
TD 1

Après une pile, une file générique

1.1 Objectif

L'objectif de cet exercice est de montrer que l'on peut réaliser du code utile par simple imitation. C'est aussi une occasion de mettre en oeuvre certains des mécanismes principaux de C++ qui ont été survolés dans l'introduction et de prendre en main l'environnement de développement en C++ (éditeur, compilateur, *makefiles*, etc.).

1.2 Énoncé du problème

Vous trouverez dans le répertoire `Fifo` que vous venez de recopier, le source de la classe générique `Stack` vue en cours accompagné d'un programme de test, ainsi que d'une *Makefile*. Compilez (il suffit de faire *make*) et exécutez le programme de test (`main_stack.exe`) dans une fenêtre **shell**. Le programme attend que vous entriez des caractères et que vous terminiez par une fin de fichier (au terminal, `^D` seul sur une ligne).

En prenant cette classe `Stack` comme modèle, on vous demande d'écrire une classe `Fifo`, également générique, qui réalise une file avec la stratégie « premier entré, premier sortie » (*first in, first out*). Une telle classe modélise une file d'attente (dans notre cas, elle est de dimension maximale fixe).

Votre classe `Fifo` devra être dotée des propriétés suivantes :

- une taille maximale `N` (une constante),
- un constructeur par défaut (c'est-à-dire sans paramètre) qui construit une file qui ne contient aucun élément (utile),
- deux opérations principales : `put()` qui place un élément dans la file, et `get()` qui retire l'élément le plus ancien de la file et retourne sa valeur,
- deux prédicats `is_full()` et `is_empty()`, comme dans `Stack`.

Bien entendu vous devrez aussi lever des exceptions en cas d'opérations impossibles. Enfin votre file devra être gérée comme un *tampon circulaire* afin que la place libérée par `get()` puisse être réutilisée par `put()`.

Dans votre programme de test, essayez d'instancier votre classe `Fifo` avec plusieurs types très différents (`char`, `double`, `pointeurs`...).

1.3 Note sur l'instanciation des classes génériques (templates)

Lorsque le compilateur instancie une classe générique comme `Stack` ou `Fifo`, il produit le code source des classes correspondant aux divers paramètres d'instanciation¹. À cause du mécanisme de compilation séparée de C/C++, le compilateur a besoin non seulement de la définition de la classe, mais encore de la définition (du corps) de toutes ses fonctions-membres. Ceci explique que, contrairement aux (bonnes) habitudes, la distinction entre fichier d'entête (`.h`) et fichier de corps (`.cpp`) est, dans le cas des templates, sans grande signification puisque le compilateur a besoin du contenu des deux. Si l'on veut cependant conserver cette séparation on peut alors inclure le fichier `.cpp` dans le fichier `.h` comme il est fait dans l'exemple de la classe `Stack`. Bien entendu, dans ce cas, le fichier `Stack.cpp` ne doit pas être compilé séparément puisqu'il le sera lors de son inclusion (voir la `Makefile`).

Noter également qu'en C++, il n'y a jamais *aucune* excuse pour éviter de protéger les fichiers d'entête contre une inclusion multiple. Vous *devez* donc encadrer vos fichiers `.h` de la séquence magique

```
#ifndef <un identificateur unique qui rappelle le nom du fichier>
#define <le même identificateur>
<le contenu du fichier>
#endif
```

¹ L'instanciation des *templates* est un mécanisme analogue à la macro-génération, mais il s'agit d'une **macro-génération dirigée par les types**.