Rapport du TP2

Licence 3, 2020/2021

Table des matières

Introduction

Ce rapport a vocation à vous faire comprendre le but de notre TP et surtout de vous expliquer comment utiliser le programme à des fins personnels.

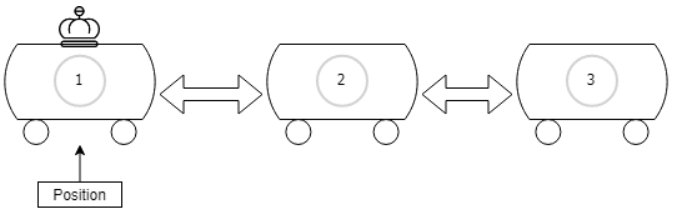
Le sujet de ce TP était de créer ce que l'on appelle une liste chaînée de type T, T étant un type quelconque. Nous avions l'ossature du code mais il nous était demandé de surcharger et d'améliorer le code afin de pouvoir utiliser des méthodes plus simples et plus explicites.

Nous verrons comment a été réalisé ce code et comment l'utiliser correctement.

Concept de liste chaînée

Concept physique

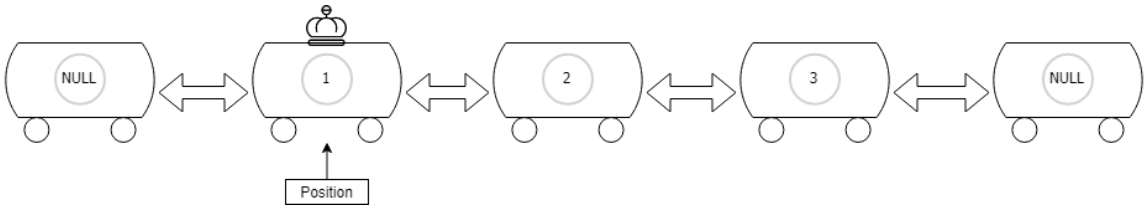
La liste chaînée est un concept en fait assez simple. Prenons un train composé de petits Wagons, il sera notre liste chaînée :



Nous voyons ici que chaque Wagon contient un numéro et est connecté avec le suivant s'il en a un et le précèdent s'il en a un, ce concept est représenté par les flèches, elles sont à double sens car si le Wagon 1 a accès au Wagon 2 alors le Wagon 2 a accès au Wagon 1. Au final, nous sommes assis dans le Wagon 1, la tête de train et les toilettes sont dans le Wagon 3, alors pour aller il faut passer du Wagon 1 au Wagon 2 et du Wagon 2 au Wagon 3, et inversement lorsque l'on voudra retourner à notre siège.

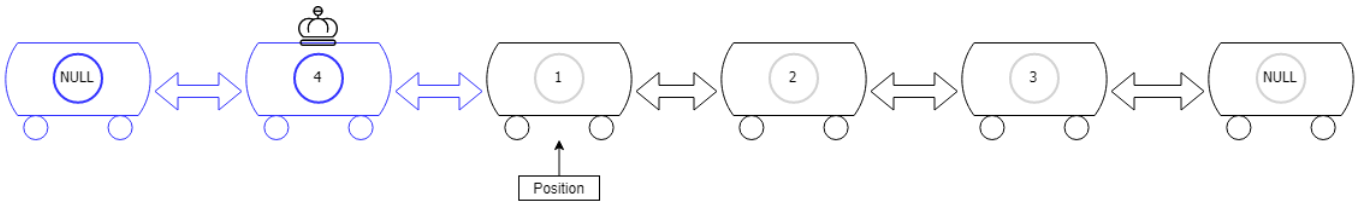
Concept informatique simplifié

Le concept précédent oubliait un élément majeur des listes chaînées en informatique.

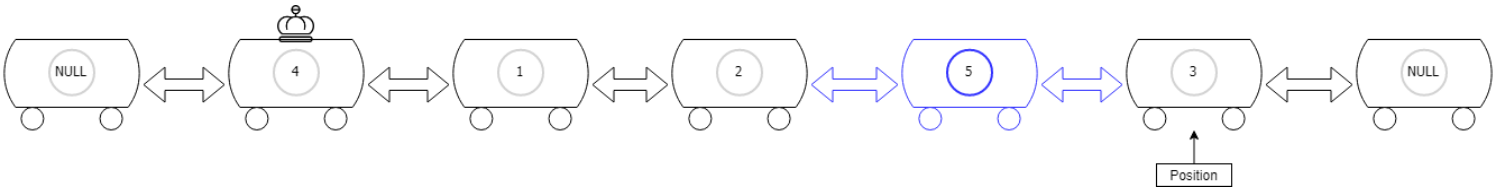


Cette version est beaucoup plus logique d'un point de vue informatique, un objet de type Wagon aura **forcément** un Wagon précèdent et un Wagon suivant, la complexité est sur le fait qu'un Wagon peut être null ce qui veut dire qu'il existe mais il n'est pas

créé en soi, ce qui veut dire qu'a l'heure actuelle, le Wagon 1 est la tête de train, mais si j'insère un Wagon numéro 4 avant le Wagon 1 alors :

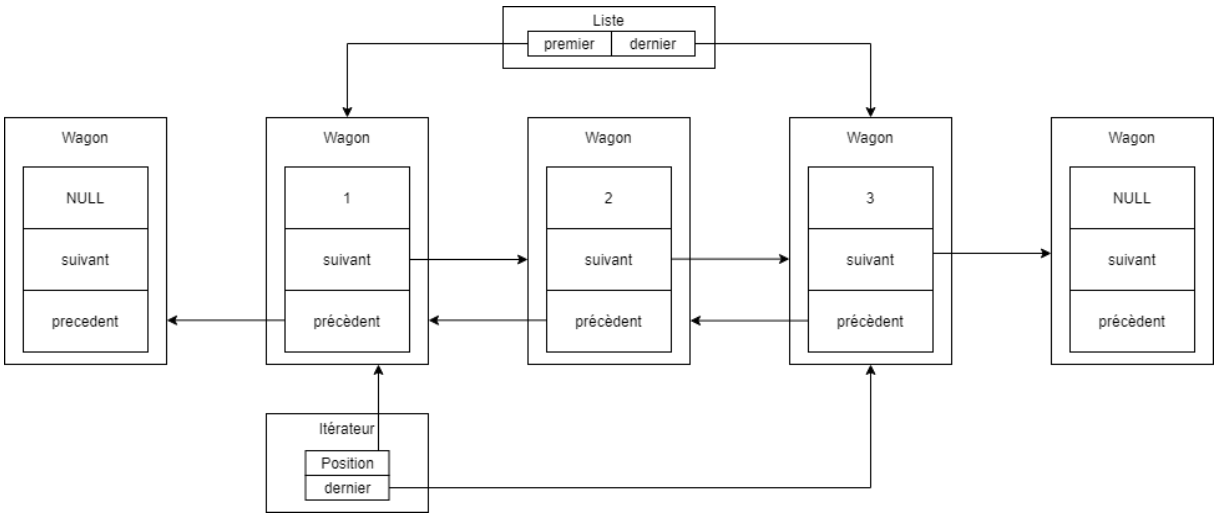


On voit que le Wagon 4 a remplacé le wagon null qui précédait le Wagon 1 et est devenu la tête de train. En fait le Wagon de valeur null est un emplacement possible mais pas encore utilisé, c'est de cette façon qu'il faut comprendre ce concept.



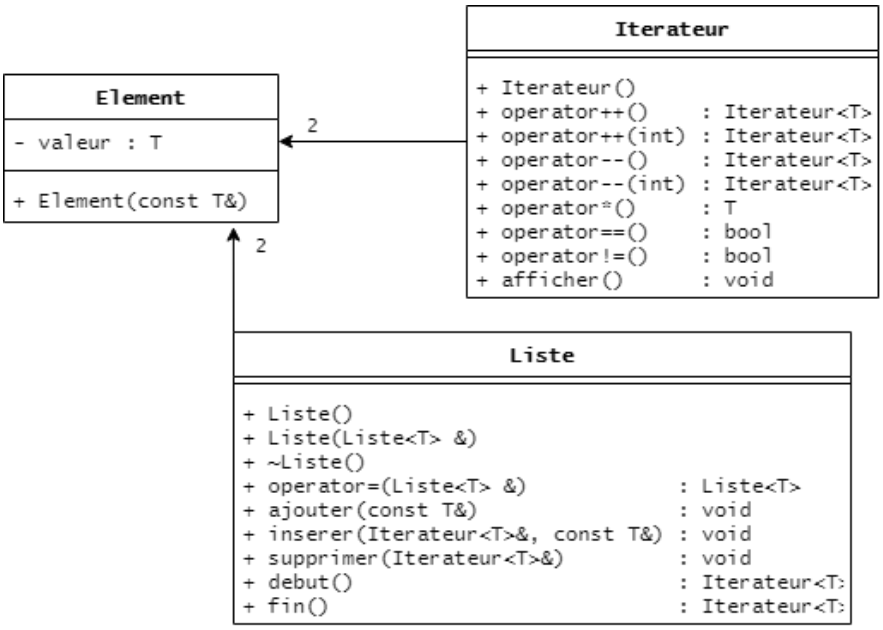
De même, l'atout d'une liste chaînée est pouvoir insérer un Wagon en plein milieu de celle-ci pour peu que l'on change de position, imaginons que notre siège est dans le Wagon 3 et que l'on insère un Wagon 5 :

Concept informatique réel



Voici la façon informatique de voir le concept. Des objets de type wagon contenant 3 variables, une variable valeur qui prend un type T quelconque, un pointeur de Wagon suivant et un pointeur de Wagon précèdent. Une Liste de Wagon qui pointe vers le premier et le dernier Wagon, qui est juste un conteneur qui générera notre Itérateur. Et d'ailleurs, on peut voir notre Itérateur en dessous de la Liste qui point un wagon via la variable position et un autre via la variable dernière.

Un code qui sonne complexe …



On peut voir sur ce diagramme de classes qu'il y a tout de même beaucoup de méthodes à utiliser, nous verrons plus tard comment les utiliser correctement.

Passons dans le vrai langage du TP et donc arrêtons d'utiliser des objets de type Wagon, utilisons plutôt des objets de type Element.

Petite clarification avant d'approfondir le sujet, nous avons utilisé le terme « type T » plusieurs fois au cours de ce rapport, je l'ai déjà précisé rapidement afin que vous ne soyez pas perdu dès le départ et de commencer en douceur. Lorsque j'utilise le terme T cela veut dire que les classes ont été écrite selon le paradigme de programmation nommé « Generic Programming Paradigm » . Plus précisément, cela veut dire que notre code ne comporte pas de type précis comme int, string, char, double, … . Dans notre cas, cela veut dire que Elément peut prendre n'importe quel type en tant que valeur, d'ailleurs dans le main de test que l'on verra plus tard ce fait est prouvé directement.

Dans le diagramme de classes vu plus haut, nous pouvons voir des operator++, operator*, operator==, … . Cela s'appelle un override, il ne s'agit pas de multiplier 2 Eléments via operator*, ils ont été override car ils étaient inutiles et cela permet d'avoir un code moins verbeux du fait que l'on n'utilise pas de fonction mais aussi plus compréhensible. En effet, operator++ permet de passer à l'Element suivant d'une variable tterateur via iterateur++ ou

++iterateur, de même pour les operator--, operator* agis comme un get et retourne l'Element de la position de l'Iterateur.

La classe Liste est relativement simple à comprendre mais elle mérite tout de même quelques clarifications sur certaines méthodes :

La méthode debut créée un Iterateur<T> avec la position placée sur le premier Element de la Liste, la methode fin, quand a elle met la position a null, il faut donc bien prendre en compte ce fait, si on fait iterateur++, par sécurité le code mettra la position sur l'Element final de la Liste.

La méthode inserer quant a elle insère un Element avant la position, comme nous avons vu plus haut dans l'exemple du train où nous insérions un Wagon 4 lorsque la position était sur le Wagon 1. Le Wagon 4 était donc le précèdent du Wagon 1 et il était devenu le premier Element de la liste, ce que nous avions exprimé en disant qu'il était devenu la tête du train.

… mais pourtant très simple à utiliser

Compiler et lancer le code

Un fichier makefile a été crée afin de pouvoir compiler le code très facilement. Une fois dans le dossier contenant les .cpp/.h et le makefile, lancez tout simplement la commande make, cela va compiler le code (et afficher quelques warnings par la même occasion, mais n'y prêtez pas attention, cela veut juste dire que le code ne respecte pas forcément les conventions d'écritures voulues par le compilateur). Une fois compilé, écrivez ./MainTemplate dans la console et magie, le test se lance en affichant tous les tests.

Le fichier test mainTemplate.cpp

Ce fichier est le fichier dans lequel j'ai placé le main du projet avec tous les tests que j'ai imaginés, c'est aussi dans ce fichier que vous pouvez écrire si vous voulez faire d'autres tests, ou bien vous pouvez créer votre propre fichier en faisant attention à import listeTemplate.cpp et à modifier le makefile. Comme vous pourrez le voir dans le fichier, la plupart des tests ont été testé, chaque méthode a été plusieurs fois via dans différents cas.