TP/Projet

Introduction

Afin de vous faire apprécier les différentes propriétés des structures de données vues en cours, nous avons pris l'initiative de mettre en place un TP/Projet englobant l'ensemble de leurs utilisations.

Le TP/Projet consiste en l'implémentation d'un code en C pour la **gestion d'une station de train** où différentes structures de données doivent être proposées afin de représenter les propriétés de chaque entité. Des fonctions reposantes sur les structures de données précédemment établies doivent être implémentées afin de simuler les comportements de chaque entité.

La fonction *main* fait appel aux fonctions en suivant un ordonnancement logique afin de synchroniser le les tâches d'une station de train.

Représentation Générale et illustration

(Voir le PowerPoint)

Diapo 1

Diapo 2

Structures de données

Dans une gare, nous retrouvons plusieurs entités, dans notre cas, nous nous limiterons à l'implémentation des entités suivantes :

- Train
- Quai
- Passager

Un *Train* doit disposer, mais ne se limite pas, aux caractéristiques suivantes :

- Identifiant unique
- Capacité maximum
- Taux de remplissage
- Liste des passagers
- Destination
- Vitesse

Un *Quai (ParkingRail)* doit disposer, mais ne se limite pas, aux caractéristiques suivantes :

- Identifiant unique
- Liste de trains
- Premier train à partir
- Nombre maximum de Train en attente

Un *Passager* doit disposer, mais ne se limite pas, aux caractéristiques suivantes :

- Un identifiant unique

- Une destination

Fonctions à implémenter

Fonctions associé aux quais (ParkingRail)

```
1.ParkingRail CreerUnParkingRail(chaine Identifiant, int Capacity);
```

Cette fonction a pour objectif de créer un quai ayant une capacité maximum d'admission de train avec un identifiant

Exemple d'appel :

```
ParkingRail PK=CreerUnParkingRail("El Moudjahid", 4);
```

```
2. int PileRemplie(ParkingRail PK)
```

Cette fonction vérifie si une pile est remplie ou non. La variable a checker est « Capacity » avec la tête de la pile

Exemple d'appel :

```
if(PileRemplie(QuaiElMoudjahid)==0){
…
}else
{
…
}
```

```
3. ParkingRail TrouverDesTrainDisponible(ParkingRail ParkingForTrain, Train *
Trains, int numTrain)
```

Rechercher des trains qui ne sont pas dans les quais (voir diapo 3).

```
4. int Empiler(Train TrainCourant, ParkingRail * ParkingForTrain)
```

Empiler un train qui arrive, à partir d'une destination particulière ou du stock des trains dans un quai particulier. Le *int* qui est retournée indique si l'empilation s'est déroulé avec succès ou bien la pile (le quai) est remplie (capacité maximum) et le train ne peut pas être empilé.

```
5. int AjouterQuais(ParkingRail * Quais,int NumQuai,chaine IdentifiantQuai,
int capacity, Train * Trains, int NumTrain) ;
```

Nous pouvons déclarer les différents quais de manière séparé comme suit :

```
ParkingRail Q1 ;

ParkingRail Q2 ;
```

Toutefois, afin de les énumérer de manière plus simplifié, nous pouvons les mettre dans un tableau. Cette fonction ajoute un quai dans un tableau et récupère les trains qui ne sont affecté à aucun quai pour le mettre dans le quai.

```
6. void UpdateQuais( ParkingRail * Quais, int NumQuai, Train * Trains, int NumTrain ) ;
```

Après avoir simulé le déroulement d'une itération, les trains qui étaient dans le départ sont devenu en trajet et n'occupent plus de place dans les quais, de ce fait, il est nécessaire d'updater l'état des quais.

```
7. Train Depiler (ParkingRail * ParkingForTrain);
```

Prendre un train d'un quai particulier et le retourner. La tête de pile doit être mise à jour.

```
8. void AfficherParkingRail(ParkingRail* ParkingRails, int NumParkingRail);
```

La fonction qui affiche l'état des quais, les trains qui y sont stationnés, leurs capacités ainsi que leurs identifiants.

```
9. int PileVide(ParkingRail * ParkingForTrain);
```

Vérifie si la pile est vide, retourne vrai (1) si la pile est vide, (0) si la pile n'est pas vide et il existe des trains dans le quai.

Fonctions associé aux passagers

```
10. Passagers CreerPassager(int IDPassagerCourant, chaine Destination) ;
```

Cette fonction a pour objectif de créer un passager ayant un identifiant et une destination particulière.

Exemple d'appel :

```
Passager * Passager1= CreerPassager (1, "Helsinki");
```

Cette fonction doit vérifier si la destination du passager existe pour ajouter le passager dans la liste et retourner un pointeur vers la structure passager. L'identifiant du passager doit être unique.

```
11. int SiListeVide(Passagers ListePassager);
```

Cette fonction a pour but de tester si la liste est des passagers est vide. Retourne 1 si la liste est vide 0 dans le cas contraire.

```
12. int TailleListe(Passagers ListePassager);
```

Cette fonction parcourt une liste et retourne sa taille.

```
13. Passagers AvoirDernierElementList(Passagers ListePassager);
```

Cette fonction parcourt une liste et retourne son dernier élément.

```
14. Passagers ReservationPlace(Passagers ListePassagers, Train * Trains, int NumTrain, chaine Destination);
```

Cette fonction a pour but de réserver une place d'un passager dans un train. Elle doit vérifier si la destination du passager et du train est la même. Elle doit aussi vérifier si il y'a de la place dans le train. Incrémenter le nombre de places prise dans le cas où la réservation est réussie.

```
15. int ExistIdDansListe(int IDRecherche,Passagers ListePassager) ;
```

Cette fonction vérifie si un ID existe dans la liste des passagers. Elle retourne 1 si l'ID existe, 0 dans le cas contraire.

```
16. Passagers AnnulerReservation(Passagers ListePassagers,int IDAnulation);
```

Annuler une réservation tend à rechercher un ID dans une liste, extraire le passager de la liste et s'assurer que le chainage du parent avec l'enfant est assuré correctement. Un pointeur vers l'élément courant et l'élément parent doit être maintenu tout au long du processus pour faire apparaitre une ligne de code similaire à celle-ci :

```
Parent->suivant=Courant->suivant ;
```

```
17. Passagers AjouterPassagerDansLaListe (Passager PassagerAAjouter, Passagers ListeDesPassagers);
```

Après que la réservation soit faite, l'embarquement effectif du passager doit être réalisé en ajoutant le passager dans la liste des passagers du train. Cette fonction retourne une liste de passager et ajoute la structure Passager (passagerAAjouter) dans la liste.

```
18. AfficherPassagers(Passagers ListePassagers) ;
```

Affiche un passager en énumérant ses propriétés. Identifiant, Destination ...

```
19. Passager SupprimerPassagerListeAttenteEtLeRetourner(Passagers * ListePassagers,int iDPassager) ;
```

Pour supprimer un passager de la liste des passagers en attente et le mettre dans la liste des passagers dans le train, cette fonction est extrêmement utile. Elle supprime un passager de la liste d'attente et le retourne pour que par la suite il soit ajouté dans la liste des passagers du train.

Fonctions associé aux Trains

```
20. Train CreerTrain(int IDTrain, int PlacesSupportees, chaine Destination);
```

Fonctions de création d'un train en attribuant les caractéristiques du train aux structures.

```
21. Train ViderTrain(Train T);
```

Lorsqu'un train rentre d'un voyage, il doit être vidé des différents passagers (liste des passagers dans le train) ainsi que le nombre de places prises.

```
22. int AjouterTrain(Train * Trains, int Id, int Capacitee, chaine
Destination) ;
```

Similairement aux fonctions AjouterParkingRail, la fonction AjouterTrain crée et ajoute des trains dans une structure de tableau faite spécialement pour stocker les différentes instances de train. Ceci permet de les parcourir facilement et de dynamiser leurs traitements d'affichage entre autre.

Comme pour les structures des quais, les trains peuvent être déclarés de manière individuelle comme suit :

```
Train1= CreerTrain(0,20,"Tokyo");

Train2= CreerTrain(1,20,"Berlin");

Train3= CreerTrain(2,10,"Nairobi");

Train4= CreerTrain(3,35,"Berlin");
```

Néanmoins, pour une meilleure gestion des structures de données, un tableau groupant l'ensemble des déclarations est préférable pour cela, une fonction spécifique a été mise en place :

```
int AjouterTrain(Train * Trains, int Id, int Capacitee, chaine Destination) ;
```

```
23. void AfficherTrains(Train * Trains, int nbrTrains) ;
```

Fonction qui affiche les états des trains. Par convention, le nombre de Km restant pour arriver nous indique l'état des trains comme suit :

```
Train.kilometreRestant==-1// le train est en attente d'une place qui se
libère dans le quai (en attente)

Train.kilometreRestant==0 // le train est dans le quai et prêt à faire
embarquer les passagers
```

```
Train.kilometreRestant>0 // le train est en marche vers une destination particulière
```

Les états des trains sont affichés ainsi que leurs taux de remplissage et le quai sur lequel ils sont stationnés.

```
24. int SiTrainRemplis(Train TrainCourant) ;
```

Retourne *1* si le train est rempli en vérifiant la variable `Train.PlacesPrises` dans la structure du Train. 0 dans le cas contraire.

NB : La variable liste des passagers dans la structure train n'est pas un indicateur du remplissage du train, elle ne fait que lister les passagers dans le train.

```
25. void MiseAJourTableauDesTrain(Train TrainPourDepart,Train * Trains,int NumTrain) ;
```

Avant qu'un train ne sorte du quai et n'entame son voyage, ses propriétés doivent être reportées sur la structure des train (tableau des trains, vus précédemment). Des informations comme la distance restante, le nombre de passagers dans le train, etc, doivent être reporté. Cette fonction a pour but d'effectuer ce genre de mises à jour.

```
26. Passagers Embarquement (ParkingRail * Quaies, int NumQuaie, Passagers PassagerPretPourEmbarquement, chaine Destinations[MapSize][2], int NumDestination, Train * Trains, int NumTrain) ;
```

Cette fonction étant la plus complexe du programme parcours les passagers du en attente et les attributs dans la liste des passagers du train. Bien évidemment, le train doit avoir la même destination que le passager. Une fois mis dans la liste des passagers du train, le passager en question ne doit plus appartenir aux passagers en attente.

Les passagers qui demandent des destinations qui n'existent pas dans notre gare (aucun train ne mène à cette destination) doit être directement rejeté.

```
29. void TrainEnMarche(ParkingRail * Quaies,int NumQuai,  Train * Trains, int NumTrain) ;
```

Cette fonction simule le train lorsqu'il est en marche, la vitesse du train n'étant pas implémenté, tous les trains roulent à une vitesse de 5Km/h. Lorsque les trains arrivent à 5Km de la gare, ils doivent choisir quel quai s'arrêter pour déposer les passagers, dans le cas où aucun quai n'est disponible, ils se mettent en attente.

Fonctions associé aux Destinations

```
30. int InitialiserLesDestinationsDisponibles(chaine Destinations[MapSize][2]) ;
```

Une matrice est implémentée pour représenter les différentes villes destinatrices ainsi que leurs distances. Plusieurs implémentations sont possibles, nous avons choisis de mettre en place une matrice représentée de la sorte :

Nom Ville	Distance
Berlin	1500 Km
Denver	2000 Km
Ouergla	50 Km
…	…
Rio	900Km

Fonctions main

La fonction main effectue les déclarations et simule l'ensemble du fonctionnement de la gare heure après heure. Elle est donnée par le code suivant :

```
int InitialiserLesDestinationsDisponibles


int main()
{


//Déclaration des structures de données
    Train Trains[MaxTrain];// tableau de train
    ParkingRail Quais[MaxParkingRails]; // tableau de quais
    int NumTrain=0;//Nombre de train qu'il y'a dans la gare
    int NumQuai=0;// nombre de quais / Parking rail qu'il y'a dans la gare


    chaine Destinations[MapSize][2]; // matrice des destinations


    int NumDestinations= InitialiserLesDestinationsDisponibles(Destinations);


    // Ajout des trains
    NumTrain=AjouterTrain(Trains,NumTrain,15,"Tokyo");
    NumTrain=AjouterTrain(Trains,NumTrain,10,"Helsinki");
    NumTrain=AjouterTrain(Trains,NumTrain,10,"Helsinki");
    NumTrain=AjouterTrain(Trains,NumTrain,10,"Tokyo");
    NumTrain=AjouterTrain(Trains,NumTrain,10,"Berlin");
    NumTrain=AjouterTrain(Trains,NumTrain,8,"Nairobi");
    NumTrain=AjouterTrain(Trains,NumTrain,10,"Denver");
    NumTrain=AjouterTrain(Trains,NumTrain,10,"Nairobi");
    NumTrain=AjouterTrain(Trains,NumTrain,10,"Berlin");
```

```
    NumTrain=AjouterTrain(Trains,NumTrain,10,"Ouergla");




    // Ajouter les Quais
    NumQuai=AjouterQuais(Quais,NumQuai,"Q1",2,Trains, NumTrain);
    NumQuai=AjouterQuais(Quais,NumQuai,"El Moudjahid",1,Trains, NumTrain);
    NumQuai=AjouterQuais(Quais,NumQuai,"Roberto Carlos",1,Trains,NumTrain);


    Passagers ListeDesPassagerEnAttente=NULL; // déclarer la liste des
passagers en attente




    //Commencer la simulation
    int NbrHeur=0;
    while(1){// nous pouvons remplacer le 1 par un nombre d'itération bien
précis !
        AfficherParkingRail(Quais,NumQuai); // afficher l'état des quais pour
voir ou se trouve les train.
        NbrHeur++;
        printf("************Heure %d***********\n\n",NbrHeur);
        srand(time(NULL));// appeler la fonction random pour générer des
nombres aléatoire de réservations et d'annulations


        int NombreAleatoireDeReservations=rand()%10+5;
        int NombreAleatoireDAnulations=rand()%5;
// nous devons nous assurer que le nombres de reservations soit strictement
superieurs aux nombre d'annulations
        printf("Nombre de reservations :
%d\n",NombreAleatoireDeReservations);
        printf("Nombre d'annulations : %d\n",NombreAleatoireDAnulations);


        for (int i=0;i<NombreAleatoireDeReservations;i++){
            int ChoisirUneDestinationAleatoire=rand()%(NumDestinations+1);

ListeDesPassagerEnAttente=ReservationPlace(ListeDesPassagerEnAttente,
Trains,NumTrain,Destinations[ChoisirUneDestinationAleatoire][0]);
```

```
// réserver des places et ajouter des passagers dans la liste d'attente, a ce stade, aucun passager n'est dans le train, rien que la variable « place prises » est incrémenter pour dire que les places sont reservées

        }



        for (int i=0;i<NombreAleatoireDAnulations;i++){

            int ChoisirUnIDPourAnulationAleatoire=rand()%TailleListe(ListeDesPassagerEnAttente);


ListeDesPassagerEnAttente=AnnulerReservation(ListeDesPassagerEnAttente,ChoisirUnIDPourAnulationAleatoire);

// briser la liste chainée « ListeDesPassagersEnAttente » et enlever les passagers qui souhaites annuler leurs reservations.

        }


        printf("\n Nombre de passagers en attente %d\n",TailleListe(ListeDesPassagerEnAttente));

        AfficherPassagers(ListeDesPassagerEnAttente);

// affichier l'état des passagers en attente avant embarquement et après que les réservations et annulations soient faites

        ListeDesPassagerEnAttente=Embarquement(Quais, NumQuai, ListeDesPassagerEnAttente, Destinations,NumDestinations, Trains,NumTrain );

// Embarquer les passager !

        AfficherTrains(Trains,NumTrain);

//afficher l'état des trains pour voir si ils ont pris des passagers avec eux.



        TrainEnMarche(Quais,NumQuai, Trains, NumTrain);

// lancer les trains qui sont les premiers dans les quais

        printf("\n Nombre de passagers en attente après embarquement %d\n",TailleListe(ListeDesPassagerEnAttente));



        AfficherPassagers(ListeDesPassagerEnAttente);

// afficher l'état des passagers en attente après embarquement, dans le cas ou votre programme marche correctement, il ne devrai y avoir aucun passager qui demande une destination qui vient d'être proposé par un train sur le départ sauf si le train est remplis a 100%.

        UpdateQuais( Quais, NumQuai, Trains, NumTrain );

Mettre a jour les quais en récupérant les train en attente et en avançant les tarins dans la pile pour proposer de nouvelle destinations.
```

```
        system("pause");// arrêter le programme pour avoir le temps de lire
les affichages

// vous pouvez ajouter un « clearscreen » ici mais ceci vous empêchera
d'avoir une vue sur l'historique



    }

    //*/


    return 0;

}
```

Ce que vous devez faire

1- Implémenter les fonctions en jaune
2- Lancer l'exécution du programme sans erreurs
3- Implémenter la vitesse des trains
4- Implémenter la possibilité de changement de destination des trains lorsqu'ils rentrent de leurs voyages
5- Implémenter la possibilité d'ajouter des trains avec de nouvelle destination au cours de l'exécution du programme
6- Implémenter la possibilité d'enlever des trains qui partent tout le temps avec peu de passagers
7- Implémenter la possibilité d'ajouter des quais cours de l'exécution du programme
8- Ajouter le cout du billet pour chaque passager et l'inclure dans une cagnotte de la gare
9- Modéliser le système de la gare suivants les variables d'optimisation et les contraintes posées tel que :

Contrainte	**Variable d'optimisation**
Nombres de passagers en attente ne doit pas dépasser une certaine valeur	Cagnotte de la gare doit augmenter
Ajouter un train engendre un cout	
Ajout des quais engendre un cout	

10- Transformer votre programme de « gestion de gare » en un jeu où le joueur doit savoir ajouter les trains ou les enlever pour ne pas avoir un grand nombre de passagers en attente. L'ajout de train ou de quai se fait avec de l'argent (cagnotte) reçu par les voyages fait par les passagers. De manière formelle il vous est demandé de trouver la stabilité du système que vous avez mis en place.

The end !
Bon courage, et n'oubliez pas de vous amuser ! ☺