

INFO0947: Projet 2  
Types Abstraits de Données

Groupe 10: Cyril RUSSE

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Contexte . . . . .	3
1.2	Enoncé . . . . .	3
1.2.1	Types abstraits . . . . .	3
1.2.2	Représentation Concrète . . . . .	3
<b>2</b>	<b>Signature des TAD</b>	<b>3</b>
2.1	Ville . . . . .	3
2.2	Gaule . . . . .	4
<b>3</b>	<b>Spécifications des TAD</b>	<b>5</b>
3.1	Ville . . . . .	5
3.1.1	Structure . . . . .	5
3.1.2	Spécification des fonctions et procédures de ville.h . . . . .	5
3.2	Gaule . . . . .	6
3.2.1	Structure en Tableau . . . . .	6
3.2.2	Structure en Liste chaînée . . . . .	6
3.2.3	Spécification des fonctions et procédures gaule.h . . . . .	7
3.2.4	Schématisation . . . . .	8
<b>4</b>	<b>Implémentation des structures</b>	<b>9</b>
4.1	Ville . . . . .	9
4.2	Gaule . . . . .	9
4.2.1	Tableau . . . . .	9

# 1 Introduction

## 1.1 Contexte

En référence à la BD Astérix et Obélix, "Le tour de Gaule d'Astérix", ce travail a pour but d'implémenter ce fameux tour sous la forme d'un type abstrait de données.

## 1.2 Enoncé

### 1.2.1 Types abstraits

Dans le cadre de ce projet, nous avons pour consigne de spécifier deux types abstraits :

#### 1. Ville

Ce type abstrait doit permettre de :

- créer une ville à partir de ses deux coordonnées X et Y et de son nom ;
- obtenir la coordonnée X d'une escale ;
- obtenir la coordonnée Y d'une escale ;
- obtenir le nom de la ville ;
- calculer la distance géographique entre deux ville ;
- enregistrer la spécialité gastronomique de la ville ;
- obtenir la spécialité gastronomique de la ville ;

#### 2. Gaule

Ce type abstrait doit permettre de :

- créer un tour de Gaule sur base de deux villes. Par définition, un tour nouvellement créé ne peut pas constituer un circuit ;
- déterminer si un tour de Gaule constitue un circuit ;
- déterminer le nombre de villes visitées durant le tour ;
- déterminer le nombre totale de spécialités gastronomiques dans le tour ;
- déterminer la spécialité gastronomique d'une ville du tour ;
- ajouter une ville à un tour ;
- supprimer une ville d'un tour ;

### 1.2.2 Représentation Concrète

Ces types abstraits doivent être représentés de 2 manières :

- Un tableau
- Une liste chaînée

# 2 Signature des TAD

## 2.1 Ville

Type :

- Ville

Utilise :

- *String*
- $\mathbb{R}$

Opérations :

- $\text{creer\_ville} : \text{String} \times \mathbb{R} \times \mathbb{R} \rightarrow \text{Ville}$
- $\text{get\_x\_ville} : \text{Ville} \rightarrow \mathbb{R}$
- $\text{get\_y\_ville} : \text{Ville} \rightarrow \mathbb{R}$

- $\text{get\_nom\_ville} : \text{Ville} \rightarrow \text{String}$
- $\text{get\_specialite\_ville} : \text{Ville} \rightarrow \text{String}$
- $\text{set\_specialite\_ville} : \text{Ville} \times \text{String} \rightarrow \text{Ville}$
- $\text{distance\_entre\_2\_villes} : \text{Ville} \times \text{Ville} \rightarrow \mathbb{R}$

Préconditions :

- $\emptyset$

Axiomes :  $\forall x, y \in \mathbb{R} \wedge \forall V_1, V_2 \in \text{Ville} \wedge \forall s, t \in \text{String}$

- $\text{distance\_entre\_2\_villes}(V_1, V_2) = \frac{\sqrt{(\text{get\_x\_ville}(V_2) - \text{get\_x\_ville}(V_1))^2 + (\text{get\_y\_ville}(V_2) - \text{get\_y\_ville}(V_1))^2}}{2}$
- $\text{distance\_entre\_2\_villes}(\text{set\_specialite\_ville}(V_1, s), \text{set\_specialite\_ville}(V_2, t)) = \text{distance\_entre\_2\_villes}(V_1, V_2)$
- $\text{get\_x\_ville}(\text{creer\_ville}(s, x, y)) = x$
- $\text{get\_y\_ville}(\text{creer\_ville}(s, x, y)) = y$
- $\text{get\_x\_ville}(\text{set\_specialite\_ville}(V_1, s)) = \text{get\_x\_ville}(V_1)$
- $\text{get\_y\_ville}(\text{set\_specialite\_ville}(V_1, s)) = \text{get\_y\_ville}(V_1)$
- $\text{get\_nom\_ville}(\text{creer\_ville}(s, x, y)) = s$
- $\text{get\_nom\_ville}(\text{set\_specialite\_ville}(V_1, s)) = \text{get\_nom\_ville}(V_1)$
- $\text{get\_specialite\_ville}(\text{set\_specialite\_ville}(V_1, s)) = s$
- $\text{get\_specialite\_ville}(\text{creer\_ville}(s, x, y)) = \text{NULL}$

## 2.2 Gaule

Type :

- *Gaule*

Utilise :

- *Ville*
- *Entiers*
- *String*
- *Booleen*

Opérations :

- $\text{cree\_nouveau\_tour} : \text{Ville} \times \text{Ville} \rightarrow \text{Gaule}$
- $\text{get\_nombre\_villes} : \text{Gaule} \rightarrow \text{Entiers}$
- $\text{ajoute\_ville} : \text{Gaule} \times \text{Ville} \rightarrow \text{Gaule}$
- $\text{supprime\_ville} : \text{Gaule} \rightarrow \text{Gaule}$
- $\text{get\_est\_circuit} : \text{Gaule} \rightarrow \text{Booleen}$
- $\text{get\_nombre\_specialite} : \text{Gaule} \rightarrow \text{Entiers}$
- $\text{get\_specialite} : \text{Gaule} \times \text{String} \rightarrow \text{String}$
- $\text{ville\_en\_double} : \text{Gaule} \times \text{Ville} \rightarrow \text{Booleen}$

Préconditions :

Axiomes :  $\forall G \in \text{Gaule}, \forall V_1, V_2 \in \text{Ville}$

- $\text{get\_nombre\_villes}(\text{ajoute\_ville}(G, V_1)) = \text{get\_nombre\_villes}(G) + 1$
- $\text{get\_nombre\_villes}(\text{supprime\_ville}(G)) = \text{get\_nombre\_villes}(G) - 1$   
si  $\text{get\_nombre\_villes}(G) > 0$   
sinon  $\text{get\_nombre\_villes}(\text{supprime\_ville}(G)) = \text{get\_nombre\_villes}(G)$
- $\text{get\_nombre\_villes}(\text{cree\_nouveau\_tour}(V_1, V_2)) = 2$
- $\text{get\_nombre\_specialite}(\text{ajoute\_ville}(G, V_1)) = \text{get\_nombre\_specialite}(G) + 1$   
si  $\text{ville\_en\_double}(G, V_1) = \text{False} \wedge \text{get\_specialite}(G, V_1) \neq \text{NULL}$   
sinon  $\text{get\_nombre\_specialite}(\text{ajoute\_ville}(G, V_1)) = \text{get\_nombre\_specialite}(G)$

- $\text{get\_nombre\_specialite}(\text{cree\_nouveau\_tour}(V_1, V_2)) = 2$   
 si  $\text{get\_specialite}(G, V_1) \neq \text{NULL} \wedge \text{get\_specialite}(G, V_2) \neq \text{NULL}$
- $\text{get\_nombre\_specialite}(\text{cree\_nouveau\_tour}(V_1, V_2)) = 1$   
 si  $\text{get\_specialite}(G, V_1) \neq \text{NULL} \wedge \text{get\_specialite}(G, V_2) = \text{NULL}$   
 $\vee \text{get\_specialite}(G, V_2) \neq \text{NULL} \wedge \text{get\_specialite}(G, V_1) = \text{NULL}$
- $\text{get\_nombre\_specialite}(\text{cree\_nouveau\_tour}(V_1, V_2)) = 0$   
 si  $\text{get\_specialite}(G, V_1) = \text{NULL} \wedge \text{get\_specialite}(G, V_2) = \text{NULL}$
- $\text{get\_est\_circuit}(\text{cree\_nouveau\_tour}(V_1, V_2)) = \text{False}$
- $\text{get\_specialite}(\text{cree\_nouveau\_tour}(V_1, V_2), V_1) = \text{get\_specialite\_ville}(V_1)$
- $\text{get\_specialite}(\text{cree\_nouveau\_tour}(V_1, V_2), V_2) = \text{get\_specialite\_ville}(V_2)$
- $\text{get\_specialite}(\text{ajoute\_ville}(G, V_1), V_1) = \text{get\_specialite\_ville}(V_1)$

## 3 Spécifications des TAD

### 3.1 Ville

#### 3.1.1 Structure

```

1  struct Ville_t{
2  char *nom;
3  float x;
4  float y;
5  char *specialite;
6  };

```

Extrait de Code 1 – Structure "Ville"

#### 3.1.2 Spécification des fonctions et procédures de ville.h

```

1  /*
2  *@pre : nom  $\neq \text{NULL}$   $\wedge x = x_0 \wedge y = y_0 \wedge \text{nom} = \text{nom}_0$ 
3  *@post : villeinit  $\wedge x = x_0 \wedge y = y_0 \wedge \text{nom} = \text{nom}_0 \wedge \text{get\_x\_ville}(\text{ville}) = \text{ville} \rightarrow x \wedge$ 
4  *get_y_ville(ville) = ville  $\rightarrow y \wedge \text{get\_nom\_ville}(\text{ville}) = \text{ville} \rightarrow \text{nom}$ 
5  */
6  Ville *creer_ville(char *nom, float x, float y);
7
8  /*
9  *@pre :  $\emptyset$ 
10 *@post : ville = NULL
11 */
12 void detruit_ville(Ville *ville);
13
14 /*
15 *@pre : ville  $\neq \text{NULL}$ 
16 *@post : ville = ville0  $\wedge \text{get\_x\_ville}(\text{ville}) = \text{ville} \rightarrow x$ 
17 */
18 float get_x_ville(Ville *ville);
19
20 /*
21 *@pre : ville  $\neq \text{NULL}$ 
22 *@post : ville = ville0  $\wedge \text{get\_y\_ville}(\text{ville}) = \text{ville} \rightarrow y$ 
23 */
24 float get_y_ville(Ville *ville);
25
26 /*

```

```

27  *@pre : ville ≠ NULL
28  *@post : ville = ville0 ∧ get_nom_ville(ville) = ville->nom
29  */
30  char *get_nom_ville(Ville *ville);
31
32  /*
33  *@pre : ville ≠ NULL
34  *@post : ville = ville0 ∧ get_specialite_ville(ville) = ville->specialite
35  */
36  char *get_specialite_ville(Ville *ville);
37
38  /*
39  *@pre : ville ≠ NULL ∧ specialite ≠ NULL ∧ specialite = specialite0
40  *@post : ville = ville0 ∧ specialite = specialite0 ∧ get_specialite_ville(ville) = specialite
41  */
42  void set_specialite_ville(Ville *ville, char *specialite);
43
44  /*
45  *@pre : ville1 = ville10 ≠ NULL ∧ ville2 = ville20 ≠ NULL
46  *@post : ville1 = ville10 ∧ ville2 = ville20 ∧
47  *distance_entre_2_villes(ville1, ville2) =
48  *√((get_x_ville(ville2) - get_x_ville(ville1))2 + (get_y_ville(ville2) - get_y_ville(ville1))2)
49  */
50  float distance_entre_2_villes(Ville *ville1, Ville *ville2);
51
52  /*
53  *@pre : ∅
54  *@post : retourne la taille mémoire de la struct Ville
55  */
56  int size_ville(void);

```

Extrait de Code 2 – Spécification des fonctions et procédures du header "ville.h"

## 3.2 Gaule

### 3.2.1 Structure en Tableau

```

1  struct Gaule_t{
2  Ville **tableau_ville;
3  int nombre_villes;
4  int est_circuit;
5  int nombre_specialites;
6  };

```

Extrait de Code 3 – Structure "Gaule" dans l'implémentation en tableau

### 3.2.2 Structure en Liste chaînée

Pour la liste chaînée, une deuxième structure vient s'ajouter. La première, comme pour les tableaux, garde les informations sur la liste et la deuxième sont les structures qui correspondront chacune à une des villes avec un pointeur sur l'élément suivant et précédent de la liste.

```

1  struct Gaule_t{
2  Cellule_Gaule *premiere_cellule;
3  Cellule_Gaule *derniere_cellule;
4  int nombre_villes;
5  int est_circuit;
6  int nombre_specialites;

```

```

7   };
8
9   struct Cellule_Gaule_t{
10  Cellule_Gaule *cellule_suivante;
11  Cellule_Gaule *cellule_precedente;
12  Ville *ville;
13  };

```

Extrait de Code 4 – Structure "Gaule" dans l'implémentation en tableau

### 3.2.3 Spécification des fonctions et procédures gaule.h

```

1  /*
2  *@pre : ville1 = ville1_0 ≠ NULL ∧ ville2 = ville2_0 ≠ NULL
3  *@post : ville1 = ville1_0 ∧ ville2 = ville2_0 ∧ tour_init
4  */
5  Gaule *cree_nouveau_tour(Ville *ville1, Ville *ville2);
6
7  /*
8  *@pre : ∅
9  *@post : tour = NULL
10 */
11 void detruit_tour(Gaule *tour);
12
13 /*
14 *@pre : tour ≠ NULL ∧ nombre_villes > 0
15 *@post : get_nombre_villes(tour) = nombre_villes
16 */
17 void set_nombre_villes(Gaule *tour, int nombre_villes);
18
19 /*
20 *@pre : tour ≠ NULL
21 *@post : get_nombre_villes = get_nombre_villes(tour)
22 */
23 int get_nombre_villes(Gaule *tour);
24
25 /*
26 *@pre : tour ≠ NULL ∧ ville ≠ NULL
27 *@post : get_nombre_villes(ajoute_ville(tour, ville)) = get_nombre_villes(tour) + 1 ∧
28 *ville ajoutée à tour
29 */
30 int ajoute_ville(Gaule *tour, Ville *ville);
31
32 /*
33 *@pre : tour ≠ NULL ∧ get_nombre_villes(tour) > 0
34 *@post : dernière ville retirée de tour
35 *^get_nombre_villes(tour) = get_nombre_villes(tour_0) - 1
36 */
37 void supprime_ville(Gaule *tour);
38
39 /*
40 *@pre : ∅
41 *@post : compare_string(chaine1, chaine2) = 0 si chaine1 = chaine2, -1 sinon
42 */
43 int compare_string(char *chaine1, char *chaine2);
44
45 /*
46 *@pre : tour ≠ NULL
47 *@post : get_est_circuit(tour) = 1

```

```

48  *si premiere ville et derniere ville de tour sont les mêmes
49  */
50  void maj_est_circuit(Gaule *tour);
51
52  /*
53  *@pre : tour ≠ NULL
54  *@post : retourne tour->est\_circuit
55  */
56  int get_est_circuit(Gaule *tour);
57
58  /*
59  *@pre : tour ≠ NULL
60  *@post : retourne tour->nombre\_specialites
61  */
62  int get_nombre_specialites(Gaule *tour);
63
64  /*
65  *@pre : tour ≠ NULL ∧ nom_ville ≠ NULL
66  *@post : ∃V1 ∈ Ville, get_nom_ville(Ville) = nom_ville ⇒ get_specialite(tour, nom_ville) =
67  *get_specialite_ville(V1)
68  */
69  char *get_specialite(Gaule *tour, char *nom_ville);
70
71  /*
72  *@pre : tour ≠ NULL ∧ nom_ville ≠ NULL
73  *@post : ville_en_double(tour, nom_ville) = 1,
74  *∃V1, V2 ∈ Ville, V1, V2 ⊂ tour ∧ get_nom_ville(V1) = get_nom_ville(V2)
75  *sinon ville_en_double(tour, nom_ville) = 0
76  */
77  int ville_en_double(Gaule *tour, char *nom_ville);

```

Extrait de Code 5 – Spécification des fonctions et procédures du header "gaule.h"

### 3.2.4 Schématisation

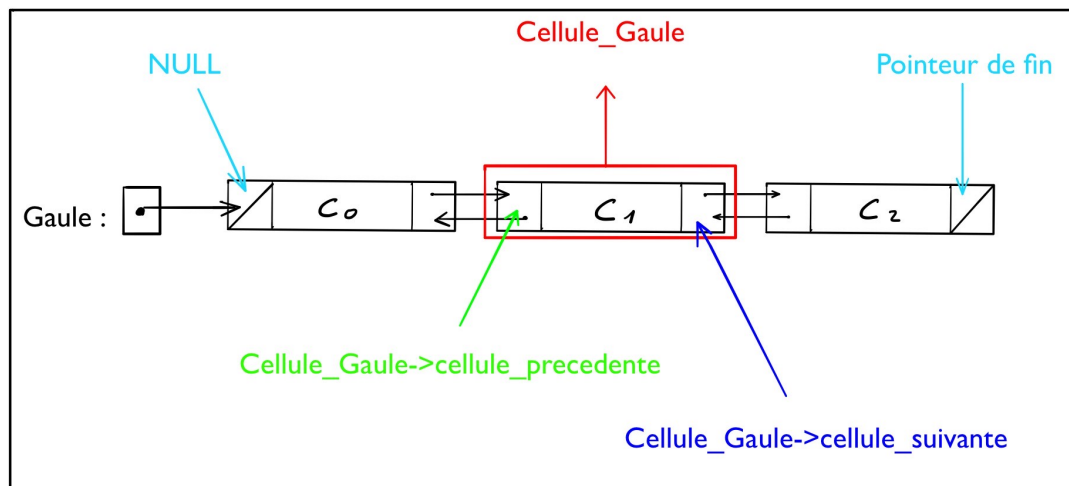


FIGURE 1 – Schéma de la strucutre sous forme de liste chaînée



## 4 Implémentation des structures

### 4.1 Ville

Le header ville.h est implémenté par le fichier ville.c. Toutes les implémentations des fonctions/procédures sont très simples. En effet, la majorité d'entre elles ne sont que des accesseurs/-setteurs et pour les autres il s'agit d'opérations mathématique ou d'allocation de mémoire très simple ayant toutes une complexité d'ordre  $O(1)$ .

### 4.2 Gaule

#### 4.2.1 Tableau

##### 1. accesseurs

Comme pour ville, certaines fonctions sont uniquement des accesseurs/setteurs et donc retournent une valeur d'un champ de Gaule. C'est notamment le cas des fonctions/procédures `get_nombre_specialites`, `get_est_circuit`, `get_nombre_villes` et `set_nombre_villes` pour lesquelles je ne rentrerai donc pas dans les détails.

##### 2. ajoute\_ville

Cette fonction va s'occuper d'ajouter au tableau de villes, la ville donnée en argument en s'assurant que celle-ci n'est pas la même que la dernière du tableau. Elle va également s'occuper de mettre à jour certains champs de la structure Gaule(`est_circuit`, `nombre_villes` et `nombre_specialites`)

- SP1 : Vérification que l'on ne rajoute pas la même ville que la dernière de la liste.
- SP2 : Mise à jour du nombre de ville du tour
- SP3 : Aggrandissement de l'espace mémoire de tableau ville et ajout de la nouvelle ville
- SP4 : Mise à jour du nombre de spécialités et du champ de Gaule permettant de savoir si le tour est un circuit ou non

##### 3. supprime\_ville

A l'inverse de `ajoute_ville`, cette procédure retire la dernière ville du tableau à qu'il ne soit pas vide.

- SP1 : Mise à jour du nombre de spécialités et du nombre de villes
- SP2 : Realloque la mémoire du tableau afin d'en retirer la dernière ville
- SP3 : Mise à jour du champ de Gaule permettant de savoir si le tour est un circuit ou non

##### 4. compare\_string

Cette fonction permet de savoir si les 2 chaînes de caractères données en argument sont les mêmes. Pour ce faire, on compare tous caractères un à un de la première chaîne avec ceux respectifs de la deuxième. Ce problème va donc parcourir au Maximum l'entièreté de la première chaîne de caractères.

Spécifions pour ce problème  $\forall s \in \text{string}, \exists i \in \text{Entier}, \text{taille}(s) = i, s[i - 1] = 0$

```
1  int i=0;
2  {Inv : chaine1 = chaine10 ∧ chaine2 = chaine20 ∧ 0 ≤ i ≤ length(chaine1)}
3  while (chaine1[i] != 0) {
4      {Inv ∧ B : chaine1 = chaine10 ∧ chaine2 = chaine20 ∧ 0 ≤ i < length(chaine1)}
5      if (chaine2[i] == 0 || (chaine1[i] != chaine2[i]))
6          return -1;
7      i++;
8  }
9  {Inv ∧ ¬B ⇒ chaine1 = chaine10 ∧ chaine2 = chaine20 ∧ i = length(chaine1)}
```

```
10 | if(chaine2[i-1]!=0)
11 |     return -1;
12 | return 0;
```

Extrait de Code 6 – Boucle pour comparer les 2 chaines de caractères

5. maj\_est\_circuit

Cette procédure redéfinit le booléen du champ est\_circuit de Gaule. Dans le cas où le tour contient au moins 3 villes, elle compare le nom de la première et dernière ville. Le tour est un circuit s'ils correspondent.

6. get\_specialite

Cette fonction retourne la specialite de la ville dont le nom a été donné en argument.

—