Série I

BONNAZ Aymeric

3 août 2025

Cette série d'exercices appartient au corpus testant vos connaissances et votre compréhension des concepts étudiés lors de l'initiation **C# INTM Strasbourg**.
Les réponses écrites doivent être données sous leurs questions.

Nom apprenant : _____

Date de réalisation : _____

Exercice I — Opérations élémentaires

La mise en place de quelques fonctionnalités, qui réalisent certaines opérations élémentaires d'arithmétique, facilite la résolution de problèmes complexes. Cet exercice implémente quelques-unes d'entre elles.

Opérations de base :

La première de ces fonctionnalités concerne les quatre opérations élémentaires suivantes : l'addition, la soustraction, la multiplication et la division. Celles-ci sont associées aux opérateurs **{+, -, *, /}**.
Prenons a, b $\in \mathbb{N}$ et un opérateur de la liste précédente.
La signature de la méthode obtenue est la suivante :

void : **BasicOperation**(int a, int b, char ope) ;

Le résultat s'affiche sous la forme suivante : a operator b = résultat .
Si l'opérateur n'existe pas ou que le résultat est indéfini, on écrit **"Opération invalide"** à la place du résultat.

1. (2 points) Implémenter la méthode **BasicOperation**.

```
3 + 4 = 7
6 / 2 = 3
3 / 0 = Opération invalide.
6 L 9 = Opération invalide.
```

FIGURE 1 – Exemples d'opérations élémentaires

Division entière :

La division définie dans la méthode précédente ne permet pas de restituer toutes les informations fournies par une division entière.
La signature de la méthode obtenue est la suivante :

void : **IntegerDivision**(int a, int b) ;

Le résultat s'affiche sous la forme suivante : a = q * b + r où q et r sont respectivement le quotient et le reste de la division entière. Si r est égal à 0, on ne l'affiche pas.
Si la division est indéfinie, on écrit **"a : b = Opération invalide"**.

2. (2 points) Implémenter la méthode **IntegerDivision**.

```
12 = -3 * -4
13 = -3 * -4 + 1
12 : 0 = Opération invalide.
```

FIGURE 2 – Exemples de divisions entières

Puissance entière :

Pour finir, la puissance d'un entier est une fonction utile dans des opérations complexes.

La signature de la méthode obtenue est la suivante :

void : **Pow**(int a, int b) ;

Le résultat s'affiche sous la forme suivante : a ^ b = résultat. Si $b < 0$, on écrit **"Opération invalide"** à la place du résultat.

3. (1 point) Implémenter la méthode **Pow**.

Exercice II — Horloge parlante

Afin d'égayer l'emploi d'une horloge parlante, celle-ci adresse à l'utilisateur un message personnalisé qui s'adapte au moment de la journée où il l'utilise. Cet exercice a pour objectif d'implémenter une telle application. L'heure est représentée comme un entier de 0 à 23.

FIGURE 3 – *Ô Soleil ! Toi sans qui les choses Ne seraient que ce qu'elles sont.* Edmond Rostand

L'affichage est de la forme suivante : Il est **heure** H, **message**
L'ensemble des messages proposés est présenté ci-dessous en fonction de l'horaire donnée :

Cas. 1 Entre minuit et avant 6 heures du matin, "Merveilleuse nuit !".

Cas. 2 Entre 6 et avant midi, "Bonne matinée !".

Cas. 3 A midi, "Bon appétit !".

Cas. 4 Entre 13 et 18 heures, "Profitez de votre après-midi !".

Cas. 5 Après 18 heures, "Passez une bonne soirée !".

La signature de la méthode est la suivante :

```
string : GoodDay(int heure) ;
```

1. (2 points) Implémenter la méthode **Journee**.

Exercice III — Construction d'une pyramide

L'existence de pharaon est vouée à régner sur la Haute et la Basse-Égypte. Aussi compliqué que puisse être ce sacerdoce, le plus important, l'œuvre de toute une vie, consiste en la construction de la pyramide qui lui ouvrira les portes du monde des morts. Cet exercice implémente un programme dessinant cette pyramide.

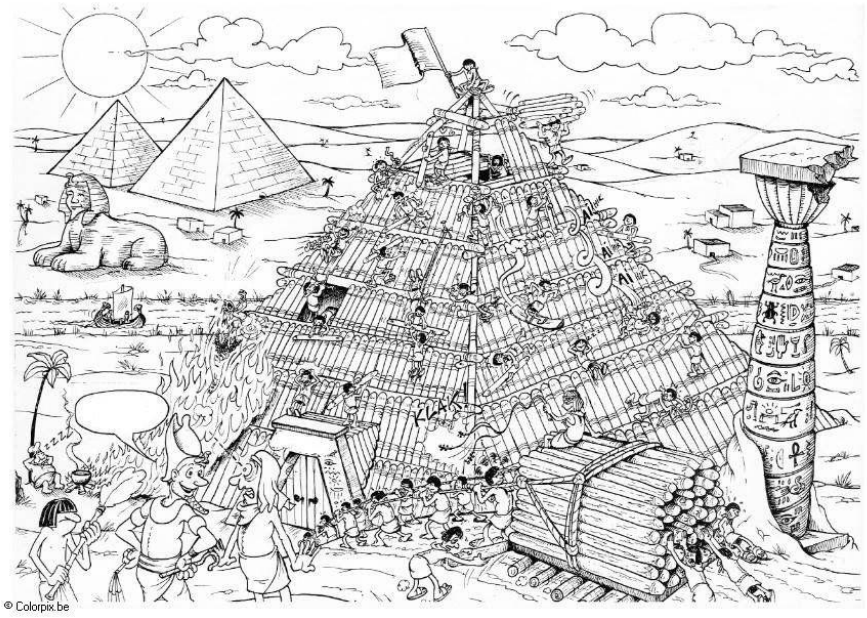

FIGURE 4 – La construction d'une pyramide

L'affichage de la pyramide s'effectue dans la console. La pyramide de hauteur N est constituée de niveaux de blocs successifs, le niveau 1 représente le sommet et le niveau N la base. La base commence au premier caractère de la ligne. À chaque descente de niveau, on ajoute un bloc sur la gauche et un sur la droite. Deux paramètres sont à prendre en compte, la **hauteur** de la pyramide notée $N \in \mathbb{N}^+$ et **l'apparence** de la pyramide, lisse ou striée. Dans une pyramide striée, les blocs des niveaux impairs et pairs sont représentés respectivement par des + et -. Les blocs sont des + pour les pyramides lisses.

1. Nombre de blocs :
   (a) (1 point) Donner le nombre de blocs pour un niveau j donné, $1 \leq j \leq N$
   (b) (½ point) Quel est le nombre total de blocs au niveau N ?

FIGURE 5 – Pyramide lisse, N = 10.

FIGURE 6 – Pyramide strillée, N = 10.

Posons **gauche**(j), **droite**(j) les positions des blocs limites respectivement à gauche et droite de la pyramide au niveau j.

2. Niveau quelconque :
   (a) (½ point) Quelle est la position du sommet de la pyramide ?
   (b) (1 point) Déterminer les formules établissant **gauche**(j), **droite**(j).

La signature de la méthode est :

```
void : PyramidConstruction(int n, bool isSmooth) ;
```

3. (2 points) Implémenter la méthode ci-dessus.

Exercice IV — Factorielle

La factorielle d'un nombre entier positif est définie comme $n! = \prod_{i=0}^{n-1} i$ avec $0! = 1$. L'objectif de cet exercice est de réaliser une méthode la calculant.
La signature de la méthode est la suivante :

```
int : Factorial(int n) ;
```

1. (2 points) Établir une version itérative de la méthode **Factorielle**.
2. Version récursive :
   (a) (½ point) Implémenter la méthode de manière récursive.
   (b) (½ point) Quelle est la version de la méthode la plus efficace ?