



# Compte rendu de programmation: TP1

ESIREM DIJON

CHAPUS Louka

PAUZIE Laureline

Groupe 3A IE

# TP n°1

Tout d'abord, nous avons choisi de commencer par le TP1 car nous n'avions jamais codé en C++ avant et d'après monsieur GINHAC, le TP1 était annoncé comme étant le plus simple. Cela nous a donc permis de nous habituer à la programmation en C++. Nous avons aussi appris à utiliser le logiciel Github et plus particulièrement Git pour pouvoir travailler à deux sans avoir à télécharger et échanger les fichiers pendant chaque séance. Cet outil a montré très efficace et pratique au cours de notre travail car nous pouvons l'utiliser à chaque fois nous changeons le code ainsi que notre binôme peut récupérer le dernier changement au code.

Le but du TP1 était d'implémenter la gestion d'une bibliothèque avec des livres, des dates, des lecteurs, des auteurs et des emprunts.

Au début de chaque classe on a mis la commande :

- *#pragma once*

Cette commande sert à n'inclure qu'une seule fois les autres classes nécessaires à l'implémentation, cela permet d'économiser de la mémoire même si nous n'avons pas créé beaucoup de classe. Nous avons aussi ajouté les commandes suivant dans les fichiers .h pour but de ne pas avoir définir une classe plusieurs fois au mauvais endroit.:

- *#ifndef*
- *#define*
- *#endif*

De plus, pour chaque classe nous avons mis des getters constants. Cependant, nous mettons rarement des setters car ils ne nous servent pas.

Pour commencer, nous avons choisi de créer la classe **Date** en première en reprenant simplement celle du cours, en ajoutant en plus les années et une méthode pour convertir la date en texte affichable. Puisque c'est notre premier TP en binôme nous avons décidé de faire cette première classe ensemble. Par la suite nous avons réparti à chacune une classe à réaliser.

Ensuite, nous avons implémenté la classe Auteur. Tous les paramètres de cette classe sont, soit des chaînes de caractères, soit une date d'où il n'y a pas vraiment de vérification à faire pour s'assurer d'avoir un objet valide: Nous sommes dans le cas où nous acceptons tous les prénoms et tous les noms. De plus, nous avons surchargé l'opérateur << pour pouvoir afficher correctement les informations d'un auteur :

- Le nom
- Le prénom
- La date de naissance

La surcharge a été réalisée par Louka et n'a posé aucun problème étant donné que nous avons déjà déclaré des chaînes de caractère.

Puis, nous avons créé la classe **Livre**, qui reprend plusieurs arguments venant des autres classes comme l'auteur ou la date de publication. De même ici, nous ne faisons pas

de test sur les paramètres ‘passer’ en ‘entrer’ car la plupart sont des chaînes de caractères. Nous considérons par défaut que lorsqu’un livre est ajouté alors il est disponible directement, d’où la commande: *_dispo = true* dans le constructeur de la classe. Nous avons fait que lorsqu’un lecteur emprunte un livre, alors son identifiant s'ajoute automatiquement dans la liste des emprunteurs et le code met la disponibilité à *false*. Enfin, comme la classe précédente, nous avons surchargé l’opérateur << pour pouvoir afficher correctement toutes les informations d’un livre, cela s’est fait sans trop de difficulté puisque nous avons repris le même code de la classe précédente en modifiant ce qu’il fallait. Dans ce TP, c’était Louka qui est chargée de faire les surcharges, Laureline est chargée de faire la base des codes et les tests.

Par la suite, nous avons implémenté la classe **Lecteur**, elle prend comme argument des chaînes de caractères et un vecteur de type *long int* pour la liste ISBN, cette liste contiendrait des livres empruntés par le lecteur. Encore une fois, nous laissons toutes les possibilités pour les chaînes de caractère. Lorsqu’un lecteur va emprunter un livre, nous allons ajouter son ISBN dans la liste prévue à cet effet ce qui permet de savoir à n’importe quel moment tous des livres empruntés par un lecteur. Et lorsque le lecteur rend le livre, nous retirons l’ISBN du livre de la liste. Pour cela, nous avons fait la méthode suivante : *remove_ISBN()*.

Cette méthode est un peu particulière, car nous ne pouvons pas faire une boucle *for* normale et supprimer l’élément qui est à supprimer car nous avons un vecteur d’où la longueur n’est pas constante. Donc si nous supprimons un élément, alors il faut faire attention au prochains indices qui seront décrémenter d'un ou faire une boucle for "à la main", en incrémentant manuellement l’indice. C’est donc ce que nous avons réalisé à l’aide d’un itérateur (ligne 27 à 35 dans lecteur.cpp). Nous avons enfin, encore une fois, surchargé l’opérateur << pour avoir accès aux informations des lecteurs. Nous distinguons deux cas, un où l’utilisateur a des livres à rendre et un où il n’en a pas et nous affichons un message différent en fonction de la situation rencontrée.

Nous passons maintenant à la classe **Emprunt**, nous n’avons pas pu utiliser cette classe ni dans d’autres classes ni dans le main puisque lorsqu’un lecteur emprunte un livre, nous passons directement le livre et le lecteur en argument de la fonction. Nous pourrions passer par la classe Emprunt mais cela ne fait qu'ajouter une couche supplémentaire non nécessaire au bon déroulement du programme car les emprunts n’apportent pas spécialement plus d’information que ce que nous avons déjà dans les autres classes. Donc, nous avons décidé de la coder avec son variable membre et ses getters mais nous avons aussi choisi de ne pas utiliser cette classe.

Enfin, pour la classe **Bibliothèque**, elle réunit toutes les autres classes afin de pouvoir avoir des commandes simples et aussi de pouvoir toutes les réunir au même endroit. Cette classe ne comporte que des vecteurs car nous ne savons pas, à priori, combien d'éléments il va y avoir dans chaque vecteur. De plus, il est possible à n’importe quel moment de changer la taille d’un vecteur en ajoutant soit un livre, soit un lecteur ou soit un auteur. Donc il est indispensable d’avoir des vecteurs ou des array. Comme les vecteurs sont des conteneurs de la STL, il est assez simple d’ajouter du contenu dans chaque vecteur avec la commande *push_back()*, nous avons donc mis en place trois méthodes pour ajouter du contenu aux trois vecteurs.

Puis, il faut pouvoir emprunter et restituer un livre à la bibliothèque, nous avons réalisé la méthode *emprunter()* qui prend directement le livre à emprunter et le lecteur qui l’emprunte comme argument (avec des pointeurs pour pouvoir modifier l’argument et non sa copie).

Ensuite nous regardons si le livre est disponible, si c'est le cas alors on ajoute l'ISBN à la liste des livres empruntés du lecteur et nous changeons le statut du livre. Si le livre n'est pas disponible alors nous afficherons un message d'erreur dans la commande. Par la suite, nous avons fait la méthode *restituer()*, comme celle précédente, elle prend les mêmes arguments, aussi avec un pointeur pour pouvoir modifier directement le paramètre 'passer' en argument et non une copie. Puis nous parcourons la liste des ISBN d'un lecteur car s'il veut rendre un livre, cela signifie que le livre est dans sa liste des ISBN. Nous avons besoin alors de comparer chaque élément au livre passé en argument, lorsque nous tombons sur le bon alors nous enlevons l'ISBN de la liste du lecteur et nous remettons le statut en disponible (= true). Pour terminer nous avons réalisé une fonction pour afficher tous les livres d'un auteur qu'ils soient empruntés ou non. Pour cette fonction, nous choisissons le nom de l'auteur puis nous parcourons la liste des livres ayant dans la bibliothèque et nous comparons le nom des auteurs auteurs à chaque fois, si les deux noms sont identiques alors le titre du livre afficherait.

Finalement, dans le main nous testons presque toutes les fonctions et les méthodes précédemment évoquées, nous commençons par créer les vecteurs d'auteur, de lecteur et de livre avec chaque fois les informations nécessaires pour création de chaque objet. Puis, nous créons une bibliothèque qui s'appelle 'BU' avec les vecteurs précédemment définis. Et enfin nous essayons de faire des différentes opérations comme 'emprunter' ou 'restituer' des livres avec différents cas de figure pour tester les erreurs. Pour terminer nous testons aussi la surcharge des opérateurs pour Livre et Lecteur.

Pour conclure, ce TP nous a permis, d'abord, de prendre en main l'outil très pratique pour la programmation Git qui permet de sauvegarder et partager les changements d'un code lorsqu'on travaille à plusieurs. Ensuite, nous avons pu voir les bases de la programmation en C++ avec les différentes classes et fonctions, comment les coder et les utiliser, comment prendre en main les notions apprises en cours et les appliquer en pratique… Après ce TP, nous sommes plus à l'aise avec la programmation et nous savons aussi notre niveau en C++. Nous avons donc décidé par la suite de tester un TP de niveau moyen et non avancé.