TP 1 : Bases du langage C++ et principes de la POO

Objectifs du TP

- Identifier les spécificités du langage C++ par rapport au C, notamment en matière de syntaxe, de typage et de modularité.
- Mettre en œuvre des fonctions avancées en C++ : surcharge, passage par référence, paramètres par défaut, fonctions `inline`.
- Utiliser l'allocation dynamique de mémoire via `new` et `delete`, ainsi que les pointeurs intelligents comme `std::unique_ptr`.
- Concevoir et manipuler des classes en C++ intégrant des membres privés, des fonctions membres, des constructeurs, des destructeurs, des attributs statiques et des opérateurs surchargés.

Rappel de cours —Notions utilisées dans ce TP

- **En-têtes standards utilisés** : `<iostream>` pour les entrées/sorties, `<string>` pour les chaînes de caractères, `<vector>` pour les tableaux dynamiques, `<memory>` pour les pointeurs intelligents, `<fstream>` pour les fichiers.
- **`using namespace std;`** : évite de préfixer les objets de la STL par `std::` dans tout le programme. **Exemple :** `cout << "Bonjour";` au lieu de `std::cout`.
- **`new / delete`** : opérateurs pour allouer et libérer dynamiquement de la mémoire manuellement. **Exemple :** `int* tab = new int[5]; delete[] tab;`
- **`std::unique_ptr`** : pointeur intelligent qui gère automatiquement la libération mémoire sans appel explicite à `delete`. **Exemple :** `std::unique_ptr<int[]> p(new int[5]);`
- **`template<typename T>`** : permet de créer des fonctions ou classes génériques. **Exemple :** `template<typename T> T traitement(T a);`
- **`inline`** : suggère au compilateur d'insérer le code de la fonction directement à l'appel. **Exemple :** `inline int carre(int x) { return x*x; }`
- **`T& nom`** : passage par référence pour éviter la copie et permettre la modification directe. **Exemple :** `void increment(int& x) { x++; }`
- **`class`** : mot-clé pour définir une classe. **Exemple :** `class Point { ... };`
- **`private / public`** : déterminent l'accessibilité des membres d'une classe. **Exemple :** `public: void afficher() { ... }` ou `private: double x;`
- **`static`** : rend un attribut ou une méthode commune à toutes les instances. **Exemple :** `static int x;`
- **Initialisation d'un attribut `static`** : un membre `static` doit être initialisé une seule fois, en dehors de la classe (généralement dans le fichier source). **Exemple :** `int MaClasse::x = 0;`

- **`operator+`** : permet de surcharger un opérateur pour qu'il fonctionne avec des objets. **Exemple :** `ClassName operator+(const ClassName& c) ...`
- **`ofstream`** : flux de sortie utilisé pour écrire dans un fichier. **Exemple :** `ofstream f("fichier.txt", ios::app);`
- **Enregistrer une ligne dans le fichier. Exemple :** `f << "text" << "\n";`

1. Préparation de l'environnement de travail (sous Dev-C++)

Avant de commencer les exercices, assurez-vous que votre environnement de développement est correctement configuré. Le TP sera réalisé avec **Dev-C++** (de préférence), ou à défaut **Code ::Blocks**.

- Ouvrez **Dev-C++** depuis le menu Démarrer ou licône du Bureau.
- Créez un nouveau projet :
  - Allez dans `Fichier > Nouveau > Projet...`
  - Choisissez `Console Application`, cliquez sur `C++`, puis nommez votre projet (ex. `TP1_CPP`).
  - Enregistrez-le dans un dossier nommé `TP_CPP_Groupe_X` (remplacez **X** par votre numéro de groupe) sur le Bureau.
- Si vous préférez créer un simple fichier source :
  - Allez dans `Fichier > Nouveau > Fichier source`.
  - Tapez votre code, puis enregistrez le fichier avec l'extension `.cpp`, par exemple `exercice1.cpp`, dans le dossier `TP_CPP_Groupe_X`.
- Pour compiler et exécuter :
  - Appuyez sur `F11` ou allez dans `Exécuter > Compiler et exécuter`.
  - Une fenêtre de console souvrira automatiquement avec l'affichage du résultat.
- Si **Dev-C++** nest pas installé, vous pouvez utiliser **Code ::Blocks** :
  - Créez un projet `Console application`, sélectionnez `C++`, puis utilisez un dossier `TP_CPP_Groupe_X` comme répertoire de travail.

Testez votre configuration avec le programme minimal suivant :

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

int main() {
    cout << "Bienvenue dans le TP C++ !" << endl;
    return 0;
}
```

2. Bases du langage C++ et structures de contrôle

Exercice 1 :

Écrire un programme C++ qui lit deux entiers à l'aide de `cin`, puis affiche leur somme, leur différence et leur produit avec `cout`.

Exercice 2 :

Écrire un programme qui réalise les étapes suivantes :

1. Parcourir les entiers de 1 à 10 à l'aide d'une boucle `for`.
2. Déclarer une variable `bool` nommée `estPair` pour indiquer si l'entier est pair.
3. Déclarer une variable `string` nommée `type`.
4. Pour chaque entier :
   - Affecter à `estPair` le résultat de l'expression `i % 2 == 0`.
   - Si `estPair` est `true`, affecter `"pair"` à `type`, sinon `"impair"`.
5. Afficher un message sous la forme `"L'entier X est type."`, où **X** est l'entier et **type** est la chaîne correspondante.

3. Fonctions avancées et mémoire dynamique en C++

Exercice 1 :

Écrire un programme qui utilise deux méthodes d'allocation dynamique en C++ :

1. Inclure les en-têtes `<iostream>`, `<string>` et `<memory>`.
2. Définir une fonction `inline void afficherTableau(int* tab, int taille, std::string titre = "Contenu :")` qui affiche les éléments d'un tableau dynamique.
3. Dans `main`, allouer deux tableaux de taille 5 :
   - `tab1` avec `new int[5]`,
   - `tab2` avec `std::unique_ptr<int[]>`.
4. Remplir les deux tableaux avec les entiers de 1 à 5.
5. Appeler une fois la fonction `afficherTableau` pour chaque tableau :
   - Pour `tab1`, utiliser le titre par défaut.
   - Pour `tab2`, utiliser un titre personnalisé. Utiliser la méthode `.get()` pour récupérer le pointeur brut depuis le `unique_ptr`.
6. Libérer manuellement `tab1` avec `delete[]` ; la mémoire de `tab2` est libérée automatiquement.

Exercice 2 :

Écrire un programme utilisant des fonctions `template` pour déterminer la valeur maximale.

1. Définir une fonction `template<typename T> T maxValeur(T a, T b)` qui retourne la plus grande des deux valeurs.

2. Définir une autre fonction
`template<typename T> T maxValeur(const std::vector<T>& v)`
qui retourne la plus grande valeur contenue dans un vecteur.
3. On suppose que le vecteur contient toujours au moins un élément.
4. Dans `main()`, tester :
   - la fonction avec deux entiers puis deux réels,
   - la fonction avec un vecteur d'entiers puis un vecteur de `double`.

4. Classes, objets, membres et surcharge d'opérateurs

Exercice 1 :

Créer une classe `Point3D` permettant de représenter et manipuler des points dans l'espace tridimensionnel. Chaque instruction ci-dessous doit être respectée.

1. Déclarer une classe nommée `Point3D`. Ses coordonnées `x`, `y` et `z` doivent être des attributs privés de type `double`.
2. Ajouter un attribut statique `int compteur` permettant de compter le nombre total d'objets `Point3D` créés. Écrire également une fonction membre statique `getCompteur()` pour y accéder.
3. Définir un constructeur à trois paramètres (`x`, `y`, `z`) initialisant les attributs. Il devra incrémenter le compteur statique à chaque appel.
4. Définir un destructeur qui affiche un message lors de la destruction d'un objet (par exemple : `Destruction du point (x, y, z)`).
5. Définir une méthode `void afficher()` qui affiche les coordonnées du point au format `(x, y, z)`.
6. Définir une méthode `void sauvegarder()` qui ajoute une ligne au fichier texte `"points.txt"` (créé s'il n'existe pas) au format `"x;y;z\n"`, en utilisant le mode d'ouverture `ios::app`.
7. Surcharger l'opérateur `+` afin qu'il retourne un nouveau point résultant de la somme des coordonnées respectives de deux objets `Point3D`.
8. Écrire un programme principal qui :
   - crée deux objets `Point3D`,
   - les affiche à l'écran,
   - additionne les deux objets à l'aide de l'opérateur `+`,
   - affiche et sauvegarde les trois points (`p1`, `p2`, `p3`) dans le fichier `points.txt`,
   - affiche le nombre total de points créés grâce à la méthode statique.