**Merci de respecter les cadres pour les réponses. Tout débordement ne sera pas pris en compte et pourra être pénalisé de 1 point maximum par exercice.**

**Question de cours (4 points) / 1pt par question**

1) Quelle est l'utilité des templates ?

|  |
| --- |
| Rendre générique des fonctions ou des classes / factoriser le code |

2) Qu’est ce que le polymorphisme ?

|  |
| --- |
| Adapter le comportement des objets au runtime / créer une interface générique dont l'implémentation réelle se trouve dans les classes filles |

3) Quels sont les problèmes liés à l’héritage en diamant ?

|  |
| --- |
| Ambiguité dans le nom des méthodes et attributs / probleme de duplication d'objet en mémoire / peu modulaire et maintenable |

4) J'ajoute dans un tableau les entiers 1,2,3,4 et 1. Quel conteneur utiliser pour récupérer les nombres 1,2,3,4 ? Justifiez en une phrase.

|  |
| --- |
| un set car on veut supprimer les doublons |

**Exercice 1 : Template et spécialisation (4 points)**

1) Ecrire une fonction permettant de calculer la moyenne d’une liste de nombres pour les types int, float, double. Vous pouvez utiliser le conteneur de votre choix. (2 points)

|  |
| --- |
| template <class T>  void moyenne(const std::vector<T>& liste)  {  float moy = 0.0f;  for (const auto& elem : liste)  {  moy += elem;  }  std::cout << moy/static\_cast<float>(liste.size()) << std::endl;  } |

2) Pour le type std ::string, on souhaite compter le nombre d’occurrences de la lettre « a ». Spécialiser la fonction précédente permettant de réaliser cela. (2 points)

|  |
| --- |
| // Attention le nom de la méthode doit être la même que dans la question précédente !  // De même le conteneur doit être le même  template<>  void moyenne(const std::vector<std::string>& liste)  {  int nbOccurence = 0;  for (const auto& elem : liste)  {  for (unsigned int i=0; i<elem.size(); ++i)  if (elem[i] == 'a') ++nbOccurence;  }  std::cout << nbOccurence << std::endl;  } |

**Exercice 2 : Compréhension de code avancée (5 points)**





Ecrire la sortie du programme (ce qui sera affiché à l'écran lors de son exécution).

|  |
| --- |
| // 0.5 par ligne correction, -0.5 par erreur ou par ligne manquante  CTOR A  CTOR C  CTOR A  CTOR B  foo B  foo A  DTOR B  DTOR A  DTOR C  DTOR A |

**Exercice 3 – Création d’une classe (7 points)**

Ecrire une classe *Matrice* qui permet de gérer des opérations simples sur des matrices (m x n) de nombres réels avec m = nombre de lignes et n = nombre de colonnes.

Vous devez coder en C++ et séparer la définition de l’implémentation

Si vous codez en C, vous aurez 0 pour l’exercice !

* Définir les attributs nécessaires (1 point)
* Implémenter les constructeurs nécessaires et le destructeur (1 point)
* Créer une fonction *add*() permettant d’additionner deux matrices (2 points)
* Créer une fonction *mul*() permettant de multiplier une matrice (m x n) un vecteur (n x 1) (2 points)
* Créer une fonction *estIdentite*() permettant de savoir si une matrice est l’identité. (1 point)

Fichier .hpp (ou .h)

|  |
| --- |
| // PLusieurs manières soit avec des new[] ou alors vector (plus facile)  #include <iostream>  #include <vector>  #include <iomanip>  class Matrix  {  private:  std::vector<std::vector<float>> m\_mat;  unsigned int m\_line = 0;  unsigned int m\_col = 0;  public:  Matrix(unsigned int l, unsigned int c);  ~Matrix();  unsigned int getCol() const { return this->m\_col; }  unsigned int getLig() const { return this->m\_line; }  void setValue(unsigned int l, unsigned int c, float v) { this->m\_mat[l][c] = v; }  float getValue(unsigned int l, unsigned int c) const { return this->m\_mat[l][c]; }  void add(const Matrix& a);  void mul(const Matrix& a);  bool isIdentity() const;  void print() const; // fonction non demandée  }; |

Fichier .cpp

|  |
| --- |
| Matrix::Matrix(unsigned int l, unsigned int c)  : m\_line(l), m\_col(c)  {  m\_mat.resize(l);  for (unsigned int i=0; i<l; ++i)  {  // Mets des 0 dans la matrice  m\_mat[i].resize(c, 0);  }  }  Matrix::~Matrix()  {  // rien à faire car le vecteur se detruit tout seul !  // Sinon faire les delete ici dans l'autre cas  }  void Matrix::add(const Matrix& a)  {  // Verifie la taille  if (a.getCol() == this->m\_col && a.getLig() == this->m\_line)  {  for (unsigned int i=0; i<m\_line; ++i)  {  for (unsigned int j=0; j<m\_col; ++j)  {  this->m\_mat[i][j] += a.getValue(i, j);  }  }  }  }  void Matrix::mul(const Matrix& a)  {  // Verifie la taille, on ne doit verifier que le nb de colonne (cf définition mathématiques)  if (a.getCol() == this->m\_col)  {  for (unsigned int i=0; i<m\_line; ++i)  {  for (unsigned int j=0; j<m\_col; ++j)  {  // Appliquer la formule ici  }  }  }  }  bool Matrix::isIdentity() const  {  // La matrice doit être carré  if (this->getCol() != this->getLig()) return false;  for (unsigned int i=0; i<m\_line; ++i)  {  for (unsigned int j=0; j<m\_col; ++j)  {  if (i == j)  {  // diagonale  if (this->m\_mat[i][j] != 1)  return false;  }  else  {  if (this->m\_mat[i][j] != 0)  return false;  }  }  }  return true;  }  // Non demandé mais utile pour debuguer  void Matrix::print() const  {  std::cout << "Matrix [" << std::endl;  for (unsigned int i=0; i<m\_line; ++i)  {  for (unsigned int j=0; j<m\_col; ++j)  {  std::cout << std::setw(4) << m\_mat[i][j] << " ";  }  std::cout << std::endl;  }  std::cout << "]" << std::endl;  }  // non demandé mais pour tester, il faut un main !  int main()  {  Matrix a(3,2), b(3,2), c(3,3);  a.setValue(0,0,10);  a.setValue(0,1,-5.2);  a.setValue(1,0,-1);  a.print();  b.setValue(0,0, 5);  b.setValue(1,1, -0.1);  b.print();  a.add(b);  a.print();  c.setValue(0, 0, 1);  c.setValue(1, 1, 1);  c.setValue(2, 2, 2);  std::cout << "IsIdentity = " << c.isIdentity() << std::endl;  c.setValue(2, 2, 1);  c.print();  std::cout << "IsIdentity = " << c.isIdentity() << std::endl;  return 0;  } |

Bon courage !