 * compteur) / nbr;
}
```

7.2 Alice et Bob 3

Du haut de ses 553, 33 mètres, la tour CN est la plus haute construction humaine au Canada. Bob désire estimer, en négligeant la friction, le temps que pendrait une balle à tomber depuis le haut de la tour. Pour ce faire, Alice prépare la classe suivante :

```
#include <vector>
using namespace std;

class Simulateur
{
    const float g = 9.8067;
    const float hauteurInitiale = 553.33;
```

18. La méthode utilisée pour approximer ce nombre s'appelle la méthode Monte-Carlo.

```
    vector<float> positions;

public:
    Simulateur();
    ~Simulateur();

    // Méthodes get et set classiques
    ...
     .
     .
     .
    ...

    void simulation(float);
};
```

La position, en mètres, de la balle en fonction du temps est donnée par l'équation suivante :

$$x(t) = y_0 - \frac{1}{2}gt^2$$

Avec y_0 la hauteur initiale, g l'accélération gravitationnelle et t le temps écoulé, en secondes, depuis le début de la chute.
Bob cherche à déterminer numériquement l'instant où la balle touche le sol, c'est-à-dire où $x(t) = 0$. Alice s'arrête avant d'écrire la définition de *simulation*, mais Bob n'a pas la moindre idée de comment continuer ! Écrire la définition de la méthode *simulation* en tenant compte des choses suivantes :

- Le paramètre de *simulation* représente le nombre de secondes écoulées entre chaque itération. Il est recommandé de tester la méthode avec un paramètre inférieur à 1.
- La méthode devra enregistrer l'ensemble des positions au cours du temps dans le vecteur dédié.
- La méthode devra s'arrêter lorsque la position devient négative et afficher le temps que cela a pris.

Correction :

```
void Simulateur::simulation(float deltaT)
{
    float position = hauteurInitiale;
    float duree = 0;
    positions.push_back(position);

    while (position > 0)
    {
        position -= g * deltaT * deltaT / 2;
        duree += deltaT;
        positions.push_back(position);
    }

    // On suppose que iostream est inclus
    cout << duree;
}
```

7.3 Tri par sélection

Écrire une fonction qui trie (du plus petit au plus grand) un vecteur d'entiers sans utiliser la fonction *sort* de la bibliothèque standard. Conseil pour l'algorithme :

1. Stocker le premier élément du vecteur dans une variable (X).
2. Pour chaque élément du vecteur, déterminer s'il est plus petit que X. Si oui, remplacer X par cet élément.
3. À la fin du parcours du vecteur, ajouter X à un autre vecteur. Supprimer X du premier vecteur (voir l'exercice 3.9 au besoin).
4. Répéter l'algorithme.

Correction :

```
vector<int> trieSelection(vector<int> vecteur)
{
    int element;
    int elementIndice;
    int taille = vecteur.size();
    vector<int> vecteurTrie;

    while(vecteurTrie.size() < taille)
    {
        element = vecteur.at(0);
        elementIndice = 0;

        for (int i = 0; i < vecteur.size(); i++)
        {
            if (vecteur.at(i) < element)
            {
                element = vecteur.at(i);
                elementIndice = i;
            }
        }

        vecteurTrie.push_back(element);
        vecteur.erase(vecteur.begin() + elementIndice);
    }

    return vecteurTrie;
}
```

7.4 Diviser pour mieux estimer

On a construit la fonction suivante :

```
float f(float x)
{
    return x * x - 2;
}
```

Transcrire l'algorithme [19] suivant en une fonction de paramètre n :

19. Il s'agit d'un algorithme de dichotomie.

$a \leftarrow 0$
$b \leftarrow 2$
Pour i allant de 0 à n :
    Si $f(a) \times f(\frac{a+b}{2}) < 0$:
        $b \leftarrow \frac{a+b}{2}$
    Sinon :
        $a \leftarrow \frac{a+b}{2}$

La fonction devra renvoyer $\frac{a+b}{2}$. Que remarquez-vous si vous élevez cette valeur au carré ?

Correction :

```
float dichotomie(int n)
{
    float a = 0;
    float b = 2;

    for (int i = 0; i < n; i++)
    {
        if( f(a) * f(b) < 0)
        {
            b = (a + b) / 2;
        }

        else
        {
            a = (a + b) / 2;
        }
    }

    return (a + b) / 2;
}
```

On obtient la valeur 2.

7.5 C'est plus ! C'est moins !

Écrire un programme demandant un nombre entier entre 1 et 100 en entrée. Le programme devra deviner votre nombre le plus efficacement possible en ayant comme seule information l'ordre, c'est-à-dire si son choix est plus petit ou plus grand que le nombre choisi. Afficher le nombre de tentatives requises par l'ordinateur pour trouver votre nombre.

Correction :

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

int main()
{
    int nombre = 0;

    while (nombre <= 0 || nombre > 100)
    {
        cout << "Veuillez donnner un nombre entre 1 et 100\n >>> ";
```

```
        cin >> nombre;
    }

    int essai = 50;
    int delta = 25;
    int tentatives = 0;

    while (essai != nombre)
    {
        if (essai > nombre)
        {
            essai -= delta;
        }

        else if (essai < nombre)
        {
            essai += delta;
        }

        delta /= 2;

        if (delta < 1)
        {
            delta = 1;
        }

        tentatives++;

        cout << "Tentative #" << tentatives << " : " << essai << endl;
    }

    cout << "Le nombre trouvé en " << tentatives << "tentatives est : " << essai ;

    return 0;
}
```

7.6 Alors, c'est qui le plus malin ?

Faire le même programme que celui du 7.5 en inversant les rôles : vous êtes celui devant deviner un nombre aléatoire entre 1 et 100 en ayant comme seule information l'ordre.

Correction :

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

int main()
{
    int essai = 0;

    random_device systeme{};
    mt19937 generateur{systeme()};
    uniform_int_distribution<int> distribution{1, 100};
```

```
    int nombre = distribution(generateur);
    int tentatives = 0;

    while (nombre != essai)
    {
        cout << "Veuillez donnner un nombre entre 1 et 100\n >>> ";
        cin >> essai;

        if (nombre > essai)
        {
            cout << "C'est plus !\n";
        }

        else if (nombre < essai)
        {
            cout << "C'est moins !\n";
        }

        tentatives++;
    }

    cout << "Bravo ! Vous avez trouvé le nombre en " << tentatives << " tentatives !";

    return 0;
}
```

7.7 Stan Robotix 4

Le tableau suivant contient le nombre d'heures de maths par semaine qu'ont 16 personnes réliées (de près ou de loin) à Stan Robotix :

```
float heures[16] = {25, 11, 6, 10, 0, 5, 4, 6, 4, 4, 4, 4, 6, 6, 4, 0};
```

Écrire un programme donnant les valeurs suivantes :

- La moyenne
- La médiane
- Le minimum
- Le maximum

Au besoin, vous pouvez trier un tableau de taille n de la manière suivante :

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    ...
    int tableau[n] = {...};
    ...
    sort(tableau, tableau + n);
    ...
    return 0;
}
```

Correction :

```cpp
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

int main()
{
    float heures[16] = {25, 11, 6, 10, 0, 5, 4, 6, 4, 4, 4, 4, 6, 6, 4, 0};

    sort(heures, heures + 16);

    float total = 0;

    for (int i = 0; i < 16; i++)
    {
        total += heures[i];
    }

    cout << "minimum, moyenne, médiane, maximum : " << heures[0] << ", " << total / float(16)
         << ", " << (heures[7] + heures[8] / 2) << ", " << heures[15];

    return 0;
}
```

7.8 Genshin Impact et les chaînes de caractères

Dans le *metagame* du jeu *Genshin Impact*, les différents personnages sont séparés en 3 catégories [20].

- *Main DPS* : personnages infligeant la majorité des dégâts dans une équipe et étant longtemps sur le terrain
- *Sup DPS* : personnages infligeant beaucoup de dégâts, mais ayant peu de temps de terrain
- *Support* : personnages ayant diverses facultés, comme le soin, la possibilité de changer des réactions, d'augmenter les statistiques des autres membres de l'équipe...

L'objectif de cet exercice est de renvoyer la catégorie de tous les personnages d'une équipe donnée en entrée.

Il est possible de stocker des chaînes de caractères en C++ de la manière suivante :

```cpp
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    string prenom;

    cout << "Quel est ton prénom ?\n";
    cin >> prenom;
    cout << "Bonjour " << prenom << " !";
}
```

20. Voir https ://genshin.gg/tier-list

On a rangé les personnages[21] dans divers vecteurs selon leur catégorie :

```
vector<string> mainDPS = {"Alhaitham", "Ayaka", "Ayato", "Cyno", "Diluc", "Eula", "Ganyu",
    "Hu Tao", "Itto", "Kaveh", "Keqing", "Klee", "Lyney", "Ningguang", "Noelle", "Nomade",
    "Raiden", "Razor", "Tartaglia", "Yanfei", "Yoimiya", "Xiao", "Xinyan"};
vector<string> subDPS = {"Albedo", "Aloy", "Amber", "Beidou", "Fischl", "Heizou", "Kaeya",
    "Tighnari", "Venti", "Xiangling", "Xingqiu", "Yae Miko", "Yelan"};
vector<string> support = {"Baizhu", "Barbara", "Bennett", "Candace", "Chongyun", "Collei",
    "Dehya", "Diona", "Dori", "Faruzan", "Gorou", "Jean", "Kazuha", "Kirara", "Kokomi",
    "Layla", "Lisa", "Lynette", "Mika", "Mona", "Nahida", "Nilou", "Qiqi", "Rosaria",
    "Sara", "Sayu", "Shenhe", "Shinobu", "Sucrose", "Thoma", "Yao Yao", "Yun Jin",
    "Zhongli"};
```

Le programme devra :

1. Demander le nom des personnages composant l'équipe (4 personnages) et les stocker dans un vecteur.
2. Afficher la catégorie de chaque personnage.
3. Afficher (gérer efficacement les différents cas) un message différent en fonction du nombre de *mainDPS*.

Correction :

```
#include <iostream>
#include <cstring>
#include <vector>
using namespace std;

int main()
{
    vector<string> equipe;
    string personnage;

    vector<string> mainDPS = {"Alhaitham", "Ayaka", "Ayato", "Cyno", "Diluc", "Eula",
        "Ganyu", "Hu Tao", "Itto", "Kaveh", "Keqing", "Klee", "Lyney", "Ningguang", "Noelle",
        "Nomade", "Raiden", "Razor", "Tartaglia", "Yanfei", "Yoimiya", "Xiao", "Xinyan"};
    vector<string> subDPS = {"Albedo", "Aloy", "Amber", "Beidou", "Fischl", "Heizou",
        "Kaeya", "Tighnari", "Venti", "Xiangling", "Xingqiu", "Yae Miko", "Yelan"};
    vector<string> support = {"Baizhu", "Barbara", "Bennett", "Candace", "Chongyun",
        "Collei", "Dehya", "Diona", "Dori", "Faruzan", "Gorou", "Jean", "Kazuha", "Kirara",
        "Kokomi", "Layla", "Lisa", "Lynette", "Mika", "Mona", "Nahida", "Nilou", "Qiqi",
        "Rosaria", "Sara", "Sayu", "Shenhe", "Shinobu", "Sucrose", "Thoma", "Yao Yao", "Yun
        Jin", "Zhongli"};

    for (int i = 0; i < 4; i++)
    {
        cout << "Veuillez donner le nom du personnage #" << i + 1 << " :\n>>> ";
        cin >> personnage;

        equipe.push_back(personnage);
    }

    int nombreMainDPS = 0;
```

21. Versions 4.0 et antérieures. Le voyageur n'est pas compté.

```
for (int i = 0; i < 4; i++)
{
    for (int j = 0; j < mainDPS.size(); j++)
    {
        if (equipe.at(i) == mainDPS.at(j))
        {
            nombreMainDPS++;

            cout << equipe.at(i) << " est un main DPS.\n";
        }
    }

    for (int j = 0; j < subDPS.size(); j++)
    {
        if (equipe.at(i) == subDPS.at(j))
        {
            cout << equipe.at(i) << " est un sub DPS.\n";
        }
    }

    for (int j = 0; j < support.size(); j++)
    {
        if (equipe.at(i) == support.at(j))
        {
            cout << equipe.at(i) << " est un support.\n";
        }
    }
}

switch (nombreMainDPS)
{
case 0:
    cout << "Je parie que tu as une team Nilou-bloom.";
    break;

case 1:
    cout << "Une team classique.";
    break;

case 2:
    cout << "Une team intéressante.";
    break;

case 3:
    cout << "Tu joues avec le feu.";
    break;

case 4:
    cout << "Comment tu fais pour 36* les abyss ?";
    break;

default:
    break;
}

return 0;
```

```
}
```

7.9 Stan Robotix et les chaînes de caractères

Voir l'exercice précédent pour l'utilisation des chaînes de caractères en C++. On a rangé le nom de certains membres (anciens ou non) de Stan Robotix dans un vecteur. On a fait de même pour leur équipe principale.

```
vector<string> noms = {"Alice", "Alban", "André", "Ines", "Kaloyan", "Majeed", "Max",
    "Maxence", "Maxime", "Mohammad", "Nina", "Raphaël", "Stéphanie", "Victor", "Zachary"};
vector<string> equipes = {"S&D", "Construction", "Programmation", "Construction",
    "Marketing", "S&D", "S&D", "Construction", "S&D", "S&D", "Construction",
    "Programmation", "S&D", "Construction", "Construction"};
```

1. Sélectionner au hasard un nom.
2. Demander la sous-équipe. Si le résultat est bon, augmenter un score.
3. Répéter l'opération au moins 12 fois.

Si l'utilisateur a au moins 70% de bonnes réponses après 12 tentatives, le programme s'arrête. Sinon, le programme continue tant que l'utilisateur n'a pas 70% de bonnes réponses.

Correction :

```
#include <iostream>
#include <cstring>
#include <vector>
#include <random>
using namespace std;

int main()
{
   vector<string> noms = {"Alice", "Alban", "André", "Ines", "Kaloyan", "Majeed", "Max",
       "Maxence", "Maxime", "Mohammad", "Nina", "Raphaël", "Stéphanie", "Victor", "Zachary"};
    vector<string> equipes = {"S&D", "Construction", "Programmation", "Construction",
        "Marketing", "S&D", "S&D", "Construction", "S&D", "S&D", "Construction",
        "Programmation", "S&D", "Construction", "Construction"};

    random_device systeme{};
    mt19937 generateur{systeme()};
    uniform_int_distribution<int> distribution{0, noms.size() - 1};

    float precision = 0;
    int position = 0, bonnesReponses = 0, reponses = 0;
    string reponse;

    while (precision < 0.7 || reponses < 12)
    {
        position = distribution(generateur);

        cout << "Question #" << reponses + 1 << " :\nQuelle est (fut) la sous-équipe de " <<
            noms.at(position) << " ?\n>>> ";
        cin >> reponse;

        if (reponse == equipes.at(position))
```

```
        {
            cout << "Bonne réponse !\n";

            bonnesReponses++;
        }

        else
        {
            cout << "Mauvaise réponse !\n";
        }

        reponses++;

        precision = (float)bonnesReponses / (float)reponses;

        cout << "Précision : " << precision << endl;
    }

    return 0;
}
```

7.10 Neurone artificiel

Les intelligences artificielles comme *ChatGPT* utilisent des modèles d'apprentissage se basant sur un réseau de neurones à plusieurs couches. L'objectif de l'exercice est de modéliser un neurone en C++.

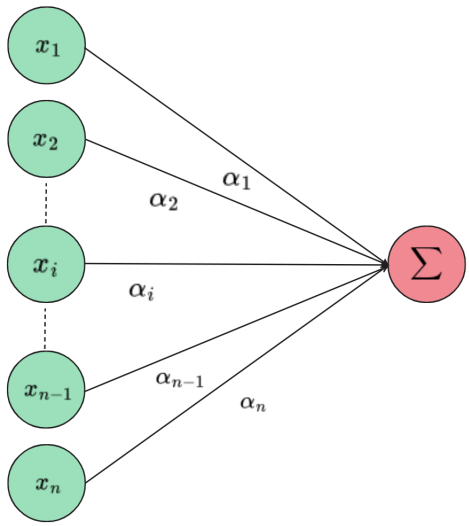

FIGURE 1 – Schéma simplifié d'un neurone artificiel à n entrées

Ce neurone prend n valeurs en entrée et donnera en sortie la valeur suivante :

$$\sum_{i=1}^{n} \alpha_i x_i = \alpha_1 x_1 + \alpha_2 x_2 + ... + \alpha_i x_i + ... + \alpha_{n-1} x_{n-1} + \alpha_n x_n$$

On représente souvent les entrées et les poids (les α_i) sous forme de vecteurs. Ainsi, si les entrées sont $X = \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix}$ et les poids $A = \begin{pmatrix} 5 \\ 8 \\ 1 \end{pmatrix}$, le neurone doit renvoyer la valeur :

$$2 \times 5 + 3 \times 8 + 5 \times 1 = 10 + 24 + 5 = 39$$

Écrire et implémenter une classe *Neurone* ayant les spécificités suivantes :

1. Deux vecteurs de *float* représentant les entrées et les poids.
2. Les méthodes *get* et *set*.
3. Une méthode qui renvoie la valeur attendue du neurone.

Correction :

```
// neurone.h

#include <vector>
using namespace std;

class Neurone
{
    vector<float> entrees;
    vector<float> poids;

public:
    Neurone();
    ~Neurone();

    vector<float> getEntrees();
    vector<float> getPoids();

    void setEntrees(vector<float>);
    void setPoids(vector<float>);

    float sortie();
};
```

```
// neurone.cpp

#include "neurone.h"

Neurone::Neurone() { }

Neurone::~Neurone() { }

std::vector<float> Neurone::getEntrees()
{
    return entrees;
}

std::vector<float> Neurone::getPoids()
{
    return poids;
}
```

```cpp
void Neurone::setEntrees(std::vector<float> Entrees)
{
    entrees = Entrees;
}

void Neurone::setPoids(std::vector<float> Poids)
{
    poids = Poids;
}

float Neurone::sortie()
{
    if (entrees.size() != poids.size())
    {
        return 0;
    }

    float total = 0;

    for (int i = 0; i < entrees.size(); i++)
    {
        total += entrees.at(i) * poids.at(i);
    }

    return total;
}
```

```cpp
// main.cpp

#include <iostream>
using namespace std;

#include "neurone.h"

int main()
{
    Neurone neurone;

    vector<float> entrees = {1, 2, 3, 4, 5};
    vector<float> poids = {0, -3, -2, 7, 1};

    neurone.setEntrees(entrees);
    neurone.setPoids(poids);

    cout << "Sortie du neurone : " << neurone.sortie();

    return 0;
}
```

7.11 En lettres s'il vous plait !

Écrire un programme demandant continuellement un chiffre (entier) à l'utilisateur. Le programme devra l'afficher en lettre.

Exemple : l'utilisateur rentre "2", le programme affiche "deux".

Correction :

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

int main()
{
    int nombre;

    while (true)
    {
        cout << "Veuillez entrer un nombre :\n>>> ";
        cin >> nombre;

        nombre %= 10;

        switch (nombre)
        {
        case 0:
            cout << "Vous avez rentré le chiffre zéro !";
            break;

        case 1:
            cout << "Vous avez rentré le chiffre un !";
            break;

        case 2:
            cout << "Vous avez rentré le chiffre deux !";
            break;

        case 3:
            cout << "Vous avez rentré le chiffre trois !";
            break;

        case 4:
            cout << "Vous avez rentré le chiffre quatre !";
            break;

        case 5:
            cout << "Vous avez rentré le chiffre cinq !";
            break;

        case 6:
            cout << "Vous avez rentré le chiffre six !";
            break;

        case 7:
            cout << "Vous avez rentré le chiffre sept !";
            break;

        case 8:
            cout << "Vous avez rentré le chiffre huit !";
            break;
```

```
        case 9:
            cout << "Vous avez rentré le chiffre neuf !";
            break;

        default:
            break;
        }

        cout << endl;
    }

    return 0;
}
```

7.12 En chiffres s'il vous plait !

Voir l'exercice 7.8 pour l'utilisation de chaînes de caractères en C++. Chercher comment convertir une chaîne de caractères en nombre *float*, puis écrire un programme demandant en entrée un nombre sous la forme d'une chaîne de caractères. Afficher ensuite la moitié de ce nombre.

Correction :

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

int main()
{
    string nombre;

    cout << "Veuillez rentrer un nombre :\n>>> ";
    cin >> nombre;

    cout << "La moitié du nombre est : " << atof(nombre.c_str()) / 2;

    return 0;
}
```

7.13 Chiffrement par décalage

Le chiffrement par décalage [22] est un exemple très ancien de cryptographie. Il consiste à remplacer chaque lettre d'un message par une autre lettre à distance constante.
Ici [23], le texte est décalé de 3 lettres vers la droite. Écrire un programme ayant les spécificités suivantes :

1. Demander en entrée un nombre entier et une chaînes de caractères (un message en minuscules).
2. Décaler chaque lettre du message en fonction du nombre en entrée. Ne rien faire pour les espaces.
3. Afficher le message chiffré.

Correction :

22. Voir https ://fr.wikipedia.org/wiki/Chiffrement_par_d%C3%A9calage
23. Illustration venant de https ://commons.wikimedia.org/wiki/File :Caesar3.svg ?uselang=fr

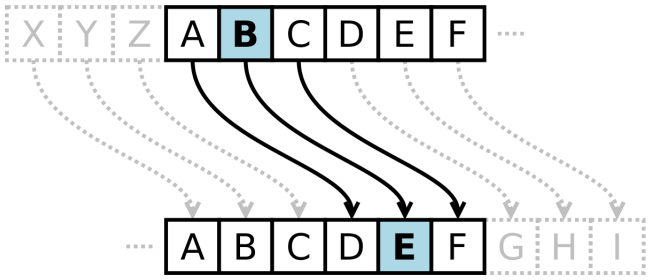

FIGURE 2 – Chiffrement par décalage

```
#include <iostream>
#include <cstring>
#include <vector>
using namespace std;

int main()
{
    string texte;
    string alphabet = "abcdefghijklmnopqrstuvwxyz";

    int cle;

    cout << "Veuillez rentrer le texte àchiffrer :\n>>> ";
    // Pour les textes avec espaces
    getline(cin, texte);
    cout << "Veuillez rentrer la clé de chiffrement :\n>>> ";
    cin >> cle;

    for (int i = 0; i < texte.length(); i++)
    {
        for (int j = 0; j < alphabet.length(); j++)
        {
            if (texte[i] == alphabet.at(j))
            {
                texte[i] = alphabet.at((j + cle) % alphabet.size());
                break;
            }
        }
    }

    cout << "Le message chiffré est : " << texte;

    return 0;
}
```

7.14 Stan Robotix 5

Le nombre 6622 possède la curieuse particularité d'avoir le même nombre de chiffres que de facteurs premiers. Écrire un programme stockant dans un tableau les facteurs premiers de 6622. Il peut être utile de déterminer tous les nombres premiers entre 2 et 50.

Correction :

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

bool premier(int nombre)
{
    for (int i = 2; i < nombre; i++)
    {
        if (nombre % i == 0)
        {
            return false;
        }
    }

    return true;
}

int main()
{
    int facteursPremiers[4];
    int position = 0;
    int testFacteur = 2;

    while (position < 4)
    {
        if (6622 % testFacteur == 0 && premier(testFacteur))
        {
            facteursPremiers[position] = testFacteur;
            position++;
        }

        testFacteur++;
    }

    for (int i = 0; i < 4; i++)
    {
        cout << "Facteur " << i << " : " << facteursPremiers[i] << endl;
    }

    return 0;
}
```

7.15 Brute Force

Voir l'exercice 7.8 pour l'utilisation des chaînes de caractères. Écrire un programme qui génère et affiche toutes les chaînes de caractères possibles de 1 à 4 caractères. Les majuscules, minuscules, chiffres et caractères spéciaux sont à prendre en compte.

Correction :

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

```cpp
int main()
{
    string alphabet =
        "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%?&*()-_=+/{}[]~,.':;<>";
    string texte = "  ";
    int compteur = 0;

    for (int i = 0; i < alphabet.length(); i++)
    {
        for (int j = 0; j < alphabet.length(); j++)
        {
            for (int k = 0; k < alphabet.length(); k++)
            {
                for (int l = 0; l < alphabet.length(); l++)
                {
                    texte[0] = alphabet[l];
                    texte[1] = alphabet[k];
                    texte[2] = alphabet[j];
                    texte[3] = alphabet[i];

                    compteur++;

                    // L'affichage peut durer plusieurs heures !
                    // Si l'on met en commentaire la ligne suivante
                    // le programme ne met que quelques secondes à
                    // prendre fin...
                    cout << "Chaine " << compteur << " : " << texte << endl;
                }
            }
        }
    }

    cout << "Fin avec " << compteur << " chaines";

    return 0;
}
```

7.16 Alice et Bob 4

En finances, l'option *call* constitue un droit d'acheter plus tard un bien à un prix fixé, indépendamment de sa valeur à la date de l'expiration du contrat.
Alice possède une compagnie en pleine effervescence. La valeur d'une action actuellement est de 100$. Bob désire acheter 100 actions de la compagnie d'Alice, mais dans un an car il n'a pas immédiatement les fonds. Bob va alors signer un contrat *call* avec sa banque, avec un prix d'exercice fixé à 120$. En échange d'intérêts fixes à verser à la banque, celle-ci assure à Bob de lui fournir dans un an 100 actions de la compagnie d'Alice à 120$ par action, et ce peu importe le véritable prix de l'action.

On peut donc résumer les différents cas possibles :

— En 2024, la valeur d'une action de la compagnie d'Alice dépasse les 120$: Bob exerce son contrat et Alice est obligé de lui vendre 100 actions à 100$.

— En 2024, la valeur d'une action de la compagnie d'Alice est en dessous de 120$: Bob n'a aucun intérêt à exercer son contrat puisqu'il peut acheter les actions sur le marché à un prix inférieur au prix fixé.

L'objectif de cet exercice est de déterminer la somme à verser à la banque à partir de laquelle elle peut espérer faire des gains peu importe la situation.

Valeur supplémentaire	$+30\$$	$-10\$$
Probabilité	0.3	0.7

Le tableau ci-dessus présente deux cas de figure, le premier où l'action vaut 130$ en 2024, l'autre où l'action vaut 90$ en 2024.
Pour que la banque puisse espérer toujours faire des gains, on démontre que la somme i à verser par action doit satisfaire l'inégalité :

$$i > p\alpha$$

avec p la probabilité d'une augmentation de la valeur de l'action et α la valeur supplémentaire.
Écrire une fonction déterminant numériquement la somme i. La fonction devra pouvoir s'adapter à différentes précisions.

Correction :

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

int main()
{
    float probabilite = 0.3;
    float gain = 30;
    float interets = 0;
    float precision = 0.1;

    // La question étant mal construite,
    // il suffit de prendre i > probabilite * gain.
    // Essayons tout de même autrement,
    // dans le cas où l'on ne connait que l'ordre :

    while (interets <= probabilite * gain)
    {
        interets += precision;
    }

    cout << "La somme àverser est " << interets;

    return 0;
}
```

7.17 La pyramide d'astérisques

Voici une tour faite à partir d'astérisques :
Écrire un programme qui demande la hauteur à l'utilisateur et qui génère la tour de hauteur associée.

Correction :

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

int main()
```

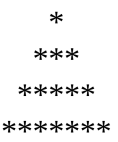

FIGURE 3 – Tour de hauteur 4

```
{
    int hauteur;

    cout << "Hauteur de la tour :\n >>> ";
    cin >> hauteur;

    cout << endl << endl;

    for (int i = 0; i < hauteur; i++)
    {
        for (int j = 0; j < hauteur - i - 1; j++)
        {
            cout << " ";
        }

        for (int j = 0; j < 2 * i + 1; j++)
        {
            cout << "*";
        }

        cout << endl;
    }

    cout << endl << endl;

    return 0;
}
```

7.18 Un grapheur de base

Le code en annexe permet d'afficher le graphe d'une fonction dans la console. Les abscisses minimales et maximales sont affichées en bas du graphe.

1. Tester le code.
2. Modifier quelques valeurs pour comprendre certaines parties (ne pas toucher au vecteur).
3. Tracer la fonction $f(x) = 2x - 4$.
4. La décaler de 10 unités vers la gauche et mettre le zoom à 1.

Pas de correction pour cet exercice. Poser des questions au besoin.

7.19 Marche aléatoire

La marche aléatoire est un modèle mathématique de processus réliés au hasard. Le modèle le plus simple consiste à augmenter ou diminuer un compteur de 1 à chaque itération, et ce de façon équiprobable. À l'aide de boucles, simuler la marche aléatoire en partant de 0 pour des durées de 10, 30 et

100 itérations. Faire une moyenne de la position atteinte à la dernière itération.

Correction :

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

int main()
{
    srand(time(0));

    int iterations = 30;
    int somme = 0;
    int position;
    int nbrAleatoire;

    int nbrSimulation = 10000;

    for (int i = 0; i < nbrSimulation; i++)
    {
        position = 0;

        for (int j = 0; j < iterations; j++)
        {
            nbrAleatoire = rand() % 2;

            if (nbrAleatoire == 1)
            {
                position++;
            }

            else
            {
                position--;
            }
        }

        somme += position;
    }

    cout << "Distance maximale en moyenne : " << float(somme) / float(nbrSimulation);

    return 0;
}
```

7.20 Produits de vecteurs

Le produit scalaire de deux vecteurs est la somme du produit terme à terme de chaque élément des vecteurs. Il s'agit d'une opération qui a de multiples applications, notamment en physique pour faire

des calculs sur des forces. Pour $v_1 = \begin{pmatrix}1\\3\\2\end{pmatrix}$ et $v_2 = \begin{pmatrix}-2\\-2\\4\end{pmatrix}$, le produit scalaire vaut :

$$\begin{aligned} v_1 \cdot v_2 &= 1 \times (-2) + 3 \times (-2) + 2 \times 4 \\ &= 0 \end{aligned}$$

Dans ce cas (quand le produit scalaire vaut 0), on dit que les vecteurs sont orthogonaux.
Écrire une fonction calculant le produit scalaire de deux vecteurs (attention, ils doivent avoir la même taille).
Il existe un autre produit sur les vecteurs : le produit vectoriel. Celui génère un nouveau vecteur au lieu de renvoyer un nombre. Le vecteur renvoyé est perpendiculaire aux deux autres. L'opération est légèrement plus complexe et se fait uniquement pour des vecteurs à 3 éléments.
Pour $w_1 = \begin{pmatrix}a\\b\\c\end{pmatrix}$ et $w_2 = \begin{pmatrix}x\\y\\z\end{pmatrix}$, le produit vectoriel vaut :

$$w_1 \times w_2 = \begin{pmatrix}bz - cy\\cx - az\\ay - bx\end{pmatrix}$$

Écrire une fonction renvoyant le produit vectoriel de deux vecteurs. Vérifier l'orthogonalité du vecteur renvoyé avec les deux premiers.

Correction :

```cpp
#include <iostream>
#include <vector>
using namespace std;

float produitScalaire(vector<float> v, vector<float> w)
{
    if (v.size() != w.size())
    {
        return 0;
    }

    float somme = 0;

    for (int i = 0; i < v.size(); i++)
    {
        somme += v.at(i) * w.at(i);
    }

    return somme;
}

vector<float> produitVectoriel(vector<float> v, vector<float> w)
{
    if (v.size() != w.size() || v.size() != 3 || w.size() != 3)
    {
        return vector<float>{0, 0, 0};
    }

    return vector<float>{v.at(1) * w.at(2) - v.at(2) * w.at(1),
```

```
                    v.at(2) * w.at(0) - v.at(0) * w.at(2),
                    v.at(0) * w.at(1) - v.at(1) * w.at(0)};
}

int main()
{
    vector<float> v = {1, 3, 2};
    vector<float> w = {-2, -2, 4};
    vector<float> produit = produitVectoriel(v, w);

    cout << "Produit scalaire de v et w : " << produitScalaire(v, w) << endl;
    cout << "Produit vectoriel de v et w : " << produit.at(0) << ", " << produit.at(1) << ",
        " << produit.at(2) << endl;
    cout << "Produit scalaire de v et v * w : " << produitScalaire(v, produit) << endl;
    cout << "Produit scalaire de w et v * w : " << produitScalaire(w, produit) << endl;

    return 0;
}
```

7.21 Link, un archer hors-pair

Dans la série *The Legend of Zelda*, Link, le protagoniste, est amené à utiliser plusieurs types d'armes, notamment des arcs. Dans cet exercice, on va estimer un angle optimal de visée pour toucher un ennemi situé à une certaine distance. En négligeant les frottements de l'air, on peut montrer qu'un projectile de vitesse initiale v_0 et lancé d'un angle α suit la trajectoire suivante :

$$y(x) = -\frac{9.81}{2}\frac{x^2}{\cos(\alpha)^2 v_0^2} + \tan(\alpha)x$$

où x représente la distance parcourue (en mètres) et $y(x)$ la hauteur du projectile. On peut approximer la vitesse initiale d'une flèche tirée par un arc par 75 m.s^{-1}. Écrire un programme déterminant l'angle optimal pour que Link puisse toucher un ennemi à 420 mètres, avec une précision de 0,1. Link peut-il toucher un ennemi à 600 mètres ? Note : les fonctions cos et tan se trouvent dans le module cmath.

Correction :

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

float trajectoire(float distanceParcourue, float vitesseInitiale, float angle)
{
    return - (9.81 / 2) * distanceParcourue * distanceParcourue / (cos(angle) * cos(angle) *
        vitesseInitiale * vitesseInitiale) + tan(angle) * distanceParcourue;
}

int main()
{
    // Dichotomie pour trouver la deuxième racine,
    // la première étant 0.

    float a;
    float b;
    float vitesseInitiale = 75;
```

```
    vector<float> distance;

    for (float angle = 0; angle < 2 * atanf(1); angle += 0.1)
    {
        a = 0.5;
        b = 700;

        while (abs(a - b) > 0.001)
        {
            if (trajectoire(a, vitesseInitiale, angle) * trajectoire((a + b) / 2,
                vitesseInitiale, angle) < 0)
            {
                b = (a + b) / 2;
            }

            else
            {
                a = (a + b) / 2;
            }
        }

        distance.push_back(abs((a + b) / 2 - 420));
    }

    float minimum = distance.at(0);
    int positionMinimum = 0;

    for (int i = 1; i < distance.size(); i++)
    {
        if (distance.at(i) < minimum)
        {
            minimum = distance.at(i);
            positionMinimum = i;
        }
    }

    cout << "L'angle optimal est : " << 0.1 * positionMinimum << " avec un écart de " <<
        distance.at(positionMinimum);

    return 0;
}
```

7.22 Encore plus de binaire

En se basant sur l'exercice 1.10, écrire un programme pour décoder un octet (8 bits). Ensuite, modifier le programme pour décoder n'importe quel nombre de bits.

Correction :

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

int main()
```

```
{
    int nbrBits;
    int bit;
    int resultat = 0;

    cout << "Nombre de bits àdécoder :\n>>> ";
    cin >> nbrBits;

    for (int i = 1; i <= nbrBits; i++)
    {
        cout << "Bits " << i << " : ";
        cin >> bit;

        resultat += bit * pow(2, nbrBits - i);
    }

    cout << "Le résultat est : " << resultat;

    return 0;
}
```

7.23 Des chiffres mais pas trop

Le concept de chiffres significatifs est intimement lié à celui de mesure. Si vous mesurez la longueur d'un objet à la règle, vous obtiendrez une précision au milimètre. Pour une mesure de $42,0$, on dit que le nombre de chiffres significatifs est 3. Utiliser cette valeur dans un calcul et exprimer le résultat avec 6 chiffres n'a alors pas de sens car il serait plus précis que votre mesure. Si l'on veut exprimer 382.93 et 0.008321 avec 3 chiffres significatifs, on obtient 383 (arrondi par excès) et 0.00832 (arrondi par défaut). Notez que les 0 après la virgule ne comptent pas.
Écrire un programme pour exprimer n'importe quel nombre positif avec un nombre spécifié de chiffres significatifs. Indice : pour les nombres plus grand que 1, il peut être utile de le décomposer en unités, dizaines, centaines... (voir l'exerice 4.10). Pour les nombres inférieurs à 1, il peut être utile de savoir la puissance de 10 à partir de laquelle il devient supérieur à 1 (exemple : pour 0,0023 c'est 1000 car $1000 \times 0,0023 = 2,3$).

Correction :

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

int main()
{
    int nbrChiffresSignificatifs;
    float nbr;

    int partieEntiere;
    float partieDecimale;

    int nbrChiffresPartieDecimale = 0;

    vector<int> decompositionPartieEntiere;
    vector<int> decompositionPartieDecimale;
```

```
float resultat = 0;

cout << "Indiquer le nombre àarrondir :\n>>> ";
cin >> nbr;

cout << "Indiquer le nombre de chiffres significatifs :\n>>> ";
cin >> nbrChiffresSignificatifs;

if (nbr > 0)
{
    partieEntiere = (int)floor(nbr);
    partieDecimale = nbr - floor(nbr);

    while (partieDecimale != 0)
    {
        decompositionPartieDecimale.push_back(int(floor(partieDecimale * 10)));
        partieDecimale = partieDecimale * 10 -
            decompositionPartieDecimale[nbrChiffresPartieDecimale];
        nbrChiffresPartieDecimale++;
    }
}

else
{
    partieEntiere = (int)ceil(nbr);
    partieDecimale = nbr + ceil(nbr);

    while (partieDecimale != 0)
    {
        decompositionPartieDecimale.push_back(int(ceil(partieDecimale * 10)));
        partieDecimale = partieDecimale * 10 -
            decompositionPartieDecimale[nbrChiffresPartieDecimale];
        nbrChiffresPartieDecimale++;
    }
}

while (partieEntiere != 0)
{
    decompositionPartieEntiere.push_back(partieEntiere % 10);
    partieEntiere = (partieEntiere - partieEntiere % 10) / 10;
}

if (decompositionPartieEntiere.size() == 0)
{
    int position = 0;

    while (decompositionPartieDecimale[position] == 0)
    {
        position++;
    }

    for (int i = position; i < position + min((int)decompositionPartieDecimale.size(),
         nbrChiffresSignificatifs); i++)
    {
        if (i == position + nbrChiffresSignificatifs - 1)
```

```
            {
                decompositionPartieDecimale[i] =
                    int(round(float(decompositionPartieDecimale[i]) +
                    float(decompositionPartieDecimale[i + 1]) / 10));
            }

            resultat += decompositionPartieDecimale[i] * pow(10, - i - 1);
        }
    }

    else
    {
        for (int i = 0; i < min((int)decompositionPartieEntiere.size(),
            nbrChiffresSignificatifs); i++)
        {
            if (i == nbrChiffresSignificatifs - 1 && (int)decompositionPartieEntiere.size() !=
                nbrChiffresSignificatifs)
            {
                decompositionPartieEntiere[(int)decompositionPartieEntiere.size() - i - 1] =
                    int(round(float(decompositionPartieEntiere[(int)decompositionPartieEntiere.size()
                    - i - 1]) +
                    float(decompositionPartieEntiere[(int)decompositionPartieEntiere.size() -
                    i - 2]) / 10));
            }

            resultat += decompositionPartieEntiere[(int)decompositionPartieEntiere.size() - i
                - 1] * pow(10, (int)decompositionPartieEntiere.size() - i - 1);
        }

        for (int i = 0; i < min((int)decompositionPartieDecimale.size(),
            nbrChiffresSignificatifs - (int)decompositionPartieEntiere.size()); i++)
        {
            if (i + min((int)decompositionPartieEntiere.size(), nbrChiffresSignificatifs) ==
                nbrChiffresSignificatifs - 1)
            {
                decompositionPartieDecimale[i] =
                    int(round(float(decompositionPartieDecimale[i]) +
                    float(decompositionPartieDecimale[i + 1]) / 10));
            }

            resultat += decompositionPartieDecimale[i] * pow(10, - i - 1);
        }
    }

    cout << "Nombre approximé : " << resultat << endl;

    return 0;
}
```

7.24 Un cercle et des polygones

Le but de cet exercice est d'approximer π en mesurant la longueur de polygones réguliers. Un polygone régulier est un polygone inscrit dans un cercle et dont les côtés sont de la même longueur. En construisant des polygones réguliers dont le nombre de côtés est de plus en plus grand, on peut approximer la circonférence d'un cercle ($2\pi r$) par le périmètre du polygone. Pour un polygone régulier à n côtés

inscrit dans le cercle de rayon 1, il est possible de montrer que la longueur d'un côté vaut :

$$l = \sqrt{\left(\cos\left(\frac{2\pi}{n}\right) - 1\right)^2 + \sin\left(\frac{2\pi}{n}\right)^2}$$

Écrire une fonction pour approximer la valeur de π par un polygone régulier à n côtés. Pour imiter le cas pratique où l'on mesure à la règle la longueur d'un côté, limiter la valeur de la racine carrée à 3 chiffres significatifs. Prendre pour valeur de π 3,14159. *Même si une valeur de π vous est donnée, elle n'est là que pour imiter la mesure avec une règle.*

Correction :

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

double piParPolygone(int nbrCotes)
{
    const double piApprox = 3.14159;

    return nbrCotes * 0.5 * sqrt( (cos(2 * piApprox / nbrCotes) - 1) * (cos(2 * piApprox /
        nbrCotes) - 1) + sin(2 * piApprox / nbrCotes) * sin(2 * piApprox / nbrCotes));
}

int main()
{
    cout << "Approximation de pi : " << piParPolygone(10);

    return 0;
}
```

7.25 Céline et les diviseurs

Pour un exercice de mathématiques, Céline a besoin de connaître certains diviseurs de 1922310, mais celui-ci en a 96 ! Aider Céline a trouver tous les diviseurs de 1922310 en les stockant progressivement dans un vecteur.

Correction :

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

int main()
{
    vector<int> diviseurs;

    cout << "Liste des diviseurs de " << 1922310 << " :\n";

    for (int i = 1; i <= 1922310; i++)
    {
        if (1922310 % i == 0)
        {
            diviseurs.push_back(i);
```

```
            cout << ">>> " << i << "\n";
        }
    }

    return 0;
}
```

7.26 De l'imagination, il en faut !

Là où les nombres "classiques" peuvent être vus comme des points sur une droite, les nombres complexes peuvent être vus comme des points sur un plan. Les nombres complexes ont aussi la particularité d'assurer la solution de certaines équations polynomiales, notamment $x^2 + 1 = 0$. On note i le nombre tel que $i^2 = -1$. Pour deux nombres complexes $z_1 = a + ib, z_2 = x + iy$ avec a, b, x, y des nombres "classiques", les opérations usuelles sont définies comme suit :

$$\begin{aligned} z_1 + z_2 &= (a + x) + i(b + y) \\ z_1 \times z_2 &= (ax - by) + i(bx + ay) \\ \frac{z_1}{z_2} &= \frac{ax + by}{x^2 + y^2} + i\frac{bx - ay}{x^2 + y^2} \end{aligned}$$

Écrire une classe NombreComplexe permettant d'encapsuler deux *float*. Écrire ensuite des fonctions AdditionComplexe, MultiplicationComplexe et DivisionComplexe prenant en paramètre deux objets NombreComplexe et renvoyant un nombre complexe. Vérifier que $\left(\frac{1+i\sqrt{3}}{2}\right)^2 = -\frac{1+i\sqrt{3}}{2}$ et que $\frac{1+i}{1-i} = i$.

Correction :

```
// complexe.h

class NombreComplexe
{
    double partieRelle;
    double partieImaginaire;

public:
    NombreComplexe();
    ~NombreComplexe();

    double getPartieRelle();
    double getPartieImaginaire();

    void setPartieReelle(double);
    void setPartieImaginaire(double);
};

NombreComplexe additionComplexe(NombreComplexe, NombreComplexe);

NombreComplexe multiplicationComplexe(NombreComplexe, NombreComplexe);

NombreComplexe divisionComplexe(NombreComplexe, NombreComplexe);
```

```
// complexe.cpp
```

```
#include "complexe.h"

NombreComplexe::NombreComplexe() { }

NombreComplexe::~NombreComplexe() { }

double NombreComplexe::getPartieRelle()
{
    return partieRelle;
}

double NombreComplexe::getPartieImaginaire()
{
    return partieImaginaire;
}

void NombreComplexe::setPartieReelle(double PartieReelle)
{
    partieRelle = PartieReelle;
}

void NombreComplexe::setPartieImaginaire(double PartieImaginaire)
{
    partieImaginaire = PartieImaginaire;
}

NombreComplexe additionComplexe(NombreComplexe z, NombreComplexe w)
{
    NombreComplexe v;

    v.setPartieReelle(z.getPartieRelle() + w.getPartieRelle());
    v.setPartieImaginaire(z.getPartieImaginaire() + w.getPartieImaginaire());

    return v;
}

NombreComplexe multiplicationComplexe(NombreComplexe z, NombreComplexe w)
{
    NombreComplexe v;

    v.setPartieReelle(z.getPartieRelle() * w.getPartieRelle() - z.getPartieImaginaire() *
        w.getPartieImaginaire());
    v.setPartieImaginaire(z.getPartieImaginaire() * w.getPartieRelle() + z.getPartieRelle() *
        w.getPartieImaginaire());

    return v;
}

NombreComplexe divisionComplexe(NombreComplexe z, NombreComplexe w)
{
    NombreComplexe v;

    v.setPartieReelle((z.getPartieRelle() * w.getPartieRelle() + z.getPartieImaginaire() *
        w.getPartieImaginaire()) / (w.getPartieRelle() * w.getPartieRelle() +
        w.getPartieImaginaire() * w.getPartieImaginaire()));
    v.setPartieImaginaire((z.getPartieImaginaire() * w.getPartieRelle() - z.getPartieRelle()
```

```
        * w.getPartieImaginaire()) / (w.getPartieRelle() * w.getPartieRelle() +
        w.getPartieImaginaire() * w.getPartieImaginaire()));

    return v;
}
```

```
// main.cpp

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

#include "complexe.h"

int main()
{
    NombreComplexe z;
    NombreComplexe w;

    z.setPartieReelle(0.5);
    z.setPartieImaginaire(sqrt(3) / 2);

    cout << "(" << z.getPartieRelle() << " + " << z.getPartieImaginaire() << "i)^2 = " <<
        multiplicationComplexe(z, z).getPartieRelle() << " + " << multiplicationComplexe(z,
        z).getPartieImaginaire() << "i" << endl;

    z.setPartieReelle(1);
    z.setPartieImaginaire(1);

    w.setPartieReelle(1);
    w.setPartieImaginaire(- 1);

    cout << "(" << z.getPartieRelle() << " + " << z.getPartieImaginaire() << "i) / " << "("
        << w.getPartieRelle() << " + " << w.getPartieImaginaire() << "i)" << " = " <<
        divisionComplexe(z, w).getPartieRelle() << " + " << divisionComplexe(z,
        w).getPartieImaginaire() << "i" << endl;

    return 0;
}
```

7.27 Jeu 2

Le jeu *Mastermind* consiste à déterminer une série de 4 couleurs différentes (et dans l'ordre) parmi 8 en faisant le moins de tentatives possibles. À chaque essai, le joueur teste une série de couleurs et la machine lui fournit le nombre de couleurs mal placées et le nombre de couleurs bien placées.
Écrire un programme permettant de jouer au *Mastermind*.

Correction :

```
// mastermind.h

enum Couleur
{
    Rouge,
```

```
    Vert,
    Bleu,
    Jaune,
    Gris,
    Blanc,
    Rose,
    Orange,

    Rien
};

class Mastermind
{
    Couleur code[4];
    Couleur essai[4];

    int tentatives;

public:
    Mastermind();
    ~Mastermind();

    int getBienPlace();
    int getMalPlace();
    int getTentatives();

    void nouvelEssai(Couleur[4]);

    void genererCode();

    void afficherCode();
};
```

```
// mastermind.cpp

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

#include "mastermind.h"

Mastermind::Mastermind()
{
    tentatives = 0;

    code[0] = Rien;
    code[1] = Rien;
    code[2] = Rien;
    code[3] = Rien;

    essai[0] = Rien;
    essai[1] = Rien;
    essai[2] = Rien;
    essai[3] = Rien;
```

```
}

Mastermind::~Mastermind() { }

int Mastermind::getBienPlace()
{
    int compteur = 0;

    for (int i = 0; i < 4; i++)
    {
        if (code[i] == essai[i])
        {
            compteur++;
        }
    }

    return compteur;
}

int Mastermind::getMalPlace()
{
    int compteur = 0;

    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (code[i] == essai[j] && i != j)
            {
                compteur++;
            }
        }
    }

    return compteur;
}

int Mastermind::getTentatives()
{
    return tentatives;
}

void Mastermind::nouvelEssai(Couleur Essai[4])
{
    essai[0] = Essai[0];
    essai[1] = Essai[1];
    essai[2] = Essai[2];
    essai[3] = Essai[3];

    tentatives++;
}

void Mastermind::genererCode()
{
    int couleur = 0;
    int compteur = 0;
```

```
    bool dejaPris[8] = {false, false, false, false, false, false, false, false};

    while (compteur < 4)
    {
        couleur = rand() % 8;

        if (!dejaPris[couleur])
        {
            dejaPris[couleur] = true;

            switch (couleur)
            {
            case 0:
                code[compteur] = Rouge;
                break;

            case 1:
                code[compteur] = Vert;
                break;

            case 2:
                code[compteur] = Bleu;
                break;

            case 3:
                code[compteur] = Jaune;
                break;

            case 4:
                code[compteur] = Gris;
                break;

            case 5:
                code[compteur] = Blanc;
                break;

            case 6:
                code[compteur] = Rose;
                break;

            case 7:
                code[compteur] = Orange;
                break;

            default:
                code[compteur] = Rien;
                break;
            }

            compteur++;
        }
    }
}

void Mastermind::afficherCode()
```

```
{
    for (int i = 0; i < 4; i++)
    {
        switch (code[i])
        {
        case Rouge:
            cout << "Couleur " << i + 1 << " : rouge.\n";
            break;

        case Vert:
            cout << "Couleur " << i + 1 << " : vert.\n";
            break;

        case Bleu:
            cout << "Couleur " << i + 1 << " : bleu.\n";
            break;

        case Jaune:
            cout << "Couleur " << i + 1 << " : jaune.\n";
            break;

        case Gris:
            cout << "Couleur " << i + 1 << " : gris.\n";
            break;

        case Blanc:
            cout << "Couleur " << i + 1 << " : blanc.\n";
            break;

        case Rose:
            cout << "Couleur " << i + 1 << " : rose.\n";
            break;

        case Orange:
            cout << "Couleur " << i + 1 << " : orange.\n";
            break;

        default:
            break;
        }
    }
}
```

```
// main.cpp

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

#include "mastermind.h"

int main()
{
    srand(time(0));
```

```
Mastermind jeu;
jeu.genererCode();

Couleur essai[4] = {Couleur::Rien, Couleur::Rien, Couleur::Rien, Couleur::Rien};

cout << "Conversion chiffre -> couleur :\n0 -> Rouge\n1 -> Vert\n2 -> Bleu\n3 -> Jaune\n4
    -> Gris\n5 -> Blanc\n6 -> Rose\n7 -> Orange\n8 -> Rien\n\n";

int couleur;

while (jeu.getBienPlace() != 4)
{
    cout << "Il y a " << jeu.getBienPlace() << " couleur(s) bien placée(s) et " <<
        jeu.getMalPlace() << " couleur(s) mal placée(s).\n";
    cout << "Début de la tentative #" << jeu.getTentatives() + 1 << " :\n";

    for (int i = 0; i < 4; i++)
    {
        cout << "Couleur #" << i + 1 << " :\n>>> ";
        cin >> couleur;

        switch (couleur)
        {
        case 0:
            essai[i] = Rouge;
            break;

        case 1:
            essai[i] = Vert;
            break;

        case 2:
            essai[i] = Bleu;
            break;

        case 3:
            essai[i] = Jaune;
            break;

        case 4:
            essai[i] = Gris;
            break;

        case 5:
            essai[i] = Blanc;
            break;

        case 6:
            essai[i] = Rose;
            break;

        case 7:
            essai[i] = Orange;
            break;
```

```
            default:
                essai[i] = Rien;
                break;
            }
        }

        jeu.nouvelEssai(essai);
    }

    return 0;
}
```

7.28 Une machine dans tous ses états !

Se référer à l'exercice 2.9 pour l'utilisation de nombres aléatoires suivant une distribution normale. Une machine à états est un algorithme pouvant fonctionner en boucle fermée (sans intervention humaine) et produisant des résultats différents en fonction d'un état mesuré.
Concevoir une machine à états fonctionnant pendant 100 itérations et générant à chaque fois une mesure aléatoire suivant une loi normale de paramètres 3 et 2.

1. Si la mesure est inférieure à 2, la machine doit "scanner"
2. Si la mesure est entre 2 et 5, la machine doit "actionner"
3. Si la mesure est entre 5 et 9, la machine doit "bouger"

Si la mesure est supérieure à 9, la machine doit s'arrêter prématurément.

Correction :

```
#include <random>
#include <ctime>
#include <iostream>
using namespace std;

void machineAEtats()
{
    random_device systeme{};
    mt19937 generateur{systeme()};
    normal_distribution<float> distribution{3.0, 2.0};

    float mesure;

    for (int iteration = 0; iteration < 100; iteration++)
    {
        mesure = distribution(generateur);

        if (mesure < 2)
        {
            cout << "Iteration #" << iteration + 1 << " : scanner.\n";
        }

        else if (mesure < 5)
        {
            cout << "Iteration #" << iteration + 1 << " : actionner.\n";
        }
```

```
        else if (mesure < 9)
        {
            cout << "Iteration #" << iteration + 1 << " : bouger.\n";
        }

        else
        {
            cout << "Arrêt prématuré.";

            break;
        }
    }
}

int main()
{
    machineAEtats();

    return 0;
}
```

7.29 Permutations

Écrire un programme générant toutes les permutations de deux éléments d'un tableau à un nombre fixe éléments.

Correction :

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

int main()
{
    int tableau[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int temp;

    int compteur = 1;

    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            temp = tableau[j];

            tableau[j] = tableau[i];
            tableau[i] = temp;

            cout << "Permutation #" << compteur << " :\n";

            for (int k = 0; k < 10; k++)
            {
                cout << "|" << tableau[k];
            }

            cout << "|\n\n";
```

```
            temp = tableau[i];

            tableau[i] = tableau[j];
            tableau[j] = temp;

            compteur++;
        }
    }

    return 0;
}
```

7.30 Écart type

L'écart type est un nombre mesurant la dispersion des valeurs d'une série statistique par rapport à sa moyenne. L'écart type, noté σ se mesure ainsi :

$$\sigma = \sqrt{\frac{S}{n} - m^2}$$

avec m la moyenne de la série et S la somme des carrés des valeurs de la série. Écrire un programme mesurant la dispersion d'une série statistique stockée dans un vecteur. Quel est l'écart type de la série de l'exercice 7.7 ?

Correction :

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

int main()
{
    vector<float> serie;

    int nbrDonnees = 0;
    float nbr;

    cout << "Nombre de données ârentrer : ";
    cin >> nbrDonnees;

    for (int i = 0; i < nbrDonnees; i++)
    {
        cout << ">>> ";
        cin >> nbr;

        serie.push_back(nbr);
    }

    float somme = 0;
    float sommeCarres = 0;

    for (int i = 0; i < serie.size(); i++)
    {
```

```
        somme += serie.at(i);
        sommeCarres += serie.at(i) * serie.at(i);
    }

    cout << "Écart-type : " << sqrt(sommeCarres / serie.size() - somme * somme /
        (serie.size() * serie.size()));

    return 0;
}
```

7.31 Décalage d'éléments 1

On peut assimiler un vecteur de 9 éléments à un tableau 3x3, où les 3 premiers éléments sont sur la première ligne, les 3 suivants sur la deuxième, etc... Écrire une fonction permettant de décaller d'un certain nombre de crans les éléments d'une ligne ou d'une colonne d'un tableau 3x3. Par exemple, décaler les éléments de la première ligne de 2 devrait donner :

$$\begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{array} \longrightarrow \begin{array}{ccc} 2 & 3 & 1 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{array}$$

Correction :

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

enum Type
{
    Ligne,
    Colonne
};

vector<int> permuter(vector<int> vecteur, Type type, int position, int decalage)
{
    int temp;

    switch(type)
    {
    case Ligne:
        temp = vecteur.at((position % 3) * 3 + (2 * decalage) % 3);

        vecteur.at((position % 3) * 3 + (2 * decalage) % 3) = vecteur.at((position % 3) * 3 +
            decalage % 3);
        vecteur.at((position % 3) * 3 + decalage % 3) = vecteur.at((position % 3) * 3);
        vecteur.at((position % 3) * 3) = temp;

        break;
```

```
    case Colonne:
        temp = vecteur.at((position % 3 + 6 * decalage) % 9);

        vecteur.at((position % 3 + 6 * decalage) % 9) = vecteur.at((position % 3 + 3 *
            decalage) % 9);
        vecteur.at((position % 3 + 3 * decalage) % 9) = vecteur.at(position % 3);
        vecteur.at(position % 3) = temp;

        break;

    default:
        break;
    }

    return vecteur;
}

int main()
{
    vector<int> vecteur = {1, 2, 3, 4, 5, 6, 7, 8, 9};

    for (int i = 0; i < 3; i++)
    {
        cout << "|" << vecteur.at(3 * i) << "|" << vecteur.at(3 * i + 1) << "|" <<
            vecteur.at(3 * i + 2) << "|\n";
    }

    cout << endl;

    vecteur = permuter(vecteur, Type::Ligne, 0, 2);

    for (int i = 0; i < 3; i++)
    {
        cout << "|" << vecteur.at(3 * i) << "|" << vecteur.at(3 * i + 1) << "|" <<
            vecteur.at(3 * i + 2) << "|\n";
    }

    return 0;
}
```

7.32 Décalage d'éléments 2

Faire le même programme que l'exercice précédent en utilisant un tableau à deux dimensions au lieu d'un vecteur et une seule fonction au lieu de deux. Indice : il est possible de passer en paramètres un tableau 3x3.

Correction :

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

enum Type
{
    Ligne,
    Colonne
```

```
};

void permuter(int tableau[3][3], Type type, int position, int decalage)
{
    int temp;
    int bonDecalage = 0;

    switch (decalage % 3)
    {
    case 1:
        bonDecalage = 2;
        break;

    case 2:
        bonDecalage = 1;
        break;

    default:
        break;
    }

    switch(type)
    {
    case Ligne:
        temp = tableau[position][0];

        tableau[position][0] = tableau[position][bonDecalage];
        tableau[position][bonDecalage] = tableau[position][(2 * bonDecalage) % 3];
        tableau[position][(2 * bonDecalage) % 3] = temp;

        break;

    case Colonne:
        temp = tableau[0][position];

        tableau[0][position] = tableau[bonDecalage][position];
        tableau[bonDecalage][position] = tableau[(2 * bonDecalage) % 3][position];
        tableau[(2 * bonDecalage) % 3][position] = temp;
        break;

    default:
        break;
    }
}

int main()
{
    int tableau[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

    for (int i = 0; i < 3; i++)
    {
        cout << "|";

        for (int j = 0; j < 3; j++)
        {
            cout << tableau[i][j] << "|";
```

```
        }

        cout << endl;
    }

    permuter(tableau, Type::Ligne, 0, 0);

    cout << "\n\n";

    for (int i = 0; i < 3; i++)
    {
        cout << "|";

        for (int j = 0; j < 3; j++)
        {
            cout << tableau[i][j] << "|";
        }

        cout << endl;
    }

    return 0;
}
```

7.33 Interpolation de Lagrange

L'interpolation de Lagrange [24] est une manière de faire passer une courbe polynomiale par des points choisis. Pour 3 couples de points (x_1, y_1), (x_2, y_2), (x_3, y_3), l'équation de la courbe est donnée par :

$$f(x) = y_1 \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} + y_2 \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} + y_3 \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}$$

Écrire une fonction donnant la courbe d'interpolation de 3 couples de points. Déterminer l'image de 4 pour les points suivants : $(0, 1)$, $(3, 1)$, $(-2, 4)$.

Correction :

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

float interpolationDegre3(vector<float> x, vector<float> y, float point)
{
    float image = 0;

    for (int i = 0; i < x.size(); i++)
    {
        image += y.at(i) * (point - x.at((i + 1) % (int)x.size())) * (point - x.at((i + 2) %
            (int)x.size())) / ((x.at(i % (int)x.size()) - - x.at((i + 1) % (int)x.size())) *
            (x.at(i % (int)x.size()) - - x.at((i + 2) % (int)x.size())));
    }

    return image;
```

24. Voir https ://fr.wikipedia.org/wiki/Interpolation_lagrangienne

```
}

int main()
{
    vector<float> x = {0, 3, -2};
    vector<float> y = {1, 1, 4};

    cout << "L'image de 4 est : " << interpolationDegre3(x, y, 4);

    return 0;
}
```

7.34 Modèle SIR discret

Le modèle SIR (Sains, Infectés, Rétablis) est un modèle mathématique de propagation d'une maladie dans une population. C'est un modèle à compartiments [25] dont on peut facilement lui rajouter ou enlever des catégories. Utiliser le modèle suivant pour écrire une fonction donnant le nombre d'individus sains, infectés et rétablis jour par jour :

1. Chaque individu sain a une probabilité de devenir un nouvel infecté de $\frac{9,437I}{P}\tau$ avec I le nombre d'infectés, P la population totale et τ la probabilité de transmission de la maladie
2. Chaque individu infecté (mais pas nouvel infecté) a une probabilité de guérison γ
3. Les individus rétablis ne peuvent pas être réinfectés
4. À chaque fin de journée, les nouveaux infectés deviennent des infectés "classiques" (ils peuvent à leur tour transmettre la maladie)

Utiliser ce modèle pour prédire l'évolution d'une maladie dans une population de 100000 habitants dont un infecté, avec une probabilité de transmission de $0,25$ et une probabilité de guérison de $0,1$.

Correction :

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

void sir(int population, int infectes, double tauxTransmission, double tauxGuerison)
{
    int sains = population - infectes;
    int retablis = 0;

    int nouveauxInfectes;

    int jour = 1;

    while (infectes > 0)
    {
        nouveauxInfectes = 0;

        for (int i = 0; i < sains; i++)
        {
            if (((double) rand() / (RAND_MAX)) < tauxTransmission * double(9.437 * infectes) /
                double(population))
```

25. Voir https ://fr.wikipedia.org/wiki/Mod%C3%A8les_compartimentaux_en_%C3%A9pid%C3%A9miologie

```
            {
                nouveauxInfectes++;
            }
        }

        sains -= nouveauxInfectes;

        for (int i = 0; i < infectes; i++)
        {
            if (((double) rand() / (RAND_MAX)) < tauxGuerison)
            {
                infectes--;
                retablis++;
            }
        }

        infectes += nouveauxInfectes;

        cout << "Jour " << jour << " : S = " << sains << " / I = " << infectes << " / R = "
             << retablis << endl;

        jour++;
    }
}

int main()
{
    srand(time(0));

    sir(100000, 1, 0.25, 0.1);

    return 0;
}
```

7.35 Prédire le chaos ?

Le mouvement Brownien est un type de mouvement aléatoire décrivant à la base le comportement des particules de l'air. Ce modèle peut aussi être utilisé dans d'autres domaines, notamment en finance où il est utiliser pour prédire les futures tendances boursières. Le programme suivant contient une fonction qui tente de prédire le coût de l'action de Nvidia au cours des 6 prochains mois (seulement les journées de travail) : [26]

```
#include <iostream>
#include <vector>
#include <cmath>
#include <random>
#include <ctime>
using namespace std;

vector<float> mouvementBrownienGeometrique(float mu, float sigma, float y0, float
    intervalle, float tempsMax)
{
```

26. Le prix de base de l'action se base sur son prix atteint le 30 avril 2024 à l'ouverture. La volatilité historique a été déterminée sur la période du 23 avril 2023 au 23 avril 2024.

```
    int iterations = int(tempsMax / intervalle);

    mt19937 generateur;
    generateur.seed(time(0));
    normal_distribution<float> gauss(0, sqrt(intervalle));

    vector<float> valeurs;
    valeurs.push_back(std::exp((mu - sigma * sigma / 2) * intervalle + sigma *
        gauss(generateur)));

    for (int i = 0; i < iterations - 1; i++)
    {
        valeurs.push_back(valeurs.at(i) * exp((mu - sigma * sigma / 2) * intervalle + sigma *
            gauss(generateur)));
    }

    for (int i = 0; i < iterations; i++)
    {
        valeurs.at(i) *= y0;
    }

    return valeurs;
}

int main()
{
    vector<float> valeurs = mouvementBrownienGeometrique(0.12, 0.41, 871.2, 0.000403, 0.05);

    for (int i = 0; i < valeurs.size(); i++)
    {
        cout << valeurs.at(i) << endl;
    }

    return 0;
}
```

En utilisant ce programme, faire une moyenne du prix de l'action de Nvidia après 6 mois sur 20, 100 et 1000 simulations.

Correction :

```
#include <iostream>
#include <cmath>
#include <random>
#include <vector>
#include <ctime>
using namespace std;

vector<float> mouvementBrownienGeometrique(float mu, float sigma, float y0, float
    intervalle, float tempsMax)
{
    int iterations = int(tempsMax / intervalle);

    mt19937 generateur;
    generateur.seed(time(0));
    normal_distribution<float> gauss(0, sqrt(intervalle));
```

```
    vector<float> valeurs;
    valeurs.push_back(std::exp((mu - sigma * sigma / 2) * intervalle + sigma *
        gauss(generateur)));

    for (int i = 0; i < iterations - 1; i++)
    {
        valeurs.push_back(valeurs.at(i) * std::exp((mu - sigma * sigma / 2) * intervalle +
            sigma * gauss(generateur)));
    }

    for (int i = 0; i < iterations; i++)
    {
        valeurs.at(i) *= y0;
    }

    return valeurs;
}

int main()
{
    vector<float> valeurs;

    int nbrSimulations = 1000;
    double somme = 0;

    for (int i = 0; i < nbrSimulations; i++)
    {
        valeurs = mouvementBrownienGeometrique(0.12, 0.41, 871.2, 0.000403, 0.05);
        somme += valeurs.at(valeurs.size() - 1);
    }

    cout << "Moyenne du prix de l'action de Nvdia après 6 mois sur " << nbrSimulations << "
        simulations : " << somme / nbrSimulations;

    return 0;
}
```

7.36 Chiffrement de Vigenère

Le chiffrement de Vigenère est une méthode de chiffrement un peu plus robuste que celui par décalage. Dans ce cas, les lettres du messages sont décalées d'un nombre différents à chaque fois. Ce nombre est déterminé par un mot, appelé clé du message. Par exemple, pour chiffrer le message "persona est une serie de jeux videos japonais" avec la clé "robotique", on procède de la manière suivante :

Persona est une serie de jeux videos japonais
robotiq uer obo tique ro boti querob otiquero

Le nombre de décalage est déterminé par la position dans l'alphabet de la lettre de la clé associée. Le message chiffré devient alors :

persona est une serie de jeux videos japonais
robotiq uer obo tique ro boti querob otiquero
gssghvq ywk ios lmhci us ksnf lchvct xtxehezg

Écrire un programme chiffrant un message par selon cette méthode. Indice : il peut être utile de chercher une méthode pour récupérer la taille d'une chaîne de caractères. Également, utiliser la méthode *getline* est une meilleure idée pour récupérer une entrée contenant des espaces.

Correction :

```
#include <iostream>
#include <cstring>
#include <vector>
using namespace std;

int main()
{
    string alphabet = "abcdefghijklmnopqrstuvwxyz";
    string message, cle;

    cout << "Message a chiffrer (en minuscules) : \n>>> ";
    getline(cin, message);
    cout << "Cle de chiffrement :\n>>> ";
    getline(cin, cle);

    vector<int> positionsCle;
    vector<int> positionsMessage;

    int nbrEspace = 0;

    for (int i = 0; i < cle.length(); i++)
    {
        for (int j = 0; j < alphabet.length(); j++)
        {
            if (cle[i] == alphabet[j])
            {
                positionsCle.push_back(j);
                break;
            }
        }
    }

    for (int i = 0; i < message.length(); i++)
    {
        for (int j = 0; j < alphabet.length(); j++)
        {
            if (message[i] == alphabet[j])
            {
                positionsMessage.push_back(j);
                break;
            }
        }

        if (message[i] == ' ')
        {
            positionsMessage.push_back(-1);
        }
    }

    for (int i = 0; i < message.length(); i++)
```

```
    {
        if (positionsMessage[i] != -1)
        {
            message[i] = alphabet[(positionsMessage[i] + positionsCle[(i - nbrEspace) %
                cle.length()]) % 26];
        }

        else
        {
            nbrEspace++;
        }
    }

    cout << "Message chiffre : \n >>> " << message;

    return 0;
}
```

7.37 « Bonjour-hi »

Écrire un programme agissant comme un caissier : en fonction du prix d'achat d'un produit et de l'argent remis par le client, il doit remettre de la monnaie avec le moins de coupures possibles. Les coupures possibles sont 100$, 50$, 20$, 10$, 5$, 2$, 1$, 25 cents, 10 cents et 5 cents. On suppose que tous les prix d'achat sont une combinaison de ces coupures.

Correction :

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

int main()
{
    float coupures[10] = {100, 50, 20, 10, 5, 2, 1, 0.25, 0.1, 0.05};
    int coupuresRendues[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

    float prixAchat;
    float sommeDonnee;

    cout << "Entrer le prix d'achat du produit :\n>>> ";
    cin >> prixAchat;

    cout << "Entrer la somme donnée par le client :\n>>> ";
    cin >> sommeDonnee;

    if (prixAchat < 0 || sommeDonnee < prixAchat)
    {
        cout << "Au moins une des variables n'a pas une valeur possible.";

        return 0;
    }

    sommeDonnee -= prixAchat;

    for (int i = 0; i < sizeof(coupures) / sizeof(float); i++)
    {
```

```
        while (sommeDonnee > 0)
        {
            if (sommeDonnee - coupures[i] > 0)
            {
                coupuresRendues[i]++;

                sommeDonnee -= coupures[i];
            }

            else
            {
                break;
            }
        }
    }

    cout << "Coupures rendues :\n";

    for (int i = 0; i < sizeof(coupures) / sizeof(float); i++)
    {
        if (coupuresRendues[i] != 0)
        {
            cout << coupuresRendues[i] << "x " << coupures[i] << "$ ";
        }
    }

    return 0;
}
```

7.38 Ça monte et ça descend !

Un circuit LC série est un ciruit composé d'un condensateur et d'une bobine branchés en série. Ce type de circuit a de nombreuses applications, notamment dans la réalisation de filtres. La tension du courant dans ce genre de circuit oscille entre deux valeurs pérdiodiquement et ce, selon une équation différentielle (l'inconnue est une fonction). Par des techniques d'analyse numérique, on peut montrer qu'une bonne approximation de la tension est donnée par :

$$u_{n+2} = 2u_{n+1} - u_n + h^2 \frac{E - u_n}{LC}$$

avec u_{n+2}, u_{n+1} et u_n les approximations de la tension aux temps (en secondes) $(n+2)h$, $(n+1)h$ et nh, h le pas entre chaque approximation, E la tension aux bornes d'un régulateur de voltage, C la capacité du condensateur et L l'inductance de la bobine.
Écrire un programme stockant dans un vecteur les approximations de la tension jusqu'au temps $t = 10s$ avec un pas de $0,0005$ et $u_0 = 0$, $u_1 = 0,00005$. Les valeurs du circuit sont les suivantes : $E = 12V$, $L = 0,5H$, $C = 0,05F$. Sachant que la tension est sensée osciller entre 0 et $24V$, l'approximation est-elle bonne (déterminer le maximum / minimum) ? Essayer différente valeur de h pour trouver une meilleure approximation.

Correction :

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

```cpp
int main()
{
    double pas = 0.00001;
    double t0 = 0;
    double y0 = 0;
    double y0Prime = 0.5;

    double E = 12;
    double L = 0.5;
    double C = 0.05;

    double tMax = 10;

    vector<double> tension = {y0, y0Prime * pas + y0};

    for (int i = 0; i < int(tMax / pas) - 2; i++)
    {
        tension.push_back(2 * tension.at(i + 1) - tension.at(i) + pas * pas * (E -
            tension.at(i)) / (L * C));
    }

    double min = tension.at(0);
    double max = tension.at(0);

    for (int i = 0; i < tension.size(); i++)
    {
        if (tension.at(i) > max)
        {
            max = tension.at(i);
        }

        if (tension.at(i) < min)
        {
            min = tension.at(i);
        }
    }

    cout << "Minimum de la tension : " << min << " maximum de la tension : " << max;

    return 0;
}
```