```
/*
-------------------------------------------------------------------------------
Nom du fichier : main.c
Nom du labo    : Laboratoire no. 2
Auteur(s)      : Eric Peronetti, Grégory Rey-Mermet, Célestin Piccin
Date creation  : 24.05.2022
Description    : Ce programme permet d'afficher les caractéristiques de différents
                 véhicules, de calculer la taxe annuelle de chaque véhicule et de
                 calculer différentes statistiques (somme, moyenne, médianne,
                 écart-type) par sous-catégorie de type de véhicules.
Remarque(s)    : L'affichage des véhicules est fait de façon décroissante en fonction
                 de la taxe annuelle
Compilateur    : Mingw-w64 g++ 11.2.0
-------------------------------------------------------------------------------
*/

#include <stdio.h>    //printf
#include <stdlib.h>   //EXIT_SUCCESS
#include <math.h>     //pow
#include <inttypes.h> //uint

#include "taxeParking.h"
#include "vehicule.h"

typedef struct {
   Vehicule vehicule;   //Le véhicule
   double taxeAnnuelle; //La taxe annuelle
} VehiculeParking;

typedef struct {
   double somme;     //La sommme
   double moyenne;   //La moyenne
   double mediane;   //La médiane
   double ecartType; //L'écart-type
} Statistiques;

/**
 * Vide le buffer
 */
void viderBuffer(void);

/**
 * Tri décroissant en fonction de la taxe annuelle
 *
 * @param parking       Parking à trier
 * @param tailleParking Taille du parking
 */
void triDecroissantParkingTaxe(const VehiculeParking* parking, size_t tailleParking);

/**
 * Compare les taxes
 *
 * @param v1    VehiculeParking 1
 * @param v2    VehiculeParking 2
 * @return      Vrai : v1 < v2 // Faux : v1 >= v2
 */
int comparaisonTaxe(const void* v1, const void* v2);

/**
 * Compare deux double
 *
 * @param d1    Double 1
 * @param d2    Double 2
 * @return      Vrai : d1 < d2 // Faux : d1 >= d2
 */
int comparaisonDouble(const void* d1, const void* d2);

/**
 * Affiche les caractéristiques et taxe annuelle due de chacun des véhicules d'un
 * parking
 *
 * @param parking       Parking à afficher
 * @param tailleParking Taille du parking
 */
void afficherParking(const VehiculeParking* parking, size_t tailleParking);

/**
```

```
 * Affichage des statistiques (somme, moyenne, médiane, écart-type)
 *
 * @param stats         Structure contenant les différentes statistiques
 * @param vehicule      Type de véhicule
 * @param sousCategorie Sous-catégorie à considérer (ex. STANDARD, HAUT_GAMME, ...)
 */
void afficherStatistiques(const Statistiques* stats,
                          TVehicule vehicule,
                          uint16_t sousCategorie);

/**
 * Calcule les différentes statistiques pour un certain type de véhicule et une
 * certaine sous-catégorie sur un parking
 *
 * @param parking       Parking
 * @param tailleParking Taille du parking
 * @param vehicule      Type de véhicule
 * @param sousCategorie Sous-catégorie
 * @return              Les statistiques pour le parking
 */
Statistiques obtenirStatistiques(const VehiculeParking* parking,
                                 size_t tailleParking,
                                 TVehicule vehicule,
                                 uint16_t sousCategorie);

int main(void) {
   const double TAXE_DEFAUT = 0; //Taxe par défaut

   //Création du parking
   VehiculeParking parking[] = {
      {camionette("FR 123451", "Ford", 3.3),                       TAXE_DEFAUT},
      {camionette("BL 267564", "Mercedes-Benz", 3.8),              TAXE_DEFAUT},
      {voitureStandard("BE 88823", "BMW", 1850, 2998, 159),        TAXE_DEFAUT},
      {voitureStandard("ZG 190002", "Dacia", 1321, 1200, 98),      TAXE_DEFAUT},
      {voitureStandard("GE 591356", "Smart", 1500, 1150, 162),     TAXE_DEFAUT},
      {voitureHautDeGamme("VD 119977", "Aston Martin", 1870, 230), TAXE_DEFAUT},
      {voitureHautDeGamme("ZH 874569", "Porsche", 2010, 678),      TAXE_DEFAUT}
   };

   size_t tailleParking = sizeof(parking) / sizeof(VehiculeParking);

   //Calcul de toutes les taxes annuelles (pour chaque véhicule)
   for (size_t i = 0; i < tailleParking; ++i) {
      parking[i].taxeAnnuelle = calculTaxeAnnuelle(&parking[i].vehicule);
   }

   //Affichage du parking par ordre décroissant des taxes annuelles
   triDecroissantParkingTaxe(parking, tailleParking);
   afficherParking(parking, tailleParking);

   //Calcul somme, moyenne, médiane, écart-type par type de véhicule
   //Utilisation de 0 avec une caminonette car le type de la camionette
   //n'est pas important et donc non utilisé
   Statistiques statsCamionette = obtenirStatistiques(parking,
                                                      tailleParking,
                                                      CAMIONETTE,
                                                      0);
   Statistiques statsVoitureStd = obtenirStatistiques(parking,
                                                      tailleParking,
                                                      VOITURE,
                                                      (uint16_t)STANDARD);
   Statistiques statsVoitureHG = obtenirStatistiques(parking,
                                                     tailleParking,
                                                     VOITURE,
                                                     (uint16_t)HAUT_GAMME);

   //Affichage des statistiques
   //Utilisation de 0 avec une caminonette car le type de la camionette
   //n'est pas important et donc non utilisé
   afficherStatistiques(&statsCamionette, CAMIONETTE, 0);
   printf("\n");
   afficherStatistiques(&statsVoitureStd, VOITURE, (uint16_t)STANDARD);
   printf("\n");
   afficherStatistiques(&statsVoitureHG, VOITURE, (uint16_t)HAUT_GAMME);

   //Fin du programme
   printf("%s", "\nPresser ENTER pour quitter...");
```

```
   viderBuffer();

   return EXIT_SUCCESS;
}

void viderBuffer(void) {
   int c;

   do {
      c = getchar();
   } while (c != '\n' && c != EOF);
}

void triDecroissantParkingTaxe(const VehiculeParking* parking, size_t tailleParking) {
   qsort((void*)parking, tailleParking, sizeof(VehiculeParking), comparaisonTaxe);
}

int comparaisonTaxe(const void* v1, const void* v2) {
   return ((VehiculeParking*)v1)->taxeAnnuelle < ((VehiculeParking*)v2)->taxeAnnuelle;
}

int comparaisonDouble(const void* d1, const void* d2) {
   return *((double*)d1) < *((double*)d2);
}

void afficherParking(const VehiculeParking* parking, size_t tailleParking) {
   for (size_t i = 0; i < tailleParking; ++i) {
      printf("-------------------------------------------------\n");
      afficherVehicule(&parking[i].vehicule);
      printf("\n");
      printf("Taxe annuelle : %.2lf CHF\n", parking[i].taxeAnnuelle);
      printf("-------------------------------------------------\n");
      printf("\n");
   }
}

void afficherStatistiques(const Statistiques* stats,
                          TVehicule vehicule,
                          uint16_t sousCategorie) {
   printf("%s", obtenirNomTVehicule(vehicule));
   switch (vehicule) {
      case VOITURE:
         //On fait ce switch si l'utilisateur entre une sous-catégorie qui n'existe
         //pas. Dans ce cas on ne va pas afficher la sous-catégorie de la voiture.
         switch ((TVoiture)sousCategorie) {
            case STANDARD:
            case HAUT_GAMME:
               printf(" %s", obtenirNomTVoiture((TVoiture)sousCategorie));
               break;
         }
         break;
      case CAMIONETTE:
      default:
         break;
   }

   printf("\n");

   printf("Somme      : %.2lf\n", stats->somme);
   printf("Moyenne    : %.2lf\n", stats->moyenne);
   printf("Mediane    : %.2lf\n", stats->mediane);
   printf("Ecart-type : %.2lf\n", stats->ecartType);
}

Statistiques obtenirStatistiques(const VehiculeParking* parking,
                                 size_t tailleParking,
                                 TVehicule vehicule,
                                 uint16_t sousCategorie) {
   double* tab;
   size_t tailleTab = 0;
   Statistiques stats = {0, 0, 0, 0};

   //Allocation dynamique
   tab = (double*) calloc(tailleParking, sizeof(double));

   if (tab) {
      //Cree un tableau avec le type de véhicule désiré et fait le calcul de la somme
```

```
      //de ses éléments
      for (size_t i = 0; i < tailleParking; ++i) {
         Vehicule v = (&parking[i])->vehicule;
         if ((vehicule == CAMIONETTE && v.tVehicule == CAMIONETTE) ||
             (vehicule == VOITURE &&
              v.typeVehicule.voiture.tVoiture == (TVoiture)sousCategorie)) {
            tab[tailleTab++] = parking[i].taxeAnnuelle;
            stats.somme += tab[tailleTab - 1];
         }
      }

      tab = (double*)realloc(tab, tailleTab * sizeof(double));
      if (tab) {
         //Tri le tableau de façon décroissante car le tableau peut ne pas être
         //encore trié
         qsort((void*)tab, tailleTab, sizeof(double), comparaisonDouble);

         //Calcul de la moyenne
         stats.moyenne = stats.somme / (double)tailleTab;

         size_t indexMilieu = (tailleTab - 1) / 2;
         //Calcul de la médianne
         if (tailleTab % 2) {
            stats.mediane = tab[indexMilieu];
         } else {
            stats.mediane = (tab[indexMilieu] + tab[indexMilieu + 1]) / 2;
         }

         //Calcul de l'écart-type
         double ecartMoyenne = 0;
         for (size_t i = 0; i < tailleTab; ++i) {
            ecartMoyenne += pow(tab[i] - stats.moyenne, 2);
         }
         stats.ecartType = sqrt(ecartMoyenne / (double)tailleTab);
      }

      //Libération de la mémoire
      free(tab);
   }

   return stats;
}
```

```
/*
-----------------------------------------------------------------------------------
Nom du fichier : vehicule.h
Nom du labo    : Laboratoire no. 2
Auteur(s)      : Eric Peronetti, Grégory Rey-Mermet, Célestin Piccin
Date creation  : 24.05.2022
Description    : Fichier contenant les structures et les déclarations nécessaires
                 pour la voiture
Remarque(s)    : -
Compilateur    : Mingw-w64 g++ 11.2.0
-----------------------------------------------------------------------------------
*/

#ifndef PRG_LABO_2_VEHICULE_H
#define PRG_LABO_2_VEHICULE_H

#include <stdint.h> //uint

//La taille maximum des différents pays semblait être 10
#define TAILLE_MAX_NUMERO_PLAQUE 10

//La plus grande marque trouvée étant "International Harvester"
#define TAILLE_MAX_MARQUE 25

typedef char NumeroPlaque[TAILLE_MAX_NUMERO_PLAQUE + 1];
typedef char Marque[TAILLE_MAX_MARQUE + 1];

typedef enum {
   CAMIONETTE,
   VOITURE
} TVehicule; //Les types de vehicules
typedef enum {
   STANDARD,
   HAUT_GAMME
} TVoiture; //Les types de voiture

//Camionette
typedef struct {
   double volTransport; //Volume de transport (m3)
} Camionette;

//Voiture
typedef struct {
   uint16_t cylindree; //Cylindrée (cm3)
   uint16_t rejetCO2;  //Rejet de CO2 (g/km)
} VoitureStd;

typedef struct {
   uint16_t puissance; //Puissance (CV)
} VoitureHG;

typedef union {
   VoitureStd voitureStd; //Voiture standard
   VoitureHG voitureHg;   //Voiture haut de gamme
} TypeVoiture;

typedef struct {
   TVoiture tVoiture;       //Type de voiture (ex. STANDARD, HAUT_GAMME, ...)
   uint16_t poids;          //Poids de la voiture (kg)
   TypeVoiture typeVoiture; //Caractéristiques de la voiture
} Voiture;

//Vehicule
typedef union {
   Camionette camionette; //Camionette
   Voiture voiture;       //Voiture
} TypeVehicule;

typedef struct {
   TVehicule tVehicule;       //Type de véhicule (ex. CAMIONETTE, VOITURE, ...)
   NumeroPlaque numeroPlaque; //Numéro de plaque
   Marque marque;             //Marque du véhicule
   TypeVehicule typeVehicule; //Caractéristiques du véhicule
} Vehicule;

/**
 * Affiche un vehicule (toutes ses caractéristiques) dans la console
```

```
 * @param vehicule Le vehicule à afficher
 */
void afficherVehicule(const Vehicule* vehicule);

/**
 * Initialisation d'un objet véhicule de type voiture standard
 *
 * @param numeroPlaque  Numéro de plaque
 * @param marque        Marque de la voiture
 * @param poids         Poids de la voiture
 * @param cylindree     Cylindrée de la voiture
 * @param rejetCO2      Quantité de rejet de CO2
 * @return              La voiture
 */
Vehicule voitureStandard(const char* numeroPlaque,
                         const char* marque,
                         uint16_t poids,
                         uint16_t cylindree,
                         uint16_t rejetCO2);

/**
 * Initialisation d'un objet véhicule de type voiture haut de gamme
 *
 * @param numeroPlaque  Numéro de plaque
 * @param marque        Marque de la voiture
 * @param poids         Poids de la voiture
 * @param puissance     Puissance de la voiture
 * @return              La voiture
 */
Vehicule voitureHautDeGamme(const char* numeroPlaque,
                            const char* marque,
                            uint16_t poids,
                            uint16_t puissance);

/**
 * Initialisation d'un objet véhicule de type camionette
 *
 * @param numeroPlaque  Numéro de plaque
 * @param marque        Marque de la camionette
 * @param volTransport  Volume de transport de la camionette
 * @return              La camionette
 */
Vehicule camionette(const char* numeroPlaque, const char* marque, double volTransport);

/**
 * Retourne le type du véhicule (ex. Camionette, Voiture, ...)
 * @param tVehicule Le type du véhicule
 * @return          Le type du véhicule en toutes lettres
 */
const char* obtenirNomTVehicule(TVehicule tVehicule);

/**
 * Retourne le type de la voiture (ex. Standard, Haut de gamme, ...)
 * @param tVoiture Le type de la voiture
 * @return         Le type de la voiture en toutes lettres
 */
const char* obtenirNomTVoiture(TVoiture tVoiture);

#endif //PRG_LABO_2_VEHICULE_H
```

```
/*
-----------------------------------------------------------------------------------
Nom du fichier : vehicule.c
Nom du labo    : Laboratoire no. 2
Auteur(s)      : Eric Peronetti, Grégory Rey-Mermet, Célestin Piccin
Date creation  : 24.05.2022
Description    : Fichier contenant les définitions nécessaires pour la voiture
Remarque(s)    : -
Compilateur    : Mingw-w64 g++ 11.2.0
-----------------------------------------------------------------------------------
*/

#include <stdio.h>    //printf, PRI
#include <inttypes.h> //uint
#include <string.h>   //strncpy

#include "vehicule.h"

#define FORMAT "%-*s : "

//Les noms des types de véhicule en toutes lettres
const char* const TVEHICULE[] = {"Camionette", "Voiture"};
//Les noms des types de voiture en toutes lettres
const char* const TVOITURE[] = {"Standard", "Haut de gamme"};

//Le nombre de caractères pour le format (le format demande un int et non un size_t)
const int NOMBRE_CARACTERE = 24;

void afficherVehicule(const Vehicule* vehicule) {
   printf(FORMAT "%s\n", NOMBRE_CARACTERE, "Type de vehicule", TVEHICULE[vehicule->tVehicule]);
   printf(FORMAT "%s\n", NOMBRE_CARACTERE, "Plaque", vehicule->numeroPlaque);
   printf(FORMAT "%s\n", NOMBRE_CARACTERE, "Marque", vehicule->marque);

   switch (vehicule->tVehicule) {
      case CAMIONETTE:
         printf(FORMAT "%.2lf\n",
                NOMBRE_CARACTERE,
                "Volume de transport [m3]",
                vehicule->typeVehicule.camionette.volTransport);
         break;
      case VOITURE:
         printf(FORMAT "%" PRIu16 "\n",
                NOMBRE_CARACTERE,
                "Poids [kg]",
                vehicule->typeVehicule.voiture.poids);

         printf(FORMAT "%s\n",
                NOMBRE_CARACTERE,
                "Categorie de voiture",
                TVOITURE[vehicule->typeVehicule.voiture.tVoiture]);

         switch (vehicule->typeVehicule.voiture.tVoiture) {
            case STANDARD:
               printf(FORMAT "%" PRIu16 "\n",
                      NOMBRE_CARACTERE,
                      "Cylindree [cm3]",
                      vehicule->typeVehicule.voiture.typeVoiture.voitureStd.cylindree);

               printf(FORMAT "%" PRIu16 "\n",
                      NOMBRE_CARACTERE,
                      "Rejet CO2 [g/km]",
                      vehicule->typeVehicule.voiture.typeVoiture.voitureStd.rejetCO2);
               break;
            case HAUT_GAMME:
               printf(FORMAT "%" PRIu16 "\n",
                      NOMBRE_CARACTERE,
                      "Puissance [CV]",
                      vehicule->typeVehicule.voiture.typeVoiture.voitureHg.puissance);
               break;
         }
         break;
   }
}

Vehicule voitureStandard(const char* numeroPlaque,
                         const char* marque,
                         uint16_t poids,
```

```
                        uint16_t cylindree,
                        uint16_t rejetCO2) {
   //Ici il n'est pas nécessaire de mettre tous les noms des paramètres mais pour
   //une question de lisibilité nous avons décidé de tous les préciser
   Vehicule v = {
      .tVehicule=VOITURE,
      .numeroPlaque="",
      .marque="",
      .typeVehicule={
         .voiture={
            .tVoiture=STANDARD,
            .poids=poids,
            .typeVoiture={
               .voitureStd={
                  .cylindree=cylindree,
                  .rejetCO2=rejetCO2
               }
            }
         }
      }
   };

   //Recopie du numéro de plaque et de la marque dans le vehicule
   strncpy(v.numeroPlaque, numeroPlaque, TAILLE_MAX_NUMERO_PLAQUE);
   strncpy(v.marque, marque, TAILLE_MAX_MARQUE);

   return v;
}

Vehicule voitureHautDeGamme(const char* numeroPlaque,
                            const char* marque,
                            uint16_t poids,
                            uint16_t puissance) {
   //Ici il n'est pas nécessaire de mettre tous les noms des paramètres mais pour
   //une question de lisibilité nous avons décidé de tous les préciser
   Vehicule v = {
      .tVehicule=VOITURE,
      .numeroPlaque="",
      .marque="",
      .typeVehicule={
         .voiture={
            .tVoiture=HAUT_GAMME,
            .poids=poids,
            .typeVoiture={
               .voitureHg={
                  .puissance = puissance
               }
            }
         }
      }
   };

   //Recopie du numéro de plaque et de la marque dans le vehicule
   strncpy(v.numeroPlaque, numeroPlaque, TAILLE_MAX_NUMERO_PLAQUE);
   strncpy(v.marque, marque, TAILLE_MAX_MARQUE);

   return v;
}

Vehicule camionette(const char* numeroPlaque, const char* marque, double volTransport) {
   //Ici il n'est pas nécessaire de mettre tous les noms des paramètres mais pour
   //une question de lisibilité nous avons décidé de tous les préciser
   Vehicule v = {
      .tVehicule=CAMIONETTE,
      .numeroPlaque="",
      .marque="",
      .typeVehicule={
         .camionette={
            .volTransport=volTransport
         }
      }
   };

   //Recopie du numéro de plaque et de la marque dans le vehicule
   strncpy(v.numeroPlaque, numeroPlaque, TAILLE_MAX_NUMERO_PLAQUE);
   strncpy(v.marque, marque, TAILLE_MAX_MARQUE);
```

```
   return v;
}

const char* obtenirNomTVehicule(TVehicule tVehicule) {
   return TVEHICULE[tVehicule];
}

const char* obtenirNomTVoiture(TVoiture tVoiture) {
   return TVOITURE[tVoiture];
}
```

```
/*
-----------------------------------------------------------------------------------
Nom du fichier : taxeParking.h
Nom du labo    : Laboratoire no. 2
Auteur(s)      : Eric Peronetti, Grégory Rey-Mermet, Célestin Piccin
Date creation  : 24.05.2022
Description    : Fichier contenant les structures et les déclarations nécessaires
                 pour les taxes du parking
Remarque(s)    : -
Compilateur    : Mingw-w64 g++ 11.2.0
-----------------------------------------------------------------------------------
*/

#ifndef PRG_LABO_2_TAXEPARKING_H
#define PRG_LABO_2_TAXEPARKING_H

#include "vehicule.h"

/**
 * Calcul de la taxe annuelle d'un véhicule
 *
 * @param vehicule  Le véhicule
 * @return          La taxe annuelle du véhicule
 */
double calculTaxeAnnuelle(const Vehicule* vehicule);

/**
 * Taxe de base d'un véhicule
 *
 * @param vehicule  Le véhicule
 * @return          La taxe de base du véhicule
 */
double calculTaxeBase(const Vehicule* vehicule);

/**
 * Taxe de spécifique d'un véhicule
 *
 * @param vehicule  Le véhicule
 * @return          La taxe spécifique du véhicule
 */
double calculTaxeSpecifique(const Vehicule* vehicule);

/**
 * Calcul de la taxe spécifique pour une camionette
 *
 * @param camionette La camionette
 * @return           La taxe spécifique
 */
double calculTaxeCamionette(const Camionette* camionette);

/**
 * Calcul de la taxe spécifique pour une voiture standard
 *
 * @param voitureStd La voiture standard
 * @return           La taxe spécifique
 */
double calculTaxeVoitureStd(const Voiture* voitureStd);

/**
 * Calcul de la taxe spécifique pour une voiture haut de gamme
 *
 * @param voitureHg La voiture haut de gamme
 * @return          La taxe spécifique
 */
double calculTaxeVoitureHg(const Voiture* voitureHg);

#endif //PRG_LABO_2_TAXEPARKING_H
```

```
/*
-----------------------------------------------------------------------------------
Nom du fichier : taxeParking.c
Nom du labo    : Laboratoire no. 2
Auteur(s)      : Eric Peronetti, Grégory Rey-Mermet, Célestin Piccin
Date creation  : 24.05.2022
Description    : Fichier contenant les définitions nécessaires pour les taxes du
                 parking
Remarque(s)    : Utilisation des "double" pour les constantes des prix car on veut
                 garder le même type de données pour chacune des variables étant
                 donné que l'une d'entre elle à la valeur 0.05.
Compilateur    : Mingw-w64 g++ 11.2.0
-----------------------------------------------------------------------------------
*/

#include <stdint.h>  //uint

#include "taxeParking.h"

//Taxe de base
const double BASE_PRIX_CAMIONNETTE = 700;
const double BASE_PRIX_VOITURE     = 400;

//Taxe spécifique
//Taxe camionette
const double SPEC_PRIX_CAMIONNETTE = 10;

//Taxe voiture standard
const uint16_t SPEC_STD_SEUIL_CYLINDREE                   = 1400;
const uint16_t SPEC_STD_SEUIL_REJET_CO2                   = 130;
const double   SPEC_STD_PRIX_PETITE_CYLINDREE_PETIT_REJET = 0;
const double   SPEC_STD_PRIX_PETITE_CYLINDREE_GROS_REJET  = 50;
const double   SPEC_STD_PRIX_GROSSE_CYLINDREE             = 0.05;

//Taxe voiture haut de gamme
const uint16_t SPEC_HG_SEUIL_PUISSANCE   = 250;
const double   SPEC_HG_PRIX_PETIT_MOTEUR = 200;
const double   SPEC_HG_PRIX_GROS_MOTEUR  = 300;
const double   SPEC_HG_PRIX_POIDS        = 20;

double calculTaxeAnnuelle(const Vehicule* vehicule) {
   return calculTaxeBase(vehicule) + calculTaxeSpecifique(vehicule);
}

double calculTaxeBase(const Vehicule* vehicule) {
   switch (vehicule->tVehicule) {
      case CAMIONETTE:
         return BASE_PRIX_CAMIONNETTE;
      case VOITURE:
         return BASE_PRIX_VOITURE;
   }
}

double calculTaxeSpecifique(const Vehicule* vehicule) {
   switch (vehicule->tVehicule) {
      case CAMIONETTE:
         return calculTaxeCamionette(&vehicule->typeVehicule.camionette);
      case VOITURE:
         switch (vehicule->typeVehicule.voiture.tVoiture) {
            case STANDARD:
               return calculTaxeVoitureStd(&vehicule->typeVehicule.voiture);
            case HAUT_GAMME:
               return calculTaxeVoitureHg(&vehicule->typeVehicule.voiture);
         }
         break;
   }
}

double calculTaxeCamionette(const Camionette* camionette) {
   return SPEC_PRIX_CAMIONNETTE * camionette->volTransport;
}

double calculTaxeVoitureStd(const Voiture* voitureStd) {
   uint16_t cylindree = voitureStd->typeVoiture.voitureStd.cylindree;
   uint16_t rejetCO2  = voitureStd->typeVoiture.voitureStd.rejetCO2;

   if (cylindree < SPEC_STD_SEUIL_CYLINDREE) {
```

```
      if (rejetCO2 < SPEC_STD_SEUIL_REJET_CO2)
         return SPEC_STD_PRIX_PETITE_CYLINDREE_PETIT_REJET;
      return SPEC_STD_PRIX_PETITE_CYLINDREE_GROS_REJET;
   }
   return SPEC_STD_PRIX_GROSSE_CYLINDREE * cylindree;
}

double calculTaxeVoitureHg(const Voiture* voitureHg) {
   uint16_t puissance = voitureHg->typeVoiture.voitureHg.puissance;
   if (puissance <= SPEC_HG_SEUIL_PUISSANCE)
      return SPEC_HG_PRIX_PETIT_MOTEUR;

   //Taxe en fonction du poids en tonnes
   return SPEC_HG_PRIX_GROS_MOTEUR + SPEC_HG_PRIX_POIDS * voitureHg->poids / 1000.0;
}
```