C++ : classes, attributs et méthodes

TP C++

Classes, attributs et méthodes

Eric Ramat
eric.ramat@univ-littoral.fr

2 novembre 2023

Durée : 3 heures

1 Objectifs

Le deuxième TP a pour objectif d'utiliser pleinement la notion de classe du C++ et quelques classes de la STL (`std::string`, `std::valarray`, `std::array`, `std::vector` et naturellement les pointeurs intelligents). Pour le moment, nous avons utilisé la notion de structure avec des méthodes. Avec les classes, il va falloir préciser la visibilité des attributs et des méthodes et qualifier les méthodes de const ou non const.

Afin d'illustrer les diverses techniques, on va implémenter une partie du diagramme UML de classes ci-dessous. Ce diagramme modélise les éléments d'un réseau routier.

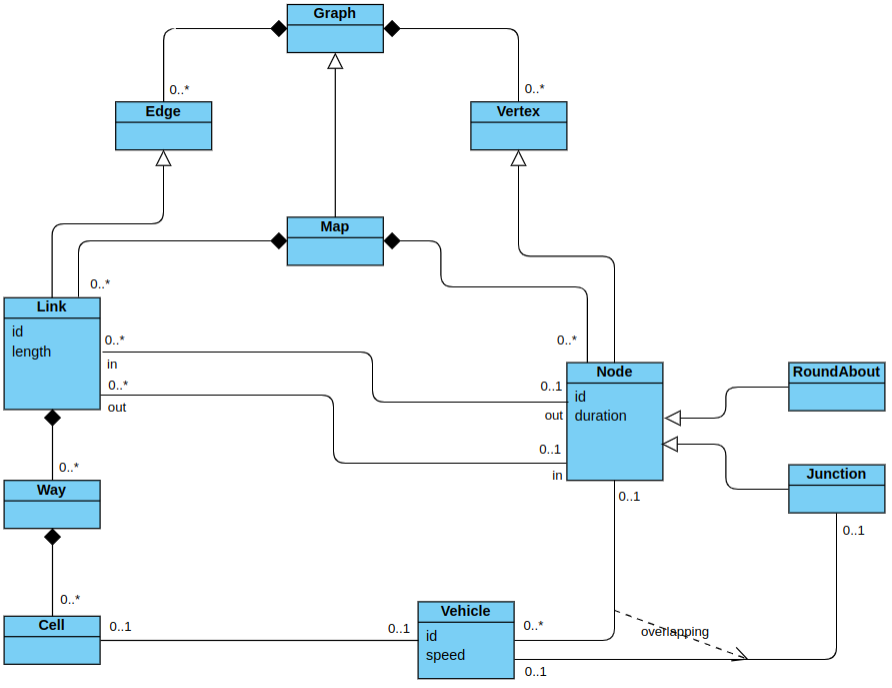


Diagramme de classes

2 Quelques définitions

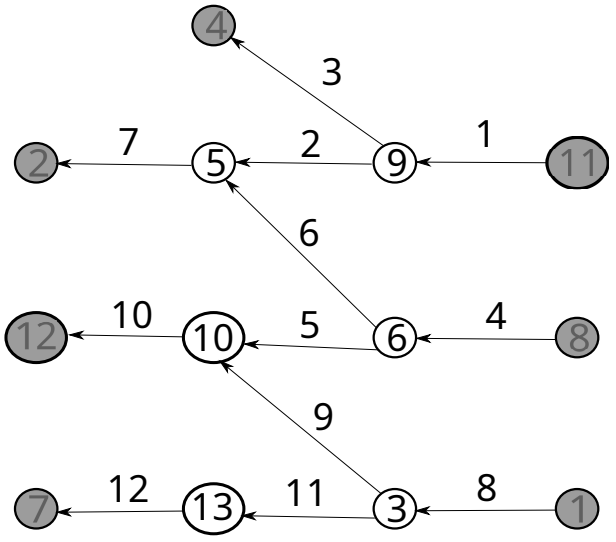
Un *link* représente un tronçon de route. Il est défini par un identifiant et une longueur exprimée en mètre arrondie à la dizaine de mètres. Un *link* est découpé en voies (*way*) et une voie est décomposée en cellules (*cell*). Une cellule est l'espace occupé potentiellement par un véhicule. La longueur d'une cellule est de 10m. Les véhicules avancent de cellule en cellule. Arrivé à la fin d'un *link* (c'est à dire dans la dernière cellule du *link*), le véhicule sort du tronçon et entre dans le *node* aval s'il existe.

Un *node* est un élément reliant des tronçons (*link*) de route. Il peut être spécialisé en carrefour plus ou moins complexe. Pour le moment, il est défini par un identifiant, une durée pour le "traverser", la liste des tronçons amonts et des tronçons avals. Un *node* peut être lié à un véhicule lorsque ce dernier traverse le *node*.

Une *map* regroupe l'ensemble des *links* et *nodes*. Elle prends en charge la représentation du réseau de routes.

Un véhicule (*vehicle*) est défini par un identifiant et sa vitesse. On considére que la vitesse sur les routes est identique partout et donc c'est la vitesse du véhicule qui défini le temps passé dans une cellule.

De manière plus abstraite, une *map* est un graphe (*Graph*) composé d'arcs (*Edge*) et de sommets (*Vertex*).



Exemple de réseau de tronçons

3 Travail demandé

Avant de commencer le TP, il est nécessaire de forker le dépôt `git` contenant le projet avec un début de code. Vous devez voir apparaître dans votre compte gitlab (gitlab.dpt-info.univ-littoral.fr) le projet "traffic". Toutes les classes mentionnées précédemment sont déjà définies avec uniquement un constructeur. Nous allons compléter les différentes classes. Il faut faire les bons choix d'implémentation (instance, référence ou pointeur intelligent) et utiliser les bons types pour représenter les relations entre les classes.

Question 1. Compléter la classe `Vehicle` en ajoutant les attributs et en modifiant le constructeur.

Question 2. Compléter la classe `Cell` en ajoutant un attribut qui "stocke" la présence d'un véhicle dans la cellule. Développer trois méthodes : *is_free* (retourne true s'il n'y a pas de véhicule présent dans

la cellule) et *push_vehicle* (place un véhicule dans la cellule).

Question 3. Compléter la classe `Way` en ajoutant un attribut qui représente l'ensemble des cellules d'une voie de circulation et en modifiant le constructeur. Pour rappel, le nombre de cellules est égale à la division entière entre la longueur exprimée en mètre de la voie et la taille d'une cellule (constante égale à 10m). Ajouter trois méthodes : *is_free* (retourne true s'il n'y a pas de véhicule présent dans la première cellule), *push_vehicle* (place un véhicule dans la première cellule) et *ready_to_exit* (retourne true si un véhicle est présent dans la dernière cellule).

Question 4. Compléter la classe `Link` (tronçon) en ajoutant les attributs (l'identifiant, la longueur, le noeud connecté à l'entrée du tronçon, le noeud connecté à la sortie du tronçon et l'ensemble des voies) et en modifiant le constructeur. Ajouter quatre méthodes : *attach_in_node* (relie le tronçon au noeud d'entrée), *attach_out_node* (relie le tronçon au noeud de sortie), *is_free* (retourne true si l'une des voies est libre) et *push_vehicle* (place un véhicule dans la première voies libres).

Question 5. Compléter la classe `Node` en ajoutant les attributs (l'identifiant, la durée de traversée, la liste des tronçons entrants, la liste des tronçons sortants et le véhicle s'il est présent) et en modifiant le constructeur. Le constructeur doit pré-réserver les emplacements dans la structure de données choisie pour les listes. Ajouter quatre méthodes : *attach_in_link* (relie le noeud à l'un des tronçons entrants), *attach_out_link* (relie le noeud à l'un des tronçons sortants), *is_free* (retourne true s'il n'y a pas de véhicules présent dans le noeud) et *push_vehicle* (place un véhicule dans le noeud).

Question 6. Compléter la classe `Map` en ajoutant les attributs (liste des tronçons et liste des noeuds). Ajouter cinq méthodes : *add_link* (ajout d'un tronçon), *add_node* (ajout d'un noeud), *attach_in_link* (relie le tronçon entrant à un noeud), *attach_out_link* (relie un noeud à un tronçon sortant) et *push_vehicle* (place un véhicule dans un noeud désigné par son id).

Toutes les méthodes doivent les identifiants des tronçons et des noeuds.

Afin de tester votre implémentation, voici le code de la fonction *main* :

```
#include "map.hpp"

using namespace traffic;

void create_map(Map &map) {
  map.add_link("L1", 200, 1);
  map.add_link("L2", 500, 1);
  map.add_link("L3", 800, 1);
  map.add_link("L4", 900, 1);
  map.add_link("L5", 300, 1);
  map.add_link("L6", 400, 1);
  map.add_link("L7", 1000, 2);
  map.add_link("L8", 500, 1);
  map.add_link("L9", 300, 1);
  map.add_link("L10", 200, 1);
  map.add_link("L11", 400, 1);
  map.add_link("L12", 700, 1);

  map.add_node("N1", 2, 0, 1);
  map.add_node("N2", 2, 1, 0);
```

```
  map.add_node("N3", 2, 1, 2);
  map.add_node("N4", 2, 1, 1);
  map.add_node("N5", 2, 2, 1);
  map.add_node("N6", 2, 1, 2);
  map.add_node("N7", 2, 1, 0);
  map.add_node("N8", 2, 0, 1);
  map.add_node("N9", 2, 1, 2);
  map.add_node("N10", 2, 2, 1);
  map.add_node("N11", 2, 0, 1);
  map.add_node("N12", 2, 1, 0);
  map.add_node("N13", 2, 1, 0);

  map.attach_out_link(0, "N1", "L8");
  map.attach_out_link(0, "N8", "L4");
  map.attach_out_link(0, "N11", "L1");
  map.attach_in_link(0, "L7", "N2");
  map.attach_in_link(0, "L10", "N12");
  map.attach_in_link(0, "L12", "N7");
  map.attach_in_link(0, "L8", "N3");
  map.attach_out_link(0, "N3", "L9");
  map.attach_out_link(1, "N3", "L11");
  map.attach_in_link(0, "L4", "N6");
  map.attach_out_link(0, "N6", "L6");
  map.attach_out_link(1, "N6", "L5");

  map.attach_in_link(0, "L1", "N9");
  map.attach_out_link(0, "N9", "L3");
  map.attach_out_link(1, "N9", "L2");

  map.attach_in_link(0, "L2", "N5");
  map.attach_in_link(1, "L6", "N5");
  map.attach_out_link(0, "N5", "L7");

  map.attach_in_link(0, "L5", "N10");
  map.attach_in_link(1, "L9", "N10");
  map.attach_out_link(0, "N10", "L10");

  map.attach_in_link(0, "L11", "N4");
  map.attach_out_link(0, "N4", "L12");

  map.attach_in_link(0, "L3", "N13");
}

int main() {
  Map map;
  auto vehicle = std::make_unique<Vehicle>("V1", 10);

  create_map(map);
  map.push_vehicle(vehicle, "N1");
  return 0;
```

```
}
```