CÉGEP DE SAINTE-FOY
ALGORITHMIQUE AVANCÉE – 420-W31-SF

Exercice 11 – La file en C++

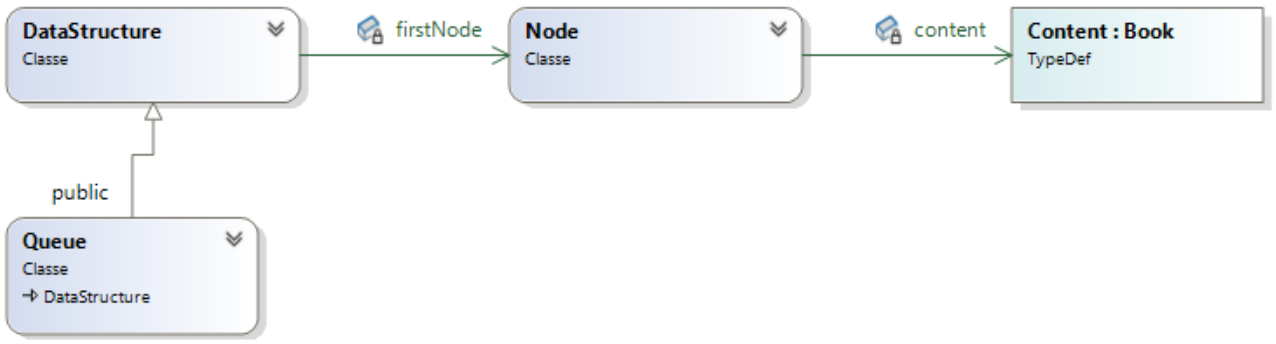
13 août 2025

Préparé par
Daniel Huot et Pierre Poulin

1 Travail à effectuer

Récupérez le projet sur GitHub puis répondez ensuite aux questions suivantes.

Cet exercice a pour but de vous faire coder une seconde structure de données dynamique : la file (*queue*). Le contenu qui sera ajouté/retiré sera encore 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 :



où ici la classe `Content` sera `Book`.

1.1 La classe `DataStructure`

La classe `DataStructure` est la même que pour la pile. Vous n'avez pas à la modifier.

1.2 La class Queue

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

a) D'avoir un stockage des données conforme (les contenus sont réellement mis en file et respectent le principe du « premier arrivé, premier servi ».
b) De respecter la composition du diagramme ci-haut
c) D'avoir un programme sans fuite de mémoire (*memory leak*)

Attention : Le codage de la Queue est un peu plus technique que celui de la classe `Stack`. N'hésitez pas à vérifier votre algorithme sur un dessin. Sinon vous allez colmater des brèches en empirant la situation.

Afin de coder la classe Queue, revoici les spécifications vues au cours (sous forme de « *checkList*). Le défi de ce travail est encore le **chaînage des pointeurs**.

Spécifications de la file

Spécifications	☑
La file est vide lorsque sa taille est zéro.	
La méthode `push_back` va créer un nœud, y insérer le contenu reçu et effectuer les chaînages pour enfiler le nœud adéquatement en fin de file.	
La méthode `front` va retourner le contenu stocké dans le premier nœud sans modifier le contenu de la file. Appeler `front` sur une file vide provoque le lancement d'une exception à l'aide de l'instruction `throw std::runtime_error("Empty queue");`	
La méthode `pop_front` va détruire le nœud se trouvant en avant de la file et de refaire les chaînages adéquatement. Appeler `pop_front` sur une file vide provoque aussi le lancement d'une exception (voir point précédent)	
Si la file est détruite et qu'il reste des nœuds, la file devra respecter sa composition forte. **Indice** : faire des appels de `pop_front` 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 file fonctionne**. En d'autres mots, vous aurez à coder la file, 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 file 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 file, 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 file. 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.

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`, `Release` et Debug **pour tous les projets**.