

# Correction des exercices

Document rédigé par Raphaël et André

---

## Table des matières

<b>1</b>	<b>Syntaxe, variables et I/O</b>	<b>4</b>
1.1	Explications 1 . . . . .	4
1.2	Erreurs 1 . . . . .	4
1.3	Complétions 1 . . . . .	4
1.4	Alice et Bob 1 . . . . .	5
1.5	Triple d'un nombre . . . . .	5
1.6	Aire d'un cercle . . . . .	6
1.7	Une expression... compliquée! . . . . .	6
1.8	Racines carrées . . . . .	7
1.9	Puissances . . . . .	7
1.10	Un peu de binaire . . . . .	8
<b>2</b>	<b>Structures conditionnelles et boucles</b>	<b>9</b>
2.1	Explications 2 . . . . .	9
2.2	Erreurs 2 . . . . .	11
2.3	Complétions 2 . . . . .	13
2.4	Du tableau à la boucle . . . . .	14
2.5	Parité . . . . .	14
2.6	Les premiers nombres premiers . . . . .	15
2.7	Stan Robotix 1 . . . . .	15
2.8	La meilleure des structures conditionnelles . . . . .	17
2.9	Nombres aléatoires 1 . . . . .	19
2.10	Nombres aléatoires 2 . . . . .	20
<b>3</b>	<b>Vecteurs, tableaux et pointeurs</b>	<b>23</b>
3.1	Tableau ou vecteur? . . . . .	23
3.2	À couvert! . . . . .	23
3.3	Une racine carrée... compliquée! . . . . .	24
3.4	Prédictions 2 . . . . .	24
3.5	Alice et Bob 2 . . . . .	24
3.6	Un vecteur bien long . . . . .	25
3.7	Stan Robotix 2 . . . . .	26
3.8	Céline et les statistiques . . . . .	27
3.9	Un vecteur bien long... mais sans pairs! . . . . .	28
<b>4</b>	<b>Fonctions</b>	<b>29</b>
4.1	Le meilleur des types de retour . . . . .	29
4.2	Une fonction... spéciale! . . . . .	29
4.3	Le plus petit des multiples . . . . .	30
4.4	Mes carrés! . . . . .	30
4.5	Céline et le jeu . . . . .	31

4.6	In french please ! . . . . .	32
4.7	Prédictions 2 . . . . .	32
4.8	Nombres aléatoires 3 . . . . .	33
4.9	SohCahToa ! . . . . .	34
4.10	Centaines, dizaines et unités . . . . .	34
4.11	Stan Robotix 3 . . . . .	35
<b>5</b>	<b>Classes</b>	<b>38</b>
5.1	Définitions . . . . .	38
5.2	Objet construit ! . . . . .	38
5.3	Le retour de la racine carrée perdue . . . . .	39
5.4	Triangle . . . . .	39
5.5	Cercle . . . . .	40
5.6	Le fameux hérisson . . . . .	41
5.7	Une architecture classique . . . . .	42
5.8	Jeu . . . . .	42
5.9	Simulation . . . . .	45
5.10	Prédictions 3 . . . . .	47
<b>6</b>	<b>Git et Github</b>	<b>49</b>
6.1	Des commandes et des explications . . . . .	49
6.2	Entraînement . . . . .	49
<b>7</b>	<b>Exercices supplémentaires</b>	<b>50</b>
7.1	Nombres aléatoires 4 . . . . .	50
7.2	Alice et Bob 3 . . . . .	50
7.3	Trie par sélection . . . . .	51
7.4	Diviser pour mieux estimer . . . . .	52
7.5	C'est plus ! C'est moins ! . . . . .	53
7.6	Alors, c'est qui le plus malin ? . . . . .	54
7.7	Stan Robotix 4 . . . . .	55
7.8	Genshin Impact et les chaînes de caractères . . . . .	56
7.9	Stan Robotix et les chaînes de caractères . . . . .	58
7.10	Neurone artificiel . . . . .	60
7.11	En lettres s'il vous plait ! . . . . .	62
7.12	En chiffres s'il vous plait ! . . . . .	64
7.13	Chiffrement par décalage . . . . .	64
7.14	Stan Robotix 5 . . . . .	65
7.15	Brute Force . . . . .	66
7.16	Alice et Bob 4 . . . . .	67
7.17	La pyramide d'astérisques . . . . .	68
7.18	Un grapheur de base . . . . .	69



# 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 brièvement ce qu'il fait.

**Correction :**

- Variable : age (nombre entier)  
Fonctionnement : le programme demande l'âge à l'utilisateur et affiche l'âge rentré.
- Variable : nombre (nombre réel / flottant)  
Fonctionnement : le programme demande un nombre réel à l'utilisateur en entrée et affiche son carré.
- 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 :**

- L'opérateur ">>" (ligne 6) n'est pas celui qui doit être placé après *cout*. Il faut le remplacer par "<<".
- Il manque un ";" à la ligne 13.
- La variable *a* ne peut pas être modifiée (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<sup>1</sup>.

**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.

---

```
#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 à l'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 :

---

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

// On cherche à calculer et afficher l'aire d'un triangle
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 :**

---

```
#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<sup>2</sup> 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<sup>3</sup> 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;
}

```

---



## 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 satisfaire pour rentrer dans les structures conditionnelles. Enfin, expliquer brièvement ce que fait le programme.

a.

---

```
#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.

---

```
#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. 

---

```
#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. 

---

```
#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.

```
#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-ci peut être infinie : sa seule 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.

---

```
#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.

---

```
#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<sup>4</sup> 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";
    }

    return 0;
}

```

---

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

## 2.6 Les premiers nombres premiers

Écrire un programme qui affiche tous les nombres premiers entre 2 et 100 (inclus).<sup>5</sup>

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 est également plus agréable de travailler avec des valeurs 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,
```

---

5. Astuce : utilisez une boucle dans une autre.

```
    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*<sup>6</sup> 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,
    Beta,
    Gamma,
    Delta,
    Elpha
};

int main()

```

---

6. Aldous Huxley, 1932

```

{

    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<sup>7</sup> :

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

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};

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

        if (nombre > 1)
        {
            plusGrand++;

```

---

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

```

    }
    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'occurrences dans chaque catégorie. Afficher par des étoiles (\*) la distribution dans l'ordre indiqué des catégories.<sup>8</sup>

**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++)
    {
        float nombre = distribution(generateur);

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

```

---

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

```

    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 :

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

**Correction :**

- Vecteur (on ne connaît pas forcément le nombre de nombres premiers entre 1 et 100)
- Tableau (la taille est fixée à 10)
- Tableau (le nombre de phases est connu à l'avance)
- Vecteur (on ne connaît 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<sup>9</sup> 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.



---

```
#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 :**

---

```
#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<sup>10</sup> des mesures à partir d'observations passées. Son programme contient deux vecteurs comme suit :

---

```
vector<int> occurrences = {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 occurrences et la moyenne des observations. Écrire un programme calculant la valeur manquante. Prédire la mesure pour 21 occurrences avec la formule suivante :

$$m = ao + b$$

Avec  $m$  la mesure et  $o$  le nombre d'occurrences.

**Correction :**

---

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

int main()
{
    vector<int> occurrences = {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};

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

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

    float b = totalObservations / observations.size() - a * totalOccurrences /
        occurrences.size();

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

---

10. La méthode utilisée est une régression linéaire

### 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  $n$ <sup>11</sup>, 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).

- Une fonction déterminant la parité d'un nombre
- Une fonction calculant l'aire d'un cercle en fonction du rayon
- Une fonction déterminant si le robot est à moins d'un mètre du mur
- Une fonction réinitialisant les paramètres du robot
- Une fonction qui affiche un message en fonction de l'heure

**Correction :**

- bool* : il n'y a que deux états possibles
- float* ou *double* : en fonction de la précision du rayon
- bool* : il n'y a que deux états possibles
- void* ou *bool* : la fonction peut ne rien renvoyer, ou renvoyer un état indiquant si la réinitialisation s'est bien faite
- 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 :

- 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);
}
```

---

- 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 :

---

```
#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 :

---

```
#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.<sup>12</sup>

Correction :

---

```
#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 Prédiction 2

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

a. 

---

```
#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 :**

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

```

---

## 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 aberrant 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 :

$$\begin{aligned}
 639 \% 10 &= 9 \\
 \frac{639 - 9}{10} \% 10 &= 63 \% 10 = 3 \\
 \frac{63 - 3}{10} \% 10 &= 6 \% 10 = 6
 \end{aligned}$$

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;
}

```

---

## 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<sup>13</sup> 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
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<sup>14</sup>, écrire et implémenter une fonction renvoyant l'action que doit faire le robot en fonction des secondes actuelles.

### Correction :

---

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

14. Elle dure en réalité 15s.

---

```
#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;
    }

    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 détruit !"

### 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 getCirconference();
    float getAire();

    void setRayon(float Rayon);
}

Cercle::Cercle() { }

Cercle::~Cercle() { }

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

```



```

float Cercle::getCirconference()
{
    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.getCirconference() << " 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 :**

---

```

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.distanceParcours(7200) << " mètres en 2h";
}

```

---

## 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 |          | Joueur 1 |         |
|---------------------|----------|----------|---------|
|                     |          | Coopérer | Tricher |
| Joueur 2            | 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<sup>15</sup>. 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*<sup>16</sup> 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

#include "joueur.h"

enum TypeVictoire
{
    Joueur1Victorieux,

```

---

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.

```

        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();
    };

```

---

```

// 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

Cr  er une classe *Simulation* permettant de simuler un nombre ind  fini (c'est-  dire    pr  ciser 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 setNombresSimulations(int);
    void setJeu(Jeu);

    void simuler();
};

```

---

```

// simulation.cpp

```

---

```

Simulation::Simulation() { }

Simulation::~Simulation() { }

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

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

int Simulation::getNombreSimulation()
{
    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;
    nombreVictoireJoueur1 = nombreVictoireJoueur2 = 0;

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

        switch(typeVictoire)
        {
            case TypeVictoire::Joueur1Victorieux:
                nombreVictoireJoueur1++;
                break;

            case TypeVictoire::Joueur2Victorieux:
                nombreVictoireJoueur2++;
                break;

            default:

```

```

        break;
    }
}
}

// main.cpp

#include "simulation.h"

#include <iostream>
using namespace std;

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() << endl << "Le joueur
    2 a << simulation.getNombreVictoiresJoueur2();
}

```

## 5.10 Prédiction 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;  
  
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 Entraînement

- a. Créer une branche à partir de *master*.
- b. Ajouter dans le répertoire Activités/2023/ un exercice<sup>17</sup>. 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 modifications.

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 :

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

Si la fonction renvoie un nombre entier, forcer la conversion de  $n$  en *float*. Que remarquez-vous<sup>18</sup> ?

**Correction :**

---

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

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

    for(int i = 0; i < n; i++)
    {
        wX = distribution(generateur);
        wY = distribution(generateur);

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

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

---

### 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 prendrait 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.

```

std::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 Trie 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<sup>19</sup> suivant en une fonction de paramètre  $n$  :

---

$a \leftarrow 0$   
 $b \leftarrow 2$   
 Pour  $i$  allant de 0 à  $n$  :

---

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

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 donner 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 donner 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 reliées (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>

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

```

---

**Correction :**

```
#include <bits/stdc++.h>
```

---

```

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

    std::sort(heures);

    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<sup>20</sup>.

- *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 :

```

#include <iostream>
#include <cstring>

int main()
{
    std::string prenom;

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

```

---

On a rangé les personnages<sup>21</sup> dans divers vecteurs selon leur catégorie :

```

std::vector<std::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"};
std::vector<std::string> subDPS = {"Albedo", "Aloy", "Amber", "Beidou", "Fischl", "Heizou",
    "Kaeya", "Tighnari", "Venti", "Xiangling", "Xingqiu", "Yae Miko", "Yelan"};

```

---

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

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



```
std::vector<std::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>

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

    std::vector<std::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"};
    std::vector<std::string> subDPS = {"Albedo", "Aloy", "Amber", "Beidou", "Fischl",
        "Heizou", "Kaeya", "Tighnari", "Venti", "Xiangling", "Xingqiu", "Yae Miko", "Yelan"};
    std::vector<std::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++)
    {
        std::cout << "Veuillez donner le nom du personnage #" << i + 1 << " : \n>>> ";
        std::cin >> personnage;

        equipe.push_back(personnage);
    }

    int nombreMainDPS = 0;

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

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

```

    }
}

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

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

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

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

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

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

case 4:
    std::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.

---

```
std::vector<std::string> noms = {"Alice", "Alban", "André", "Ines", "Kaloyan", "Majeed",
    "Max", "Maxence", "Maxime", "Mohammad", "Nina", "Raphaël", "Stéphanie", "Victor",
    "Zachary"};
std::vector<std::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>

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

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

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

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

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

        if (reponse == equipes.at(position))
        {
            std::cout << "Bonne réponse !\n";

            bonnesReponses++;
        }

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

```

    }

    reponses++;

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

    std::cout << "Précision : " << precision << std::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++.

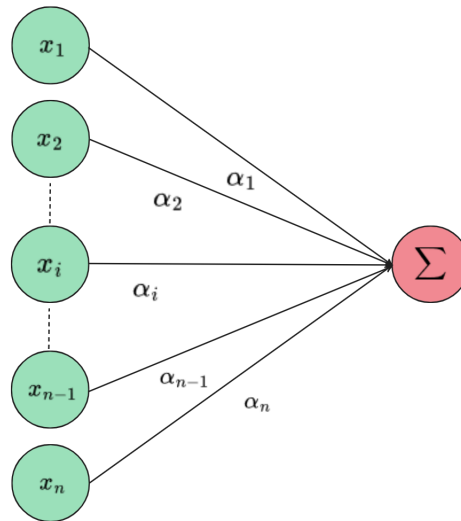


FIGURE 1 – Schéma simplifié d'un neurone artificiel à  $n$  entrées

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

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

On représente souvent les entrées et les poids (les  $\alpha_i$ ) 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>
```

```
class Neurone
```

```
{
```

```
    std::vector<float> entrees;
```

```
    std::vector<float> poids;
```

```
public:
```

```
    Neurone();
```

```
    ~Neurone();
```

```
    std::vector<float> getEntrees();
```

```
    std::vector<float> getPoids();
```

```
    void setEntrees(std::vector<float>);
```

```
    void setPoids(std::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;
```

```
}
```

```
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;
}

```

---

```

// main.cpp

```

---

```

#include <iostream>

#include "neurone.h"

int main()
{
    Neurone neurone;

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

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

    std::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<sup>22</sup> est un exemple très ancien de cryptographie. Il consiste à remplacer chaque lettre d'un message par une autre lettre à distance constante.

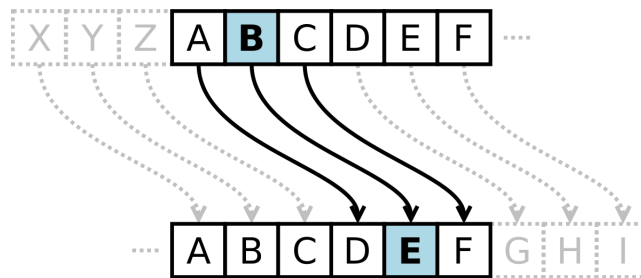


FIGURE 2 – Chiffrement par décalage

Ici<sup>23</sup>, 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.

---

22. Voir [https://fr.wikipedia.org/wiki/Chiffrement\\_par\\_d%C3%A9calage](https://fr.wikipedia.org/wiki/Chiffrement_par_d%C3%A9calage)

23. Illustration venant de <https://commons.wikimedia.org/wiki/File:Caesar3.svg?uselang=fr>



Correction :

---

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

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

    int cle;

    cout << "Veuillez rentrer le texte à chiffrer :\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;

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";
}

```

---

## 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  $\alpha$  la valeur supplémentaire. Écrire une fonction déterminant numériquement la somme  $i$ . La fonction devra pouvoir s'adapter à différentes précisions.

**Correction :**

---

```
#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  $i > \text{probabilite} * \text{gain}$ .
    // Essayons tout de même autrement,
    // dans le cas où l'on ne connaît que l'ordre :

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

    cout << "La somme à verser est " << interets;

    return 0;
}
```

---

## 7.17 La pyramide d'astérisques

Voici une tour faite à partir d'astérisques :

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

FIGURE 3 – Tour de hauteur 4

Écrire un programme qui demande la hauteur à l'utilisateur et qui génère la tour de hauteur associée.

**Correction :**

---

```
#include <iostream>

using namespace std;

int main()
{
    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.