INFO0947 : Compléments de Programmation
Récursivité & Types Abstraits de Données

B. Donnet, S. Liénardy
Université de Liège

1 Type Abstrait

Dans ce travail, nous allons nous intéresser à un voyageur de commerce. [1] En particulier, il vous est demander de spécifier deux types abstraits, une *escale* (Sec. 1.1) et un *voyage* basé sur une séquence d'escales (Sec. 1.2).

1.1 Escale

Une *escale*, sur une carte, est définie par ses coordonnées géographiques X et Y, sous la forme de deux nombres réels. Le type abstrait **Escale** doit permettre de :

— créer une escale à partir de ses deux coordonnées X et Y ;
— obtenir la coordonnée X d'une escale ;
— obtenir la coordonnée Y d'une escale ;
— calculer la distance géographique entre deux escales ;
— enregistrer la vente d'un produit [2] ;
— obtenir le nombre de ventes effectuées dans cette escale.

Dans notre cas, les coordonnées X et Y sont exprimées au format du degré décimal (par exemple, pour Liège, (50.6337300 ° , 5.5674900 °)) et non sous la forme degré/minutes/secondes (soit, pour Liège, 50 ° 38.0238', 5 ° 34.0494')

La *distance géographique* entre deux points A et B (exprimés sous le format du degré décimal) est définie comme suit :

$$d(A, B) = acos(sin(X_A) \times sin(X_B) + cos(X_A) \times cos(X_B)) \times cos(Y_A - Y_B).$$

La valeur de d est obtenue dans une unité correspondant au rayon de la spère terrestre ($R = 6366$km). Si l'arc cosinus rend une valeur en radiant (à vous de vérifier dans la documentation de `math.h`), il suffit de multipler le résultat par R pour obtenir la valeur de d en km. [3]

1.2 Voyage

Un *voyage* est défini comme une suite ordonnée d'escales. Si le point de départ du voyage (i.e., toute première escale) correspond au point d'arrivée (i.e., dernière escale), on dit que le voyage constitue un *circuit*.

Le type abstrait **Voyage** doit permettre de :

— créer un voyage sur base de deux escales. Par définition, un voyage nouvellement créé ne peut pas constituer un circuit ;
— déterminer si un voyage constitue un circuit ;
— déterminer le nombre d'escales d'un voyage ;

1. De manière générale, un voyageur de commerce est un représentant d'une société, voyageant de ville en ville, afin de vendre des produits (historiquement, via du porte à porte).
2. peu importe lequel, ce n'est pas vraiment la question ici
3. Source : `http://www.ipnas.ulg.ac.be/garnir/donneesGPS/TexteTP_calcul.pdf`

— déterminer le nombre totale de ventes d'un voyage ;
— déterminer la distance totale parcourue dans un voyage ;
— ajouter une escale à un voyage ;
— supprimer une escale d'un voyage ;

2 Représentation Concrète

À l'instar de ce qui a été fait au cours théorique, vous devrez proposer deux représentations concrètes implémetant vos types abstraits.

La première devra être une représentation basée sur un **tableau**.

La deuxième représentation devra être basée sur une **liste chainée**. [4] Pour cette représentation, quand cela s'avère possible et pertinent, il vous est demandé d'implémenter les fonctionnalités de manière récursive.

Attention, n'oubliez pas de fournir une représentation concrète pour tous les types abstraits que vous devez définir.

3 Tests Unitaires

Nous vous demandons, aussi, dans ce travail de tester la validité d'une partie de vos deux implémentations à l'aide de tests unitaires. Pour ce faire, vous utiliserez la librairie `seatest` présentée dans le cours INFO0030. En particulier, nous voulons que vous testiez la fonctionnalité qui retourne le nombre d'escales d'un voyage et celle qui permet d'ajouter une escale à un voyage.

4 Aspects Pratiques

Le travail à rendre se compose d'une archive `tar.gz` [5]. Votre archive portera la nom suivant : `tad-groupeXX.tar.gz`, où `XX` fait référence à l'identifiant de votre groupe. [6]

Une fois décompressée, votre archive devra donner naissance à deux répertoires : `rapport/` (cfr. Sec. 4.1) et `code/` (Sec. 4.2). Tout non-respect des consignes se verra sanctionné par 2 points en moins dans la note finale de votre projet. [7]

4.1 Rapport

Votre rapport devra être rédigé via l'outil LaTeX en utilisant l'en-tête du template LaTeX disponible sur la page web du cours. Dans le template, vous trouverez les commandes suivantes :

```
\title{INFO0947: Titre du Projet}
\author{Groupe XX: Prénom\_Nom, Prénom\_Nom}
```

où `Titre du Projet` doit être remplacé par le titre de ce projet et où `XX` désigne votre numéro de binôme. Vous ferez suivre votre numéro de groupe par les prénoms et noms des différents membres du groupe. Une fois compilé, votre document LaTeX devra produire un fichier PDF dont le nom sera `tad-XX.pdf`, où `XX` représente toujours votre numéro de groupe. Ce rapport ne pourra pas compter plus de **12 pages**. [8]

Le dossier `rapport/` sera composé des éléments suivants :

4. Il n'est pas nécessaire, ici, de passer par une représentation en TAD de la liste chainée.
5. Nous vous renvoyons à la formation sur la ligne de commande, donnée au Q1, pour la compression des fichiers dans une archive `tar.gz`.
6. Pour rappel, il est interdit de changer de groupe entre les différents projets. Votre identifiant doit nécessairement être un nombre composé de deux chiffres (compris entre 01 et 50).
7. Comme pour les projets précédents, les archives ne se décompressant pas convenablement engendrent une note finale nulle pour ce projet.
8. Sans compter la page de garde, ni son verso.

— votre rapport au format `.tex` ainsi que toutes les sources utiles à la compilation ;
— votre rapport déjà compilé au format `.pdf`.

Votre rapport devra contenir tous les éléments nécessaires à la compréhension de votre code source. Soit :

— la spécification complète des types abstraits (i.e., signature et sémantique) ainsi que la justification de la complétude et consistance des axiomes ;
— la description de la structure implémentant vos TADs ;
— la description des structures de données basées sur un tableau et sur une liste que vous mettrez en place ;
— la discussion des avantages et inconvénients de chaque structure de données ;
— votre découpe en sous-problèmes, la description de chaque sous-problème et comment ils s'emboîtent ;
— la spécification (formelle) de chaque procédure/fonction concrète (pour les deux représentations) ;
— quand cela s'avère nécessaire, un schéma représentant la situation générale et l'invariant formel que vous tirez de cet invariant. Si nécessaire, vous pouvez introduire de nouvelles notations (à condition de bien les expliquer). Il n'est pas demandé d'écrire les assertions intermédiaires ;
— pour les implémentations récursives, il est demandé de justifer votre construction récursive en répondant aux trois questions (paramètre récursif, cas de base, formulation récursive – cfr. Chapitre 4, Slides 42 → 64) ;
— l'évaluation et la justification de la complexité de chacune des opérations en fonction du type d'implémentation ;
— une explication de la façon dont vous avez écrit votre librairie de tests unitaires.

Soyez clairs et précis dans vos explications. N'hésitez pas à ajouter un schéma si vous le jugez nécessaire. Dans ce cas, expliquez-le : un schéma seul ne suffit pas !

Soignez votre orthographe : au delà de trois fautes, chaque faute vous fera perdre 0,25 points.

4.2 Code

Votre code source devra être rédigé en langage C. Votre code source devra être accompagné d'un Makefile [9] rédigé par vos soins.

Votre code source sera adéquatement découpé en headers et modules. Les différents sous-problèmes identifiés dans votre rapport devront apparaitre clairement dans votre code. Les headers décrivant les interfaces de vos types abstraits devront s'appeler `escale.h` et `voyage.h`. Vous devrez en outre fournir les modules implémentant ces headers. Le premier, appelé `escale.c` implémente le point dans l'espace à deux dimensions. Ensuite, vous devrez fournir deux autres modules, soit `voyage_tableau.c` correspond à l'implémentation basée sur un tableau. Le deuxième, appelé `voyage_liste.c`, correspond à l'implémentation basée sur une liste.

En outre , vous fournirez une librairie de tests unitaires basée sur `seatest` [10] comme indiqué à la Sec. 3.

L'exécution de la commande

```
$>make test_array
```

doit produire un fichier binaire exécutable appelé `voyage_tableau_test`. Ce fichier exécutable sera situé dans le même répertoire que celui où se trouve le code source. Son exécution doit permettre de tester la validité de votre code pour l'implémentation des deux fonctionnalités demandées pour le TAD via un tableau. La fonction `main` de cet exécutable devra être implémentée dans un fichier appelé `voyage_tableau_test.c`

L'exécution de la commande

```
$>make test_list
```

9. cfr. le cours INFO0030, « Partie 2 – Chapitre 1 : Compilation ».
10. cfr. le cours INFO0030, « Partie 2 – Chapitre 2 : Tests ». La page web du cours INFO0030 contient les sources de la librairie `seatest`. Vous n'oublierez pas de joindre ces sources à votre archive.

doit produire un fichier binaire exécutable appelé `voyage_liste_test`. Ce fichier exécutable sera situé dans le même répertoire que celui où se trouve le code source. Son exécution doit permettre de tester la validité de votre code pour l'implémentation des deux fonctionnalités demandées pour le TAD via une liste. La fonction `main` de cet exécutable devra être implémentée dans un fichier appelé `voyage_liste_test.c`.

Ces différents fichiers source doivent pouvoir être compilés et utilisés sans la présence du fichier contenant la fonction `main`.

Il est **interdit** [11] d'utiliser des fonctions ou procédures de librairies tierces (y compris la librairie standard du C), à l'exception de `assert.h`, `stdlib.h`, `math.h` et `seatest.h`.

5 Agenda

5.1 Milestones

Pendant la durée du projet, nous vous proposons deux *milestones*. [12] L'objectif de ces milestones est de rencontrer chaque groupe individuellement afin de discuter de l'avancement du projet et corriger le tir le cas échéant. Ces rencontres sont organisées resp. la 3e et la 2e semaine qui précèdent la remise du travail final. À l'issue de ces rencontres, aucune note ne sera affectée. L'objectif n'est donc pas d'évaluer formellement votre travail mais bien de vous aider à réaliser le projet et à en tirer des enseignements.

Les deux milestones sont les suivants :

— **Semaine du 22/04/2019**. Le milestone portera sur les axiomes. Afin de permettre à l'équipe pédagogique de préparer au mieux cette rencontre, il vous est demandé de nous (i.e., Benoit Donnet **et** Simon Liénardy) envoyer par email un petit document (2 pages maximum, format PDF, rédigé en LaTeX) décrivant votre compréhension du problème et vos axiomes. En particulier, il sera intéressant de fournir un schéma décrivant la notion de polyligne. La deadline pour l'envoi de ce document est le samedi 20/04/2019, 12h00 (n'oubliez pas d'indiquer la chaîne de caractères [INFO0947] dans le sujet de votre email).

— **Semaine du 29/04/2019**. Le milestone portera sur les structures de données et une partie de votre code (et donc les invariants). Afin de permettre à l'équipe pédagogique de préparer au mieux cette rencontre, il vous est demandé de nous (i.e., Benoit Donnet **et** Simon Liénardy) envoyer par email un petit document (format PDF, rédigé en LaTeX) décrivant vos structures de données et vos invariants (situation générale, dessin, et invariant formel). La deadline pour l'envoi de ce document est le samedi 27/04/2019, 12h00 (n'oubliez pas d'indiquer la chaîne de caractères [INFO0947] dans le sujet de votre email).

La participation à ces milestones est obligatoire pour tous les membres du binôme. Aucun groupe ni étudiant ne peut s'y soustraire. Pour définir le moment auquel votre groupe viendra rencontrer l'équipe pédagogique, deux sujets de discussion sont créés dans le forum du cours, sur la plateforme e-campus. Il vous suffit de lire et de suivre les instructions détaillées dans le premier message de chaque sujet.

Veuillez faire débuter les sujets des mails envoyés à l'équipe pédagogique par la chaine "**[INFO0947]**".

5.2 Deadline

Les archives `tar.gz` finales doivent être soumises sur la plateforme `http://submit.montefiore.ulg.ac.be` avant le **13/05/2019** à **18h00** (l'heure du serveur faisant foi). Toute soumission postérieure à cette date ne sera pas prise en compte. Comme l'heure de l'horloge du serveur de soumission et celle de votre ordinateur peuvent être légèrement différentes, il est **fortement déconseillé** de soumettre votre projet à 17h59 et 59 secondes : ne prenez pas de risque inutile.

11. Cette restriction ne s'applique pas aux fichiers `*-test.c` où sont implémentés les exécutables de tests.
12. « Étapes » en français.