```
/*
-------------------------------------------------------------------------------
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;
}
```