Programmation Orientée Objet — Feuille 0 : Encapsulation

Aurélien MOINEL, Aymeric FERRON, Rémi DEHENNE, Sébastien DELPEUCH, Tom MOËNNE-LOCCOZ

Septembre 2020

Algorithme sur les tableaux

1. Compacter les éléments provoque un problème pendant le parcours du tableau dans la fonction `ab__aller_arret_suivant()`. En prenant le scénario exemple (`simple.c`), expliquer, avec un dessin, le problème.

Considérons la représentation mémoire d'un autobus de trois passagers présentée en Figure 1.

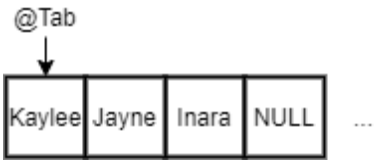

Fig. 1 : Représentation mémoire d'un autobus de trois passagers

Lors de la suppression d'un passager, la méthode actuelle remplace le contenu du pointeur vers ce passager par `NULL`, comme illustré en Figure 2.

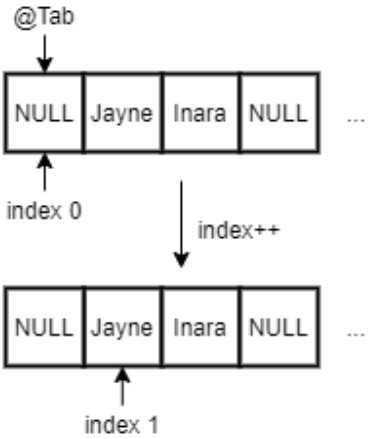

Fig. 2 : Représentation mémoire d'un autobus après suppression d'un passager sans compactage

Les cases du tableau sont parcourues itérativement de sorte qu'à chaque arrêt, tout passager puisse changer d'état.

En revanche, en compactant le tableau après la suppression d'un passager à l'indice n, les passagers suivants sont décalés d'un indice vers la gauche : le passager à l'indice $n+1$ est ainsi déplacé en indice n, etc (cf. Figure 3). Toutefois, l'index de parcours du tableau étant incrémenté à chaque itération, le prochain passager traité est celui d'indice $n+1$ (anciennement $n+2$) : le passager d'indice n (anciennement $n+1$) n'a donc pas été traité.

Ainsi, avec une telle implémenation, un passager ne peut pas changer d'état à un arrêt si le passager stocké à la case précédente du tableau est descendu du bus.

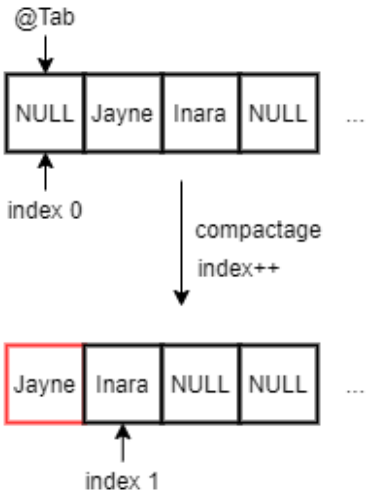

FIG. 3 : Représentation mémoire d'un autobus après suppression d'un passager avec compactage

La structuration du code

2. La recommandation habituelle est d'avoir des parties indépendantes. Dans le cours, cette indépendance est définie vis-à-vis de quoi ?

L'indépendance est définie par la possibilité de modifier la réalisation d'un code sans impacter le code qui l'utilise. Cela consiste à fournir une couche d'abstraction et à masquer les détails propres à l'implémentation.

Étude de la dépendance entre les trois parties (scénario, autobus, passager)

L'indépendance porte sur les deux points :

1. Le code de changement de l'état d'un passager standard.

- Quelle partie de code (`autobus.c` ou `ps_standard.c`) se charge de la modification de l'état d'un passager standard ?

Les fonctions internes du fichier `autobus.c` se chargent de la modification de l'état d'un passager standard.

- Expliquez vis-à-vis de quoi la partie autobus n'est pas indépendante de la partie passager standard.

La partie `autobus` n'est pas indépendante de la partie `ps_standard`, car les fonctions agissant sur `autobus` modifient directement les champs de la structure `ps_standard`. Ainsi tout changement opéré sur cette structure force à modifier le code des fonctions d'`autobus`. Il y a donc dépendance entre la partie `passager` et la partie `autobus`.

- Quelle solution ne modifiant pas le prototype des fonctions existantes faut-il adopter pour permettre l'indépendance ?

Le code client ne doit pas accéder directement aux champs de la structure `ps_standard`, mais utiliser des fonctions d'interface fixées et indépendantes de la réalisation. Pour cela, les attributs doivent être masqués.

- Quelle technique de programmation avez-vous appliquée pour permettre cette indépendance ?

L'encapsulation permet cette indépendance.

2. Sur le découpage en trois fichiers en-tête.

- Pourquoi ne pas avoir déclaré les deux structures de données et toutes les fonctions dans ces deux fichiers en-tête ?

Le fichier d'en-tête `__internes.h` contient l'ensemble des définitions de structures et des déclarations de fonctions propres à la réalisation et ne doivent pas être exposées au code « client » afin de garantir le principe d'encapsulation.

Les fonctions d'interface, accessibles au client, doivent ainsi nécessairement être déclarées un ou plusieurs autre(s) fichier(s) considérés « publics » : ici, `ps_standard.h` et `autobus.h`.

- Concluez sur l'intention du fichier du fichier en-tête `__internes.h`. Pourquoi en langage C n'est-ce qu'une intention ?

L'intention du fichier en-tête `__internes.h` est donc d'assurer le principe d'encapsulation dans le programme. Néanmoins, il ne s'agit que d'une intention, car rien n'empêche le client d'inclure le fichier en-tête directement dans les fichiers sources.