main.c

```
/*
 ------------------------------------------------------------------------------------
 Laboratoire : N°5 - Port, Bateaux et Taxes
 Fichier     : main.c
 Auteur(s)   : Alec Berney, Quentin Forestier, Victoria Logan
 Date        : 29.05.2020

 But         : Tester les différents éléments implémentés pour modéliser un Port
               contenant plusieurs Bateaux de différents types, selon lesquels
               sont dues des taxes annuelles.

 Remarque(s) : - Le but de ce laboratoire est de pratiquer l'allocation dynamique,
                 les structures, les unions et les types énumérés.

 Compilateur : MinGW-gcc 6.3.0
 ------------------------------------------------------------------------------------
 */

#include <stdio.h>
#include <stdlib.h>

#include "Bateau.h"
#include "Port.h"
#include "Outils.h"

#define NB_BATEAUX_PORT 15

int main()
{
   Bateau* voile1 = creeBateauAVoile("Voilier1", 199);
   Bateau* voile2 = creeBateauAVoile("Voilier2", 200);
   Bateau* voile3 = creeBateauAVoile("Voilier3", 210);

   Bateau* peche1 = creeBateauPeche("Peche1", 10, 19);
   Bateau* peche2 = creeBateauPeche("Peche2", 15, 20);
   Bateau* peche3 = creeBateauPeche("Peche3", 20, 25);
   Bateau* peche4 = creeBateauPeche("Peche4", 25, 30);

   Bateau* plaisance1 = creeBateauPlaisance("Plaisance1",  99, 32, "Proprietaire1");
   Bateau* plaisance2 = creeBateauPlaisance("Plaisance2", 100, 35, "Proprietaire2");
   Bateau* plaisance3 = creeBateauPlaisance("Plaisance3", 110, 20, "Proprietaire3");
   Bateau* plaisance4 = creeBateauPlaisance("Plaisance4", 120, 30, "Proprietaire4");
   Bateau* plaisance5 = creeBateauPlaisance("Plaisance5", 130, 40, "Proprietaire5");

   Port* p = construitPort(NB_BATEAUX_PORT);

   ajouteBateau(p,     voile1);
   ajouteBateau(p,     peche1);
   ajouteBateau(p,     voile2);
   ajouteBateau(p, plaisance1);
   ajouteBateau(p, plaisance2);
   ajouteBateau(p,     peche2);
   ajouteBateau(p, plaisance3);
   ajouteBateau(p,     peche4);
   ajouteBateau(p,     voile3);
   ajouteBateau(p, plaisance4);
   ajouteBateau(p,     peche3);
   ajouteBateau(p, plaisance5);

   printf("Affiche tous les bateaux du port : \n");
   printf("=================================\n");
   afficherBateauxPort(p);

   printf("Affiche tous les calculs en rapport avec les taxes : \n");
   printf("=====================================================\n");
   afficherTaxesParType(p);
```

```
    detruitPortEtBateaux(p);

    return EXIT_SUCCESS;
}
```

Bateau.h

```
/*
 ------------------------------------------------------------------------------
 Laboratoire : N°5 - Port, Bateaux et Taxes
 Fichier     : Bateau.h
 Auteur(s)   : Alec Berney, Quentin Forestier, Victoria Logan
 Date        : 29.05.2020

 But         : Définir différentes structures servant à modéliser trois types
               de Bateau (bateau de plaisance, bateau de pêche, voilier), et
               des fonctions utiles à la gestion de ces Bateau.

 Remarque(s) : -

 Compilateur : MinGW-gcc 6.3.0
 ------------------------------------------------------------------------------
 */

#ifndef LABO_MOI_BATEAU_H
#define LABO_MOI_BATEAU_H

#include <inttypes.h>
#include <stdbool.h>

// ------------------ PUBLIC ----------------------

typedef char* Nom;

// Utilite du bateau à moteur
typedef struct
{
   uint8_t longueurBateau; // en m
   Nom nomProprietaire;

} Plaisance;

typedef struct
{
   uint8_t quantiteAutoriseePoissons; // En tonne

} Peche;

typedef enum UTILITE_BATEAU {PECHE, PLAISANCE} UtiliteBateau;

typedef union
{
   Plaisance* plaisance;
   Peche*     peche;

} Utilite;


// Motorisation du bateau
typedef struct
{
   uint16_t surfaceVoilure; // En m^2

} Voile;

typedef struct
{
   uint16_t      puissanceMoteur; // En CV(cheveaux)
   UtiliteBateau utiliteBateau;
   Utilite       utilite;

} Moteur;
```

```
typedef enum TYPE_BATEAU {A_VOILE, MOTORISE} TypeBateau;

typedef union
{
   Moteur* moteur;
   Voile*  voile;

} Motorisation;


typedef struct
{
   Nom          nom;
   TypeBateau   typeBateau;
   Motorisation motorisation;

} Bateau;

// ------------------ Fonctions publics -----------------------

Bateau* creeBateauPeche(Nom nomBateau, uint16_t puissanceMoteur,
                        uint8_t quantiteAutoriseePoissons);

Bateau* creeBateauPlaisance(Nom nomBateau, uint16_t puissanceMoteur,
                            uint8_t longueurBateau, Nom nomProprietaire);

Bateau* creeBateauAVoile(Nom nomBateau, uint16_t surfaceVoilure);

void detruitBateau(Bateau* b);

bool estMotorise(const Bateau* b);

bool estAVoile(const Bateau* b);

bool estUtilePeche(const Bateau* b);

bool estUtilePlaisance(const Bateau* b);

// --------------  Getter / Setter ----------------

// Plaisance
void setLongueurBateau(Bateau* b, uint8_t nouvelleLongueur);

const uint8_t* getLongueurBateau(const Bateau* b);

void setNomProprietaire(Bateau* b, Nom nouveauNom);

const Nom* getNomProprietaire(const Bateau* b);


// Pêche
void setQuantiteAutoriseePoissons(Bateau* b, uint8_t nouvelleQuantiteEnTonne);

const uint8_t* getQuantiteAutoriseePoissons(const Bateau* b);


// Bateau à voile
void setSurfaceVoilure(Bateau* b, uint16_t nouvelleSurface);

const uint16_t* getSurfaceVoilure(const Bateau* b);


// Bateau à moteur
void setPuissanceMoteur(Bateau* b, uint16_t nouvellePuissanceEnCV);

const uint16_t* getPuissanceMoteur(const Bateau* b);

void setUtiliteBateauMoteur(Bateau* b, UtiliteBateau nouvelleUtiliteBateau);
```

```
const UtiliteBateau* getUtiliteBateau(const Bateau* b);

void setPeche(Bateau* b, Peche* p);

const Peche* getPeche(const Bateau* b);

void setPlaisance(Bateau* b, Plaisance* p);

const Plaisance* getPlaisance(const Bateau* b);


// Bateau
void setNom(Bateau* b, Nom nouveauNom);

const Nom* getNom(const Bateau* b);

void setTypeBateau(Bateau* b, TypeBateau nouveauType);

const TypeBateau* getTypeBateau(const Bateau* b);

void setMoteur(Bateau* b, Moteur* m);

const Moteur* getMoteur(const Bateau* b);

void setVoile(Bateau* b, Voile* v);

const Voile* getVoile(const Bateau* b);

#endif //LABO_MOI_BATEAU_H
```

Bateau.c

```
/*
 ------------------------------------------------------------------------------
 Laboratoire : N°5 - Port, Bateaux et Taxes
 Fichier     : Bateau.c
 Auteur(s)   : Alec Berney, Quentin Forestier, Victoria Logan
 Date        : 29.05.2020

 Compilateur : MinGW-gcc 6.3.0
 ------------------------------------------------------------------------------
 */
#include <stdlib.h>
#include "Bateau.h"

// ------------------ PUBLIC ----------------------

// Cree un bateau a peche en utilisant l'allocation dynamique
// ATTENTION : Nécessite une destruction via la fonction detruitBateau
Bateau* creeBateauPeche(Nom nomBateau, uint16_t puissanceMoteur,
                        uint8_t quantiteAutoriseePoissons)
{
   Peche*  p = (Peche*)  malloc(sizeof( Peche));
   Moteur* m = (Moteur*) malloc(sizeof(Moteur));
   Bateau* b = (Bateau*) malloc(sizeof(Bateau));

   setNom(b, nomBateau);

   setTypeBateau(b, MOTORISE);

   setMoteur(b, m);

   setPuissanceMoteur(b, puissanceMoteur);

   setUtiliteBateauMoteur(b, PECHE);

   setPeche(b, p);

   setQuantiteAutoriseePoissons(b, quantiteAutoriseePoissons);

   return b;
}

// Cree un bateau de plaisance en utilisant l'allocation dynamique
// ATTENTION : Nécessite une destruction via la fonction detruitBateau
Bateau* creeBateauPlaisance(Nom nomBateau, uint16_t puissanceMoteur,
                            uint8_t longueurBateau, Nom nomProprietaire)
{
   Plaisance* p = (Plaisance*) malloc(sizeof(Plaisance));
   Moteur*    m = (Moteur*)    malloc(sizeof(   Moteur));
   Bateau*    b = (Bateau*)    malloc(sizeof(   Bateau));

   setNom(b, nomBateau);

   setTypeBateau(b, MOTORISE);

   setMoteur(b, m);

   setPuissanceMoteur(b, puissanceMoteur);

   setUtiliteBateauMoteur(b, PLAISANCE);

   setPlaisance(b, p);

   setLongueurBateau(b, longueurBateau);

   setNomProprietaire(b, nomProprietaire);
```

```
   return b;
}

// Cree un bateau a voile en utilisant l'allocation dynamique
// ATTENTION : Nécessite une destruction via la fonction detruitBateau
Bateau* creeBateauAVoile(Nom nomBateau, uint16_t surfaceVoilure)
{
   Voile*  v = (Voile*)  malloc(sizeof( Voile));
   Bateau* b = (Bateau*) malloc(sizeof(Bateau));

   setNom(b, nomBateau);

   setTypeBateau(b, A_VOILE);

   setVoile(b, v);

   setSurfaceVoilure(b, surfaceVoilure);

   return b;
}

void detruitBateau(Bateau* b)
{
   if (estAVoile(b))
   {
      free((Voile*) getVoile(b));
   }
   else if (estMotorise(b))
   {
      if (estUtilePeche(b))
      {
         free((Peche*) getPeche(b));
      }
      else if (estUtilePlaisance(b))
      {
         free((Plaisance*) getPlaisance(b));
      }
      free((Moteur*) getMoteur(b));
   }

   free(b);
}

bool estMotorise(const Bateau* b)
{
   return *getTypeBateau(b) == MOTORISE;
}

bool estAVoile(const Bateau* b)
{
   return *getTypeBateau(b) == A_VOILE;
}

bool estUtilePeche(const Bateau* b)
{
   if (estMotorise(b))
   {
      return *getUtiliteBateau(b) == PECHE;
   }
   return false;
}

bool estUtilePlaisance(const Bateau* b)
{
   if (estMotorise(b))
   {
      return *getUtiliteBateau(b) == PLAISANCE;
   }
   return false;
```

```
}


// --------------  Getter / Setter ----------------

// Plaisance
void setLongueurBateau(Bateau* b, uint8_t nouvelleLongueur)
{
   if (estUtilePlaisance(b))
   {
      b->motorisation.moteur->utilite.plaisance->longueurBateau = nouvelleLongueur;
   }
}

const uint8_t* getLongueurBateau(const Bateau* b)
{
   if (estUtilePlaisance(b))
   {
      return &getPlaisance(b)->longueurBateau;
   }

   return NULL;
}

void setNomProprietaire(Bateau* b, Nom nouveauNom)
{
   if (estUtilePlaisance(b))
   {
      b->motorisation.moteur->utilite.plaisance->nomProprietaire = nouveauNom;
   }
}

const Nom* getNomProprietaire(const Bateau* b)
{
   if (estUtilePlaisance(b))
   {
      return &getPlaisance(b)->nomProprietaire;
   }

   return NULL;
}


// Pêche
void setQuantiteAutoriseePoissons(Bateau* b, uint8_t nouvelleQuantiteEnTonne)
{
   if (estUtilePeche(b))
   {
      b->motorisation.moteur->utilite.peche->quantiteAutoriseePoissons
                                                = nouvelleQuantiteEnTonne;
   }
}

const uint8_t* getQuantiteAutoriseePoissons(const Bateau* b)
{
   if (estUtilePeche(b))
   {
      return &getPeche(b)->quantiteAutoriseePoissons;
   }

   return NULL;
}


// Bateau à voile
void setSurfaceVoilure(Bateau* b, uint16_t nouvelleSurface)
{
   if (estAVoile(b))
   {
```

```
      b->motorisation.voile->surfaceVoilure = nouvelleSurface;
   }
}

const uint16_t* getSurfaceVoilure(const Bateau* b)
{
   if (estAVoile(b))
   {
      return &getVoile(b)->surfaceVoilure;
   }

   return NULL;
}


// Bateau à moteur
void setPuissanceMoteur(Bateau* b, uint16_t nouvellePuissanceEnCV)
{
   if (estMotorise(b))
   {
      b->motorisation.moteur->puissanceMoteur = nouvellePuissanceEnCV;
   }
}

const uint16_t* getPuissanceMoteur(const Bateau* b)
{
   if (estMotorise(b))
   {
      return &getMoteur(b)->puissanceMoteur;
   }

   return NULL;
}

void setUtiliteBateauMoteur(Bateau* b, UtiliteBateau nouvelleUtiliteBateau)
{
   if (estMotorise(b))
   {
      b->motorisation.moteur->utiliteBateau = nouvelleUtiliteBateau;
   }
}

const UtiliteBateau* getUtiliteBateau(const Bateau* b)
{
   if (estMotorise(b))
   {
      return &getMoteur(b)->utiliteBateau;
   }

   return NULL;
}

void setPeche(Bateau* b, Peche* p)
{
   if (estUtilePeche(b))
   {
      b->motorisation.moteur->utilite.peche = p;
   }
}

const Peche* getPeche(const Bateau* b)
{
   if (estUtilePeche(b))
   {
      return getMoteur(b)->utilite.peche;
   }

   return NULL;
}
```

```
void setPlaisance(Bateau* b, Plaisance* p)
{
   if (estUtilePlaisance(b))
   {
      b->motorisation.moteur->utilite.plaisance = p;
   }
}

const Plaisance* getPlaisance(const Bateau* b)
{
   if (estUtilePlaisance(b))
   {
      return getMoteur(b)->utilite.plaisance;
   }

   return NULL;
}

// Bateau
void setNom(Bateau* b, Nom nouveauNom)
{
   b->nom = nouveauNom;
}

const Nom* getNom(const Bateau* b)
{
   return &b->nom;
}

void setTypeBateau(Bateau* b, TypeBateau nouveauType)
{
   b->typeBateau = nouveauType;
}

const TypeBateau* getTypeBateau(const Bateau* b)
{
   return &b->typeBateau;
}


void setMoteur(Bateau* b, Moteur* m)
{
   if (estMotorise(b))
   {
      b->motorisation.moteur = m;
   }
}

const Moteur* getMoteur(const Bateau* b)
{
   if (!estMotorise(b))
   { return NULL; }

   return b->motorisation.moteur;
}

void setVoile(Bateau* b, Voile* v)
{
   if (estAVoile(b))
   {
      b->motorisation.voile = v;
   }
}

const Voile* getVoile(const Bateau* b)
{
   if (!estAVoile(b))
   { return NULL; }
```

```
    return b->motorisation.voile;
}
```

Port.h

```
/*
 ------------------------------------------------------------------------------
 Laboratoire : N°5 - Port, Bateaux et Taxes
 Fichier     : Port.h
 Auteur(s)   : Alec Berney, Quentin Forestier, Victoria Logan
 Date        : 29.05.2020

 But         : Définir différentes structures servant à modéliser un Port contenant
               différents types de Bateau, et
               prototyper des fonctions utiles à la gestion d'un Port.
               Pour se faire, on définit également différentes macros qui servent
               à stocker les différentes valeurs constantes avec lesquelles nous
               travaillons ici.

 Remarque(s) : -

 Compilateur : MinGW-gcc 6.3.0
 ------------------------------------------------------------------------------
 */

#ifndef INF2_LABO5_PORT_H
#define INF2_LABO5_PORT_H

#include "Bateau.h"

#define TAXE_BASE_VOILIER 50.0
#define TAXE_BASE_BATEAU_MOTEUR 100.0

#define TAILLE_VOILURE_TAXE_SUPP 200.0
#define TAXE_SPECIFIQUE_MAX_VOILIER 25.0
#define TAXE_SPECIFIQUE_MIN_VOILIER 0.0

#define TONNES_POISSONS_TAXE_SUPP 20.0
#define TAXE_SPECIFIQUE_MAX_PECHE 100.0
#define TAXE_SPECIFIQUE_MIN_PECHE 0.0

#define PUISSANCE_MOTEUR_TAXE_SUPP 100.0
#define TAXE_SPECIFIQUE_MAX_PLAISANCE(longueurEnM) ((longueurEnM) * 15.0)
#define TAXE_SPECIFIQUE_MIN_PLAISANCE 50.0

typedef struct Port
{
   Bateau**  bateaux;
   uint16_t capacite;
   uint16_t   taille;
} Port;

Port* construitPort(uint16_t capacite);

// Détruit le port et tous les bateaux présents
void detruitPortEtBateaux(Port* p);

// Renvoie la position du bateau en cas de réussite
// Renvoie -1 si le bateau n'est pas le port.
int trouveBateau(const Port* p, const Bateau* b);

void ajouteBateau(Port* p, Bateau* b);

// L'ordre des bateaux dans le port peut-être modifié
void supprimeBateau(Port* p, const Bateau* b);

double calculTaxeAnnuelle(const Bateau* b);

#endif //INF2_LABO5_PORT_H
```

Port.c

```
/*
 ------------------------------------------------------------------------------
 Laboratoire : N°5 - Port, Bateaux et Taxes
 Fichier     : Port.c
 Auteur(s)   : Alec Berney, Quentin Forestier, Victoria Logan
 Date        : 29.05.2020

 Compilateur : MinGW-gcc 6.3.0
 ------------------------------------------------------------------------------
 */

#include "Port.h"
#include <stdlib.h>
#include <string.h>

Port* construitPort(uint16_t capacite)
{
   Port* p = (Port*) malloc(sizeof(Port));

   p->taille = 0;
   p->capacite = capacite;

   p->bateaux = (Bateau**) malloc(sizeof(Bateau*) * capacite);

   return p;
}

void detruitPortEtBateaux(Port* p)
{
   for (uint16_t i = 0; i < p->taille; ++i)
   {
      detruitBateau(p->bateaux[i]);
   }
   free(p);
}

int trouveBateau(const Port* p, const Bateau* b)
{
   for (uint16_t i = 0; i < p->taille; ++i)
   {
      if (p->bateaux[i] == b)
      {
         return i;
      }
   }
   return -1;
}

void ajouteBateau(Port* p, Bateau* b)
{
   if (p->taille < p->capacite)
   {
      p->bateaux[p->taille] = b;
   }
   else
   {
      p->capacite *= 2;
      Bateau** tmp = p->bateaux;

      p->bateaux = realloc(p->bateaux, sizeof(Bateau*) * p->capacite);

      memcpy(p->bateaux, tmp, sizeof(Bateau*) * p->taille);
      p->bateaux[p->taille] = b;

   }
   ++p->taille;
}

void supprimeBateau(Port* p, const Bateau* b)
```

```
{
   int index = trouveBateau(p, b);
   if (index != -1)
   {
      // Ecrase le pointeur à l'index trouvé par
      // le pointeur du dernier bateau du tableau
      p->bateaux[index] = p->bateaux[p->taille - 1];
      p->bateaux[p->taille - 1] = NULL;
      --p->taille;
   }
}

double calculTaxeAnnuelle(const Bateau* b)
{
   double resultat = 0.0;

   if (estUtilePlaisance(b))
   {
      resultat += TAXE_BASE_BATEAU_MOTEUR;
      if (*getPuissanceMoteur(b) < PUISSANCE_MOTEUR_TAXE_SUPP)
      {
         resultat += TAXE_SPECIFIQUE_MIN_PLAISANCE;
      }
      else
      {
         resultat += TAXE_SPECIFIQUE_MAX_PLAISANCE(*getLongueurBateau(b));
      }
   }
   else if (estUtilePeche(b))
   {
      resultat += TAXE_BASE_BATEAU_MOTEUR;
      if (*getQuantiteAutoriseePoissons(b) < TONNES_POISSONS_TAXE_SUPP)
      {
         resultat += TAXE_SPECIFIQUE_MIN_PECHE;
      }
      else
      {
         resultat += TAXE_SPECIFIQUE_MAX_PECHE;
      }
   }
   else if (estAVoile(b))
   {
      resultat += TAXE_BASE_VOILIER;
      if (*getSurfaceVoilure(b) < TAILLE_VOILURE_TAXE_SUPP)
      {
         resultat += TAXE_SPECIFIQUE_MIN_VOILIER;
      }
      else
      {
         resultat += TAXE_SPECIFIQUE_MAX_VOILIER;
      }
   }
   return resultat;
}
```

Outils.h

```
/*
 ------------------------------------------------------------------------------
 Laboratoire : N°5 - Port, Bateaux et Taxes
 Fichier     : Outils.h
 Auteur(s)   : Alec Berney, Quentin Forestier, Victoria Logan
 Date        : 29.05.2020

 But         : Différentes fonctions utiles à différentes statistiques
               (somme, moyenne et médiane) de taxes sur les Bateau d'un Port,
               ainsi qu'à l'affichage de ces dernières, de Bateau et de Port.
               Pour se faire, les constantes globales stockant le type de Bateau
               (Voilier ou Motorise), l'utilité d'un bateau (Peche ou Plaisance),
               ainsi que tous les types de Bateau (Voilier, Motorise Peche,
               Motorise Plaisance) sont ici déclarées.

 Remarque(s) : - Si le tableau trié contient un nombre pair d'éléments, la médiane
                 est calculée en prenant l'élément d'indice égal à la partie entière
                 INFERIEURE de ((n+1)/2) -1.

 Compilateur : MinGW-gcc 6.3.0
 ------------------------------------------------------------------------------
 */

#ifndef INF2_LABO5_OUTILS_H
#define INF2_LABO5_OUTILS_H

#include "Port.h"

#define NB_TYPE_BATEAUX 3

enum TypeBateau {VOILIER, BATEAU_PECHE, BATEAU_PLAISANCE};

extern const char* TYPE_BATEAU_AFFICHAGE[2];
extern const char* UTILITE_BATEAU_AFFICHAGE[2];
extern const char* TOUS_TYPES_BATEAU_AFFICHAGE[NB_TYPE_BATEAUX];

uint16_t* getNbBateauxParType(const Port* p);

void getTabTaxesParBateaux(const Port* p, double* tabTaxesPlaisance, double* tabTaxesPeche,
double* tabTaxesVoilier);

// Fonction de comparaison pour le tri (qsort)
int cmpfunc(const void* a, const void* b);

// Fonction qui remplit le tableau tab Taxes avec la moyenne des taxes annuelles, par type
double* calculerMoyenneTaxesAnnuellesParType(const Port* p);

// Fonction qui remplit le tableau tab Taxes avec la médiane des taxes annuelles, par type
double* calculerMedianeTaxesAnnuellesParType(const Port* p);

// Fonction qui remplit le tableau tab Taxes avec la somme des taxes annuelles, par type
double* calculerSommeTotaleTaxesAnnuellesParType(const Port* p);

void afficherTaxesParType(const Port* p);

void afficherBateauxPort(const Port* p);

void afficherBateau(const Bateau* b);

void afficherTabTaxesParType(double* tabTaxesParType);

#endif //INF2_LABO5_OUTILS_H
```

Outils.c

```
/*
 ------------------------------------------------------------------------------
 Laboratoire : N°5 - Port, Bateaux et Taxes
 Fichier     : Outils.c
 Auteur(s)   : Alec Berney, Quentin Forestier, Victoria Logan
 Date        : 29.05.2020

 Compilateur : MinGW-gcc 6.3.0
 ------------------------------------------------------------------------------
 */
#include <stdlib.h>
#include <stdio.h>
#include "Outils.h"

const char* TYPE_BATEAU_AFFICHAGE[2] = {"Voilier", "Motorise"};
const char* UTILITE_BATEAU_AFFICHAGE[2] = {"Peche", "Plaisance"};
const char* TOUS_TYPES_BATEAU_AFFICHAGE[NB_TYPE_BATEAUX] = {"Voilier",
                                                            "Motorise Peche",
                                                            "Motorise Plaisance"};

uint16_t* getNbBateauxParType(const Port* p)
{
   uint16_t* tabNbBateaux = calloc(NB_TYPE_BATEAUX, sizeof(uint16_t));

   for (size_t i = 0; i < p->taille; ++i)
   {

      if (estUtilePlaisance(p->bateaux[i]))
      {
         ++tabNbBateaux[BATEAU_PLAISANCE];
      }
      else if (estUtilePeche(p->bateaux[i]))
      {
         ++tabNbBateaux[BATEAU_PECHE];
      }
      else
      {
         ++tabNbBateaux[VOILIER];
      }
   }

   return tabNbBateaux;
}

void getTabTaxesParBateaux(const Port* p, double* tabTaxesPlaisance,
                           double* tabTaxesPeche, double* tabTaxesVoilier)
{
   size_t nbPlaisance = 0, nbPeche = 0, nbVoilier = 0;
   for (size_t i = 0; i < p->taille; ++i)
   {
      Bateau* b = p->bateaux[i];
      if (estUtilePeche(b))
      {
         tabTaxesPeche[nbPeche] = calculTaxeAnnuelle(b);
         ++nbPeche;
      }
      else if (estUtilePlaisance(b))
      {
         tabTaxesPlaisance[nbPlaisance] = calculTaxeAnnuelle(b);
         ++nbPlaisance;
      }
      else
      {
         tabTaxesVoilier[nbVoilier] = calculTaxeAnnuelle(b);
         ++nbVoilier;
      }
```

```
   }
}

int cmpfunc(const void* a, const void* b)
{
   return ((int) (*(double*) a - *(double*) b));
}


double* calculerMoyenneTaxesAnnuellesParType(const Port* p)
{
   double* tabMoyennesTaxeParType = calloc(NB_TYPE_BATEAUX, sizeof(double));
   tabMoyennesTaxeParType = calculerSommeTotaleTaxesAnnuellesParType(p);

   size_t nbBateauxPlaisanceDansPort = 0,
          nbBateauxPecheDansPort     = 0,
          nbBateauxVoilierDansPort   = 0;

   for (size_t i = 0; i < p->taille; ++i)
   {
      if (estMotorise(p->bateaux[i]))
      {
         if (estUtilePlaisance(p->bateaux[i]))
         {
            ++nbBateauxPlaisanceDansPort;
         }
         else
         {
            ++nbBateauxPecheDansPort;
         }

      }
      else
      {
         ++nbBateauxVoilierDansPort;
      }
   }

   tabMoyennesTaxeParType[BATEAU_PLAISANCE] /= nbBateauxPlaisanceDansPort;
   tabMoyennesTaxeParType[BATEAU_PECHE]     /= nbBateauxPecheDansPort;
   tabMoyennesTaxeParType[VOILIER]          /= nbBateauxVoilierDansPort;

   return tabMoyennesTaxeParType;
}

double* calculerMedianeTaxesAnnuellesParType(const Port* p)
{
   double* tabMedianTaxesParType = malloc(NB_TYPE_BATEAUX * sizeof(double));

   uint16_t* tmp = getNbBateauxParType(p);

   size_t nbPlaisance = tmp[BATEAU_PLAISANCE];
   size_t nbPeche     = tmp[BATEAU_PECHE];
   size_t nbVoilier   = tmp[VOILIER];

   double tabTaxesPlaisance[nbPlaisance];
   double tabTaxesPeche[nbPeche];
   double tabTaxesVoilier[nbVoilier];

   getTabTaxesParBateaux(p, tabTaxesPlaisance, tabTaxesPeche, tabTaxesVoilier);

   qsort(tabTaxesPlaisance, nbPlaisance, sizeof(double), cmpfunc);
   qsort(    tabTaxesPeche,     nbPeche, sizeof(double), cmpfunc);
   qsort(  tabTaxesVoilier,   nbVoilier, sizeof(double), cmpfunc);

   tabMedianTaxesParType[VOILIER]          = tabTaxesVoilier[(nbVoilier + 1) / 2 - 1];
   tabMedianTaxesParType[BATEAU_PECHE]     = tabTaxesPeche[(nbPeche + 1) / 2 - 1];
   tabMedianTaxesParType[BATEAU_PLAISANCE] = tabTaxesPlaisance[(nbPlaisance + 1) / 2 - 1];
```

```
   return tabMedianTaxesParType;
}

double* calculerSommeTotaleTaxesAnnuellesParType(const Port* p)
{
   double* tabSommesTaxeParType = calloc(NB_TYPE_BATEAUX, sizeof(double));

   for (size_t i = 0; i < p->taille; ++i)
   {

      if (estUtilePlaisance(p->bateaux[i]))
      {
         tabSommesTaxeParType[BATEAU_PLAISANCE] += calculTaxeAnnuelle(p->bateaux[i]);
      }
      else if (estUtilePeche(p->bateaux[i]))
      {
         tabSommesTaxeParType[BATEAU_PECHE] += calculTaxeAnnuelle(p->bateaux[i]);
      }
      else
      {
         tabSommesTaxeParType[VOILIER] += calculTaxeAnnuelle(p->bateaux[i]);
      }
   }
   return tabSommesTaxeParType;
}

void afficherTaxesParType(const Port* p)
{
   // taxesParType[i] : i = 0 <-> voilier, i = 1 <-> peche, i = 2 <-> plaisance
   double* tabTaxesParType = calloc(NB_TYPE_BATEAUX, sizeof(double));

   printf("SOMME TOTALE des taxes annuelles dues par type de bateau : \n");
   tabTaxesParType = calculerSommeTotaleTaxesAnnuellesParType(p);
   afficherTabTaxesParType(tabTaxesParType);

   printf("MONTANT MOYEN des taxes annuelles dues par type de bateau : \n");
   tabTaxesParType = calculerMoyenneTaxesAnnuellesParType(p);
   afficherTabTaxesParType(tabTaxesParType);

   printf("MONTANT MEDIAN des taxes annuelles dues par type de bateau : \n");
   tabTaxesParType = calculerMedianeTaxesAnnuellesParType(p);
   afficherTabTaxesParType(tabTaxesParType);
}


void afficherBateauxPort(const Port* p)
{
   for (size_t i = 0; i < p->taille; ++i)
   {
      afficherBateau(p->bateaux[i]);
      printf("\n");
   }
}

void afficherBateau(const Bateau* b)
{
   printf("Nom                 : %s \n", *getNom(b));
   printf("Type de bateau      : %s \n", TYPE_BATEAU_AFFICHAGE[*getTypeBateau(b)]);

   if (estMotorise(b))
   {
      printf("Puissance du moteur : %d [CV] \n", *getPuissanceMoteur(b));
      printf("Utilite du bateau   : %s \n",
             UTILITE_BATEAU_AFFICHAGE[*getUtiliteBateau(b)]);

      if (estUtilePlaisance(b))
      {
         printf("Longueur du bateau  : %d [m^2] \n", *getLongueurBateau(b));
         printf("Nom du proprietaire : %s \n", *getNomProprietaire(b));
```

```
      }
      else
      {
         printf("Quantite maximum de peche autorisee : %d [tonnes] \n",
                *getQuantiteAutoriseePoissons(b));
      }
   }
   else
   {
      printf("Surface de la voilure : %d [m^2] \n", *getSurfaceVoilure(b));
   }
}

void afficherTabTaxesParType(double* tabTaxesParType)
{
   for (size_t i = 0; i < NB_TYPE_BATEAUX; ++i)
   {
      printf("%-19s : %g \n", TOUS_TYPES_BATEAU_AFFICHAGE[i], tabTaxesParType[i]);
   }
   printf("\n");
}
```