Cégep de Sainte-Foy
Algorithmique avancée – 420-W31-SF

Exercice 10 – La pile en C++

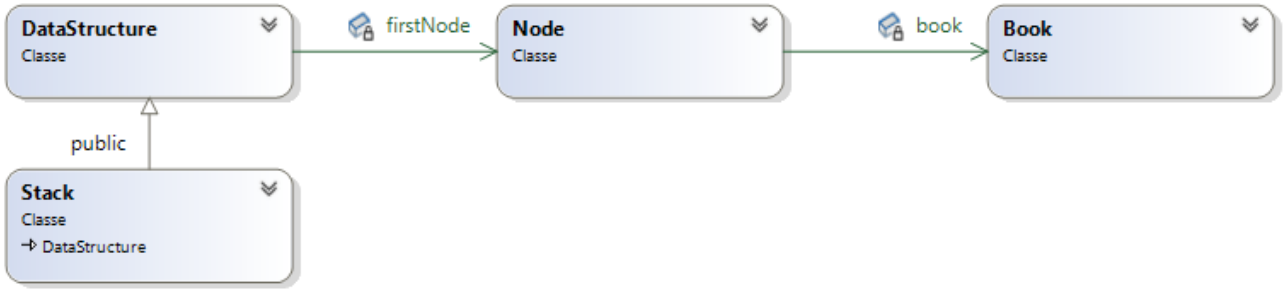
20 mai 2025

Préparé par
Daniel Huot et Pierre Poulin

1 Travail à effectuer

Récupérez le projet sur GitHub et répondez aux questions suivantes.

Cet exercice a pour but de vous faire coder une première structure de données dynamique : la pile (*stack*). Le contenu qui sera empilé/dépilé sera un livre, dont la classe est fournie avec l'énoncé. Le diagramme de classes de l'exercice n'est pas étranger à celui vu au cours :



1.1 La classe `DataStructure`

Prenez quelques instants pour analyser le code de la classe `DataStructure`.

a) Complétez ensuite le code des méthodes `size()` et `isEmpty()`.
b) Le destructeur de la classe est déclaré `virtual`. Indiquez en commentaires dans le code pourquoi il en est ainsi.

1.2 La class `Stack`

Le but de l'exercice est de coder les méthodes de la classe `Stack` pour faire fonctionner la structure c'est-à-dire :

c) D'avoir un stockage des données conforme (les contenus sont réellement empilés sur un **`push`** et dépilés sur un **`pop`**)
d) De respecter la composition du diagramme ci-haut
e) D'avoir un programme sans fuite de mémoire (*memory leak*)

Conseil : Implémentez la méthode d'affichage fournie. En parcourant la pile depuis le premier nœud, vous aurez un visuel à la console qui vous aidera grandement.?

Afin de coder la classe `Stack`, voici à nouveau les spécifications vues au cours (sous forme de « *checkList*). Le défi de ce travail est sans contredit le **chaînage des pointeurs**. Il ne faut pas hésiter à dessiner sur du papier notre approche avant de commencer à lier des pointeurs de façon arbitraire, ce qui peut rapidement conduire à des anomalies de fonctionnement.

Spécifications de la Pile

Spécifications	☑
Lorsque le pointeur vers le premier nœud est **`nullptr`**, c'est l'indicateur que la pile est vide.	
La méthode **`push`** va créer un nœud, y insérer le contenu reçu et effectuer les chaînages pour empiler le nœud adéquatement.	
La méthode **`top`** va retourner le contenu stocké dans le premier nœud sans modifier le contenu de la pile. Appeler **`top`** sur une pile vide provoque le lancement d'une exception à l'aide de l'instruction `throw std::runtime_error("Empty Stack");`	
La méthode **`pop`** va détruire le nœud se trouvant sur le dessus de la pile et de refaire les chaînages adéquatement. Appeler **`pop`** sur une pile vide provoque aussi le lancement d'une exception (voir point précédent)	
Si la pile est détruite et qu'il reste des nœuds, la pile devra respecter sa composition forte. Indice : faire des appels de **`pop`** à répétition.	

1.3 Utilisation et tests unitaires

La solution de cet exercice **ne sera pas donnée** car vous êtes en train de construire du code essentiel au TP2. Votre pile sera évaluée lors de celui-ci. Il est fortement conseillé de terminer cet exercice avant le TP2 car **l'allocation du temps du TP2 prend pour acquis que votre structure de pile fonctionne**. En d'autres mots, vous aurez à coder la pile, donc aussi bien le faire maintenant.

Il va de soi que la robustesse de votre structure est primordiale pour réussir le TP2. Une série de tests complète est donc à produire sur votre pile et fera partie de son évaluation. Afin de vous aider à débuter, un fichier de départ (dans le projet de tests) est donné avec ce travail pour vous guider sur les tests à écrire. Il est fortement recommandé d'écrire vos tests **au fur et à mesure** que vous progressez dans la pile, le but étant d'arriver au TP2 avec une structure fonctionnelle et sans faille, autrement, le problème ne pourra pas être résolu.

Les fichiers fournis contiennent un fichier nommé **`DescriptionsTests.docx`**. Commencez par y inscrire une description de tous les cas de tests pertinents pour la pile. Inscrivez également dans la dernière colonne s'il s'agit d'un cas général, limite ou d'exception.

Vous êtes libres de générer vos tests de la manière que vous souhaitez. Cependant, pour chaque test que vous ajouterez, inscrivez en commentaire au-dessus du test à quelle situation du fichier **`DescriptionsTests.docx`** il se rapporte.

1.4 Défi

La codification du constructeur de copie de la pile représente un bon défi. Essayez de le coder et écrivez un test unitaire pour vous assurer qu'il fonctionne adéquatement

Vous risquez de faire une première version qui insère les éléments … dans l'ordre inverse!

Conseil : Une solution récursive est très appropriée ici. Je le répète : c'est un bon défi!

2 Modalités de remise

Remettez votre projet C++ sur LÉA, dans la section travaux, à l'intérieur d'une archive *Zip*. Supprimez les dossiers temporaires `.vs`, `extern`, et `Release` et `Debug` pour tous les projets.