UNIVERSITÉ LIBRE DE BRUXELLES ULB

FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

INFO-F105

Deuxième projet (C++)

Année académique 2017–2018

Résumé

Ceci est l'énoncé du quatrième projet du cours « Langages de programmation 1 ». On vous demande de représenter un magasin et son contenu avec des types de données abstraites (ADT). Il faudra principalement tenir compte de trois types d'objets conceptuels : un objet en vente, un rayon dans lequel les premiers objets se trouvent et un magasin qui contient des rayons.

Introduction

Une chaîne de supermarchés veut se lancer dans la digitalisation. Elle vous demande d'implanter une représentation d'un supermarché à haut niveau. Seuls trois types d'objets doivent être pris en compte : un magasin, un rayon, et un article. L'abstraction du magasin sera interactive dans le sens où des objets peuvent être ajoutés ou supprimés, mais seulement dans un sens limité, car nous ne prenons pas en compte les clients réels, etc.

Modalités de réalisation

Pour réaliser la création des trois ADT susmentionnés, trois classes sont essentielles. Si vous ressentez le besoin de créer des classes ou des méthodes supplémentaires, n'hésitez pas, mais adhérez pleinement à celles décrites.

La classe *Store*, qui comportera un nom associé au magasin et une collection de rayons.

Une classe *Aisle*, qui représentera un rayon, et qui contiendra donc une collection d'articles, une capacité maximale d'articles que celui-ci peut contenir, ainsi qu'un numéro de rayon.

Finalement, une classe *Item* qui servira comme ADT pour les articles. Un article a un nom et un prix.

Chaque classe contiendra les méthodes décrites ci-dessous.

Classe *Store* :

- `Store(string, int, int)` constructeur qui prend le nom, le nombre de rayons et le nombre d'articles par rayon comme arguments.
- `getActifs()` renvoie la valeur totale de tous les articles dans le magasin (somme).
- `describe()` renvoie une description des articles présents dans le magasin : prix moyen, prix maximal, prix moyen par allée, etc. (vous êtes libre de proposer d'autres mesures de description pertinentes).
- `receiveShipment([item_0, int_0, ..., item_n, int_n])` prend une liste contenant des articles et leurs quantités respectifs et les ajoute au magasin si celui-ci a suffisamment d'espace. S' il n'y a pas assez d'espace, une erreur est renvoyée.
- `sell([[item_0, int_0, ..., item_n, int_n]])` prend une liste contenant des articles et leurs quantités respectives et les retire du magasin, si celui-ci en a suffisamment. Sinon, elle met cette quantité à zéro.
- `getAisle(int)` renvoie un pointeur vers le rayon avec le numéro de rayon correspondant.

Classe *Aisle* :

- `Aisle(int, int)` constructeur qui prend le numéro du rayon et la capacité de ce rayon.
- `describe()` imprime les éléments présents au rayon, ainsi que leurs prix et leurs quantités.
- `add(item, int)` ajoute le nombre spécifié d'articles au rayon, si celui-ci a assez de capacité libre. Sinon une erreur est renvoyée.
- `remove(item, int)` supprime le nombre spécifié d'éléments du rayon s'il y en a suffisamment.
- `getNumber()` renvoie le numéro de rayon.

Classe *Item* :

- `Item(string, float)` constructeur qui prend le nom de l'article et son prix.
- `getName()` renvoie le nom de l'article.
- `getPrice()` renvoie le prix de l'article.

Les implémentations doivent être placées dans des fichiers cpp, les déclarations dans des fichiers hpp.

À titre complémentaire, on vous demande d'écrire un fichier qui teste votre implantation. Ce fichier contiendra des appels à toutes les méthodes des différentes classes (à l'exception des constructeurs qui sont déjà appelés dans une autre fonction, comme le constructeur du *rayon* à l'intérieur du constructeur du *magasin*). Ce fichier se nommera *Test.cpp*.

Essayez de penser à tous les cas limites et effets de bord possibles et couvrez-les dans votre implantation.

Le projet doit être écrit en C++ strictement standard. Vous devez ainsi l'avoir compilé en respectant les directives indiquées sur l'UV (version du compilateur, options, etc.), sans susciter le moindre avertissement (*warning*).

Compilation

Votre code doit se compiler en utilisant le « *Makefile* » suivant :

PROGRAMME 1 – Makefile

```
1  FLAGS= -std=c++14 -ggdb3 -Wpedantic -Wall -Wextra -Wconversion -Weffc++ -Wstrict-null
       -sentinel -Wold-style-cast -Wnoexcept -Wctor-dtor-privacy -Woverloaded-virtual -
       Wsign-promo -Wzero-as-null-pointer-constant -Wsuggest-final-types -Wsuggest-
       final-methods -Wsuggest-override
2  CXX= g++
3  MAIN= Test.cpp
4  OUT= CalTest.out
5
6  all: $(OUT)
7
8  $(OUT): Store.o Aisle.o Item.o $(MAIN)
9    $(CXX) $(FLAGS) $(MAIN) Store.o Aisle.o Item.o -o $(OUT)
10
11 Item.o: Item.cpp Item.hpp
12   $(CXX) $(FLAGS) -c Item.cpp
13
14 Aisle.o: Aisle.cpp Aisle.hpp
15   $(CXX) $(FLAGS) -c Aisle.cpp
16
17 Store.o: Store.cpp Store.hpp
18   $(CXX) $(FLAGS) -c Store.cpp
```

```
19
20 .PHONY : clean
21 clean:
22   rm *.o *.out;
```

ATTENTION !

Tous les *warnings* non résolus qui n'ont pas de justification explicite (par un commentaire dans le code, par exemple) seront sanctionnés d'un point sur la note finale. Et un code qui ne compile pas ne sera pas corrigé.

Indications complémentaires

Vous devez soumettre un fichier compressé `.zip` qui contient un seul dossier. Ce dossier doit s'appeler <nom>_<prenom> (pas de majuscules ni de caractères accentués), le fichier zip doit porter le même nom. Par exemple, Alan Turing doit soumettre un fichier `turing_alan.zip` qui contient le dossier `turing_alan` avec son projet.

Les fichiers à soumettre sont :

- `Store.hpp` et `Store.cpp` contenant le code C++ de l'ADT ;
- `Aisle.hpp` et `Aisle.cpp` contenant le code C++ de la classe *Aisle* ;
- `Item.hpp` et `Item.cpp` contenant le code C++ de la classe *Item* ;
- `Test.cpp` qui contient une séries de test pour l'ADT *Store*.
- `Makefile` qui produit un fichier exécutable `CalTest.out`.

Consignes pour la remise du projet

À respecter scrupuleusement !

1. Chaque fichier doit commencer par un commentaire indiquant **votre nom** et **votre section**.
2. Le fichier est à remettre exclusivement via l'UV.
3. Un fichier unique `.zip` doit être remis.
4. Votre code doit être **commenté**.
5. Date limite : le **mardi 22 mai 2018** avant **8 heures** ;
   Après ce moment, les projets sont considérés comme **en retard** et ne seront plus acceptés.
6. Questions : nversbra@ulb.ac.be