Remise à niveau en C - 2

I Présentation du sujet de TP

La partie qui suit est directement tirée du TP1 C++ d'Eric Ramat.

1 Extrait du TP1

Ce TP est le premier d'une série qui a pour objectif de développer un simulateur de trafic urbain. A partir d'un plan des rues, carrefours, ..., il faut pouvoir simuler, dans un premier temps, le déplacement de véhicules sur des voies simples. Puis nous ajouterons les carrefours avec des stops, des feux, ... Dans un troisième temps, des réseaux de rues plus complexes ainsi que des générateurs de véhicules seront introduits. Dans ce TP, on va s'intéresser aux véhicules, aux rues et aux jonctions. Le simulateur doit nous donner une trace des déplacements des véhicules.

2 Structures de données

Le code contient quatre structures (map, link, node et vehicle).

Un véhicule (vehicle) est simplement défini par un identifiant.

Un node est un élément reliant des tronçons (link) de route. Il sera plus tard 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.

Un link représente un tronçon de route ou de rue dans un milieu urbain. Il est défini par un identifiant et une longueur exprimée en mètre. Un link est découpé en cellules (cell). Une cellule est l'espace occupé potentiellement par un véhicule. 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 et entre dans un node s'il existe un node rattaché au link.

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

II Prototypes des fonctions

Voici les prototypes des fonctions qu'il faudra développer :

1 map.h

```
struct Map *create_map(unsigned int link_number, unsigned int
↪  node_number);

void destroy_map(struct Map *map);

void add_link_to_map(struct Link *link, struct Map *map);

void add_node_to_map(struct Node *node, struct Map *map);

void run_map(struct Map* map, unsigned int time);
```

2 node.h

```
struct Node *create_node(char *id, unsigned int duration, unsigned int
↪  in_number, unsigned int out_number);

void destroy_node(struct Node *node);

void add_vehicle_to_node(struct Vehicle *vehicle, struct Node *node,
↪  unsigned int time);

void attach_input_link_to_node(struct Link *link, struct Node *node);

void attach_output_link_to_node(struct Link *link, struct Node *node);
```

3 link.h

```
struct Link *create_link(char *id, unsigned int length);

void destroy_link(struct Link *link);

void add_vehicle_to_link(struct Vehicle *vehicle, struct Link *link,
↪  unsigned int time);

void attach_in_node_to_link(struct Node *node, struct Link *link);

void attach_out_node_to_link(struct Node *node, struct Link *link);
```

```
    struct Vehicle *go_out_link(struct Link *link, unsigned int time);

    struct Vehicle *run_link(struct Link *link, unsigned int time);
```

4 vehicle.h

```
    struct Vehicle *create_vehicle(char *id);

    void destroy_vehicle(struct Vehicle *vehicle);
```

III Compilation

Pour ce projet, nous allons écrire un makefile pour faciliter la compilation. En plus du fichier `./main` et des autres fichiers sources, on crée un le fichier `./makefile` suivant :

```
all : main.o node.o link.o map.o vehicle.o
        gcc -o TP1 main.o node.o link.o map.o vehicle.o
        rm -f *.o
main.o : main.c
        gcc -c -Wall main.c

structure.o : node.c
        gcc -c -Wall node.c

link.o : link.c
        gcc -c -Wall link.c

map.o : map.c
        gcc -c -Wall map.c

vehicle.o : vehicle.c
        gcc -c -Wall vehicle.c

clean :
        rm -f TP1 *.o
```

Il suffit alors de rentrer la commande `make all` pour compiler le projet.