College of Computing **Université Mohammed VI Polytechnique**

TP6 - Pointeurs, Allocation Dynamique et Smart Pointers en C++

Imad Kissami

24 Février 2025

Instructions

— Créer un `Makefile` pour compiler tous les exercices.
— Chaque exercice doit être sauvegardé sous le format : `exo{numero_exercice}.cpp`.
— L'exécution se fait avec : `./exo1` (pour l'exercice 1).
— Tous les fichiers doivent être regroupés dans un dossier **TP6_Nom_Prénom**.

Exercice 1 : Gestion dynamique d'une matrice

Objectif :
— Allouer dynamiquement une matrice de taille $n \times m$.
— Initialiser la matrice avec des valeurs croissantes.
— Implémenter une fonction de transposition de la matrice.
— Libérer la mémoire allouée.
Output attendu :

```
Matrice initiale :
1 2 3
4 5 6
7 8 9

Matrice transposée :
1 4 7
2 5 8
3 6 9
```

Exercice 2 : Gestion d'un graphe orienté avec pointeurs

Objectif :
— Implémenter une classe `Graph` utilisant des pointeurs dynamiques pour représenter un graphe orienté.
— Ajouter une méthode pour effectuer un parcours en profondeur (`DFS`).
— Libérer la mémoire dynamiquement allouée.
Output attendu :

```
Parcours en profondeur à partir du sommet 0 :
0 1 3 2
```

Exercice 3 : Gestion d'un arbre binaire de recherche

Objectif :
— Créer une classe `BST` (Binary Search Tree) avec allocation dynamique.
— Implémenter les fonctions d'insertion, de recherche et de suppression.
— Afficher l'arbre en parcours infixe (`in-order`).
Output attendu :

```
Insertion des éléments : 5 3 7 2 4 6 8

Parcours infixe :
2 3 4 5 6 7 8

Recherche de 4 : Trouvé
Suppression de 4
Parcours infixe après suppression :
2 3 5 6 7 8
```

Exercice 4 : Simulation de gestion de mémoire avec `unique_ptr`

Objectif :

— Créer une `struct MemoryBlock` pour simuler l'allocation de mémoire.
— Utiliser des `unique_ptr` pour gérer les blocs de mémoire.
— Implémenter des fonctions de fusion et de libération des blocs.

Output attendu :

```
Bloc de mémoire alloué de 100 unités.
Bloc de mémoire alloué de 200 unités.
Fusion des deux blocs : 300 unités.
Libération de la mémoire.
```

Exercice 5 : Implémentation d'un cache LRU avec `shared_ptr` et `weak_ptr`

Objectif :

— Implémenter un cache de type LRU (Least Recently Used) utilisant `shared_ptr` pour gérer les ressources.
— Utiliser `weak_ptr` pour éviter les cycles.
— Mettre en place une politique d'éviction des éléments les moins récemment utilisés.

Output attendu :

```
Accès aux clés : 1 2 3 1 4
Cache actuel :
4 1 3 2

Accès à la clé 5 -> Eviction de la clé 2
Cache actuel :
5 4 1 3
```