

/\*

-----

Nom du fichier : main.c  
Auteur(s) : Émilie Bressoud, Olin Bourquin, Timothée Van Hove  
Date création : 23.05.22  
Description : programme principal qui crée le parking contenant des voitures de luxe, voitures standards et des camionnettes.  
Les véhicules sont triés par ordre décroissant de taxe.  
Premier affichage dans la console des caractéristiques et de la taxe annuelle chaque véhicule. Second affichage dans la console des statistiques du parking, (pour chaque type de véhicule: la somme, la moyenne, la médiane et l'écart-type des taxes annuelles dues).  
  
Remarque(s) : le parking contient intentionnellement un nombre de véhicules de même type différent (pair et impair), ce qui permet de tester les deux calculs de la fonction médiane.  
Le programme se termine automatiquement après l'affichage des statistiques  
  
Compilateurs : Apple clang 13.0.0 et MinGW-W64 11.2.0

-----

```
*/
#include "vehicule.h"
#include "parking.h"
#include <stdlib.h> //EXIT_SUCCESS

//taille du parking défini ici pour pouvoir connaître la taille du tableau lors de sa compilation
#define TAILLE_MAX_TABLEAU 9

int main(void) {

    //création du parking
    Vehicule parking[TAILLE_MAX_TABLEAU] = {
        voitureStandard("Peugeot", "VD 123456", 1200, 1720, 155),
        voitureHautDeGamme("Rolls-Royce", "ZH 12345", 389, 2150),
        camionnette("Renault", "VS 1234", 15.99),
        voitureHautDeGamme("Audi", "GE 1234", 322, 1750),
        voitureStandard("Citroen", "GR 1234", 1099, 1199, 120),
        voitureHautDeGamme("BMW", "UR 1234", 285, 1885),
        camionnette("Opel", "FR 1234", 12.5),
        voitureStandard("Fiat", "GL 1234", 1130, 1589, 130),
        voitureStandard("Seat", "JU 1234", 1220, 1679, 145),
    };

    trierParkingParTaxeDecroissante(parking, TAILLE_MAX_TABLEAU);
    afficherParking(parking, TAILLE_MAX_TABLEAU);
    calculerStatistiques(parking, TAILLE_MAX_TABLEAU);

    return EXIT_SUCCESS;
}
```

/\*

-----

Nom du fichier : vehicule.h  
Auteur(s) : Émilie Bressoud, Olin Bourquin, Timothée Van Hove  
Date création : 23.05.22  
Description : fichier contenant la déclaration et la définition de la structure Vehicule, des fonctions de création des 3 types de véhicules différents (voiture standard, camionnette et voiture haut de gamme) donné par le struct Vehicule, des fonctions permettant de compter le nombre de véhicules de chaque type et fonctions de comparaisons entre types de véhicules.

Remarque(s) : utilisation d'Union entre une voiture et une camionnette et entre une voiture de luxe et une voiture standard. Cela permet d'économiser de la mémoire, sans pour autant rendre le code moins lisible.  
On suppose que les plaques d'immatriculation sont de la forme AA 000000 (Suisse).

Compilateurs : Apple clang 13.0.0 et MinGW-W64 11.2.0

\*/

```
#ifndef TAXES_VEHICULE_H
#define TAXES_VEHICULE_H

#include <stdint.h> //uint16_t, size_t

#define TAILLE_MAX_MARQUE 20

// 2 caractères pour le canton, 1 caractère pour l'espace et 6 caractères maximum pour le numéro
// d'immatriculation en Suisse
#define TAILLE_MAX_IMMATRICULATION 9

typedef char Immatriculation[TAILLE_MAX_IMMATRICULATION + 1];
typedef char Marque[TAILLE_MAX_MARQUE + 1];

typedef enum {
    VOITURE, CAMIONNETTE
} TypeVehicule;

typedef enum {
    STANDARD, HAUT_DE_GAMME
} TypeVoiture;

typedef struct {
    uint16_t puissance; // en [CV]
} VoitureHautdeGamme;

typedef struct {
    uint16_t cylindree; // en [cm3]
    uint16_t co2; // en [g/km]
} VoitureStandard;

typedef union {
    VoitureStandard standard;
    VoitureHautdeGamme hautDeGamme;
} SpecificitesVoiture;

typedef struct {
    uint16_t poids; // en [kg]
    TypeVoiture typeVoiture;
    SpecificitesVoiture specificitesVoiture;
} Voiture;

typedef struct {
    double volume; // en [m3]
} Camionnette;

typedef union {
    Voiture voiture;
    Camionnette camionnette;
} SpecificitesVehicule;
```

```
typedef struct {
    Immatriculation immatriculation;
    Marque marque;
    TypeVehicule typeVehicule;
    SpecificitesVehicule specificitesVehicule;
} Vehicule;

/*****
// Fonctions de création de véhicules
*****/

Vehicule voitureHautDeGamme(const char* marque, const char* immatriculation, uint16_t puissance,
                           uint16_t poids);

Vehicule voitureStandard(const char* marque, const char* immatriculation, uint16_t poids,
                         uint16_t cylindree, uint16_t co2);

Vehicule camionnette(const char* marque, const char* immatriculation, double volume);

/*****
// Fonctions de comparaison et de comptage
*****/

// Fonction permettant de construire l'immatriculation et la marque dans une structure Vehicule
Vehicule attributsVehicule(Vehicule* v, const char* immatriculation, const char* marque);

// Fonction de comparaison du type de vehicule entre 2 véhicules utilisés dans qsort
int compTypeVehicules(const void* v1, const void* v2);

// Fonction de comparaison du type de voitures entre 2 véhicules utilisés dans qsort
int compTypeVoitures(const void* v1, const void* v2);

// Compte le nombre d'éléments d'un sous-type de Vehicule dans un tableau
size_t compterTypeVehicule(Vehicule* v, size_t nbVehicules, TypeVehicule type);

// Compte le nombre d'éléments d'un sous type de Voiture dans un tableau
size_t compterTypeVoiture(Vehicule* v, size_t nbVehicules, TypeVoiture type);

#endif
```

/\*

```

-----
Nom du fichier      : vehicule.c
Auteur(s)          : Émilie Bressoud, Olin Bourquin, Timothée Van Hove
Date création      : 23.05.22
Description         : Implémentation des fonctions de vehicule.h

Remarque(s)        : utilisation de strncpy, ce qui permet de copier la chaine de caractères sans dépasser
                     : la taille.
                     : les enum sont casté en int pour pouvoir faire des comparaisons.
Compilateurs       : Apple clang 13.0.0 et MinGW-W64 11.2.0
-----

```

\*/

```

#include "vehicule.h"
#include <string.h> //strncpy()

```

```

Vehicule attributsVehicule(Vehicule* v, const char* immatriculation, const char* marque) {
    strncpy(v->immatriculation, immatriculation, TAILLE_MAX_IMMATRICULATION);
    strncpy(v->marque, marque, TAILLE_MAX_MARQUE);
    return *v;
}

```

```

Vehicule voitureHautDeGamme(const char* marque, const char* immatriculation, uint16_t puissance,
                             uint16_t poids) {

    VoitureHautdeGamme vhg = {puissance};
    Voiture v = {poids, HAUT_DE_GAMME, {.hautDeGamme = vhg}};
    Vehicule voiture = {"", "", VOITURE, {.voiture = v}};

    return attributsVehicule(&voiture, immatriculation, marque);
}

```

```

Vehicule voitureStandard(const char* marque, const char* immatriculation, uint16_t poids,
                           uint16_t cylindree, uint16_t co2) {

    VoitureStandard vs = {cylindree, co2};
    Voiture v = {poids, STANDARD, {.standard = vs}};
    Vehicule voiture = {"", "", VOITURE, {.voiture = v}};

    return attributsVehicule(&voiture, immatriculation, marque);
}

```

```

Vehicule camionnette(const char* marque, const char* immatriculation, double volume) {
    Camionnette c = {volume};
    Vehicule camionnette = {"", "", CAMIONNETTE, {.camionnette = c}};

    return attributsVehicule(&camionnette, immatriculation, marque);
}

```

```

int compTypeVehicules(const void* v1, const void* v2) {
    return (int) ((Vehicule*) v2)->typeVehicule - (int) ((Vehicule*) v1)->typeVehicule;
}

```

```

int compTypeVoitures(const void* v1, const void* v2) {
    return (int) ((Voiture*) v1)->typeVoiture - (int) ((Voiture*) v2)->typeVoiture;
}

```

```

size_t compterTypeVehicule(Vehicule* v, size_t nbVehicules, TypeVehicule type) {
    size_t nb = 0;
    for (size_t i = 0; i < nbVehicules; i++) {
        if (v[i].typeVehicule == type) {
            ++nb;
        }
    }
    return nb;
}

```

```

size_t compterTypeVoiture(Vehicule* v, size_t nbVehicules, TypeVoiture type) {
    size_t nb = 0;

```

```
for (size_t i = 0; i < nbVehicules; i++) {
    if (v[i].typeVehicule == VOITURE && v[i].specificitesVehicule.voiture.typeVoiture == type) {
        ++nb;
    }
}
return nb;
}
```

/\*

```
-----
Nom du fichier      : taxe.h
Auteur(s)          : Émilie Bressoud, Olin Bourquin, Timothée Van Hove
Date création      : 23.05.22
Description         : fonction permettant de calculer les taxes, soit annuelle ou spécifique
Remarque(s)        :
Compilateurs       : Apple clang 13.0.0 et MinGW-W64 11.2.0
-----
```

\*/

```
#ifndef TAXES_TAXE_H
#define TAXES_TAXE_H

#include "vehicule.h"

// Calcule la taxe de base d'un vehicule
uint16_t taxeBase(const Vehicule* v);

// Calcule la taxe spécifique d'une voiture haut de gamme
double taxeSpecifiqueVoitureHG(const VoitureHautdeGamme* v, uint16_t poids);

// Calcule la taxe spécifique d'une voiture standard
double taxeSpecifiqueVoitureStd(const VoitureStandard* v);

// Calcule la taxe spécifique d'une voiture
double taxeSpecifiqueVoiture(const Voiture* v);

// Calcule la taxe spécifique d'une camionnette
double taxeSpecifiqueCamionnette(const Camionnette* c);

// Calcule la taxe spécifique d'un véhicule
double taxeSpecifique(const Vehicule* v);

// Calcule la taxe annuelle d'un véhicule
double calculerTaxe(const Vehicule* v);

// Calcule la taxe annuelle d'un tableau de véhicules. Retourne un tableau de double
// Note : Ne pas oublier de libérer la mémoire allouée depuis l'extérieur de la fonction.
double* taxesParVehicule(Vehicule* v, size_t n);

#endif //TAXES_TAXE_H
```

/\*

```

-----
Nom du fichier      : taxe.c
Auteur(s)          : Émilie Bressoud, Olin Bourquin, Timothée Van Hove
Date création      : 23.05.22
Description         : Implémentation des fonctions de taxe.h
Remarque(s)        :
Compilateurs       : Apple clang 13.0.0 et MinGW-W64 11.2.0
-----

```

\*/

```

#include "taxe.h"
#include <math.h> //sqrt(), pow()
#include <stdlib.h> // calloc, size_t
#include <stdio.h> // printf

const uint16_t TAXE_BASE_CAMIONNETTE      = 700,
                TAXE_BASE_VOITURE          = 400,
                FACTEUR_TAXE_VOLUME        = 10,
                TAXE_SPEC_VOITURE_STD_PETITE = 0,
                TAXE_SPEC_VOITURE_STD_MOYENNE = 50,
                LIMITE_CYLINDREE_VOITURE_STD = 1400,
                LIMITE_CO2_VOITURE_STD      = 130,
                LIMITE_PUISSANCE_VOITURE_HG = 250,
                TAXE_SPEC_VOITURE_HG_PETITE = 200,
                TAXE_SPEC_VOITURE_HG_GROSSE = 300;

const double TAXE_SPEC_VOITURE_STD_GROSSE = 0.05,
              FACTEUR_TAXE_SPEC_VOITURE_HG_GROSSE = 0.02;

uint16_t taxeBase(const Vehicule* v) {
    if (v->typeVehicule == VOITURE) {
        return TAXE_BASE_VOITURE;
    }
    return TAXE_BASE_CAMIONNETTE;
}

double taxeSpecifiqueVoitureHG(const VoitureHautdeGamme* v, uint16_t poids) {
    if (v->puissance <= LIMITE_PUISSANCE_VOITURE_HG) {
        return (double) TAXE_SPEC_VOITURE_HG_PETITE;
    }
    return (double) (FACTEUR_TAXE_SPEC_VOITURE_HG_GROSSE * poids +
                     TAXE_SPEC_VOITURE_HG_GROSSE);
}

double taxeSpecifiqueVoitureStd(const VoitureStandard* v) {
    if (v->cylindree < LIMITE_CYLINDREE_VOITURE_STD) {
        if (v->co2 < LIMITE_CO2_VOITURE_STD) {
            return (double) TAXE_SPEC_VOITURE_STD_PETITE;
        }
        return (double) TAXE_SPEC_VOITURE_STD_MOYENNE;
    }
    return TAXE_SPEC_VOITURE_STD_GROSSE * v->cylindree;
}

double taxeSpecifiqueVoiture(const Voiture* v) {
    if (v->typeVoiture == STANDARD) {
        return taxeSpecifiqueVoitureStd(&v->specificitesVoiture.standard);
    }
    return taxeSpecifiqueVoitureHG(&v->specificitesVoiture.hautDeGamme, v->poids);
}

double taxeSpecifiqueCamionnette(const Camionnette* c) {
    return c->volume * FACTEUR_TAXE_VOLUME;
}

double taxeSpecifique(const Vehicule* v) {
    if (v->typeVehicule == VOITURE) {
        return taxeSpecifiqueVoiture(&v->specificitesVehicule.voiture);
    }
}

```

```
    return taxeSpecifiqueCamionnette(&v->specificitesVehicule.camionnette);
}

double calculerTaxe(const Vehicule* v) {
    return (double) taxeBase(v) + taxeSpecifique(v);
}

double* taxesParVehicule(Vehicule* v, size_t n) {
    // Allocation du tableau de taxes
    double* taxes = (double*) calloc(n, sizeof(double));

    // Le programme se termine si plus de mémoire disponible
    if (taxes == NULL) {
        printf("Plus de memoire disponible\n Arret du programme");
        exit(EXIT_FAILURE);
    }

    for (size_t i = 0; i < n; ++i) {
        taxes[i] = calculerTaxe(v + i);
    }
    return taxes;
}
```



/\*

```
-----
Nom du fichier      : parking.h
Auteur(s)          : Émilie Bressoud, Olin Bourquin, Timothée Van Hove
Date création      : 23.05.22
Description         : contient les fonctions d'affichage de l'ensemble des caractéristiques d'un véhicule
                    : et des différentes taxes (la somme, la moyenne, la médiane et l'écart-type)
Remarque(s)        :
Compilateurs       : Apple clang 13.0.0 et MinGW-W64 11.2.0
-----
```

```
*/
#ifndef TAXES_PARKING_H
#define TAXES_PARKING_H

#include <stdio.h>                //size_t
#include "vehicule.h"

// Trie un tableau de vehicules par ordre CAMIONNETTE puis VOITURE
void trierParkingParTypeVehicule(Vehicule* parking, size_t nbVehicules);

//comparaison de taxes entre chaque véhicule
int compTaxesVehicules(const void* v1, const void* v2);

void calculerStatistiques(Vehicule* v, size_t nbVehicules);

void trierParkingParTaxeDecroissante(Vehicule* parking, size_t nbVehicules);

void afficherParking(const Vehicule* parking, size_t nbVehicules);

#endif //TAXES_PARKING_H
```

/\*

```
-----
Nom du fichier      : parking.c
Auteur(s)          : Émilie Bressoud, Olin Bourquin, Timothée Van Hove
Date création      : 23.05.22
Description         : Implémentation des fonctions de parking.h
Remarque(s)        :
Compilateurs       : Apple clang 13.0.0 et MinGW-W64 11.2.0
-----
```

\*/

```
#include "parking.h"
#include "statistiques.h"
#include "taxe.h"
#include "affichage.h"
#include <stdlib.h> //qsort()
```

```
int compTaxesVehicules(const void* v1, const void* v2) {
    double taxeV1 = calculerTaxe((Vehicule*) v1);
    double taxeV2 = calculerTaxe((Vehicule*) v2);
    if (taxeV1 < taxeV2)
        return 1;
    else if (taxeV1 == taxeV2)
        return 0;
    else return -1;
}
```

```
void trierParkingParTaxeDecroissante(Vehicule* parking, size_t nbVehicules) {
    qsort(parking, nbVehicules, sizeof(Vehicule), compTaxesVehicules);
}
```

```
void trierParkingParTypeVehicule(Vehicule* parking, size_t nbVehicules) {
    qsort(parking, nbVehicules, sizeof(Vehicule), compTypeVehicules);
}
```

```
void afficherParking(const Vehicule* parking, size_t nbVehicules) {
    for (size_t i = 0; i < nbVehicules; ++i) {
        printf("Vehicule no %zd \n", i + 1);
        afficherVehicule(parking + i);
        afficherTaxeVehicule(parking + i);
        printf("\n");
    }
}
```

```
void calculerStatistiques(Vehicule* v, size_t nbVehicules) {
```

```
    //Trier pour mettre les camionnettes au début
    trierParkingParTypeVehicule(v, nbVehicules);
```

```
    //Compter le nombre de voitures et camionnettes
    size_t nbCamionnettes = compterTypeVehicule(v, nbVehicules, CAMIONNETTE);
    size_t nbVoitures = compterTypeVehicule(v, nbVehicules, VOITURE);
```

```
    //Compter ne nombre de voitures haut de gamme et voitures standard
    size_t nbVoituresStandard = compterTypeVoiture(v + nbCamionnettes, nbVoitures, STANDARD);
    size_t nbVoituresHautDeGamme = compterTypeVoiture(v + nbCamionnettes, nbVoitures, HAUT_DE_GAMME);
```

```
    double* taxesCamionnettes = taxesParVehicule(v, nbCamionnettes);
    double* taxesVoituresStandard = taxesParVehicule(v + nbCamionnettes, nbVoituresStandard);
    double* taxesVoituresHautDeGamme = taxesParVehicule(v + nbCamionnettes, nbVoituresHautDeGamme);
```

```
    printf("Camionnettes\n");
    afficherStatistiques(taxesCamionnettes, nbCamionnettes);
    printf("\n");
```

```
    printf("Voitures standard\n");
    afficherStatistiques(taxesVoituresStandard, nbVoituresStandard);
    printf("\n");
```

```
    printf("Voitures haut de gamme\n");
```

```
    afficherStatistiques(taxesVoituresHautDeGamme, nbVoituresHautDeGamme);  
    printf("\n");  
  
    //Libérer la mémoire allouée dans taxesParVehicule()  
    free(taxesCamionettes);  
    free(taxesVoituresStandard);  
    free(taxesVoituresHautDeGamme);  
}
```

```
/*
-----
Nom du fichier      : affichage.h
Auteur(s)           : Émilie Bressoud, Olin Bourquin, Timothée Van Hove
Date création       : 23.05.22
Description          : fichier contenant les fonctions permettant d'afficher les véhicules, les
                      caractéristiques des véhicules et les statistiques.
Remarque(s)         :
Compilateurs        : Apple clang 13.0.0 et MinGW-W64 11.2.0
-----
*/

#ifndef TAXES_AFFICHAGE_H
#define TAXES_AFFICHAGE_H

#include "vehicule.h"

// Permet d'aligner l'affichage des chiffres
void afficherAligner(const char* texte, int16_t espace);

void afficherTypeVehicule(const Vehicule* v);

void afficherSpecVoitureHg(const VoitureHautdeGamme* v);

void afficherSpecVoitureStd(const VoitureStandard* v);

void afficherSpecVoiture(const Voiture* v);

void afficherSpecCamionnette(const Camionnette* c);

void afficherVehicule(const Vehicule* v);

// Arrondi la valeur réelle à 5 centimes près
double arrondiCentimesPres(uint16_t centimes, double valeur);

void afficherStatistiques(double* taxes, size_t nbVehicules);

void afficherTaxeVehicule(const Vehicule* v);

#endif
```

/\*

```

-----
Nom du fichier      : affichage.c
Auteur(s)          : Émilie Bressoud, Olin Bourquin, Timothée Van Hove
Date création      : 23.05.22
Description         : Implémentation des fonctions de affichage.h
Remarque(s)        :
Compilateurs       : Apple clang 13.0.0 et MinGW-W64 11.2.0
-----

```

\*/

```

#include "affichage.h"
#include <inttypes.h> //PRIu16
#include <stdio.h>    //printf()
#include <math.h>     //round()
#include "taxe.h"
#include "statistiques.h"

const char* const UNITE_PUISSANCE = "[CV]";
const char* const UNITE_POIDS = "[kg]";
const char* const UNITE_VOLUME = "[m3]";
const char* const UNITE_POLLUTION = "[g/km]";
const char* const UNITE_CYLINDREE = "[cm3]";
const char* const DEVISE = "CHF";
const char* const TYPE_VEHICULE[] = {"Voiture", "Camionnette"};
const char* const TYPE_VOITURE[] = {"Standard", "Haut de gamme"};
const uint16_t CENTIMES = 5;
const int16_t ESPACE_AFFICHAGE = -24; //alignement gauche

void afficherAligner(const char* texte, int16_t espace) {
    printf("%*s : ", espace, texte);
}

void afficherTypeVehicule(const Vehicule* v) {
    afficherAligner("Type de vehicule", ESPACE_AFFICHAGE);
    printf("%s ", TYPE_VEHICULE[v->typeVehicule]);
    if (v->typeVehicule == CAMIONNETTE)
        printf("\n");
    else
        printf("%s\n", TYPE_VOITURE[v->specificitesVehicule.voiture.typeVoiture]);
}

void afficherSpecVoitureHg(const VoitureHautdeGamme* v) {
    afficherAligner("Puissance", ESPACE_AFFICHAGE);
    printf("%"PRIu16 " %s\n", v->puissance, UNITE_PUISSANCE);
}

void afficherSpecVoitureStd(const VoitureStandard* v) {
    afficherAligner("Cylindree", ESPACE_AFFICHAGE);
    printf("%"PRIu16 " %s\n", v->cylindree, UNITE_CYLINDREE);

    afficherAligner("CO2", ESPACE_AFFICHAGE);
    printf("%"PRIu16 " %s\n", v->co2, UNITE_POLLUTION);
}

void afficherSpecVoiture(const Voiture* v) {
    if (v->typeVoiture == STANDARD) {
        afficherSpecVoitureStd(&v->specificitesVoiture.standard);
    } else {
        afficherSpecVoitureHg(&v->specificitesVoiture.hautDeGamme);
    }
    afficherAligner("Poids", ESPACE_AFFICHAGE);
    printf("%"PRIu16 " %s\n", v->poids, UNITE_POIDS);
}

void afficherSpecCamionnette(const Camionnette* c) {
    afficherAligner("Volume", ESPACE_AFFICHAGE);
    printf("%.2f %s\n", c->volume, UNITE_VOLUME);
}

```

```
void afficherVehicule(const Vehicule* v) {
    afficherTypeVehicule(v);

    afficherAligner("Immatriculation", ESPACE_AFFICHAGE);
    printf("%s\n", v->immatriculation);

    afficherAligner("Marque", ESPACE_AFFICHAGE);
    printf("%s\n", v->marque);

    if (v->typeVehicule == VOITURE)
        afficherSpecVoiture(&v->specificitesVehicule.voiture);
    else
        afficherSpecCamionnette(&v->specificitesVehicule.camionnette);
}

double arrondiCentimesPres(uint16_t centimes, double valeur) {
    //arrondi au centime près (2 chiffres après la virgule)
    valeur = valeur * 100 / centimes;
    valeur = round(valeur);
    valeur = valeur * centimes / 100;
    return valeur;
}

void afficherStatistiques(double* taxes, size_t nbVehicules) {
    double SommeArrondie = arrondiCentimesPres(CENTIMES, somme(taxes, nbVehicules));
    afficherAligner("Somme", ESPACE_AFFICHAGE);
    printf("%.2f\n", SommeArrondie);

    double moyenneArrondie = arrondiCentimesPres(CENTIMES, moyenne(taxes, nbVehicules));
    afficherAligner("Moyenne", ESPACE_AFFICHAGE);
    printf("%.2f\n", moyenneArrondie);

    double medianeArrondie = arrondiCentimesPres(CENTIMES, mediane(taxes, nbVehicules));
    afficherAligner("Mediane", ESPACE_AFFICHAGE);
    printf("%.2f\n", medianeArrondie);

    double ecartTypeArrondi = arrondiCentimesPres(CENTIMES, ecartType(taxes, nbVehicules));
    afficherAligner("Ecart-type", ESPACE_AFFICHAGE);
    printf("%.2f\n", ecartTypeArrondi);
}

void afficherTaxeVehicule(const Vehicule* v) {
    double taxe = arrondiCentimesPres((int) CENTIMES, calculerTaxe(v));
    afficherAligner("Taxe du vehicule", ESPACE_AFFICHAGE);
    printf("%.2f %s\n", taxe, DEVISE);
}
```

/\*

```
-----
Nom du fichier      : statistiques.h
Auteur(s)           : Émilie Bressoud, Olin Bourquin, Timothée Van Hove
Date création       : 23.05.22
Description          : fonctions permettant de calculer les statistiques
Remarque(s)         : les fonctions n'ont pas besoin d'utiliser un objet de type vehicule, ce qui les
                      : permet d'être génériques
Compilateurs        : Apple clang 13.0.0 et MinGW-W64 11.2.0
-----
```

\*/

```
#ifndef TAXES_STATISTIQUES_H
#define TAXES_STATISTIQUES_H
```

```
#include <stdio.h> //size_t
```

```
/**
// Fonctions générales de calcul statistique à partir d'un tableau de double
**/
```

```
double mediane(const double valeurs[], size_t n);
```

```
double somme(const double valeurs[], size_t n);
```

```
double moyenne(const double valeurs[], size_t n);
```

```
double variance(const double valeurs[], size_t n);
```

```
double ecartType(const double valeurs[], size_t n);
```

```
#endif //TAXES_STATISTIQUES_H
```

/\*

```
-----
Nom du fichier      : statistiques.c
Auteur(s)          : Émilie Bressoud, Olin Bourquin, Timothée Van Hove
Date création      : 23.05.22
Description         : Implémentation des fonctions de statistiques.h
Remarque(s)        :
Compilateurs       : Apple clang 13.0.0 et MinGW-W64 11.2.0
-----
```

\*/

```
#include "statistiques.h"
#include <math.h>
```

```
double mediane(const double valeurs[], size_t n) {
    if (n % 2 == 0) {
        return (valeurs[n / 2] + valeurs[n / 2 - 1]) / 2;
    }
    return valeurs[n / 2];
}
```

```
double somme(const double valeurs[], size_t n) {
    double sommeTaxes = 0;
    for (size_t i = 0; i < n; ++i) {
        sommeTaxes += valeurs[i];
    }
    return sommeTaxes;
}
```

```
double moyenne(const double valeurs[], size_t n) {
    return somme(valeurs, n) / (double) n;
}
```

```
double variance(const double valeurs[], size_t n) {
    double variance = 0;
    double moy = moyenne(valeurs, n);
    for (size_t i = 0; i < n; ++i) {
        variance += pow(valeurs[i] - moy, 2.f);
    }
    variance /= (double) n;
    return variance;
}
```

```
double ecartType(const double valeurs[], size_t n) {
    return sqrt(variance(valeurs, n));
}
```