INFO0947 : Compléments de Programmation
Récursivité & Types Abstraits de Données

B. Donnet, G. Brieven
Université de Liège

1 Type Abstrait

Une *course à étapes* (ou course par étapes) est une course cycliste qui se déroule sur plusieurs jours, par opposition aux *courses classiques* (aussi dénommées courses d'un jour). Les courses à étapes les plus populaires sont le Tour de France (la Grande Boucle), le Tour d'Italie (le Giro) et le Tour d'Espagne (la Vuelta). [1]

Une course à étapes est représentée par un certain nombre d'*escales* [2]. Une paire d'escales représente une étape de la course. Par exemple, le Tour de Wallonie pourrait être représenté par la liste d'escales suivantes :

(Mons, Namur, Arlon, Liège, Gembloux, Wavre, Mons)

Ce Tour de Wallonie est composé de six étapes :
- (Mons, Namur)
- (Namur, Arlon)
- (Arlon, Liège)
- ...
- (Wavre, Mons)

L'objectif de ce travail est d'implémenter, sous la forme d'un type abstrait de données une course à étapes. Dans ce travail, il vous est demander de spécifier deux types abstraits, une *escale* (Sec. 1.1) représentant une des villes par laquelle passe le tour et une *course* basée sur une séquence d'escales (Sec. 1.2).

1.1 Escale

Une *escale*, sur la course à étapes, 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 et de son nom ;
- obtenir la coordonnée X d'une escale (e.g., 5.5796662 pour Liège) ;
- obtenir la coordonnée Y d'une escale (e.g., 50.6325574 pour Liège) ;
- obtenir le nom de l'escale (e.g., Liège) ;
- calculer la distance géographique entre deux escales ;
- enregistrer le meilleur temps pour atteindre cette escale [3] ;
- obtenir le meilleur temps pour cette escale.

1.2 Course par Étapes

Une *course par étapes* est définie comme une suite ordonnée d'escales. Par définition, le point de départ de la course (e.g., Mons) peut parfois être le même que le point d'arrivée. Dans ce cas, la course

1. Source : Wikipedia
2. Une escale est un lieu d'arrêt ou de relâche et de ravitaillement
3. Peu importe lequel, en fait, cela n'a pas vraiment d'importance

forme ce qu'on appelle un *circuit*. Ce n'est pas toujours le cas (e.g., le Tour de France 2020 a démarré à Nice pour se terminer à Paris).

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

— créer une course par étapes sur base de deux escales. Par définition, une course nouvellement créée ne peut pas constituer un circuit. Aussi le meilleur temps de l'escale de départ est forcément 0 ;
— déterminer si une course constitue un circuit ;
— déterminer le nombre d'escales de la course ;
— déterminer le nombre d'étapes de la course ;
— déterminer le meilleur temps total nécessaire pour faire toute la course ;
— déterminer le meilleur temps d'une escale de la course ;
— ajouter une escale à une course ;
— supprimer une escale d'une course ;

2 Représentation Concrète

À l'instar de ce qui a été fait au cours théorique, vous devrez proposer deux représentations concrètes implémentant 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 meilleur temps total nécessaire pour faire toute la course et celle qui permet d'ajouter une escale à une course.

4 Aspects Pratiques

Le travail à rendre se compose d'une archive `tar.gz`[5]. Votre archive portera le nom `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(nous vous renvoyons à la formation donnée le CSS, le 06/03/2025, pour la rédaction d'un document en LaTeX) en utilisant le template LaTeXdisponible sur eCampus (Sec. Projets).

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. De plus, toute archive ne pouvant être décomposée (c'est-à-dire une archive corrompue) ou ne produisant pas les dossiers `rapport/` et `code/` lors de la décompression engendrera une note finale **nulle**.

Le template est organisé comme suit :

— `main.tex`. C'est le fichier principal. ous trouverez une section « TITRE » délimitée par des commentaires. Il suffit de compléter les définitions des macros `\intitule`, `\GrNbr`, `\PrenomUN`, `\NomUN`, `\PrenomDEUX` et `\NomDEUX` par, respectivement, le titre du projet, votre numéro de groupe, vos prénoms et noms. **Attention !** Ne modifiez pas les dix lignes appelées « *Zone protégée* ».

— `introduction.tex`. Dans ce fichier, vous rédigez l'introduction de votre rapport (inutile de recopier l'énoncé du projet, essayez d'y mettre du vôtre).

— `tad.tex`. Dans ce fichier, vous spécifiez formellement vos TADs (syntaxe et sémantique – une sous-section/TAD). N'oubliez pas de justifier la complétude de vos axiomes. À titre d'informations, le fichier `tad.tex` contient la spécification abstraite du TAD `Vector` vu au cours théorique.

— `structures.tex`. Dans ce fichier, vous présentez vos structures de données en C afin de représenter `Escale`, `Course` (représenté sur base d'un tableau) et `Course` (représenté sur base d'une liste chainée).

— `specifications.tex`. Dans ce fichier, vous spécifiez formellement les constructeurs (trois spécifications [8]), deux transformateurs de votre choix, relatifs à `Course` (quatre spécifications), et cinq observateurs de votre choix. Si nécessaire, introduisez des notations.

— `invariants.tex`. Dans ce fichier, indiquez et décrivez tous les Invariants de Boucle nécessaires. Pour chaque Sp nécessitant un Invariant de Boucle (une sous-section/Sp), donnez l'Invariant Graphique et l'Invariant Formel correspondant. Pour l'Invariant Graphique, nous vous rappelons qu'il est possible d'exporter les Invariants Graphiques au format PDF depuis le Gli.

— `recursivite.tex`. Dans ce fichier, donnez une définition récursive pour chaque fonctionnalité pour laquelle c'est pertinent. Veillez à ce que votre définition récursive respecte le format vu a cours (cfr. Chap. 4). Expliquez brièvement votre définition.

— `complexite.tex`. Dans ce fichier, vous devez étudier la complexité de votre code. Il vous est demandé, pour chaque type d'implémentation (via un tableau, et via une liste) de grouper les fonctionnalités présentant la même complexité. Ensuite, sélectionnez une fonctionnalité par groupe et justifiez formellement sa complexité en découpant le code en instructions, évaluant la complexité exacte et justifiant par quoi elle peut être bornée (voir Chap. 3).

— `tests.tex`. Dans ce fichier, vous décrivez comment vous avez implémenté les différents tests unitaires. Pensez à justifier vos choix.

— `conclusion.tex`. Dans ce fichier, vous indiquez la conclusion de votre rapport.

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 !

Pour compiler votre rapport, renommez `"main.tex"` en `"course-XX.tex"` et utiliser la commande suivante : `"pdflatex course-XX.tex"`

Une fois compilé, votre document LaTeXdevra produire un fichier PDF dont le nom sera `course-XX.pdf`, où `XX` représente toujours votre numéro de groupe. Ce rapport ne pourra pas compter plus de **12 pages**. [9]

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

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

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

— votre rapport déjà compilé au format `.pdf`.

4.2 Code

Votre code source devra être rédigé en langage C. Votre code source devra être accompagné d'un Makefile [10] 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 `course.h`. Vous devrez en outre fournir les

8. une pour le constructeur de `Escale` et deux pour les constructeurs de `Course`, puique `Course` repose sur 2 implémentations. Les 2 spécifications seront bien différentes dans le cas de l'implémentation via un tableau et via une liste puisque les champs des structures de données sont différents.
9. Sans compter la page de garde, ni son verso.
10. cfr. le cours INFO0030, « Partie 2 – Chapitre 1 : Compilation ».

modules implémentant ces headers. Le premier, appelé `escale.c` implémente `escale.h`. Ensuite, vous devrez fournir deux autres modules, soit `course_tableau.c`correspond à l'implémentation basée sur un tableau. Le deuxième, appelé `course_liste.c`, correspond à l'implémentation basée sur une liste.

En outre , vous fournirez une librairie de tests unitaires basée sur `seatest` [11] comme indiqué à la Sec. 3.

L'exécution de la commande

```
$>make test_array
```

doit produire un fichier binaire exécutable appelé `course_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 `course_tableau_test.c`.

L'exécution de la commande

```
$>make test_list
```

doit produire un fichier binaire exécutable appelé `course_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 `course_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** [12] 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*. [13] 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 14/04/2025**. Le milestone portera sur les axiomes. Afin de permettre à l'équipe pédagogique de préparer au mieux le feedback, il vous est demandé de nous (i.e., Benoit Donnet et Géraldine Brieven) décrivant votre compréhension du problème et vos axiomes. En particulier, il peut être intéressant de fournir un schéma illustrant les concepts derrière votre TAD. La deadline pour l'envoi de ce document est le samedi 12/04/2025, 12h00. Vous devez spécifier, dans l'objet de votre email, la chaîne de caractères suivantes : `[INFO0947] Projet 2 - Milestone 1`
- **Semaine du 06/05/2025**. Le milestone portera sur les structures de données et les Invariants de Boucle. Afin de permettre à l'équipe pédagogique de préparer au mieux le feedback, il vous est demandé de nous (i.e., Benoit Donnet et Géraldine Brieven) fournir une production décrivant toutes vos structures de données (en C) et vos Invariants de Boucle (Invariants Graphiques et Invariant Formel) pour supporter l'implémentation basée sur un tableau. La deadline pour l'envoi de ce document est le samedi 04/05/2025, 12h00. Vous devez spécifier, dans l'objet de votre email, la chaîne de caractères suivantes : `[INFO0947] Projet 2 - Milestone 2`

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

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

5.2 Deadline

Les archives `tar.gz` finales doivent être soumises sur Plateforme de Soumission avant le lundi **12/05/2025** 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.