UNIVERSITÉ DE SHERBROOKE
DÉPARTEMENT D'INFORMATIQUE

IFT 339

Laboratoire#3 : La chaîne de tableaux

Hiver 2019

Le but de ce laboratoire est de vous familiariser davantage avec les listes chaînées et les tableaux, ainsi que de continuer à pratiquer les classes, les pointeurs et l'allocation dynamique. Vous devez implémenter le type abstrait Liste avec une *chaîne de tableaux*.

La date de remise officielle de ce labo est le vendredi 1er mars 2019 (selon le plan de cours). Toutefois, je vais tolérer les retards jusqu'au 10 mars à 23h59. Vous devez remettre, sur `opus.dinf.usherbrooke.ca`, le fichier chainetab.h développé au cours de ce laboratoire.
NOTE : il n'y a pas de script automatique de vérification sur turnin cette fois-ci.

Implémentation d'une chaîne de tableaux (100 points)

Nous vous demandons d'implémenter le type abstrait Liste avec une *chaîne de tableaux*. Le principe de cette structure de données est similaire au *deque* de la SL qui utilise un tableau de tableaux, à l'exception que l'on remplace le tableau au premier niveau par une liste doublement chaînée.

Les cellules de cette liste chaînée pointent toutes sur un tableau de la même taille et les éléments sont stockés dans ces tableaux. En anglais, cette structure s'appelle une *Unrolled Linked List*[1], et je n'ai pas trouvé de terme français. Pour le nom de la classe, j'ai choisi pour vous le nom chainetab. La Figure 1 illustre l'état possible d'une chaîne de tableaux.

La structure contient un pointeur DEBUT_CHAINE vers la première cellule (ou nullptr si vide), ainsi qu'un pointeur FIN_CHAINE vers la dernière cellule (ou nullptr si vide). Tous les tableaux ont la même taille TABSIZE, et ils contiennent toujours au moins un élément. Le premier tableau peut avoir de l'espace inutilisé au début (pour des *push_front*), et le dernier tableau peut avoir de l'espace inutilisé à la fin (pour des *push_back*). La variable membre POSPREMIER indique la première position du premier tableau qui est utilisée, et POSDERNIER la dernière position du dernier tableau utilisée. Ces deux variables sont indicées à 0.

Vous devez compléter le fichier chainetab.h avec l'implémentation des méthodes[2] suivantes :

1. `https://en.wikipedia.org/wiki/Unrolled_linked_list`
2. Le terme *méthode* veut dire "fonction membre".

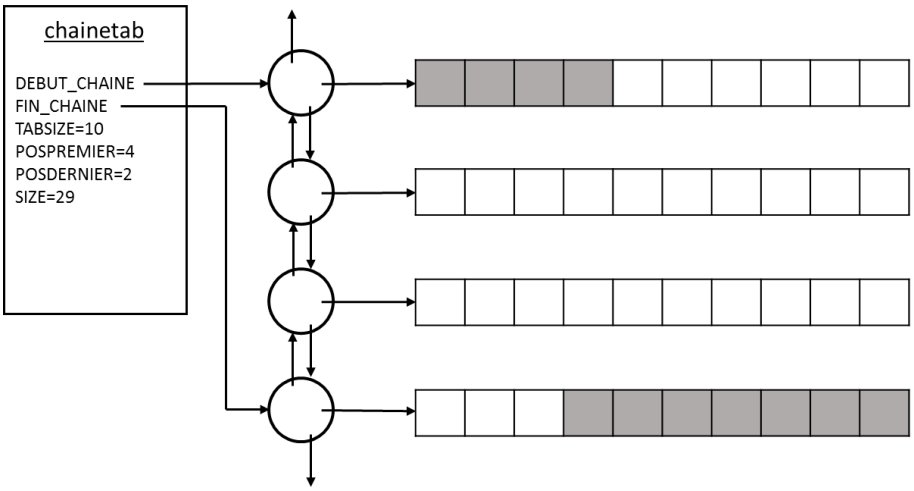

FIGURE 1 – État général d'une chaîne de tableaux. Les pointeur dans le vide sont des nullptr.

- chainetab(size_t tabsize) : le constructeur reçoit la taille des sous-tableaux pointés par vos cellules. Cette taille ne peut plus être modifiée par la suite (sinon, il faudrait vous faire implémenter une fonction pour modifier cette taille, ce qui peut être un bon exercice mais ce n'est pas demandé). La liste est initialement vide.
- operator[ ](size_t i) : retourne le i-ème élément. Ceci demande de parcourir la chaîne jusqu'au tableau contenant ce i-ème élément, puis de calculer quelle position du sous-tableau retourner. Vous n'avez pas besoin de faire de vérification sur la validité de i : c'est le problème de l'appelant.
- push_front(const TYPE& val) : ajoute un élément au début de la liste. Si le premier sous-tableau a un espace de libre en avant, vous le prenez. Sinon, il faut ajouter un nouveau sous-tableau en tête de liste et utiliser son dernier espace (voir page suivante).
- push_back(const TYPE& val) : ajoute un élément à la fin de la liste. Si le dernier sous-tableau a un espace de libre à la fin, vous le prenez. Sinon, il faut ajouter un nouveau sous-tableau en fin de liste et utiliser son premier espace (voir page suivante).
- pop_front() : enlève le premier élément de la liste. Vous n'avez pas à vérifier si la liste est vide. Supprimez une cellule si elle devient inutile.
- pop_back() : enlève le dernier élément de la liste. Vous n'avez pas à vérifier si la liste est vide. Supprimez une cellule si elle devient inutile.
- clear() : efface tout, nettoie la mémoire, met la liste vide.

La première insertion entraîne la création d'une cellule et utilise la première position de son sous-tableau, que ce soit un push_front() ou un push_back(). Certaines parties du code ont déjà été implémentées pour vous. Il est recommandé de les regarder si vous avez besoin d'inspiration. La méthode affichier_contenu() peut être utile. Vous avez le droit d'ajouter des méthodes privées, au besoin, mais les méthodes qui sont présentes doivent rester. La page suivante illustre comment faire les insertions.

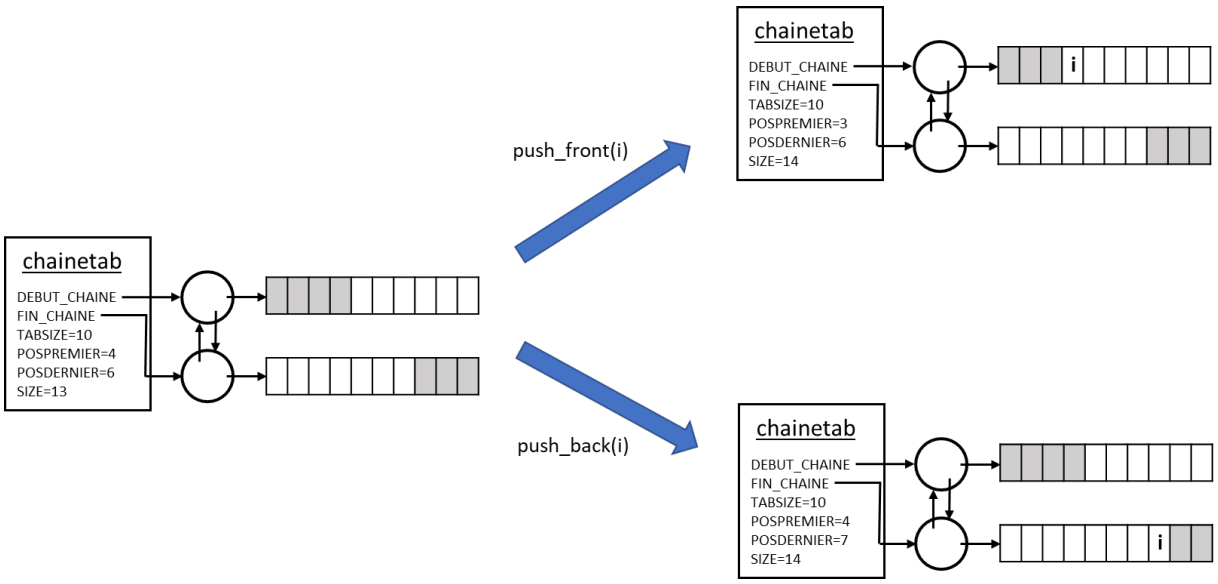

FIGURE 2 – Insertion, cas généraux.

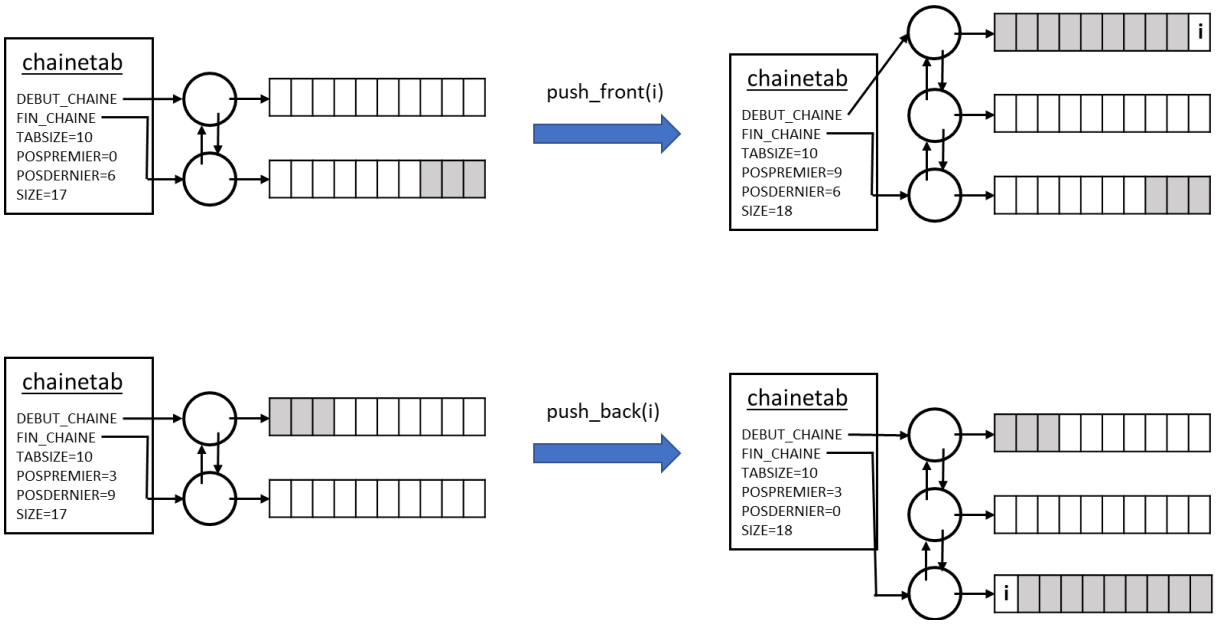

FIGURE 3 – Insertion, cas limite.

Un peu de réflexion (0 points)

Il est sain de réfléchir aux aspects de complexité de vos implémentations (je ne demande aucune réponse dans cette section - vous pouvez vous contenter d'implémenter sans réfléchir si vous voulez). Combien de temps prend un push_back, dans le pire cas ? La réponse dépend du temps qu'il faut au système pour créer un tableau de taille TABSIZE. Quel est l'avantage du chainetab par rapport au vector ? Par rapport au deque tel que vu en classe ? Et en quoi le vector ou le deque sont-ils meilleurs que le chainetab ? Aurait-il été plus intéressant de faire un tableau de listes chaînées ?

Remise du travail

Ne remettez qu'un seul fichier dont le nom doit être `chainetab.h`. Ne remettez pas d'autre fichier ni surtout d'exécutable. Votre fichier sera recompilé. Il ne doit y avoir qu'une seule remise par équipe, à partir d'un seul CIP. Les noms des co-équipiers doivent être clairement indiqués en entête de chaque fichier soumis.

Pour soumettre votre travail, connectez-vous, dans un navigateur, au serveur `opus.dinf.usherbrooke.ca` en utilisant l'un de vos CIP, puis choisissez le cours `IFT339` et le projet `labo3`. Chargez votre fichier `chainetab.h` et soumettez-le.