C++ 2A
TD/TP 3
Pointeurs, références, allocation dynamique

1 Pointeurs et références

1.1 Pointeurs

```
#include <iostream>

void fct( int * p ) {
  *p += 2;
}

int main() {
  int a = 5;
  int * p_a = &a;
  *p_a += 3;
  fct( p_a );
  std::cout << a << std::endl;
  return 0;
}
```

1. Quel est le résultat affiché par ce code ?
2. Que se passe t'il si on initialise `p_a` avec `int * p_a = nullptr;` (C++11) ?

1.2 Références

1. Reprendre le code précédent en remplaçant l'utilisation d'un pointeur par une référence.
2. Vérifier que le résultat obtenu est correct.

2 Allocation dynamique manuelle

1. Allouer dans la pile un tableau statique de 1000 entiers et le remplir (1, 2, ...). Tester ce code.
2. Passer le taille de ce tableau à 10 millions d'entiers et retester ce nouveau code.
3. Allouer dans le tas un tableau de 10 millions d'entiers et tester ce nouveau code. Ne pas oublier de supprimer le tableau à la fin du programme.
4. Est-il possible de créer un tableau de personnes (cf. classe `Personne` du TP1). Rappel : la classe `Personne` ne contient pas, et ne doit pas contenir de constructeur par défaut.
5. Expliquer pourquoi et donner le message d'erreur affiché à la compilation.
6. Pourquoi et comment cela fonctionne en Java ?
7. Porter cet solution en C++. Comment supprimer le tableau ? Ajouter des affichages dans les constructeurs et le destructeur de la classe et observer ce qui se passe.

3 Pointeurs intelligents

3.1 `unique_ptr`

1. Créer un `std::unique_ptr` pour gérer un pointeur sur une personne et manipuler l'instance à travers lui. Créer une définition à l'aide de **`using`** pour simplifier le code.
2. Créer une nouvelle personne à partir de la précédente.
3. Utiliser la fonction `std::make_unique` disponible en C++14 (`-std=c++14`)pour arriver au même résultat.
4. Reprendre l'exercice précédent en utilisant un tableau de `std::unique_ptr` de `Personne`.

3.2 `shared_ptr`

1. Créer un `std::shared_ptr` pour gérer un pointeur sur une personne et manipuler l'instance à travers lui.
2. Quel est la différence entre un `std::unique_pointer` et un `std::shared_pointer` ?
3. Créer plusieurs `std::shared_ptr` de personnes et insérer les dans différents tableaux.
4. Utiliser la méthode `use_count` pour afficher le nombre de fois qu'une même personne est référencée.

4 Conteneurs standards

La bibliothèque standard C++ propose différents conteneurs dynamiques afin d'encapsuler la gestion de la mémoire dynamique.

4.1 `std::vector`

1. Donner la déclaration d'un `std::vector` de 1000 entiers non signés.
2. Donner un exemple d'utilisation d'un tableau défini par `std::vector< Personne * >`.
3. Quels sont les problèmes de cette approche ?
4. Quel type de *smart pointer* peut-on utiliser ?
5. Créer l'implantation correspondante. Ajouter si besoin un affichage dans le destructeur de la classe `Personne` pour voir s'il est bien appelé pour chaque élément du tableau.
6. Ajouter différentes personnes dans le `std::vector`.

4.2 `std::list`

1. Reprendre l'exercice ci-dessus en remplaçant le `std::vector` pour une `std::list`.
2. Quelles sont les modifications à apporter ?