Licence Informatique — Université de Nantes
ASD2 (X22I020) — 2019–20

Projet : les consignes à bagages

(encadré sur les 3 dernières séances de TP)

Règles à respecter :

1. Ce projet doit être réalisé en **binômes**, c'est-à-dire dire en groupes de 2 personnes.
2. Un fichier archive au format `zip` ou `tar.gz` contenant le rapport au format `pdf` et les programmes sources `C++` (sans les binaires) doit être déposé sur `madoc` **au plus tard le jeudi 30 avril 2020 à 18h** par un et un seul membre du binôme dans l'espace devoir correspondant à son groupe de TP.

Partie 1

Une consigne à bagages est représentée par une liste de casiers. On peut déposer un bagage dans un casier libre et ainsi obtenir un ticket en retour. Avec un ticket, on peut récupérer le bagage correspondant. Un utilisateur ne sait pas où se trouve son bagage. En supposant que la consigne n'est pas pleine, le choix du casier lors du dépôt d'un bagage respecte la stratégie suivante : le bagage est mis dans le casier libre dont la dernière utilisation est la plus ancienne.

Le travail dans cette partie consiste à développer deux structures de données représentant les tickets et les consignes. Pour représenter les bagages, on peut définir le type `bagage` comme un alias de type, par exemple sur celui des chaînes de caractères.

Un ticket encapsule un code secret généré aléatoirement lors de sa construction. Ce code peut être représenté par une chaîne de caractères d'une longueur fixe assez grande. La SDA du type `ticket` comporte les 4 opérations suivantes : un constructeur sans paramètres, le calcul d'un code de hachage, un test d'égalité et un test de différence de deux tickets. Dans le code `C++`, on surchargera les opérateurs `operator==` et `operator!=` pour implémenter ces tests. De plus, on spécialisera le type standard `std::hash` pour le type `ticket`, ce qui permettra par la suite de manipuler des tables de hachage dont les clés sont des tickets.

Le type `consigne` peut être représenté par une liste de casiers dont le nombre est donné lors de sa construction. La SDA du type `consigne` comporte les 4 opérations suivantes : un constructeur, un test permettant de savoir si une consigne est pleine (aucun casier n'est libre), une opération pour déposer un bagage et une opération pour récupérer un bagage. Des opérations pourront être ajoutées dans la SDC pour les besoins de l'implémentation (donc par le biais de méthodes privées dans le code `C++`). Un objectif important est de pouvoir implémenter les opérations de dépôt et de retrait d'un bagage **en temps constant** (ou constant en moyenne). Ainsi, on peut se poser les deux questions suivantes. Comment trouver un casier libre lors d'un dépôt, sachant en plus qu'on doit choisir celui dont la dernière utilisation est la plus ancienne ? Comment retrouver le casier où se trouve un bagage avec le ticket qui a été émis lors de son dépôt ?

Le travail demandé dans cette partie se décompose ainsi :

— Dans le rapport, définir les SDA des types `ticket` et `consigne`. Cela consiste à réaliser un tableau des opérations et, pour chaque opération, à donner sa signature, son rôle et sa précondition.

— Dans le rapport, définir les SDC des types `ticket` et `consigne`. Cela consiste pour chaque SDC à définir un type de données en montrant sa représentation en mémoire et à donner les complexités des opérations (donner les ordres de grandeur sur la base du code développé, sans écrire les algorithmes en pseudo-code).

— Implémenter les classes `ticket` et `consigne` en séparant le code en fichiers `hpp` et `cpp`. Développer également un programme de test et un `Makefile` pour compiler l'application. La vérification des pré-conditions se fera par le biais d'assertions ou d'exceptions standards.

On donne un barême à titre indicatif sur 12 points : 2 pour la SDA/SDC `ticket`, 5 pour la SDA/SDC `consigne` et 5 pour le code `C++`. On utilisera les critères d'évaluation suivants. Une SDA doit être clairement définie. Une SDC doit être performante et on regardera particulièrement ici la complexité temporelle des opérations. Le code doit respecter les standards du `C++` et de la programmation objet : séparation du code, visibilité des membres dans les classes, encapsulation, surcharge des opérateurs si besoin, bonne utilisation des structures de données de la librairie standard, lisibilité du code, commentaires, indentation, etc.

Enfin, le meilleur conseil à donner est de bien réfléchir selon les objectifs fixés avant de commencer à coder !

Partie 2

On considère maintenant que les casiers et les bagages ont des volumes en litres. Par exemple, une consigne de n casiers peut être définie à partir d'une liste de couples (n_i, v_i) telle qu'elle possède n_i casiers de volume v_i pour tout i, avec $n = \sum_i n_i$. Ainsi, un bagage de volume v doit être déposé dans un casier de volume suffisant. De plus, on s'intéresse ici à une stratégie de dépôt optimale : parmi tous les casiers libres, on choisit le casier de volume minimal et supérieur à v dont la dernière utilisation est la plus ancienne.

Un bagage possède donc un volume. On pourra ainsi implémenter une classe abstraite `bagage` qui pourra donner lieu à des sous-classes pour avoir des types de bagages différents. Cela nécessitera de gérer des pointeurs sur des bagages au sein d'une consigne.

Par rapport à la partie précédente, le type `ticket` ne change pas. Le type `consigne` évolue en `vconsigne` avec une opération supplémentaire dans la SDA permettant de tester s'il existe au moins un casier libre dont le volume est supérieur à un volume donné. La SDC doit évoluer pour bien sûr prendre en compte le volume des casiers mais aussi pour réaliser la stratégie de dépôt optimale décrite ci-dessus de manière très efficace. Pour cela, faut-il manipuler des listes ordonnées ? Des files avec priorité ? Toute latitude est laissée pour explorer ces idées et bien d'autres encore.

Le travail demandé dans cette partie se décompose ainsi :

— Dans le rapport, définir la SDA du type `vconsigne`.

— Dans le rapport, définir la SDC du type `vconsigne`. Justifier la complexité des opérations de la SDA et expliquer pourquoi cette SDC a été choisie.

— Implémenter les classes `bagage` et `vconsigne`.

On donne un barême à titre indicatif sur 7 points : 1 pour la SDA, 3 pour la SDC et 3 pour le code.

Un dernier point

Avec pour le moment un barême de 19 points il reste un point à donner. Disons donc que ce point sera attribué si le rapport est de bonne facture : bien écrit, complet, compréhensible, propre, avec de belles illustrations, sans fautes d'orthographe, etc. Disons même qu'il sera doublé si le rapport est excellent. Pour résumer, ce rapport comportera les sections suivantes :

1. SDA et SDC `ticket` ;
2. SDA et SDC `consigne` ;
3. SDA et SDC `vconsigne`.

Ultime vérification

Il est vivement conseillé de vérifier que le fichier archive est bien constitué avant de le déposer sur `madoc`. La décompression est-elle possible (sur une autre machine, par une autre personne) ? Contient-il le rapport au format `pdf` et le code `C++` (les sources sans les binaires) ? Décompressez, c'est le moment, et vérifiez !