INFO0947: Récursivité Types Abstraits de Données

Groupe 33: Pavlov Aleksandr, Gendebien Alexandre

Table des matières

1 Introduction

Dans le cadre du cours INFO-0947, nous avons dû résoudre un problème donné et créer un algorithme en C capable de :

Créer une course de vélo fictive, composée d'escales représentées par des villes (et leurs coordonnées). Déterminer le meilleur temps (le plus petit) qu'un cycliste a mis pour parcourir la distance qui sépare deux villes, ainsi que le meilleur temps pour finir la course. Calculer le nombre total d'étapes qui composent la course et vérifier si la course forme un circuit.

Ce problème sera entièrement documenté en LaTeX.

2 Spécifications Abstraites

À titre d'exemple, voici les spécifications abstraites pour le TAD `Vector` telles que vues dans le cours théorique (Chapitre 5). À adapter en fonction de vos besoins pour le projet.

2.1 TAD Vector

2.1.1 Syntaxe

Type : Vector

Utilise : Integer
Element

Opérations : create : Integer $\rightarrow$ Vector
set : Vector $\times$ Integer $\times$ Element $\rightarrow$ Vector
get : Vector $\times$ Integer $\rightarrow$ Element
size : Vector $\rightarrow$ Integer

2.1.2 Sémantique

Préconditions : $\forall i \in$ Integer, $\forall e \in$ Element, $\forall v \in$ Vector
$\forall i \geq 0$, create(i)
$\forall i, 0 \leq i <$ size(v), get(v, i)
$\forall i, 0 \leq i <$ size(v), set(v, i, e)

Axiomes : $\forall e \in$ Element, $\forall v \in$ Vector, $\forall i, j \in$ Integer
size(create(i)) = i
size(set(v, i, e)) = size(v)

$$\text{get(set(v, i, e), j)} = \begin{cases} e & \text{If } i = j \\ \text{get}(v, j) & \text{Otherwise} \end{cases}$$

		Opérations Internes	
		create(·)	set(·)
Observateurs	get(·)	$\emptyset$	✓
	size(·)	✓	✓

2.2 TAD Escale

2.2.1 Syntaxe

Type : Escale

Utilise : String (nom)
Float (coordonnées)

Opérations : escale_create : String × Float × Float → Escale
escale_get_name : Escale → String
escale_get_x : Escale → Float
escale_get_y : Escale → Float
escale_get_best_time : Escale → Float
escale_set_best_time : Escale × Float → Integer
escale_distance : Escale × Escale → Float

2.2.2 Sémantique

Préconditions : $\forall n \in$ String, $\forall x, y \in$ Float
escale_create(n, x, y)
$\forall e \in$ Escale, escale_get_name(e), escale_get_x(e), escale_get_y(e), escale_get_best_time(e)
$\forall e \in$ Escale, $\forall t \in$ Float, escale_set_best_time(e, t)
$\forall e1, e2 \in$ Escale, escale_distance(e1, e2)

Axiomes : $\forall e \in$ Escale, $\forall n \in$ String, $\forall x, y, t \in$ Float
escale_get_name(escale_create(n, x, y)) = n
escale_get_x(escale_create(n, x, y)) = x
escale_get_y(escale_create(n, x, y)) = y
escale_get_best_time(escale_set_best_time(e, t)) = t
escale_distance(e1, e2) = $\sqrt{(e1_x - e2_x)^2 + (e1_y - e2_y)^2}$

		Opérations Internes	
		escale_create(·)	escale_set_best_time(·)
	escale_get_name(·)	✓	∅
Observateurs	escale_get_x(·)	✓	∅
	escale_get_y(·)	✓	∅
	escale_get_best_time(·)	✓	✓

2.3 TAD Course

2.3.1 Syntaxe

Type : Course

Utilise : Escale
Integer (index, comptage)
Float (temps total, meilleur temps)

Opérations : course_create : Escale × Escale → Course

course_is_circuit : Course $\rightarrow$ Boolean
course_get_escales_count : Course $\rightarrow$ Integer
course_get_stages_count : Course $\rightarrow$ Integer
course_total_time : Course $\rightarrow$ Float
course_best_time_at : Course $\times$ Integer $\rightarrow$ Float
course_append : Course $\times$ Escale $\rightarrow$ Course
course_pop : Course $\rightarrow$ Course

2.3.2 Sémantique

Préconditions : $\forall e1, e2 \in$ Escale, course_create(e1, e2)
$\forall c \in$ Course, course_is_circuit(c), course_get_escales_count(c), course_get_stages_count(c), course_total_time(c)
$\forall c \in$ Course, $\forall i \in$ Integer, course_best_time_at(c, i)
$\forall c \in$ Course, $\forall e \in$ Escale, course_append(c, e)
$\forall c \in$ Course, course_pop(c)

Axiomes : $\forall c \in$ Course, $\forall e1, e2 \in$ Escale, $\forall t \in$ Float
course_get_escales_count(course_create(e1, e2)) = 2
course_is_circuit(c) = (première escale = dernière escale)
course_total_time(course_create(e1, e2)) = distance(e1, e2) / meilleur vitesse
course_best_time_at(c, i) = *temps minimal enregistré pour l'étape i*
course_get_stages_count(course_append(c, e)) = course_get_stages_count(c) + 1

		Opérations Internes	
		course_create(·)	course_append(·)
Observateurs	course_is_circuit(·)	✓	$\emptyset$
	course_get_escales_count(·)	✓	✓
	course_get_stages_count(·)	✓	✓
	course_total_time(·)	✓	✓

3 Structures de Données

Pour implémenter les différents TAD, nous avons choisi deux types de structures de données : le tableau dynamique et la liste chaînée.

3.1 Tableau Dynamique

```
typedef struct Escale {
    char *name;
    double x;
    double y;
    double time;
} Escale;

typedef struct Course {
    size_t escales_size;
    size_t escales_count;
```

```
        Escale **escales;
    } Course;
```

Listing 1 – Structure de données (tableau)

3.1.1 Avantages

- Accès rapide aux éléments par leur indice ($O(1)$).
- Moins de surcharge mémoire due aux pointeurs supplémentaires.
- Facile à parcourir séquentiellement.

3.1.2 Inconvénients

- Redimensionnement coûteux si la taille initiale est insuffisante ($O(n)$).
- Ajout et suppression au milieu nécessitent un déplacement des éléments ($O(n)$).

3.2 Liste Chaînée

```
    typedef struct Escale {
        char *name;
        double x;
        double y;
        double time;
    } Escale;

    typedef struct Course {
        Escale *escale;
        Course *next;
    } Course;
```

Listing 2 – Structure de données (liste chainée)

3.2.1 Avantages

- Insertion et suppression en temps constant ($O(1)$) sans déplacement des éléments.
- Taille flexible sans besoin de redimensionnement.

3.2.2 Inconvénients

- Accès séquentiel aux éléments ($O(n)$) au lieu d'un accès direct.
- Surcharge mémoire due aux pointeurs supplémentaires.

Ces choix de structures de données permettent de répondre aux différentes exigences du problème. Le tableau est idéal pour un accès rapide et indexé, tandis que la liste chaînée convient mieux aux modifications fréquentes et dynamiques de la course.

3.3 Structure de données

4 Specifications

5 Invariants

Invariant formel :
$escale = escale_0$

$\wedge$
$0 < i < escale\_count$
$\wedge$
$total\_time = \sum_{i=0}^{escale\_count-1} \texttt{get\_time}(escales[i])$

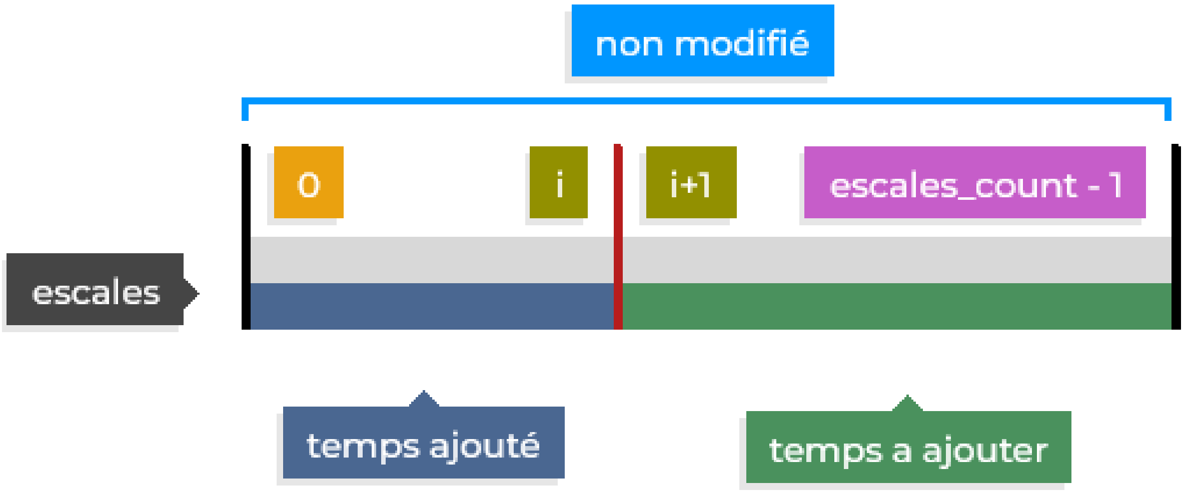

FIGURE 1 – Invariant graphique

6 Implémentations Récursives

Rappel de la signature de la fonction en C :
Définition récursive :

$$\text{nameF(arg1,..., argN)} = \begin{cases} \ldots & \text{if } \ldots \\ \ldots & \text{if } \ldots \\ \ldots & \\ \ldots & \text{otherwise} \end{cases}$$

7 Complexité

8 Tests Unitaires

9 Conclusion