POLYTECH DIJON

TRAVAUX DIRIGÉS

Programmation Orientée Objets / C++

Listes chainées - Template

OBJECTIFS

A travers cet exercice, vous manipulerez les notions suivantes en C++ :

- Structure de données de type liste chainée bidirectionnelle,
- Allocation dynamique et pointeurs
- Templates C++

PRINCIPE

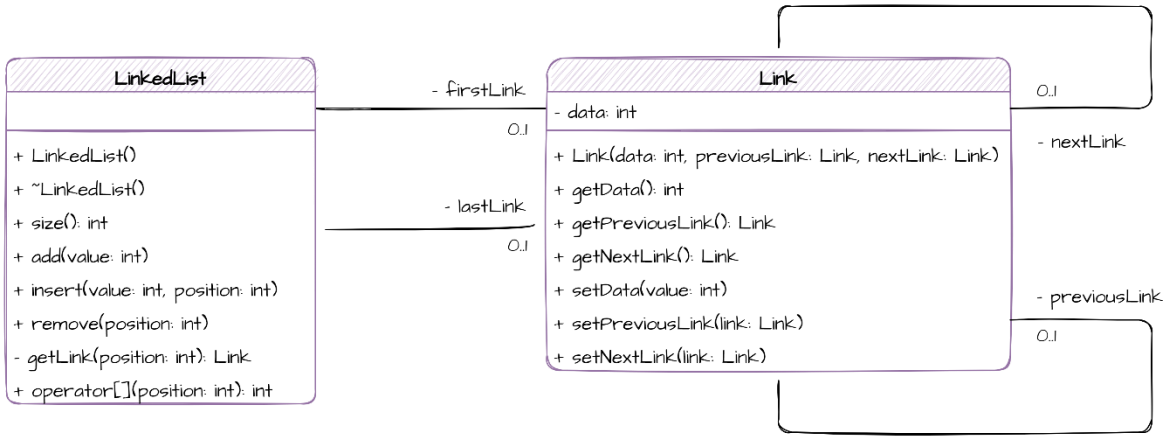
Le principe de la liste chainée est assez simple. Savez-vous ce qu'est un mousqueton ?

Maintenant oui. Imaginez à présent une chaine composée de mousquetons accrochés les uns aux autres :

Chaque maillon de cette chaine représente une donnée qui est liée à la donnée suivante et à la donnée précédente.

Quant à vous, vous tenez les deux extrémités de la chaine ce qui vous permet de savoir où elle commence et où elle se termine.

Dans le cadre d'une liste chainée d'entiers, nous pouvons réaliser la modélisation suivante :



PROJET

- Créez une nouveau projet C++ vide.
- Ajoutez un fichier main.cpp et testez le bon fonctionnement de votre programme.

CLASSE LINK

- Ajoutez la classe **`Link`** à votre projet et implémentez là à partir de la modélisation ci-dessus.

> **Note**
> Les cardinalités **`0..1`** indique que l'attribut peut contenir une valeur ou aucune, ce qui en C++ se traduit par la valeur **`nullptr`**. Ceci devrait vous donner un indice sur le type des attributs **`previousLink`** et **`nextLink`**.

- Dans votre fonction main, instanciez un objet de la classe Link pour en tester le bon fonctionnement. Vous fournirez la valeur **`nullptr`** aux paramètres **`previousLink`** et **`nextLink`** du constructeur.

CLASSE LINKEDLIST

- Ajoutez une classe **`LinkedList`** au projet.
- Dans un premier temps, ne codez que la structure de la classe : la déclaration du **`.h`** et la structure des fonctions dans le **`.cpp`** mais sans le contenu, sauf pour les méthodes qui doivent retourner quelque chose où vous retournerez une valeur par défaut.

Une fois que la structure est prête et que tout compile correctement, vous allez pouvoir coder votre liste chainée étape par étape.

Constructeur

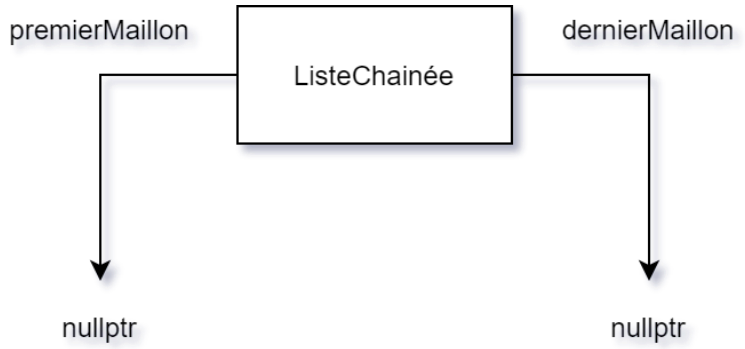
➢ Implémentez le code du constructeur.

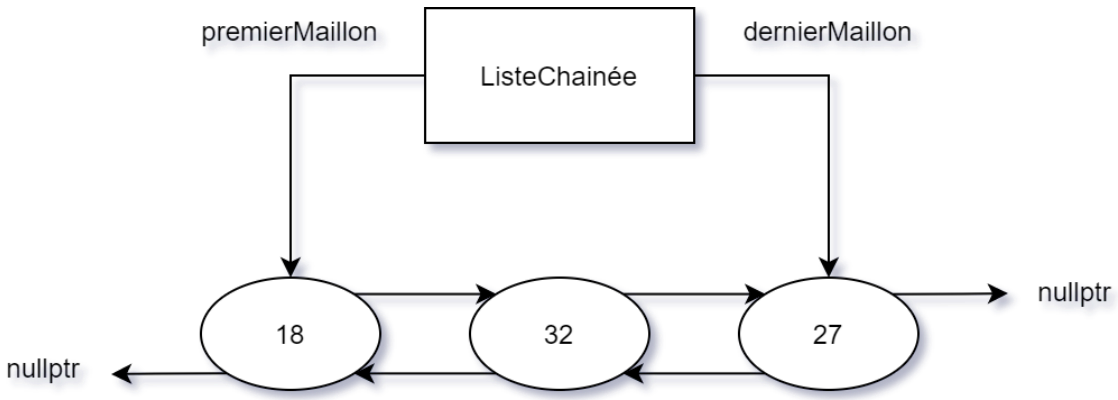
> **Note**
> La liste étant vide à sa création, les attributs **`firstLink`** et **`lastLink`** seront initialisées à **`nullptr`**.

add

➢ Implémentez la méthode **`add`** qui ajoute un maillon à la fin de la liste chainée, maillon créé à partir de la valeur passée en paramètre.

> **Attention**
> Deux cas de figure sont à prendre en compte ici : ajout dans une liste vide et ajout dans une liste non vide.





size

- Implémentez la méthode **`size`** qui parcourt la liste chainée du premier au dernier élément pour en déterminer la taille.
- Dans la fonction **`main`**, créez une liste chainée et ajoutez plusieurs valeurs avant d'en afficher la taille.
- Testez le bon fonctionnement et ne passez à la suite que lorsque tout est bon.

getLink

- Implémentez la méthode **`getLink`** qui retourne le maillon dont la position est passée en paramètre.

  La méthode lèvera une exception si la position donnée n'est pas cohérente avec la taille de la liste.

  La méthode **`getLink`** retournera un **`Link*`**.

insert

- Implémentez la méthode **`insert`** qui crée un maillon à partir de la donnée passée en paramètre et insère le maillon créé avant le maillon dont la position est passée en paramètre.

  Une exception sera levée si la position donnée n'est pas cohérente avec la taille de la liste.

- Testez le bon fonctionnement de la méthode **`insert`** depuis la fonction **`main`**.

remove

- Implémentez la méthode **`remove`** qui supprime le maillon dont la position est passée en paramètre.

  Une exception sera levée si la position donnée n'est pas cohérente avec la taille de la liste.

- Testez le bon fonctionnement de la méthode **`remove`** depuis la fonction **`main`**.

operator[]

- Implémentez l'opérateur [ ] qui retourne la donnée du maillon dont la position est donnée en paramètre.

  Une exception sera levée si la position donnée n'est pas cohérente avec la taille de la liste.

- Testez le bon fonctionnement de l'opérateur [ ] depuis la fonction **`main`**.

Destructeur

Depuis le début de l'exercice, vous faites des **`new`** en pagaille. Il va être temps de faire le ménage et de détruire ce que votre liste a créé en mémoire, lorsqu'elle est supprimée.

- Implémentez le destructeur de la classe **`LinkedList`**.
- Ajoutez également un destructeur à la classe **`Link`** dont le seul rôle sera d'afficher dans la console un message du type « Maillon détruit (valeur de la donnée) ».
- Testez le bon fonctionnement et vérifier que tous les maillons sont bien détruits.
- Profitez en pour vérifier la méthode remove…

LISTE CHAINEE GENERIQUE

Bravo ! Vous avez créé une structure de données qui gère une liste d'entiers. Cependant, si vous avez besoin de manipuler des **`floats`** ou des **`strings`**, vous devrez refaire des structures de données pour chacun de ces types.

Heureusement, C++ apporte la généricité et propose d'utiliser des **`templates`** de classes pour résoudre ce problème.

- En reprenant le cours, transformez le code de votre liste chainée pour en faire un **`template`**.
- Testez le résultat en créant et manipulant des listes chainées de différents types.