# ALGORITHME D'ETABLISSEMENT DES PRIX

# Sommary :

```
29. static double convert_string_double(std::string ss);
30. static std::string convert_int_string(int value);
31. static std::string convert_double_string(double value);
32. static void free_price_sector();
33. static void free_price_circulation_category();
34. static void free_basic_price();
35. static void free_benefice_niani();
36. static void free_correspondance( );
37. static void free_vehicule_circulation();
38. static void free_circulation();
39. static double getprice_sector(int sector,int type_vehicle);
40. static double getprice_circulation_category(int circulation_category);
41. static double getbasic_price(int type_vehicle);
42. static double getbenefices_niani(int distance_metter);
43. static double price(int distance_metter,int sector,int type_vehicle,int
    circulation_category=0);
```

## C. Politiques d'établissement des prix (partie technique).

# A. Spécificité des fichiers utilisés

Les prix seront stockés dans des fichiers csv que les algorithmes ouvriront lorsqu'ils recevront des requêtes. Ce procédé permettra de modifier les prix en temps réel sans affecter le fonctionnement du système via des algorithmes extérieurs gérés par les ingénieurs qui se chargeront d'établir les prix.

Les fichiers des prix seront stockés dans un sous-dossier où se trouve les algorithmes pour permettre un accès plus rapide à ces fichiers à l'algorithme et la modification de ces fichiers demandera d'avoir un mot de passe pour éviter une suppression, des attaques informatiques ou des incohérences.

Les différents champs des fichiers csv utilisés seront organisés en ligne, séparé par des points virgules avec la première ligne réservée pour l'entête des champs. Ici nous avons listé les fichiers qui sont utilisés en ce moment même.

1. Le fichier **PRICE_SECTOR.csv**

Dans ce fichier sera stocké les prix au mètres appliqués en fonction du secteur et du type de véhicule. Le fichier est un fichier .csv comportant plusieurs lignes et trois colonnes. La première ligne a pour argument sector, vehicle et price séparés par des points virgules (;) : cette ligne constitue l'entête de notre fichier. A part la première ligne, les autres lignes comporteront : le secteur qui sera un nombre entier, le véhicule qui sera aussi un nombre entier et le prix par mètre qui sera un réel. Ces trois champs seront séparés par un point-virgule.

**Exemple :**

```
sector;vehicle;price
10;5;1500
14;2;2000
96;7;800
```

2. Le fichier PRICE_CILCULATION_CATEGORY.csv

Ce fichier contient les prix en fonction de l'intensité de la circulation. L'intensité de la circulation sera catégorisée en différentes catégories et un prix sera appliqué en fonction de la catégorie considéré.

Le fichier sera un fichier .csv comportant plusieurs lignes et deux colonnes. La première ligne a pour argument catégorie, price séparés par des points virgules (;). : cette ligne constitue l'entête de notre fichier. A part la première ligne, les autres lignes comporteront : la catégorie de la circulation qui sera un entier et le prix associé à cette catégorie qui est un réel.

**Exemple :**

```
categorie;price
5;1500
2;2000
7;800
```

3. Le fichier BASIC_PRICE.csv

Dans ce fichier sera stocké les prix de base en fonction du type de véhicule. C'est-à-dire le prix minimale pour qu'un véhicule se mette en mouvement.

Le fichier sera un fichier .csv comportant plusieurs lignes et deux colonnes. La première ligne a pour argument catégorie, price séparés par des points virgules (;). : cette ligne constitue l'entête de notre fichier. A part la première ligne, les autres lignes comporteront : la catégorie du vehicule qui sera un entier et le prix associé à cette catégorie qui est un réel.

**Exemple :**

```
categorie;price
5;1500
2;2000
7;800
```

4. Le fichier BENEFICES_NIANI.csv

Dans ce fichier sera stocké les bénéfices de la société NIANI au mètres appliqués en fonction de la distance de la course. Le fichier est un fichier .csv comportant plusieurs lignes et deux colonnes. La première ligne a pour argument category et price_category séparés par des points virgules (;) : cette ligne constitue l'entête de notre fichier. A part la première ligne, les autres lignes comporteront : la catégorie qui sera un nombre entier et le prix de cette catégorie qui sera un réel. Ces trois champs seront séparés par un point-virgule.

```
categorie;price_categori
5;1500
2;2000
7;800
```

5. Le fichier CORRESPONDANCE.csv

Dans ce fichier sera stocké les correspondances entre les codes(catégories) représenter les secteurs donnés et leur véritable nom (qui seront des noms de villes, de pays, de quartier) en plus de cela, nous aurons 4 colonnes qui représentent chacune des coordonnées GPS d'un point dont les 4 forment un carrés qui représentent le lieu donné.

Le fichier est un fichier .csv comportant plusieurs lignes et 6 colonnes. La première ligne a pour argument secteur, nom_secteur, coordonnees1, coordonnees2, coordonnees3, coordonnees4 séparés par des points virgules (;) : cette ligne constitue l'entête de notre fichier. A part la première ligne, les autres lignes comporteront : le secteur qui sera un nombre entier, le nom du secteur qui est une chaine de caractère, les coordonnées Gps des quatre points qui seront un couples . Ces champs seront séparés par un point-virgule.

**Exemple :**

```
secteur;nom_secteur;coordonnees1;coordonnees2;coordonnees3;coordonnees4
5;dakar;(85692,962656);(856554,64518)(645651,865426)(564165,564165)(156412,65432)
79;thies;(85692,962656);(856554,64518)(645651,865426)(564165,564165)(156412,65432)
63;kaolack;(85692,962656);(856554,64518)(645651,865426)(564165,564165)(156412,65432)
75;mboro;(85692,962656);(856554,64518)(645651,865426)(564165,564165)(156412,65432)
89;kabir;(85692,962656);(856554,64518)(645651,865426)(564165,564165)(156412,65432)
```

6. Le fichier VEHICULE_CIRCULATION.csv

Dans ce fichier sera stocké les correspondances entre les codes(catégories) représenter les véhicules et leur véritable nom (qui seront les noms comme moto, vélo…).

Le fichier est un fichier .csv comportant plusieurs lignes et 2 colonnes. La première ligne a pour argument categorie, nom_vehicle séparés par des points virgules (;) : cette ligne constitue l'entête de notre fichier. A part la première ligne, les autres lignes comporteront : la catégorie du véhicule qui sera un nombre entier et le nom du véhicule qui est une chaine de caractère.

```
categorie;nom_vehicule
5;moto
9;jakarta
```

7. Le fichier CIRCULATION.csv

Dans ce fichier sera stocké les correspondances entre les codes(catégories) représenter les états de la circulation et leur véritable nom (qui seront des noms qu'on leurs a donné).

Le fichier est un fichier .csv comportant plusieurs lignes et 2 colonnes. La première ligne a pour argument categorie, nom_circulation séparés par des points virgules (;) : cette ligne constitue l'entête de notre fichier. A part la première ligne, les autres lignes comporteront : la catégorie de la circulation qui sera un nombre entier et le nom de la catégorie qui est une chaine de caractère.

```
categorie;nom_circulation
1;pas de circulation
2;inondation
3;bouchon
```

8. Le fichier UPDATE_TIME.csv

Dans ce fichier sera stocké le temps de mise à jour des données en seconde.

C'est un fichier csv avec un seul argument qui est un nombre entier qui représente le nombre de second nécessaire pour la mise à jour du temps.

```
600
```

## B. Spécificité des fonctions en c++

Les fonctions utilisées dans cette section sont écrites en C++ et seront utilisées pour répondre à une requête sur les prix d'un trajet. Les données seront lues depuis les fichiers et chargées en mémoire RAM par intervalle de temps. Cela permettra de pouvoir modifier les données tout en laissant le système opérationnel. L'inconvénient de cette méthode est que toute modification faite sur les données prendra une durée comprise entre 0 et la période de renouvellement des relectures des données pour être prise en compte. Le fait de charger les données dans la RAM permettra une vitesse d'accès aux données plus élevée et les relectures par intervalle de temps de les modifier sans arrêter le système ni utiliser un système de requête sur la mémoire qui prendrait beaucoup de temps et qui ne permettrait pas une vérification.

Pour les variables, notons que l'utilisation des conteneurs permets de stocker autant de donner qu'on veut cela évite ainsi de supposer la taille des données à l'avance. L'utilisation des pointeurs est justifiée par le fait de vouloir éviter des copies des données (passage de valeur) lors de passages de données

en argument de fonction. L'utilisation du mot clé static permet d'éviter de déclarer des variables et ains avoir des fonctions et des variables de classe.

Toutes les fonctions définies dans cette partie sont définies dans la classe ''Price'' et donc nécessite l'utilisation de cast avec le nom de cette classe.

1. static std::vector<std::vector<std::string> > *price_sector;

Ce conteneur vecteur est destiné à stocker dans la mémoire RAM le tableau représentant le fichier PRICE_SECTOR.csv. Les données seront stockées en String : nous utiliserons les fonctions de conversion en cas de besoin.

2. static std::vector<std::vector<std::string> > *price_circulation_category;

Ce conteneur vecteur est destiné à stocker dans la mémoire RAM le tableau représentant le fichier PRICE_CIRCULATION_CATEGORY.csv. Les données seront stockées en String : nous utiliserons les fonctions de conversion en cas de besoin.

3. static std::vector<std::vector<std::string> > *basic_price;

Ce conteneur vecteur est destiné à stocker dans la mémoire RAM le tableau représentant le fichier BASIC_PRICE.csv. Les données seront stockées en String : nous utiliserons les fonctions de conversion en cas de besoin.

4. static std::vector<std::vector<std::string> > *benefice_niani;

Ce conteneur vecteur est destiné à stocker dans la mémoire RAM le tableau représentant le fichier BENEFICE_NIANI.csv. Les données seront stockées en String : nous utiliserons les fonctions de conversion en cas de besoin.

5. static std::vector<std::vector<std::string> > *correspondance;

Ce conteneur vecteur est destiné à stocker dans la mémoire RAM le tableau représentant le fichier CORRESPONDANCE.csv. Les données seront stockées en String : nous utiliserons les fonctions de conversion en cas de besoin.

6. static std::vector<std::vector<std::string> > *vehicule_circulation;

Ce conteneur vecteur est destiné à stocker dans la mémoire RAM le tableau représentant le fichier VEHICULE_CIRCULATION.csv. Les données seront stockées en String : nous utiliserons les fonctions de conversion en cas de besoin.

7. static std::vector<std::vector<std::string> > *circulation;

Ce conteneur vecteur est destiné à stocker dans la mémoire RAM le tableau représentant le fichier CIRCULATION.csv. Les données seront stockées en String : nous utiliserons les fonctions de conversion en cas de besoin.

8. static unsigned long periode_seconde;

Cette variable contient le temps en seconde utilisé pour recharger les données en mémoire.

9. static unsigned long getperiode_seconde();

Cette fonction return la variable période seconde.

10. static std::vector<std::vector<std::string> > getprice_sector();

Cette fonction return une copie du tableau contenu dans price_sector.

11. static std::vector<std::vector<std::string> > getprice_circulation_category();

Cette fonction return une copie du tableau contenu dans price_circulation_category.

12. static std::vector<std::vector<std::string> > getbasic_price();

Cette fonction return une copie du tableau contenu dans basic_price.

13. static std::vector<std::vector<std::string> > getbenefice_niani();

Cette fonction return une copie du tableau contenu dans benefice_niani.

14. static std::vector<std::vector<std::string> > getcorrespondance();

Cette fonction return une copie du tableau contenu dans correspondance.

15. static std::vector<std::vector<std::string> > getvehicule_circulation();

Cette fonction return une copie du tableau contenu dans vehicule_circulation.

16. static std::vector<std::vector<std::string> > getcirculation();

Cette fonction return une copie du tableau contenu dans circulation.

17. static void load_periode_seconde(unsigned long periode_seconde, const char* namefile);

Cette fonction charge le temps de mise à jour des données contenue dans le fichier namefile dans la variable période_seconde.

18. static void load_price_sector( std::vector<std::vector<std::string> >* price_sector, const char* namefile);

Cette fonction charge les données de price_sector contenue dans le fichier ayant pour nom namefile dans le conteneur price_sector.

19. static void load_price_circulation_category(std::vector<std::vector<std::string> >* price_circulation_category, const char* namefile);

Cette fonction charge les données de price_circulation_category contenue dans le fichier ayant pour nom namefile dans le conteneur price_circulation_category.

20. static void load_basic_price(std::vector<std::vector<std::string> > *basic_price, const char* namefile);

Cette fonction charge les données de basic_price contenue dans le fichier ayant pour nom namefile dans le conteneur basic_price.

21. static void load_benefice_niani(std::vector<std::vector<std::string> > *benefice_niani, const char* namefile);

Cette fonction charge les données de benefice_niani contenue dans le fichier ayant pour nom namefile dans le conteneur benefice_niani.

22. static void load_correspondance( std::vector<std::vector<std::string> >* correspondance, const char* namefile);

Cette fonction charge les données de correspondance contenue dans le fichier ayant pour nom namefile dans le conteneur correspondance.

23. static void load_vehicule_circulation(std::vector<std::vector<std::string> >* vehicule_circulation, const char* namefile);

Cette fonction charge les données de vehicule_circulation contenue dans le fichier ayant pour nom namefile dans le conteneur vehicule_circulation.

24. static void load_circulation(std::vector<std::vector<std::string> >* circulation, const char* namefile);

Cette fonction charge les données de circulation contenue dans le fichier ayant pour nom namefile dans le conteneur circulation.

25. static void load_data_per_time(unsigned long periode);

Cette fonction charge les données des différents conteneurs par intervalle de temps égale au à la période spécifier. La première appelle à cette fonction charge les données directement dans les conteneurs.

26. static int convert_string_int(std::string ss);

Cette fonction convertie une chaine de caractère représentant un entier qui est ss en une valeur entière numérique.

27. static double convert_string_double(std::string ss);

Cette fonction convertie une chaine de caractère représentant un réel qui est ss en une valeur réelle numérique.

28. static std::string convert_int_string(int value);

Cette fonction convertie un valeur numérique représentant un entier qui est value en une chaine de caractère.

29. static std::string convert_double_string(double value);

Cette fonction convertie un valeur numérique représentant un réel qui est value en une chaine de caractère.

30. static void free_price_sector();

Cette fonction libère la mémoire qui a été alloué pour le tableau price vector.

31. static void free_price_circulation_category();

Cette fonction libère la mémoire qui a été alloué pour le tableau circulation category.

32. static void free_basic_price();

Cette fonction libère la mémoire qui a été alloué pour le tableau basic price.

33. static void free_benefice_niani();

Cette fonction libère la mémoire qui a été alloué pour le tableau benefice niani.

34. static void free_correspondance( );

Cette fonction libère la mémoire qui a été alloué pour le tableau correspondance.

35. static void free_vehicule_circulation();

Cette fonction libère la mémoire qui a été alloué pour le tableau vehicule circulation.

36. static void free_circulation();

Cette fonction libère la mémoire qui a été alloué pour le tableau circulation.

37. static double getprice_sector(int sector,int type_vehicle);

Cette fonction retourne le prix correspondant à un secteur donné et à un type de véhicule.

38. static double getprice_circulation_category(int circulation_category);

Cette fonction retourne le prix correspondant à une de la circulation.

39. static double getbasic_price(int type_vehicle);

Cette fonction retourne le prix correspondant au basic price d'un type vehicule.

40. static double getbenefices_niani(int distance_metter);

Cette fonction retourne les bénéfices de niani au mettre en fonction de la distance parcourue.

41. static double price(int distance_metter,int sector,int type_vehicle,int circulation_category=0);

Cette fonction retourne le prix d'un trajet.

## C. Politiques d'établissement des prix (partie technique)

Le prix d'un trajet sera déterminé en fonction de 4 paramètres : la distance à parcourir, le secteur ou s'effectue la course, le type de véhicule utilisé et la catégorie de la circulation (circulation dense, utilisation de l'autoroute à péage, période de fête… ).

C'est 4 paramètres seront déterminés en fonction des données csv qui sont stockés dans le fichier price. Les données rechargées tous les 10mn : tous les 10mn, les fichiers seront ouverts et les données contenues dans ces fichiers seront utilisées pour mettre à jour les prix appliqués. Cette manière de faire permettra aux ingénieurs de prix de modifier les prix appliqués à tout moment sans pour autant arrêter le système ou recompiler les données. L'inconvenant de cette méthode est que toute modification faites prendra une durée comprise entre 0 et 10mn pour être prise en compte : les modifications ne seront pas instantanées mais toute fois, on peut modifier la durée de mise à jour des prix.