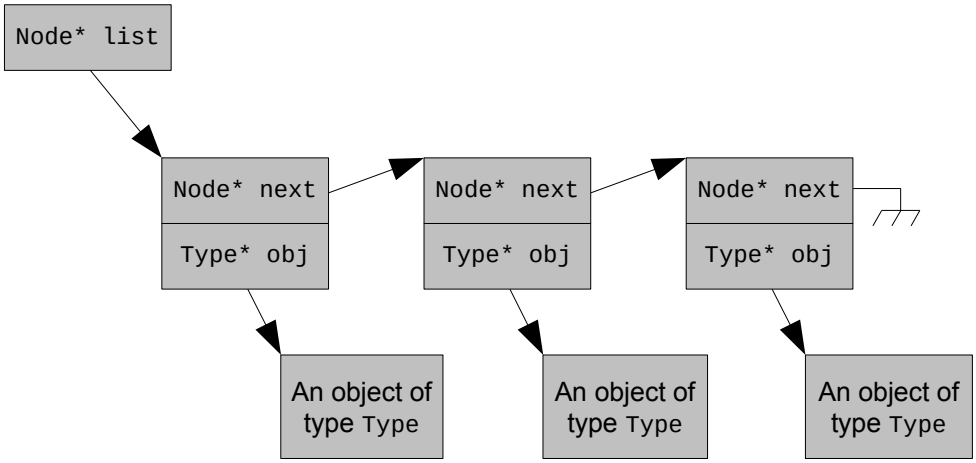
1. Principe d'une liste chaînée

Les listes chaînées sont des structures de données très classiques, il est nécessaire de bien les comprendre.

Elles sont généralement utilisées lorsque l'on doit manipuler des collections d'objets dans lesquelles on est amené à faire de nombreuses insertions et/ou suppressions (imaginez ce que ça donnerait si l'on utilisait un tableau !).

Une liste chaînée est constituée d'un ensemble de nœuds, chacun composé de l'adresse d'un objet et de l'adresse du nœud suivant (ou `nullptr` si c'est le dernier). Il suffit donc de disposer de l'adresse du premier nœud de la liste pour pouvoir la manipuler (de la même façon qu'il suffit de disposer de l'adresse de la première cellule d'un tableau).



2. Liste de vecteurs

a) Écrire la classe `Node` minimale correspondant au schéma ci-dessus pour le type `Vector`

b) Écrire un constructeur prenant comme paramètre un Vector (réfléchir aux différents modes de passage de paramètres). Tester ce constructeur grâce à `gdb`

c) Implémenter les getters que vous jugez nécessaires

d) Implémenter les setters que vous jugez nécessaires

e) Réfléchissez à l'utilisation des constructeurs par défaut et de copie. Agissez en conséquence

3. Manipulation d'une liste

La classe `Node` que vous venez d'implémenter suffit pour manipuler une liste. Cependant, son utilisation ne serait pas très pratique car son API est pauvre. Par exemple, si l'utilisateur veut ajouter un élément à la fin d'une liste, il lui faudra d'abord créer un `Node` puis le lier au dernier de la liste (sans compter le cas particulier où la liste est vide !). Il serait plus efficace et plus sûr de lui lui fournir un outil clés-en-main.

En pratique, on définit une classe (e.g. `List`) qui s'appuie sur la classe `Node` pour fournir à l'utilisateur une interface plus complète. La classe `List` contient un attribut (e.g. `head_`) qui pointe sur le premier élément de la liste. Cette classe peut aussi contenir un attribut (e.g. `nb_elts_`) contenant le nombre d'éléments de la liste. On a donc 2 classes : `Node`, dont le rôle est en premier lieu de stocker les données, et `List`, dont le rôle est de fournir à l'utilisateur des méthodes de manipulation de la liste.

a) Créer une classe `List` minimale qui permettra de manipuler une liste

b) Ajouter les méthodes `PushBack` (resp. `PopBack`) permettant d'ajouter (resp. supprimer) un élément en fin de liste

c) Implémenter une méthode `Insert` permettant d'insérer un élément dans la liste

d) Le constructeur de `Node` n'est, et ne devrait être utilisé que dans la classe `List`. Trouvez un moyen d'imposer cette contrainte

e) Faites les modifications nécessaires pour obtenir une liste doublement chaînée (chaque nœud possède un pointeur à la fois sur le nœud suivant et sur le nœud précédent)