

# 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'escaliers 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  $\text{\LaTeX}$ .

## 2 Spécifications Abstraites

### 2.1 TAD Escalier

#### 2.1.1 Syntaxe

**Type :** Escalier

**Utilise :** Integer

String (nom)

Float (coordonnées)

Boolean

**Opérations :**  $\text{escalier\_create} : \text{String} \times \text{Float} \times \text{Float} \rightarrow \text{Escalier}$

$\text{escalier\_get\_name} : \text{Escalier} \rightarrow \text{String}$

$\text{escalier\_get\_x} : \text{Escalier} \rightarrow \text{Float}$

$\text{escalier\_get\_y} : \text{Escalier} \rightarrow \text{Float}$

$\text{escalier\_get\_best\_time} : \text{Escalier} \rightarrow \text{Float}$

$\text{escalier\_set\_best\_time} : \text{Escalier} \times \text{Float}$

$\text{escalier\_distance} : \text{Escalier} \times \text{Escalier} \rightarrow \text{Float}$

$\text{escalier\_equal} : \text{Escalier} \times \text{Escalier} \rightarrow \text{Boolean}$

#### 2.1.2 Sémantique

**Préconditions :**  $\forall n \in \text{String}, \forall x, y \in \text{Float}, \forall e \in \text{Escalier}$

$\forall x, -180 \leq x \leq 180, \forall y, -90 \leq y \leq 90, \text{escalier\_create}(n, x, y)$

$\text{escalier\_get\_name}(e)$

$\text{escalier\_get\_x}(e)$

$\text{escalier\_get\_y}(e)$

$\text{escalier\_get\_best\_time}(e)$

$\forall t \geq 0 \text{ Float}, \text{escalier\_set\_best\_time}(e, t)$

$\forall e1, e2 \in \text{Escalier}, \text{escalier\_distance}(e1, e2)$

**Axiomes :**  $\forall e \in \text{Escalier}, \forall n \in \text{String}, \forall x, y, t \in \text{Float}$

$\text{escalier\_get\_name}(\text{escalier\_create}(n, x, y)) = n$

$\text{escalier\_get\_x}(\text{escalier\_create}(n, x, y)) = x$

$\text{escalier\_get\_y}(\text{escalier\_create}(n, x, y)) = y$

$\text{escalier\_get\_best\_time}(\text{escalier\_create}(n, x, y)) = 0$

$\text{escalier\_distance}(\text{escalier\_create}(n1, x1, y1), \text{escalier\_create}(n2, x2, y2)) = \text{Haversine formula}$

$\text{escale\_equal}(\text{escale\_create}(n1, x, y), \text{escale\_create}(n2, x, y)) = \text{true}$   
 $\text{escale\_equal}(\text{escale\_create}(n1, x1, y1), \text{escale\_create}(n2, x2, y2)) = \text{false}$

		Opérations Internes	
		$\text{escale\_create}(\cdot)$	$\text{escale\_set\_best\_time}(\cdot)$
Observateurs	$\text{escale\_get\_name}(\cdot)$	✓	∅
	$\text{escale\_get\_x}(\cdot)$	✓	∅
	$\text{escale\_get\_y}(\cdot)$	✓	∅
	$\text{escale\_get\_best\_time}(\cdot)$	✓	∅
	$\text{escale\_distance}(\cdot)$	✓	∅
	$\text{escale\_equal}(\cdot)$	✓	∅

## 2.2 TAD Course

### 2.2.1 Syntaxe

**Type :** Course

**Utilise :** Escale

Integer (index, comptage)

Float (temps total, meilleur temps)

Boolean

**Opérations :**  $\text{course\_create} : \text{Escale} \times \text{Escale} \rightarrow \text{Course}$

$\text{course\_is\_circuit} : \text{Course} \rightarrow \text{Boolean}$

$\text{course\_get\_escales\_count} : \text{Course} \rightarrow \text{Integer}$

$\text{course\_get\_stages\_count} : \text{Course} \rightarrow \text{Integer}$

$\text{course\_total\_time} : \text{Course} \rightarrow \text{Float}$

$\text{course\_best\_time\_at} : \text{Course} \times \text{Integer} \rightarrow \text{Float}$

$\text{course\_append} : \text{Course} \times \text{Escale} \rightarrow \text{Course}$

$\text{course\_pop} : \text{Course} \rightarrow \text{Course}$

### 2.2.2 Sémantique

**Préconditions :**  $\forall e, e1, e2 \in \text{Escale}, \forall n \in \text{String}, \forall x, y, t \in \text{Float} \forall i \in \text{Integer}$

$\forall e1, e2 \in \text{Escale}, \text{escale\_get\_best\_time}(e1) = 0 \wedge \text{escale\_equal}(e1, e2) = \text{false}, \text{course\_create}(e1, e2)$

$\text{course\_is\_circuit}(c)$

$\text{course\_get\_escales\_count}(c)$

$\text{course\_get\_stages\_count}(c)$

$\text{course\_total\_time}(c)$

$\forall i \in \text{Integer}, 0 \leq i < \text{course\_get\_escales\_count}(c), \text{course\_best\_time\_at}(c, i)$

$\text{course\_append}(c, e)$

$\forall c \in \text{Course}, \text{course\_get\_escales\_count}(c) > 0, \text{course\_pop}(c)$

**Axiomes :**  $\forall c \in \text{Course}, \forall e, e1, e2, e3 \in \text{Escale}, \forall n \in \text{String}, \forall x, y, t \in \text{Float} \forall i \in \text{Integer}$

$\text{course\_create}(e1, e2) = \text{course\_pop}(\text{course\_append}(\text{course\_create}(e1, e2), e3))$

$\text{course\_is\_circuit}(\text{course\_create}(e1, e2)) = \text{false}$

```

course_is_circuit(course_append(course_create(e1, e2), e1)) = true
course_is_circuit(course_append(course_create(e1, e2), e3)) = false
course_is_circuit(course_pop(course_create(e1, e2))) = false
course_get_escales_count(course_create(e1, e2)) = 2
course_get_escales_count(course_append(c, e)) = course_get_escales_count(c) + 1
course_get_escales_count(course_pop(c)) = course_get_escales_count(c) - 1
course_get_stages_count(course_create(e1, e2)) = 1
course_get_stages_count(course_append(c, e)) = course_get_stages_count(c) + 1
course_get_stages_count(course_pop(c)) = course_get_stages_count(c) - 1
course_get_stages_count(course_pop(course_pop(course_create(e1, e2)))) = 0
course_total_time(course_create(e1, e2)) = escale_get_best_time(e1) + escale_get_best_time(e2)
course_total_time(course_append(c, e)) = course_total_time(c) + escale_get_best_time(e)
course_total_time(course_pop(course_append(c, e))) = course_total_time(c)
course_best_time_at(course_create(e1, e2), 1) = escale_get_best_time(e2)
course_best_time_at(course_append(c, e), course_get_escales_count(c)) = escale_get_best_time(e)
course_best_time_at(course_pop(course_create(e1, e2)), 0) = escale_get_best_time(e1)

```

		Opérations Internes		
		course_create(·)	course_append(·)	course_pop(·)
Observateurs	course_is_circuit(·)	✓	✓	✓
	course_get_escales_count(·)	✓	✓	✓
	course_get_stages_count(·)	✓	✓	✓
	course_total_time(·)	✓	✓	✓
	course_best_time_at(·)	✓	✓	✓

### 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 Escale

```

1  typedef struct Escale {
2      char *name;
3      double x;
4      double y;
5      double time;
6  } Escale;

```

Listing 1 – Structure de Escale

#### 3.2 Course (Tableau)

```

1  typedef struct Course {
2      size_t escales_size;
3      size_t escales_count;

```

```

4      Escale **escales;
5  } Course;

```

Listing 2 – Structure de Course (tableau)

### 3.2.1 Avantages

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

### 3.2.2 Inconvénients

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

## 3.3 Course (Liste Chaînée)

```

1  typedef struct Course {
2      Escale *escale;
3      Course *next;
4  } Course;

```

Listing 3 – Structure de Course (liste chaînée)

### 3.3.1 Avantages

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

### 3.3.2 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.

## 3.4 Structure de données

# 4 Specifications

## 4.1 Constructeur

$\text{course\_create\_list} : \text{Escale} \times \text{Escale} \rightarrow \text{Course}$

pre :  $e_1 \neq e_2 \wedge e_1 \neq \text{NULL} \wedge e_2 \neq \text{NULL} \wedge \text{escale\_time}(e_1) = 0$

post :

taille course à la création = 2

course  $\rightarrow$  escale =  $e_1$

(course  $\rightarrow$  next)  $\rightarrow$  escale =  $e_2$

$\text{course\_create\_array} : \text{Escale} \times \text{Escale} \rightarrow \text{Course}$   
 $\text{pre} : e_1 \neq e_2 \wedge e_1 \neq \text{NULL} \wedge e_2 \neq \text{NULL} \wedge \text{escale\_time}(e_1) = 0$        $\text{escales}[0] = e_1 \quad \text{escales}[1] = e_2$   
 $\text{post} :$   
 $\text{escale\_count} \text{escale\_count} \text{escale\_count} \text{escale\_count} = 2$

$\text{escale\_create} : \text{String} \times \mathbb{R} \times \mathbb{R} \rightarrow \text{Escale}$   
 $\text{pre} : \text{name} \neq \text{NULL}$   
 $\text{post} :$   
 $\text{escale} \rightarrow \text{name} = \text{name}$   
 $\text{escale} \rightarrow x = x$   
 $\text{escale} \rightarrow y = y$   
 $\text{escale\_get\_best\_time}(e) = 0$

## 4.2 Transformateur

$\text{escale\_set\_best\_time} : \text{Escale} \times \mathbb{R} \rightarrow \text{Void}$   
 $\text{pre} : e \neq \text{NULL}$   
 $\text{post} :$   
 $\text{escale\_get\_best\_time}(e) = \text{best\_time}$

$\text{course\_append} : \text{Course} \times \text{Escale} \rightarrow \text{Course}$   
 $\text{pre} : C \neq \text{NULL} \wedge e \neq \text{NULL}$   
 $\text{post} :$   
 $\text{course\_size}(C') = \text{course\_size}(C) + 1$   
 $\text{course\_last}(C') = e$

## 4.3 Observateur

$\text{course\_is\_circuit} : \text{Course} \rightarrow \mathbb{B}$   
 $\text{pre} : C \neq \text{NULL}$   
 $\text{post} :$   
 si première étape = dernière étape return :true  
 sinon return :false

$\text{course\_best\_time} : \text{Course} \times \mathbb{N} \rightarrow \mathbb{R}$   
 $\text{pre} : C \neq \text{NULL} \wedge 0 \leq \text{index} < \text{course\_size}(C)$   
 $\text{post} :$   
 $\text{result} = \text{escale\_time}(\text{course\_at}(C, \text{index}))$

`escale_get_x : Escale → ℝ`

pre :  $e \neq \text{NULL}$

post :

result = `escale_x(e)`

`escale_get_y : Escale → ℝ`

pre :  $e \neq \text{NULL}$

post :

result = `escale_y(e)`

`escale_distance : Escale × Escale → ℝ`

pre :  $e_1 \neq \text{NULL} \wedge e_2 \neq \text{NULL}$

post :

$$d = 2R \arcsin \left( \sqrt{\sin^2 \left( \frac{\Delta\phi}{2} \right) + \cos(\phi_1) \cos(\phi_2) \sin^2 \left( \frac{\Delta\lambda}{2} \right)} \right)$$

## 5 Invariants

### 5.1 Invariant du temps total

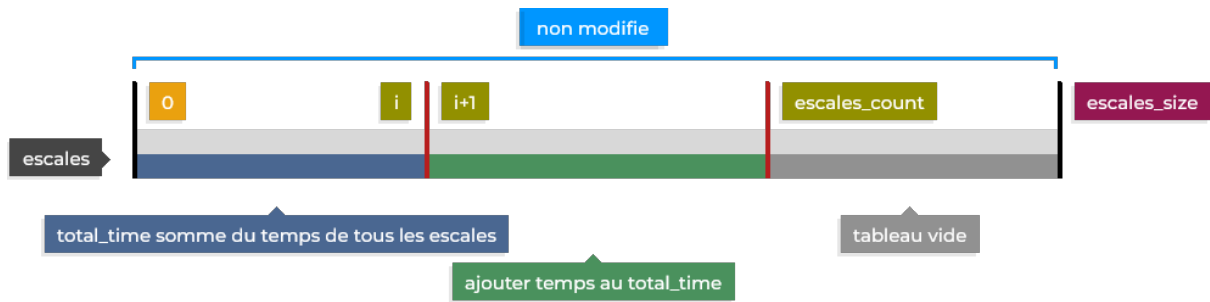


FIGURE 1 – Invariant graphique

**Invariant formel :**

$escale = escale_0$

$\wedge$

$0 < i < escale\_count$

$\wedge$

$total\_time = \sum_{i=0}^{escale\_count-1} escale\_get\_best\_time(escales[i])$

## 6 Implémentations Récursives

Définition récursive :

`course_get_escales_count(course) =`  
 $\begin{cases} \text{return } 0; & \text{if } course == \text{NULL} \\ \text{return } course\_get\_escales\_count(course \rightarrow next) + 1; & \text{otherwise} \end{cases}$



```

    course_get_stages_count(course) =
    {
        return 0;                                if course == NULL
        return 0;                                if course -> next == NULL
        return course_get_stages_count(course -> next) + 1;  otherwise
    }

    course_total_time(course) =
    {
        return 0;                                if course == NULL
        return (
            course_total_time(course -> next) +
            escale_get_best_time(course -> escale)
        );                                       otherwise
    }

    course_best_time_at(course, index) =
    {
        return escale_get_best_time(course -> escale);        if index == 0
        return course_best_time_at(course -> next, index - 1);  otherwise
    }

    course_append(course, escale) =
    {
        course = malloc(sizeof(Course));
        course -> escale = escale;
        course -> next = NULL;
        return course;                                if course == NULL

        course -> next = course_append(course -> next, escale);
        return course;                                otherwise
    }

    course_pop(course) =
    {
        free(course -> escale);
        free(course);
        return NULL;                                if course -> next == NULL

        course -> next = course_pop(course -> next);
        return course;                                otherwise
    }

    course_free(course) =
    {
        return;                                if course -> next == NULL

        course_free(course -> next);
        free(course -> escale);
        free(course);                                otherwise
    }

    course_last(course) =
    {
        return course;                                if course -> next == NULL

        return course_last(course -> next);  otherwise
    }

```

## 7 Complexité

### 7.1 Fonctions sur Escale

```
    escale_create :  $\mathcal{O}(1)$   
    escale_get_name :  $\mathcal{O}(1)$   
        escale_get_x :  $\mathcal{O}(1)$   
        escale_get_y :  $\mathcal{O}(1)$   
    escale_get_best_time :  $\mathcal{O}(1)$   
    escale_set_best_time :  $\mathcal{O}(1)$   
        escale_distance :  $\mathcal{O}(1)$   
        escale_equal :  $\mathcal{O}(1)$ 
```

### 7.2 Opérations sur Course (tableau dynamique)

Soit  $n$  le nombre d'escales,  $S$  la capacité du tableau.

```
        course_create :  $\mathcal{O}(1)$   
        course_is_circuit :  $\mathcal{O}(1)$   
    course_get_escales_count :  $\mathcal{O}(1)$   
        course_get_stages_count :  $\mathcal{O}(1)$   
        course_total_time :  $\mathcal{O}(n)$   
        course_best_time_at( $i$ ) :  $\mathcal{O}(1)$   
    course_append :  $\begin{cases} \mathcal{O}(1) \\ \mathcal{O}(n) \text{ (réallocation)} \end{cases}$   
        course_pop :  $\mathcal{O}(1)$ 
```

### 7.3 Opérations sur Course (liste chaînée)

Soit  $n$  le nombre d'escales.

```
        course_create :  $\mathcal{O}(1)$   
        course_is_circuit :  $\mathcal{O}(n)$   
    course_get_escales_count :  $\mathcal{O}(n)$   
        course_get_stages_count :  $\mathcal{O}(n)$   
        course_total_time :  $\mathcal{O}(n)$   
        course_best_time_at( $i$ ) :  $\mathcal{O}(i)$   
        course_append :  $\mathcal{O}(n)$   
        course_pop :  $\mathcal{O}(n)$ 
```

## 8 Tests Unitaires

Les tests unitaires sont situés dans le dossier `test/` et utilisent le framework `seatest`. Pour chaque implémentation (`course_liste` et `course_tableau`), deux fonctions principales sont testées :

- `test_course_append` : vérifie que l'ajout d'escales modifie correctement le nombre d'escales dans la course.
- `test_course_total_time` : vérifie que la somme des temps des escales est correcte après modification des temps et suppression d'éléments.

Les assertions utilisées sont `assert_ulong_equal` pour les entiers et `assert_double_equal` pour les valeurs réelles.

**Justification des choix :** Les tests ciblent les opérations principales et les cas limites.

**Limite :** Seules les fonctions d'ajout et de calcul du temps total sont testées. Les autres fonctions publiques ne sont pas couvertes par les tests actuels.

## 9 Conclusion

pour conclure ce rapport, nous pouvons dire que nous avons réussi à répondre au problème dans sa globalité. En créant un programme capable de créer une course fictive et d'en utiliser tous les éléments qui la composent afin de trouver par quelles ville la course passe, le meilleur temps qu'à mis un cycliste pour parcourir la distance entre deux villes, ou encore si la course forme un circuit.