À la manière d'une Javadoc, ce document recense la totalité des fonctions déclarées et utilisées du programme TCHOU-TCHOU. Nous vous conseillons d'utiliser **CTRL + F** pour taper la fonction que vous recherchez. Pour le fonctionnement INTERNE des fonctions, se référer aux codes sources directement ! Le document explique uniquement : "Pourquoi ? Avec quoi ? Pour donner quoi ?" de chaque fonction.

reseau.h & reseauAccesseur.h

------ | structure de Gestion du réseau | ------

----- Reseau ----- *typedef struct s_reseau* Reseau*
C'est un pointeur sur la structure s_reseau, il va nous permettre d'utiliser de manière dynamique notre réseau !

----- Gare ----- *typedef struct s_gare* Gare*
C'est un pointeur sur la structure s_gare. Il nous permet de gérer nos gares de manière dynamique.

----- Trajet ----- *typedef struct s_trajet* Trajet*
C'est un pointeur sur la structure s_trajet. Il nous permet de gérer l'ensemble des trajets de chaque gare de manière dynamique.

------ | fonctions de Gestion du réseau | ------

----- initReseau ----- *Reseau initReseau()*
Implémente en mémoire le réseau routier complet (avec les gares et les trajets initialisés, trains et voyageurs également) à partir de deux fichiers texte.
Paramètre : AUCUN
Retour : Reseau un pointeur vers l'emplacement du réseau en mémoire

----- initGare ----- *int initGare(Reseau, FILE*)*
Implémente en mémoire une gare dans le réseau routier à partir d'un fichier texte (sauvegarde précédente).
Paramètre : Reseau un pointeur vers l'emplacement du réseau en mémoire, FILE* un pointeur vers le fichier reseau.txt
Retour : 0 si la gare a pu être stockée en mémoire, 1 sinon.

----- initTrajet ----- *int initTrajet(Gare, FILE*)*
Implémente en mémoire le(s) trajet(s) d'une gare donné en paramètre à partir d'un fichier texte (sauvegarde précédente).
Parametre : GARE La gare à qui appartient le trajet que l'on va entrer en mémoire, FILE* un pointeur vers le fichier trajet.txt
Retour : 0 si le trajet a pu être stocké en mémoire, 1 sinon.

----- sauvReseau ----- *Reseau sauvReseau(Reseau)*
Sauvegarde dans les fichiers trajet.txt et reseau.txt le réseau routier.
Parametre : Reseau un pointeur vers l'emplacement du reseau en mémoire
Retour : Reseau un pointeur vers l'emplacement du reseau en mémoire

----- closeReseau ----- *void closeReseau(Reseau)*
libère la mémoire prise par le réseau.
Parametre : Reseau un pointeur vers l'emplacement du reseau en mémoire
Retour : void

----- ajouterUneGare ----- *int ajouterUneGare(Reseau, char*)*
Ajoute une gare, dont le nom est passé en parametre, au réseau routier.
Parametre : Reseau un pointeur vers l'emplacement du reseau en memoire, char* le nom de la gare a ajouté
Retour : 0 si la gare a été ajouté avec succès, 1 sinon

----- ajouterUnTrajet ----- *int ajouterUnTrajet(Gare, Gare, int)*
Ajoute un trajet entre deux gares (aller-retour) avec un temps donné dans le réseau routier.
Parametre : Gare Un pointeur vers une des deux gares du trajet, Gare Un pointeur vers la seconde gare du trajet, int le temps en minute de trajet entre les deux gares
Retour : 0 si le trajet a pu être implémenté au réseau, 1 sinon

----- retirerUnTrajet ----- *int retirerUnTrajet(Reseau, Gare, Gare)*
Retire un trajet entre deux gares (aller-retour) avec un temps donné dans le réseau routier. Egalement le retire des potentiels trains qui l'utilise.
Parametre : *Reseau,* le pointeur vers l'emplacement du réseau en mémoire, Gare Un pointeur vers une des deux gares du trajet, Gare Un pointeur vers la seconde gare du trajet.
Retour : 0 si le trajet a bien etait retiré, 1 sinon

----- retirerUneGare ----- *int retirerUneGare(Reseau, Gare)*
Ajoute un trajet entre deux gares (aller-retour) avec un temps donné dans le réseau routier. Egalement la retire des potentiels trains qui l'utilise.
Parametre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, Gare Un pointeur vers la gare.
Retour : 0 si la gare a pu être enlevé du réseau, 1 sinon

------ | Accesseur sur les structures de Gestion du réseau | ------

Afin de pouvoir accéder aux paramètres des 3 structures s_gare, s_reseau et s_trajets dans tous le programme, nous avons besoin d'utiliser des accesseurs sur ces paramètres.

----- gareHead ----- *Gare gareHead(Reseau)*
Parametre : Reseau un pointeur vers l'emplacement du reseau en memoire
Retour : Gare un pointeur vers la Gare en tête de la liste doublement chainée de gare du réseau

----- gareTail ----- *Gare gareHead(Reseau)*
Parametre : Reseau un pointeur vers l'emplacement du reseau en memoire
Retour : Gare un pointeur vers la Gare en queue de la liste doublement chainée de gare du réseau

----- tailleReseau ----- *int tailleReseau(Reseau)*
Parametre : Reseau un pointeur vers l'emplacement du reseau en memoire
Retour : int le nombre de gare dans le réseau.

-----headTrainReseau ----- *Train headTrainReseau(Reseau)*
Parametre : Reseau un pointeur vers l'emplacement du reseau en memoire
Retour : Train un pointeur vers l'emplacement du premier train de la liste en mémoire.

----- tailTrainReseau ----- *Train tailTrainReseau(Reseau)*
Parametre : Reseau un pointeur vers l'emplacement du reseau en memoire
Retour : Train un pointeur vers l'emplacement du dernier train de la liste en mémoire.

----- nbTrainReseau ----- *int nbTrainReseau(Reseau)*
Parametre : Reseau un pointeur vers l'emplacement du reseau en memoire
Retour : int le nombre de train dans le réseau

----- nomDeGare ----- *char* nomDeGare(Gare)*
Parametre : Gare un pointeur vers une Gare du reseau
Retour : char* le nom de la gare passé en parametre

----- nbTrajetDeLaGare ----- *int nbTrajetDeLaGare(Gare)*
Parametre : Gare un pointeur vers une Gare du reseau
Retour : int le nombre de trajets amenant à la Gare passé en parametre (son nombre de "voisin")

----- trajetHeadDeLaGare ----- *Trajet trajetHeadDeLaGare(Gare)*
Parametre : Gare un pointeur vers une Gare du reseau
Retour : Trajet Un pointeur vers l'emplacement du trajet en tête de la liste chainée des trajets de la Gare

----- trajetTailDeLaGare ----- *Trajet trajetTailDeLaGare(Gare)*
Parametre : Gare un pointeur vers une Gare du reseau
Retour : Trajet un pointeur vers l'emplacement du trajet en queue de la liste chainée des trajets de la Gare

----- gareNext ----- *Gare gareNext(Gare)*
Parametre : Gare un pointeur vers une Gare du reseau
Retour : Gare un pointeur vers la Gare suivante du réseau (suivante au sens de la liste chainée, pas de lien avec les itinireraires, trajets, chemin entre les gares)

----- garePrevious ----- *Gare garePrevious(Gare)*
Parametre : Gare un pointeur vers une Gare du reseau
Retour : Gare un pointeur vers la Gare précédente du réseau (précédente au sens de la liste chainée, pas de lien avec les itinireraires, trajets, chemin entre les gares)

----- tempsDuTrajet ----- *int tempsDuTrajet(Trajet tr);*
Parametre : Trajet un pointeur vers l'emplacement d'un trajet entre deux gares en mémoire.
Retour : int le temps de trajet en minute du trajet passé en paramètre

----- gareArvDuTrajet ----- *Gare gareArvDuTrajet(Trajet tr);*
Parametre : Trajet un pointeur vers l'emplacement d'un trajet entre deux gares en mémoire.
Retour : Gare un pointeur vers la gare d'arrivee du trajet donne en parametre

----- trajetNext ----- *Trajet trajetNext(Trajet tr);*
Paramètre : Trajet un pointeur vers l'emplacement d'un trajet entre deux gares en mémoire.
Retour : Trajet un pointeur vers l'emplacement du trajet suivant en mémoire de la Gare (suivante au sens de la liste chainée)

parcoursGraphe.h & itineraireAccesseur.h

------ | Structures de gestion d'Itineraire & de la Recherche | ------

----- Itineraire ----- *typedef struct s_itineraire* Itineraire*
C'est un pointeur sur la structure s_Itineraire, il va nous permettre d'utiliser de manière dynamique notre Itineraire !

----- Sommet ----- *typedef struct s_sommet* Sommet*
C'est un pointeur sur la structure s_sommet, il va nous permettre d'utiliser de manière dynamique notre Sommet !

----- Ensemble ----- *typedef struct s_ensemble* Ensemble*
C'est un pointeur sur la structure s_ensemble, il va nous permettre d'utiliser de manière dynamique notre Ensemble !

------ | fonctions de Gestion de la recherche | ------

----- ajoutSommet ----- *int ajoutSommet(Ensemble, Trajet, Sommet)*
Permet d'ajouter et d'initialiser un sommet lors de la recherche d'itineraire entre deux gares.
Paramètre : *Ensemble*, un pointeur vers l'emplacement de l'ensemble de recherche en mémoire, *Trajet*, un pointeur vers l'emplacement du trajet entre le Sommet exsitant et la gare que l'on veut ajouter en tant que sommet, *Sommet*, un pointeur vers l'emplacement du Sommet existant (appelé sommet Père)
Retour : 0 si le trajet a pu être ajoute à l'ensemble, 1 sinon;

----- majDistance ----- *int majDistance(Trajet, Sommet, Sommet)*
Permet de mettre à jour la distance entre deux sommets lors de la recherche d'un itineraire le plus court entre deux gares.
Paramètre : *Trajet*, un pointeur vers l'emplacement du trajet entre les deux Sommets , *Sommet*, un pointeur vers l'emplacement du premier Sommet , *Sommet*, un pointeur vers l'emplacement du second Sommet
Retour : 0

----- testVoisin ----- *int testVoisin(Ensemble, Sommet)*
Test tous les voisins d'un sommet lors de la recherche de l'itineraire le plus court entre deux gares.
Paramètre : *Ensemble*, un pointeur vers l'emplacement de l'ensemble de recherche en mémoire, *Sommet*, un pointeur vers l'emplacement du Sommet existant duquel on regarde les voisins (sommets/gare séparé par un trajet)
Retour : 0

----- rechercheItineraire ----- *Itineraire rechercheItineraire(Reseau, Gare, Gare)*
Lance la recherche d'un itineraire le plus court entre deux gares.
Paramètre : *Reseau*, un pointeur vers l'emplacement du réseau en mémoire, *Gare*, la gare de départ du passager, *Gare* la gare d'arrivée du passager
Retour : Itineraire contenant l'itineraire entre les deux gares, NULL en cas de problème.

----- initialisationGraphe ----- *Ensemble initialisationEnsemble(Gare)*
Initialise le graphe pour la recherche d'itineraire le plus court entre deux gares.
Paramètre :*Gare*, la gare de départ du passager
Retour : *Ensemble*, l'ensemble avec les premiers voisins de la Gare initialisé

----- freeGrapheRecherche ----- *void freeGrapheRecherche(Ensemble)*
Libère la mémoire occupée par l'éxécution de la fonction rechercheItineraire().
Paramètre : *Ensemble*, l'ensemble de recherche
Retour : */*

------ | Fonctions de Recherche/Manipulation dans le Réseau | ------

----- rechercheGare ----- *Gare rechercheGare(Reseau, char*)*
Parcourt l'ensemble des gares pour trouver celle qui correspond au char* passé en paramètre
Paramètre : *Reseau*, un pointeur vers l'emplacement du réseau en mémoire, *char**, le nom de la gare à rechercher
Retour : Gare si la gare existe, NULL sinon

----- rechercheTrajet ----- *Trajet rechercheTrajet(Reseau, Gare, Gare)*
Parcourt le réseau pour trouver, s'il existe, le trajet entre les deux gares passées en paramètre.
Paramètre : *Reseau*, un pointeur vers l'emplacement du réseau en mémoire, *Gare*, une des deux gares avec lesquelles nous souhaitons vérifier s'il existe un trajet, *Gare* la seconde gare.
Retour : Trajet s'il existe, NULL sinon

----- creerItineraireVide ----- *Itineraire creerItineraireVide()*
Creer un Itineraire vide
Paramètre : */*
Retour : *Itineraire,* un itineraire vide sans information.

----- ajouteTrajetItineraire ----- *int ajouteTrajetItineraire(Itineraire, Gare, Trajet)*
Ajoute un trajet à l'itineraire
Paramètre : *Itineraire,* l'itineraire auquel on doit ajouter le trajet, *Gare* la gare de depart du trajet a ajouter, *Trajet*, le trajet à ajouter à l'itineraire
Retour : 0

----- itineraireRep ----- *Itineraire itineraireRep(Reseau, Itineraire, char*, char*)*
Permet d'initialiser l'itineraire des voyageurs issu de répertoire.txt
Paramètre : *Reseau*, le pointeur vers l'emplacement du réseau en mémoire, *Itineraire,* l'itineraire auquel on doit appliquer les modifications, *char** , le nom de la nouvelle gare de Départ, *char**, le nom de la nouvelle gare d'arrivée.
Retour : *Itineraire*, l'itineraire avec les attributs mis à jours.

----- changerArv ----- *Itineraire changerArv(Itineraire, int)*
Permet de raccourcir l'itineraire, en changeant l'arrivée.
Paramètre : *Itineraire,* l'itineraire auquel on doit appliquer les modifications, *int,* le rang du trajet, dans la liste de trajet de l'itineraire, dont la gare d'arrivée devient l'arrivée de l'itineraire.
Retour : *Itineraire*, l'itineraire mis à jours.

------ | Accesseur sur Structure | ------

----- gareDepItineraire ----- *Gare gareDepItineraire(Itineraire)*
Paramètre : *Itineraire,* l'itineraire dont on veut récupérer l'information.
Retour : *Gare*, la Gare de depart de l'itineraire

----- gareArvItineraire ----- *Gare gareDepItineraire(Itineraire)*
Paramètre : *Itineraire,* l'itineraire dont on veut récupérer l'information.
Retour : *Gare*, la Gare d'arrivée de l'itineraire

----- tempsItineraire ----- *int tempsItineraire(Itineraire)*
Paramètre : *Itineraire,* l'itineraire dont on veut récupérer l'information.
Retour : *int,* le temps en minute de durée de l'itineraire

----- listeTrajetItineraire ----- *Trajet listeTrajetItineraire(Itineraire, int)*
Paramètre : *Itineraire,* l'itineraire dont on veut récupérer l'information, *int,* l'indice du trajet à récupérer dans l'itinéraire.
Retour : *Trajet,* le trajet à l'indice int passé en parametre de l'itineraire

----- nbEtapeItineraire ----- *int nbEtapeItineraire(Itineraire)*
Paramètre : *Itineraire,* l'itineraire dont on veut récupérer l'information.
Retour : *int,* le nombre d'étape (de trajet intermédiaire) dans l'itineraire

train.h & voyageur.h & trainVoyageurAccesseur.h

------ | Les Structures de gestion des Trains et Voyageurs | ------

----- Train ----- *typedef struct s_train* Train*
Représente un train en mémoire

----- Place ----- *typedef struct s_train* Train*
Représente une place en mémoire

----- Voyageur ----- *typedef struct s_train* Train*
Représente un voyageur en mémoire

------ | Fonctions pour Initialiser et Sauvegarder | ------

----- initItineraireTrain ----- *Itineraire initItineraireTrain(Reseau, FILE*)*
Permet d'initialiser l'itineraire du train suivant du fichier.
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *FILE** pointeur vers le fichier contenant les trains
Retour : L'itineraire du train

----- initTrain ----- *Train initTrain(Reseau, FILE*, FILE*)*
Permet d'initialiser un train et toutes ses données.
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *FILE** pointeur vers le fichier contenant les trains, *FILE**, pointeur vers le fichier contenant les voyageurs et place du train.
Retour : Le train

----- sauvTrain ----- *int sauvTrain(Reseau, FILE*, FILE*)*
Permet de sauvegarder dans les fichiers .txt les données de l'ensemble des trains ainsi que leur place et passagers.
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *FILE** pointeur vers le fichier contenant les trains, *FILE**, pointeur vers le fichier contenant les voyageurs et place du train.
Retour : 0

----- initVoyageur ----- *Voyageur initVoyageur(Reseau, Place, FILE*)*
Permet d'initialiser un voyageur à partir d'une place
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *Place,* la place surlequel le voyageur est assis, *FILE**, pointeur vers le fichier contenant les voyageurs et place du train.
Retour : Le voyageur

----- initPlace ----- *Place initPlace(Reseau, FILE*)*
Permet d'initialiser une place à partir du fichier voyageur.txt
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *FILE**, pointeur vers le fichier contenant les voyageurs et place du train.
Retour : La place

----- initRepertoire ----- *Voyageur initRepertoire(Reseau, FILE*)*
Permet d'initialiser un voyageur partir du fichier repertoire.txt
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *FILE**, pointeur vers le fichier contenant le répertoire des voyageurs.
Retour : Le voyageur

----- sauvVoyageur ----- *int sauvVoyageur(Train, FILE*)*
Permet de sauvegarder l'ensemble des places et des voyageurs d'un train

Paramètre : *Train,* le train que l'on veut sauvegarder*, FILE**, pointeur vers le fichier contenant les voyageurs et place du train.
Retour : 0

----- sauvRepertoire ----- *int sauvRepertoire(Reseau, FILE*)*
Permet de sauvegarder l'ensemble des voyageurs dans le répertoire.
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *FILE**, pointeur vers le fichier contenant le répertoire des voyageurs.
Retour : 0

------ | Fonctions pour manipuler les voyageurs & places | ------

----- suppVoyageur ----- *void suppVoyageur(Reseau, char*)*
Permet de supprimer un voyageur à la fois dans le répertoire et sur les places
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *char** l'id du voyageur à supprimer
Retour : void

----- rechercheVoyageur ----- *int rechercheVoyageur(Reseau, char*)*
Permet d'afficher l'itineraire d'un voyageur en détails.
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *char** l'id du voyageur à supprimer
Retour : 0

----- creerVoyageur ----- *Voyageur creerVoyageur(Reseau, Itineraire)*
Permet de créer un voyageur en mémoire associé à l'itinéraire (réservation. A la fois dans le répertoire et placé sur les places de trains.
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *Itineraire,* l'itineraire que l'on souhaite associer au voyageur.
Retour : Le voyageur intégré dans la liste directe du réseau (répertoire)

----- tirerNumVoyageur ----- *void tirerNumVoyageur(Reseau, Voyageur)*
Permet de donner un numero d'identification unique au voyageur.
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *Voyageur,* le voyageur à qui l'on souhaite attribuer le numéro.
Retour : void

----- mettreSurUnePlace ----- *Voyageur mettreSurUnePlace(Reseau, Train, Gare, Gare, Itineraire)*
Permet de positionner un voyageur sur la place adéquate en fonction de son itineraire, et du bout de son itineraire correspondant au train.
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *Train,* le train dans lequel on doit placer notre voyageur, *Gare,* la première gare à partir de laquelle le voyageur sera dans le train, *Gare,* la dernière gare où le voyageur sera dans le train, *Itineraire,* l'itineraire général du voyageur à ajouter.

Retour : Le voyageur connecté à sa place (sans informations à part son bout d'itineraire entre les deux gares)

----- creerPlaceVide ----- *Place creerPlaceVide(Train, int)*
Permet de créer une place vide associé à un train.
Paramètre : *Train,* auquel on doit ajouter la place, *int,* le rang d'ajout de la place
Retour : la place vide avec l'identifiant associé.

----- modifVoyageur ----- *Voyageur modifVoyageur(Reseau, char*)*
Permet de modifier l'itineraire d'un voyageur.
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *char** l'id du voyageur dont on doit modifier l'itineraire.
Retour : le nouveau voyageur relié à la liste du réseau.

------ | Fonctions pour manipuler les trains | ------

----- ajouterTrain ----- *Train ajouterTrain(Reseau)*
Permet à l'utilisateur de créer un train
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire
Retour : le Train ajouté.

----- suppTrain ----- *int suppTrain(Reseau, char*)*
Permet de supprimer le train dont l'identifiant est passé en paramètre
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *char*,* le numéro d'identification du train
Retour : 0; 1 en cas d'erreur

----- rechercheTrain ----- *Train rechercheTrain(Reseau, char*)*
Permet de renvoyer le train correspondant à l'id passé en paramètre
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *char*,* l'id que l'on recherche
Retour : Train s'il existe, NULL sinon.

----- trainPasVide ----- *int trainPasVide(Train t)*
Permet de savoir si le train contient ou non des passagers
Paramètre : *Train,* le train à tester
Retour : 0 si le train est vide, 1 sinon

----- gareDansTrain ----- *int gareDansTrain(Gare, Train t)*
Permet de savoir si le train contient la gare passé en paramètre
Paramètre : *Gare,* la gare à tester, *Train,* le train à tester
Retour : 1 si oui, 0 sinon

----- verifierTrain ----- *int verifierTrain(Reseau, Gare)*
Permet de vérifier si un train empêche la Gare d'être supprimé

Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *Gare,* la gare que l'on veut supprimer.
Retour : 0 s'il n'y a pas de problème, 1 sinon.

----- trajetDansTrain ----- *int trajetDansTrain(Trajet, Gare, Train t)*
Permet de savoir si le train contient le trajet passé en paramètre
Paramètre : *Trajet,* le trajet à tester *Gare,* la gare de départ du trajet, *Train,* le train à tester
Retour : 1 si oui, 0 sinon

----- verifierTrainTrajet ----- *int verifierTrainTrajet(Reseau, Gare, Trajet)*
Permet de vérifier si un train empêche le Trajet d'être supprimé
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *Gare,* la gare de départ du trajet que l'on veut supprimer, *Trajet,* le trajet que l'on veut supprimer
Retour : 0 s'il n'y a pas de problème, 1 sinon.

----- suppGareDansTrain ----- *int suppGareDansTrain(Reseau, Gare)*
Permet de supprimer la gare dans les itineraires de trains
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *Gare,* la gare que l'on veut supprimer.
Retour : 0

----- suppTrajetDansTrain ----- *int suppGareDansTrain(Reseau, Gare, Trajet)*
Permet de supprimer lle Trajet dans les itineraires de trains
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *Gare,* la gare de départ du trajet que l'on veut supprimer, *Trajet,* le trajet que l'on veut supprimer.
Retour : 0

------ | Fonctions pour manipuler la mémoire | ------

----- ajtTrain ----- *void ajtTrain(Reseau, Train)*
Permet d'ajouter le train à la liste.
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *Train,* le train à ajouter
Retour : void

----- enleverTrain ----- *void enleverTrain(Reseau)*
Retire 1 au nombre de train en mémoire.
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire
Retour : void

----- enleverTrainHead ----- *Train enleverTrainHead(Reseau, Train)*
Retire le train en tête de la liste dans le réseau.
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *Train,* le train à enlever

Retour : le train qui vient d'être enlevé de la mémoire

----- enleverTrainTail ----- *Train enleverTrainTail(Reseau, Train)*
Retire le train en queue de la liste dans le réseau
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire, *Train,* le train à enlever
Retour : le train qui vient d'être enlevé de la mémoire

----- modifItineraireTrainTrajet ----- *Itineraire modifItineraireTrainTrajet(Train, int)*
Manipule la liste de trajet de l'itineraire du train pour retirer un trajet
Paramètre : *Train,* le train associé à l'itinéraire que l'on souhaite modifier, *int,* le rang du trajet à supprimer.
Retour : le nouvel itineraire du train

----- modifItineraireTrain ----- *Itineraire modifItineraireTrain(Train, int)*
Manipule la liste de trajet de l'itineraire du train pour retirer une gare
Paramètre : *Train,* le train associé à l'itinéraire que l'on souhaite modifier, *int,* le rang du trajet dont la gare de depart est celle à supprimer.
Retour : le nouvel itineraire du train

----- ajtVoyMemoire ----- *void ajtVoyMemoire(Reseau)*
Retire 1 au nombre de voyageur du réseau.
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire
Retour : void

----- suppVoyMemoire ----- *void suppVoyMemoire(Voyageur, Reseau)*
Raccroche le voyageur à la liste du réseau (répertoire).
Paramètre : *Voyageur*, le voyageur à raccrocher, *Reseau,* un pointeur vers l'emplacement du réseau en mémoire
Retour : void

----- chgHeadVoy ----- *void chgHeadVoy(Reseau, Voyageur)*
Supprime le voyageur de tête de la liste du réseau (répertoire).
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire,*Voyageur*, le voyageur à supprimer.
Retour : void

----- chgTailVoy ----- *void chgTailVoy(Reseau, Voyageur)*
Supprime le voyageur de queue de la liste du réseau (répertoire).
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire,*Voyageur*, le voyageur à supprimer.
Retour : void

------ | Affichage | ------

----- affichageEtatReseau ----- *void affichageEtatReseau(Reseau)*
Affiche les trajets non parcouru par un train.
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire
Retour : void

----- ensembleVoyageur ----- *int ensembleVoyageur(Reseau)*
Affiche l'ensemble des places de chaque train avec leurs voyageurs dessus.
Paramètre : *Reseau,* un pointeur vers l'emplacement du réseau en mémoire
Retour : void

------ | Accesseur sur les Trains et les Voyageurs | ------

----- placeDuTrain ----- *Place placeDuTrain(Train, int)*
Paramètre : *Train*, le train auquel on veut récupérer une place, *int,*le rang de la place à récupérer
Retour : la place du train et au rang int

----- idTrain ----- *char* idTrain(Train)*
Paramètre : *Train*, un train
Retour : le numero d'identification du train

----- cheminTrain ----- *Itineraire cheminTrain(Train)*
Paramètre : *Train*, un train
Retour : l'Itineraire du train

----- trainNext ----- *Train trainNext(Train)*
Paramètre : *Train*, un train
Retour : le train suivant dans la liste du réseau

----- trainPrevious ----- *Train trainPrevious(Train)*
Paramètre : *Train*, un train
Retour : le train précédant dans la liste du réseau

----- nbVoyageurSurLaPlace ----- *int nbVoyageurSurLaPlace(Place)*
Paramètre : *Place,* la place dont on cherche à récupérer les informations
Retour : le nombre de voyageur assis sur la place

----- idPlace ----- *char* idPlace(Place)*
Paramètre : *Place,* la place dont on cherche à récupérer les informations
Retour : le numéro d'identification de la place

----- voyageurHeadPlace ----- *Voyageur voyageurHeadPlace(Place)*
Paramètre : *Place,* la place dont on cherche à récupérer les informations
Retour : le voyageur en tête de la liste des voyageurs sur la place

----- voyageurTailPlace ----- *Voyageur voyageurTailPlace(Place)*
Paramètre : *Place,* la place dont on cherche à récupérer les informations
Retour : le voyageur en queue de la liste des voyageurs sur la place

----- idVoyageur ----- *char* idVoyageur(Voyageur)*
Paramètre : *Voyageur,* le voyageur dont on cherche à récupérer les informations
Retour : le numéro d'identification du voyageur

----- nomVoyageur ----- *char* nomVoyageur(Voyageur)*
Paramètre : *Voyageur,* le voyageur dont on cherche à récupérer les informations
Retour : le nom du voyageur

----- prenomVoyageur ----- *char* prenomVoyageur(Voyageur)*
Paramètre : *Voyageur,* le voyageur dont on cherche à récupérer les informations
Retour : le prenom du voyageur

----- cheminVoyageur ----- *Itineraire cheminVoyageur(Voyageur)*
Paramètre : *Voyageur,* le voyageur dont on cherche à récupérer les informations
Retour : l'itineraire du voyageur

----- voyageurNext -----*Voyageur voyageurNext(Voyageur)*
Paramètre : *Voyageur,* le voyageur dont on cherche à récupérer les informations
Retour : levoyageur suivant dans la liste

menu.h

Les fonctions de menu.h ont un rôle similaire à une interface graphique. Elles servent principalement à afficher sur la console des informations à l'utilisateur et gèrent l'interaction avec ce dernier.

-------| fonctions d’affichage des menus principaux |-------

--- **afficheMenuPrincipal** --- *void afficheMenuPrincipal()*
affiche sur la console le Menu principal
Paramètre : *aucun*
Retour : void

--- **afficheMenuAdmin** --- *void afficheMenuAdmin()*
affiche sur la console le Menu administrateur accessible seulement après une connexion avec un identifiant et un mot de passe.
Paramètre : *aucun*
Retour : void

--- **afficheMenuControleur** --- *void afficheMenuControleur()*

affiche sur la console le Menu administrateur accessible seulement après une connexion avec un identifiant et un mot de passe.
Paramètre : *aucun*
Retour : void

--- **afficheMenuClient** --- *void afficheMenuClient()*

affiche sur la console le Menu Client accessible à tous les utilisateurs
Paramètre : *aucun*
Retour : void

--- **afficheErreurMenu** --- *void afficheErreurMenu()*

affiche sur la console un message d’erreur lorsque l’utilisateur fait une saisie erroné
Paramètre : *aucun*
Retour : void

--- **afficheMessageQuitter** --- *void afficheMessageQuitter()*

affiche sur la console un message d'au revoir à l'utilisateur lorsque celui-ci décide d'arrêter le programme
Paramètre : *aucun*
Retour : void

-------| fonctions d’affichage dans le menu administrateur |-------

--- **menuAdminVerification**--- *int menuAdminVerification(Reseau r)***;**
Affiche sur la console des instructions à l'utilisateur et utilise verifierPwdAdmin() afin de savoir si l'utilisateur a le droit d'accéder au menuAdmin.
Si oui utilise fonction menuAdmin(r).
refuse l'accès sinon.
Paramètre : *Reseau r*
Retour : int 0

--- menuAdmin--- *int menuAdmin(Reseau r)***;**
Affiche sur la console le menu Administrateur. La fonction attend une saisie de l'utilisateur et exécute d'autres fonctions en conséquence.
Paramètre : *Reseau r*
Retour : int 0
--- menuGestionTrajet--- *int menuGestionTraje (Reseau r);*
Affiche sur la console le menu des actions possibles pour la gestion des trajets. La fonction attend une saisie de l'utilisateur pour exécuter la fonction associée a l'action demandée.
Paramètre : Réseau r
retour : int 0
---menuExportationJSON--- *int menuExportationJSON();*
Affiche sur la console le menu des actions possibles pour l'exportation. La fonction attend une saisie de l'utilisateur pour exécuter la fonction associée a l'action demandée.
Paramètre : *aucun*
Retour : int 0

---menuGestionAdministration---*int menuGestionAdministration()***;**
demande à l'utilisateur la personne dont il veut modifier les données et exécute la fonction de modification en conséquence.
Paramètre : aucun
retour : int 0

---menuAjouterTrain--- *int menuAjouterTrain(Reseau r);*
affiche un message d'information et appelle la fonction ajouterTrain pour rajouter un train à un trajet.
Paramètre : réseau r
retour : int 0

---menuSuppTrain--- *int menuSuppTrain(Reseau r);*

demande le train à supprimer à l'utilisateur et appelle la fonction suppTrain pour supprimer le train choisi
Paramètre : réseau r
retour : int 0

---menuModifTrain--- *int menuModifTrain(Reseau r);*
demande le train à modifier à l'utilisateur et appelle la fonction pour modifier le train choisi
Paramètre : réseau r.
retour : int 0

---menuAjouteGare--- *int menuAjouteGare(Reseau r);*
demande à l'utilisateur un nom de Gare (sans espace et avec chiffre) et ajoute cette gare au réseau r.
Paramètre : réseau r.
retour : int 0
---menuSupGare--- *int menuSupGare(Reseau r);*
demande à l'utilisateur un nom de Gare parmi les gares du réseau r et supprime cette gare au réseau r.
Paramètre : réseau r
retour : int 0
---menuAjouteTrajet--- *int menuAjouteTrajet(Reseau r);*
demande à l'utilisateur deux noms de Gare parmi les gares du réseau r ainsi qu'un temps de trajet t . et la fonction créer un trajet entre les 2 gares de temps t dans le réseau r.
Paramètre : réseau r
retour : int 0

---menuSupTrajet--- *int menuSupTrajet(Reseau r);*
demande à l'utilisateur un nom de Gare parmi les gares du réseau r et un second nom parmi les gares qui sont connectées à la première puis supprime le trajet entre les 2 gares le réseau r.
Paramètre : réseau r
retour : int 0

---menuChoixModification--- *int menuChoixModification(int):*
demande à l'utilisateur la donnée qu'il souhaite modifier parmi : mot de passe , nom et prénom. et appelle la fonction menuModification.
Paramètre : int représentant la personne dont on va modifier les données.
retour : int 0

---menuModification--- *int menuModification(int index, int n)*
demande à l'utilisateur de saisir son nouveau mot de passe, nom ou prénom en fonction de la valeur de index. puis appel la fonction modifierDonnee pour effectuer la modification.
Paramètre : int index correspond à la donnée à modifier. int n correspond à la personne à qui on modifie les données.
retour : int 1 si erreur, 0 sinon

-------| fonctions d'affichage dans le menu contrôleur |-------

---menuControleurVerification--- *int menuControleurVerification(Reseau r)*
Affiche sur la console des instructions à l'utilisateur et utilise verifieLogControleur() afin de savoir si l'utilisateur a le droit d'acceder au menu Controleur
Si oui utilise fonction menuControleur(int i, reseau r)
refuse l'accès sinon.
Paramètre : Réseau r
retour : int 0

---menuControleur--- *int menuControleur(int n, reseau r);*
Affiche le menu des Controleurs et attends une saisie de l'utilisateur pour savoir quel menu il lance.
Paramètre : int n indique quel contrôleur est connecté ; Réseau r
retour : int 0

---menuRechercheControleur--- *int menuRechercheControleur(Reseau r);*
demande au contrôleur le numéro d'un client et lui affiche les informations de voyage associées à ce numéro client.
Paramètre : Réseau r
retour : int 0

-------| fonctions d'affichage dans le menu client |-------

---menuClient--- *int menuClient(Reseau r);*
Affiche le menu Client et attends une saisie de l'utilisateur pour savoir quel menu il lance.

Paramètre : Réseau r
retour : int 0
---menuRechercheEtChoix--- *int menuRechercheEtChoix(Reseau r);*
Demande à l'utilisateur sa gare de départ et d'arrivée et lui affiche les informations associées sur ce trajet: nombre de train , escale , durée.
Paramètre : Réseau r
retour : int 0
---menuReservation--- *int menuReservation(Reseau r);*
Demande à l'utilisateur sa gare de départ et d'arrivée puis son nom et prénom et lui réserve une place. et affiche à l'utilisateur son identifiant de voyage utile pour les modifications.
Paramètre : Réseau r
retour : int 0
--menuModificationVoyage--- *int menuModificationVoyage(Reseau r)*
demande à l'utilisateur son numéro client. Affiche les informations de voyage du client. Demande confirmation de modification. Si oui appel de la fonction modifVoyageur() rien sinon.
Paramètre : Réseau r
retour : int 0

pwd.h

-------| fonctions de saisie sécurisé et gestion mot de passe |-------

---verifierPwdAdmin--- *int verifierPwdAdmin();*
fonction de vérification de l'identifiant et du mot de passe de l'administrateur.

La fonction va demander l'identifiant et le mot de passe à l'utilisateur et va comparer les données saisies avec celles sauvegardées.
Paramètre : aucun
retour : int 1 si il y a eu une erreur , 0 sinon

---verifierLogControleur--- *int verifierLogControleur();*
fonction de vérification de l'identifiant et du mot de passe d'un contrôleur.
La fonction va demander l'identifiant et le mot de passe à l'utilisateur et va comparer les données saisies avec celles sauvegardées .
Paramètre : aucun
retour : int 1 si il y a eu une erreur , 0 sinon

---lire--- *int lire(char* chaine ,int longueur,FILE* fichier);*
Cette fonction va lire une ligne dans un fichier et remplacer le '\n' de fin de ligne par '\0' de fin de chaîne pour stocker la chaîne dans un tableau.
Paramètre : char* adresse du tableau où l'on souhaite stocker notre string qui va être lu; int taille max de la String qu'on souhaite lire; File* pointeur vers le fichier qu'on souhaite lire.
retour : int 0 si il y a eu une erreur , 1 sinon

---lireLong--- *long lireLong();*
Cette fonction va utiliser la fonction lire sur stdin et va convertir la saisie en long. Cela évite un bug du programme si l'utilisateur entre une valeur non désirée dans les menus.
Paramètre : aucun
retour : un long , 0 si il y a eu une erreur ,

---modiferDonnee--- *void modiferDonnee(char* nomfile,int index ,char* texte);*
Cette fonction va remplacer la ligne numéro "index" dans le fichier "nomfile" avec la chaîne de caractères de "texte".
Paramètre : char* nom du fichier à lire; int index numéro de la ligne à modifier; char* texte : la chaîne de caractères qu'on souhaite écrire sur le fichier.
retour : void

---crypterVigenere--- *void crypterVigenere(char* pwd ,char* pwdCrypt);*
Crypte le mot de passe contenu dans pwd selon le principe du chiffre de Vigenère et l'ajout de Salt et stocke ce mot de passe crypter dans pwdCrypt
Paramètre : char* pwd pointeur sur le tableau stockant le mot de passe en clair; char* pwdCrypt pointeur sur le tableau qui va stocker le mot de passe une fois crypté.
retour : void

gestionJson.h

---voyageurJson--- *void voyageurJson();*
convertie les données de repertoire.txt en format json dans voyageur.json
Paramètre : aucun
retour : void

--- trajetJson--- *void trajetJson()***;**
convertie les données de train.txt en format json dans listeTrajet.json
Paramètre : aucun
retour : void