# Conception Objets avancée : TP 3

Giuseppe Lipari

March 29, 2021

## Introduction

Le but de ce TP est de comprendre la programmation par *templates*. Nous allons implementer des fonctions pour faire l'union et l'intersection des deux containers.

*Attention* : des fonctions similaires sont déjà disponibles dans la librairie standard : `std::set_intersection` et `std::set_union`. Nous utiliserons des noms un peu différents pour éviter toute confusion.

### Définitions

L'intersection de deux ensembles $A$ et $B$ est l'ensemble $C$ qui contient tous les éléments qui sont présents dans le deux ensembles :

$$C = A \cap B = x | x \in A \wedge x \in B$$

L'union de deux ensembles $A$ et $B$ est l'ensemble $C$ qui contient tous les éléments qui sont présents dans un de deux ensembles:

$$C = A \cup B = x | x \in A \vee x \in B$$

L'ensemble $C$ ne contient pas de doublons.

L'objectif **final** est d'implanter ces deux fonctions de manière la plus générale possible : les fonctions doivent fonctionner sur n'importe quel container, et sur n'importe quel type de donnée contenue.

## Question 1: vecteurs d'entiers

Écrire deux fonctions, `set_intersection` et `set_union` qui, à partir de deux vecteurs d'entiers, remplissent un vecteur qui contient l'intersection (l'union, respectivement) de deux vecteurs d'entiers.

Voici le prototype de la fonction `set_intersection_nt()` :

```
void set_intersection_nt(std::vector<int>::const_iterator a_begin,
                         std::vector<int>::const_iterator a_end,
```

```
                        std::vector<int>::const_iterator b_begin,
                        std::vector<int>::const_iterator b_end,
                        std::back_insert_iterator<std::vector<int>> c_begin);
```

(`nt` indique la version non-template). La fonction sera utilisé comme dans le programme suivant:

```
vector<int> vec_a = {1, 2, 3, 4};
vector<int> vec_b = {3, 4, 5, 6};
vector<int> vec_c;

set_intersection_nt(begin(vec_a), end(vec_a), begin(vec_b), end(vec_b),
                    back_inserter(vec_c));
```

La fonction `set_union_nt` a le même prototype et on l'utilise de la même manière.

Testez les deux fonctions, surtout dans les cas limites (intersection nulle, l'union de deux ensembles vides, etc.)

### Réponse

*Écrire ici votre réponse, et indiquez les tests que vous avez écrit.*

## Question 2: Généralisation sur les containers

Généralisez les fonctions développé dans la question 2 en utilisant des templates. Il faut donc écrire deux fonctions templates, `set_intersection(...)` et `set_union(...)`. Il y a deux type de généralisation possible :

- Type d'objet contenu : par exemple, faire l'intersection entre deux vecteurs de `string`.
- Type de container : par exemple, faire l'union entre deux `list<int>`.

Nous vous demandons de faire les deux généralisations au même temps. En particulier, avec la même fonction template `set_intersection`, il doit être possible de faire l'intersection entre deux `vector<string>`, et entre deux `list<int>`.

Écrire des tests pour vérifier cela.

### Réponse

*Écrire votre réponse ici, et indiquez les tests que vous avez écrit.*

## Question 3: Containers d'objets

Créer de vecteurs et des listes d'objets de type `MyClass`, et essayez de faire des intersections et des unions avec les fonctions développées dans la Question 2. Vérifiez que tout est correct.

Quel sont les caracteristiques minimales de la classe pour pouvoir faire l'intersection et l'union ?

### Réponse

*Indiquez les test que vous avez développé. Écrire ici les caractéristiques minimales de la class* `MyClass`.

## Question 4

Est-ce qu'on peut appeler `set_intersection_t` sur deux containers qui contiennent objets de type différents ?

Si non, pourquoi ? Si oui, quel sont le conditions minimales sur les types des objets, et quel est le résultat finale ?

### Réponse

*Écrire ici votre réponse.*

## Question 5

Essayez d'appliquer vos fonctions sur des `map<int, string>`. Est-ce que ça marche ? Si oui, pourquoi ? Si non, pourquoi ?

### Réponse

*Écrire ici votre réponse.*

## Question 6

1. Généraliser la fonction `set_intersection_t` avec un paramétre template additionnel `F` qui represents une fonction de comparaison entre les objets des deux containers.

2. Appliquer la fonction `set_intersection_t` sur une `map<int, string>` and sur un `vector<int>`, le resultat sera produit sur une `map<int, string>` :

```cpp
bool my_compare(const std::pair<int, string> &x, int y);

map<int, string> a, b;
vector<int> v = {1, 3, 5};

a[1] = "A"; a[2] = "B"; a[3] = "C"; a[4] = "D";

set_intersection_t(begin(a), end(a), begin(v), end(v),
                   inserter(b, b.end()), my_compare);

// b should contain (1, "A") and (3, "C") and nothing else.
```

(a) Testez le résultat.

## Réponse

*Réponse dans le code*