using System;

using System.IO;

using System.Collections.Generic;

using System.Linq;

namespace Probleme

{

// ATTENTION : Modifier les chemins des fichiers txt se trouvant dans le main

class Program

{

#region FICHIER

public static int Comptage(string nomFich)

{

int lineCount = 0;

using (var reader = File.OpenText(nomFich))

{

while (reader.ReadLine() != null)

{

lineCount++;

}

return lineCount;

}

/\*

int lineCount = File.ReadAllLines(nomFich).Length;

return lineCount;

\*/

} // Permet de compter le nombre de ligne d'un fichier

public static void TestLecture(string nomFichier)

{

StreamReader fichierALire = new StreamReader(nomFichier);

string ligne = "";

while (fichierALire.Peek() > -1)

{

ligne = fichierALire.ReadLine();

Console.WriteLine(ligne);

}

fichierALire.Close();

} // Peut changer en fonction du chemin // Permet de lire un fichier

#region Transforme fichier txt en tableau de la classe désigné

// Explication en commentaire dans la première fonction LectureFichierClients (les 3 autres sont identiques, sauf que la classe qu'on utilise change)

public static Client[] LectureFichierClients(string nomFich, bool ok)

{

int lines = Comptage(nomFich); // Compte le nombre de lignes du fichier

StreamReader fichierALire = new StreamReader(nomFich); // Lecture du fichier dont le chemin d'accès est nomFich

Client[] contacts = new Client[lines]; // Initialise le tableau de Clients

string ligne = "";

int i = 0;

char[] sep = new char[1] { ';' }; // Définit le séparateur

string[] datas = new string[8]; // Nombre de string définissant un Client

while (fichierALire.Peek() > -1) // Permet de lire tout le fichier

{

ligne = fichierALire.ReadLine(); // Lit la ligne du fichier

if (ok == true) // true = si on veut afficher || false = si on veut pas afficher les lignes du fichiers

{

Console.WriteLine(ligne);

}

datas = ligne.Split(sep); // On sépare les données de la ligne en fonction du séparateur ici ";"

contacts[i] = new Client(datas[0], datas[1], datas[2], datas[3], datas[4], datas[5], datas[6], datas[7], datas[8]); // crée client[i] qui désigne la i-1 ème ligne

i++; // incrément pour la boucle while

}

fichierALire.Close(); // On ferme le fichier qu'on lisait

return contacts; // Retourne le tableau de client

}

public static Commande[] LectureFichierCommandes(string nomFich, bool ok)

{

StreamReader fichierALire = new StreamReader(nomFich);

int lines = Comptage(nomFich);

Commande[] commandes = new Commande[lines];

string ligne = "";

int i = 0;

char[] sep = new char[1] { ';' };

string[] datas = new string[8];

while (fichierALire.Peek() > -1)

{

ligne = fichierALire.ReadLine();

if (ok == true)

{

Console.WriteLine(ligne);

}

datas = ligne.Split(sep);

commandes[i] = new Commande(datas[0], datas[1], datas[2], datas[3], datas[4], datas[5], datas[6], datas[7], datas[8]);

i++;

}

fichierALire.Close();

return commandes;

}

public static Chauffeur[] LectureFichierChauffeurs(string nomFich, bool ok)

{

int lines = Comptage(nomFich);

StreamReader fichierALire = new StreamReader(nomFich);

Chauffeur[] contacts = new Chauffeur[lines];

string ligne = "";

int i = 0;

char[] sep = new char[1] { ';' };

string[] datas = new string[13];

while (fichierALire.Peek() > -1)

{

ligne = fichierALire.ReadLine();

if (ok == true)

{

Console.WriteLine(ligne);

}

datas = ligne.Split(sep);

contacts[i] = new Chauffeur(datas[0], datas[1], datas[2], datas[3], datas[4], datas[5], datas[6], datas[7], datas[8], datas[9], datas[10], datas[11], datas[12], datas[13]);

i++;

}

fichierALire.Close();

return contacts;

}

public static Salarie[] LectureFichierSalaries(string nomFich, bool ok)

{

StreamReader fichierALire = new StreamReader(nomFich);

int lines = Comptage(nomFich);

Salarie[] contacts = new Salarie[lines];

string ligne = "";

int i = 0;

char[] sep = new char[1] { ';' };

string[] datas = new string[10];

while (fichierALire.Peek() > -1)

{

ligne = fichierALire.ReadLine();

if (ok == true)

{

Console.WriteLine(ligne);

}

datas = ligne.Split(sep);

contacts[i] = new Salarie(datas[0], datas[1], datas[2], datas[3], datas[4], datas[5], datas[6], datas[7], datas[8], datas[9], datas[10]);

i++;

}

fichierALire.Close();

return contacts;

}

#endregion

#endregion

#region UTILITAIRE DATE

#region Définition année

public static int Annee() // Initialise l'année

{

int annee = 0;

do

{

Console.WriteLine("Saississez l'année : ");

annee = Convert.ToInt32(Console.ReadLine());

} while (annee < 1900 || annee > DateTime.Now.Year + 20);

return annee;

}

public static bool Bissextile(int annee) // fonction bissextile étudié plusieurs fois

{

if (((annee % 4 == 0) && (annee % 100 != 0)) || (annee % 400 == 0))

{

return true;

}

return false;

}

#endregion

#region Définition mois

public static int Mois() // Initialise le mois

{

int mois = 0;

do

{

Console.WriteLine("Saississez le mois : ");

mois = Convert.ToInt32(Console.ReadLine());

} while (mois < 1 || mois > 12);

return mois;

}

#endregion

#region Définiton jour

public static int Jour(int mois, int annee) // parametre avec mois et année car jour varie en fonction

{

int jour = 0;

if (mois == 2)

{

if (Bissextile(annee) == true)

{

do

{

Console.WriteLine("Saississez le jour : ");

jour = Convert.ToInt32(Console.ReadLine());

} while (jour < 1 || jour > 29);

return jour;

}

else

{

do

{

Console.WriteLine("Saississez le jour : ");

jour = Convert.ToInt32(Console.ReadLine());

} while (jour < 1 || jour > 29);

return jour;

}

}

if ((mois == 4) || (mois == 6) || (mois == 9) || (mois == 11))

{

do

{

Console.WriteLine("Saississez le jour : ");

jour = Convert.ToInt32(Console.ReadLine());

} while (jour < 1 || jour > 30);

return jour;

}

else

{

do

{

Console.WriteLine("Saississez le jour : ");

jour = Convert.ToInt32(Console.ReadLine());

} while (jour < 1 || jour > 31);

return jour;

}

}

#endregion

#region Split Date

public static int[] SplitDate(string date)

{

string[] words = date.Split('/'); // On sépare les "/"

int[] jma = new int[words.Length];

for (int i = 0; i < words.Length; i++)

{

jma[i] = Convert.ToInt32(words[i]);

}

return jma; // Obtien tableau avec jour, mois, année séparés

}

#endregion

#endregion

#region AUTRES FONCTIONS

#region SAISIE NOM

public static string SaisirNom() // Permet de saisir un nom

{

Console.WriteLine("\n\nSaisir nom du collègue : ");

string nom = Console.ReadLine();

return nom;

}

#endregion

#endregion

#region CLIENT

#region SAUVEGARDE FICHIER CLIENT

static void SauvegardeClients(Client[] clients, string nomFich) // Permet de sauvegarder les modifications réalisés dans le fichier txt

{

StreamWriter fichEc = new StreamWriter(nomFich); // on réecrit le fichier

for (int i = 0; i < clients.Length; i++) // réécrit fichier 1 à 1

{

if (clients[i] != null) // Regrade si clients est null ou non

{

fichEc.WriteLine(clients[i].ToString()); // Ecriture du clients avec le séparateur nécéssaire pour la lecture du fichier txt, qui est ";"

}

}

fichEc.Close(); // ferme le fichier

}

#endregion

#region CREATION CLIENT

public static Client CreationClient() // Permet de créer un client en tapant ces attributs

{

#region Partie création d'un client

// Création des attributs Client un par un

Console.Write("\nCREATION DU CLIENT\n");

Console.WriteLine("\nNuméro de Sécurité Sociale :");

string numeroSS = Console.ReadLine();

Console.WriteLine("\nNom :");

string nom = Console.ReadLine();

Console.WriteLine("\nPrenom :");

string prenom = Console.ReadLine();

Console.WriteLine("\nDate de naissance :");

int annee = Annee();

int mois = Mois();

int jour = Jour(mois, annee);

string naissance = jour + "/" + mois + "/" + annee;

Console.WriteLine("\nAdresse :");

string adresse = Console.ReadLine();

Console.WriteLine("\nMail :");

string mail = Console.ReadLine();

Console.WriteLine("\nNuméro de téléphone :");

string num = Console.ReadLine();

Console.WriteLine("\nSexe : (M si Homme / F si Femme)");

string sex = Console.ReadLine();

#endregion

Client newClient = new Client(numeroSS, nom, prenom, naissance, adresse, mail, num, sex, "0"); // On initialise le nouveau client créé // initialise le montant à 0, car le client n'a pas passé de commande

return newClient; // Retourne le client créé

}

#endregion

#region AJOUT CLIENT

static void AjoutClient(Client[] clients, string nomFich) // Permet d'ajouter un client au fichier txt qui lui est lié

{

int nb = clients.Length; // longueur du tableau

int res = 1;

Client newContact = CreationClient(); // On créé un nouveau client à ajouter au fichier txt

for (int i = 0; i < nb; i++) // On parcourt le fichier txt 1 par 1

{

if (clients[i] != null) // Vérifie que client est bien différent de null

{

if ((clients[i].Nom.ToUpper() == newContact.Nom.ToUpper()) && (clients[i].Prenom.ToUpper() == newContact.Prenom.ToUpper())) // On checrhe à savoir si le client existe déjà

{

Console.WriteLine("\nClient déjà existant : ERROR\n");

res = 0; // permet d'écrire ou non si le client existe, ici permet de savoir que le client existe

break;

}

}

}

// Si le client existe pas, ce qui se vérifie par : res = 1

if (res == 1)

{

StreamWriter fichEc = new StreamWriter(nomFich); // On écrit le fichoier pour le modifier

if (nb == 0) // Si le fichier est vide

{

fichEc.WriteLine(newContact.ToString()); // On écrit le nouveau client créé dans le fichier

fichEc.Close(); // On ferme le fichier pour stopper l'écriture

}

else

{

Client[] clients2 = new Client[nb]; // On créé un tableau de longueur des nb clients du fichier

for (int j = 0; j < nb; j++)

{

clients2[j] = clients[j]; // On Initialise les valeurs du tableau un par un

fichEc.WriteLine(clients2[j].ToString()); // On écrit ces valeurs dans le fichier

}

clients2[nb - 1] = newContact; // On rajoute à la dernière case du tableau le nouveau client

fichEc.WriteLine(clients2[nb - 1].ToString()); // On l'écrit dans le fichier

fichEc.Close(); // On ferme le fichier pour stopper l'écriture

}

}

SauvegardeClients(LectureFichierClients(nomFich, false), nomFich); // On sauvegrade les modifications

}

#region Ajout d'un client avec le client défini en paramètre

static void AjoutAvecClientConnu(Client[] clients, string nomFich, Client client) // Réalise le même travail que la fonction AjoutClient, sauf qu'ici le client à ajouter est déja connu, et est placé en paramètre

{

int n = clients.Length;

Client newContact = client; // On initialise le nouveaux clients comme le client placé en paramètre

int res = 1;

for (int i = 0; i < n; i++)

{

if (clients[i] != null)

{

if ((clients[i].Nom.ToUpper() == newContact.Nom.ToUpper()) && (clients[i].Prenom.ToUpper() == newContact.Prenom.ToUpper()))

{

Console.WriteLine("\nClient déjà existant : ERROR\n");

res = 0;

break;

}

}

}

if (res == 1)

{

StreamWriter fichEc = new StreamWriter(nomFich);

if (n == 0)

{

fichEc.WriteLine(newContact.ToString());

fichEc.Close();

}

else

{

Client[] clients2 = new Client[n];

for (int j = 0; j < n; j++)

{

clients2[j] = clients[j];

fichEc.WriteLine(clients2[j].ToString());

}

clients2[n - 1] = newContact;

fichEc.WriteLine(clients2[n - 1].ToString());

fichEc.Close();

}

}

SauvegardeClients(LectureFichierClients(nomFich, false), nomFich);

}

#endregion

#endregion

#region RECHERCHE CLIENT

public static Client RechercheClientRetourner(Client[] clients) // Ce code permet de rechercher un client au sein du fichier texte associé aux clients

{

// On initialise le Nom et Prénom du client qu'on souhaite rechercher pour le trouver au sein du fichier

Console.WriteLine("\nDonnez le nom du client : ");

string nom = Console.ReadLine();

Console.WriteLine("\nDonnez le prénom du client : ");

string prenom = Console.ReadLine();

for (int i = 0; i < clients.Length; i++) // On parcourt le fichier un par un

{

if (clients[i] != null) // vérifie que clients[i] est bien différent de null

{

if ((clients[i].Nom.ToUpper() == nom.ToUpper()) && (clients[i].Prenom.ToUpper() == prenom.ToUpper())) // On compare le nom et prénom de chaque client, voir si ils correspondent au nom et prénom donnés

{

Console.Write("\nLe client " + nom.ToUpper() + " " + prenom.ToUpper() + " a été trouvé.\n");

Console.WriteLine(clients[i].AfficheClient()); // On affiche le client si il est trouvé de manière propre

return clients[i]; // On retourne le client trouvé

}

}

}

Console.Write("\nLe client " + nom.ToUpper() + " " + prenom.ToUpper() + " n'est pas dans le fichier client.\n");

return null; // On retourne null si aucun client ne correspond

}

#endregion

#region SUPPRIMER CLIENT

static void SupprimerClient(Client[] clients, string nomFich) // Permet de supprimer un client du fichier texte

{

// On initialise le Nom et Prénom du client qu'on souhaite supprimer pour le trouver au sein du fichier

Console.WriteLine("\nDonnez un client à supprimer : ");

Console.WriteLine("\nNom :");

string nom = Convert.ToString(Console.ReadLine());

Console.WriteLine("\nPrénom :");

string prenom = Convert.ToString(Console.ReadLine());

for (int i = 0; i <= clients.Length - 1; i++) // On parcourt le fichier un par un

{

if (clients[i] != null) // vérifie que clients[i] est bien différent de null

{

if ((clients[i].Nom.ToUpper() == nom.ToUpper()) && (clients[i].Prenom.ToUpper() == prenom.ToUpper())) // On compare le nom et prénom de chaque client, voir si ils correspondent au nom et prénom donnés

{

Console.WriteLine("\nLe client " + nom + " " + prenom + " a été supprimé.\n");

clients[i] = null; // On initialise le client à supprimer à null

}

}

}

SauvegardeClients(clients, nomFich); // On sauvegarde la modification au sein du fichier texte

}

#endregion

#region MODIFIER CLIENT

static void ModifierClient(Client[] clients, string nomFich) // Ce code permet de modifier un client au sein du fichier

{

// On initialise le Nom et Prénom du client qu'on souhaite rechercher pour le trouver au sein du fichier

Console.WriteLine("\nDonnez un client à modifier : ");

Console.WriteLine("\nNom :");

string nom = Convert.ToString(Console.ReadLine());

Console.WriteLine("\nPrénom :");

string prenom = Convert.ToString(Console.ReadLine());

for (int i = 0; i <= clients.Length - 1; i++) // On parcourt le fichier un par un

{

if (clients[i] != null) // vérifie que clients[i] est bien différent de null

{

if ((clients[i].Nom.ToUpper() == nom.ToUpper()) && (clients[i].Prenom.ToUpper() == prenom.ToUpper())) // On compare le nom et prénom de chaque client, voir si ils correspondent au nom et prénom donnés

{

clients[i] = CreationClient(); // On modifie le clients[i] en créant un nouveau client directement grâce à la fonction CreationClient

Console.WriteLine("\nLe client " + nom.ToUpper() + " " + prenom.ToUpper() + " a été modifié.\n");

Console.WriteLine(clients[i].AfficheClient()); // On affiche le client modifié de manière propre pour voir les modifications

}

}

}

SauvegardeClients(clients, nomFich); // On sauvegrade les modifications

}

#endregion

#region TRIER CLIENT

#region Transformation de liste en tableau (et inversement)

public static List<Client> TransformerList(Client[] clients) // Collections génériques

{

List<Client> listeClient = new List<Client>(); // Initialise une liste de client

int n = clients.Length; // longueur du tableau clients

for (int i = 0; i < n; i++) // Parcours le tableau

{

listeClient.Add(clients[i]); // Ajoute à la liste le clients[i] du tableau

}

return listeClient; // On retourne la liste créée

}

public static Client[] TransformerTab(List<Client> clients) // Collections génériques

{

int n = clients.Count; // longueur de la liste de clients

Client[] tabClient = new Client[n]; // Initialise un tableau de client

for (int i = 0; i < n; i++) // Parcours la liste

{

tabClient[i] = clients[i]; // Ajoute au tableau le clients[i] de la liste

}

return tabClient; // On retourne le tableau créé

}

#endregion

#region ENSEMBLE DES TRIES

// On sauvegarde le fichier trié à chaque tri afin de pouvoir réaliser séparément ou successivement les tries comme demandé

public static void TrierOrdreAlpha(List<Client> mylist, string nomFich) //Tri avec délégation

{

Console.WriteLine("\nTri par ORDRE ALPHABETIQUE \n");

mylist.Sort((x, y) => x.Prenom.CompareTo(y.Prenom)); // Tri prénom

mylist.Sort((x, y) => x.Nom.CompareTo(y.Nom)); // Tri nom

mylist.ForEach(x => Console.WriteLine(x.ToString())); // Affiche les clients par ordre alphabétique

SauvegardeClients(TransformerTab(mylist), nomFich); // Sauvegarde les clients triés dans le fichier txt

}

public static void TrierVille(List<Client> mylist, string nomFich) //Tri avec délégation

{

Console.WriteLine("\nTri par VILLE \n");

mylist.Sort((x, y) => x.Adresse.CompareTo(y.Adresse)); // Tri ville

mylist.ForEach(x => Console.WriteLine(x.ToString())); // Affiche les clients par Localisation

SauvegardeClients(TransformerTab(mylist), nomFich); // Sauvegarde les clients triés dans le fichier txt

}

public static void TrierMontant(List<Client> mylist, string nomFich) //Tri avec interface

{

Console.WriteLine("\nTri selon le MONTANT\n");

mylist.Sort(mylist[0].Compare); // Compare montant

mylist.ForEach(x => Console.WriteLine(x.ToString())); // Affiche les clients par Localisation

SauvegardeClients(TransformerTab(mylist), nomFich); // Sauvegarde les clients triés dans le fichier txt

}

#endregion

#region MODULE TRI

static void TriClients(string nomFichClients) // Fonction avec SWITCH permettant de choisir le tri voulu

{

string n;

int s = 0;

while (s < 1)

{

Console.WriteLine("\n" +

"\n ~~~~~~~~~~~~~~~~~~~~ Module TRI ~~~~~~~~~~~~~~~~~~~~" +

"\n1. Tri par ordre alphabétique " +

"\n2. Tri par ville " +

"\n3. Tri selon le Montant " +

"\n4. Affichage " +

"\n5. QUITTER ");

do

{

Console.WriteLine("\nQue souhaitez-vous faire ? (Donnez un nombre entre 1 et 5)");

n = Console.ReadLine();

} while (Convert.ToInt32(n) < 1 || Convert.ToInt32(n) > 5);

switch (n)

{

case "1": { TrierOrdreAlpha(TransformerList(LectureFichierClients(nomFichClients, false)), nomFichClients); break; }

case "2": { TrierVille(TransformerList(LectureFichierClients(nomFichClients, false)), nomFichClients); break; }

case "3": { TrierMontant(TransformerList(LectureFichierClients(nomFichClients, false)), nomFichClients); break; }

case "4": { LectureFichierClients(nomFichClients, true); break; }

case "5": { Console.WriteLine("\nRetour au module CLIENT\n"); s++; break; }

}

}

}

#endregion

#endregion

#endregion

#region MODULE CLIENT

static void ModClient(string nomFichClients) // Fonction avec SWITCH permettant de choisir l'action du module client qu'on veut

{

string n;

int s = 0;

while (s < 1)

{

Console.WriteLine("\n" +

"\n ~~~~~~~~~~~~~~~~~~~~ Module CLIENT ~~~~~~~~~~~~~~~~~~~~" +

"\n1. Entrer un nouveau client " +

"\n2. Rechercher un client " +

"\n3. Supprimer un client " +

"\n4. Modifier un client " +

"\n5. Affichage " +

"\n6. Trier " +

"\n7. QUITTER ");

do

{

Console.WriteLine("\nQue souhaitez-vous faire ? (Donnez un nombre entre 1 et 7)");

n = Console.ReadLine();

} while (Convert.ToInt32(n) < 1 || Convert.ToInt32(n) > 7);

switch (n)

{

case "1": { AjoutClient(LectureFichierClients(nomFichClients, false), nomFichClients); break; }

case "2": { RechercheClientRetourner(LectureFichierClients(nomFichClients, false)); break; }

case "3": { SupprimerClient(LectureFichierClients(nomFichClients, false), nomFichClients); break; }

case "4": { ModifierClient(LectureFichierClients(nomFichClients, false), nomFichClients); break; }

case "5": { LectureFichierClients(nomFichClients, true); break; }

case "6": { TriClients(nomFichClients); break; }

case "7": { Console.WriteLine("\nRetour au MENU\n"); s++; break; }

}

}

}

#endregion

#region ENTREPRISE

#region ARBRE ENTREPRISE

static Arbre\_N\_aire ArbreEntreprise() // Initialise l'organigramme donné dans l'énoncé

{

#region CREATION ENTREPRISE

// Initialise les salariés de base de l'organigramme

Salarie CEO = new Salarie("0001", "Dupond", "Franck", "03/10/1956", "PARIS", "Dupond@", "0600000001", "M", "01/01/1990", "Directeur Général", "100000");

Salarie DirectCom = new Salarie("0010", "Fiesta", "Ford", "14/04/1952", "PARIS", "Fiesta@", "0600000002", "F", "01/01/1990", "Directrice Commerciale", "80000");

Salarie Com1 = new Salarie("0011", "Forge", "Henry", "02/04/1954", "PARIS", "Forge@", "0600000003", "M", "01/01/1993", "Commercial", "50000");

Salarie Com2 = new Salarie("0012", "Fermi", "La", "08/10/1960", "PARIS", "Fermi@", "0600000004", "F", "01/01/1995", "Commerciale", "45000");

Salarie DirectOp = new Salarie("0020", "Fetard", "Teuf", "05/06/1961", "PARIS", "Fetard@", "0600000005", "M", "01/01/1990", "Directeur des Opérations", "80000");

Salarie ChefEq1 = new Salarie("0210", "Royal", "Louis", "18/10/1970", "PARIS", "Royal@", "0600000006", "M", "01/01/1994", "Chef d'Equipe", "45000");

Salarie Chauff11 = new Salarie("0211", "Romu", "Ald", "25/11/1961", "PARIS", "Romu@", "0600000007", "M", "01/01/1998", "Chauffeur", "45000");

Salarie Chauff12 = new Salarie("0212", "Romi", "Scheinder", "17/06/1978", "PARIS", "Romi@", "0600000008", "F", "01/01/2016", "Chauffeur", "45000");

Salarie Chauff13 = new Salarie("0213", "Roma", "Italia", "05/02/1975", "PARIS", "Roma@", "0600000009", "M", "01/01/2020", "Chauffeur", "45000");

Salarie ChefEq2 = new Salarie("0220", "Prince", "Henriette", "01/12/1974", "PARIS", "Prince@", "0600000010", "F", "01/01/1998", "Chef d'Equipe", "45000");

Salarie Chauff21 = new Salarie("0221", "Rome", "Paris", "05/07/1980", "PARIS", "Rome@", "0600000011", "F", "01/01/2012", "Chauffeur", "45000");

Salarie Chauff22 = new Salarie("0222", "Rimou", "Marc", "09/06/1981", "PARIS", "Rimou@", "0600000012", "F", "01/01/2015", "Chauffeur", "45000");

Salarie DirectRH = new Salarie("0030", "Joyeuse", "Sadless", "05/06/1961", "PARIS", "Joyeuse@", "0600000013", "F", "01/01/1990", "Directrice RH", "80000");

Salarie RH1 = new Salarie("0031", "Couleur", "Blanche", "04/10/1984", "PARIS", "Couleur@", "0600000014", "F", "01/01/1999", "Formation", "45000");

Salarie RH2 = new Salarie("0032", "Toutlemonde", "Dorice", "24/06/1986", "PARIS", "ToutLeMonde@", "0600000015", "F", "01/01/2014", "Contrats", "45000");

Salarie DirectFinance = new Salarie("0040", "GripSous", "Mone", "07/06/1975", "PARIS", "Gripsous@", "0600000016", "M", "01/01/1990", "Directeur Financier", "80000");

Salarie DirectCompt = new Salarie("0410", "Piscou", "Can", "28/02/1976", "PARIS", "Piscou@", "0600000017", "M", "01/01/1990", "Direction Comptable", "80000");

Salarie Compt1 = new Salarie("0411", "Fournier", "Four", "30/07/1977", "PARIS", "Fournier@", "0600000018", "F", "01/01/1992", "Comptable", "55000");

Salarie Compt2 = new Salarie("0412", "Gautier", "Yves", "05/03/1978", "PARIS", "Gautier@", "0600000019", "F", "01/01/1999", "Comptable", "50000");

Salarie ControlGest = new Salarie("0420", "GrosSous", "Money", "05/06/1979", "PARIS", "GrosSous@", "0600000020", "M", "01/01/1992", "Controleur de gestion", "65000");

// Création des noeuds associés à chaques salariés

Noeud\_N DirectComN = new Noeud\_N(DirectCom);

Noeud\_N Com1N = new Noeud\_N(Com1);

Noeud\_N Com2N = new Noeud\_N(Com2);

Noeud\_N DirectOpN = new Noeud\_N(DirectOp);

Noeud\_N ChefEq1N = new Noeud\_N(ChefEq1);

Noeud\_N Chauff11N = new Noeud\_N(Chauff11);

Noeud\_N Chauff12N = new Noeud\_N(Chauff12);

Noeud\_N Chauff13N = new Noeud\_N(Chauff13);

Noeud\_N ChefEq2N = new Noeud\_N(ChefEq2);

Noeud\_N Chauff21N = new Noeud\_N(Chauff21);

Noeud\_N Chauff22N = new Noeud\_N(Chauff22);

Noeud\_N DirectRHN = new Noeud\_N(DirectRH);

Noeud\_N RH1N = new Noeud\_N(RH1);

Noeud\_N RH2N = new Noeud\_N(RH2);

Noeud\_N DirectFinanceN = new Noeud\_N(DirectFinance);

Noeud\_N DirectComptN = new Noeud\_N(DirectCompt);

Noeud\_N Compt1N = new Noeud\_N(Compt1);

Noeud\_N Compt2N = new Noeud\_N(Compt2);

Noeud\_N ControlGestN = new Noeud\_N(ControlGest);

#endregion

#region ARBRE ENTREPRISE

// Création de l'arbre n-aires associé à l'organigramme

Arbre\_N\_aire arbre = new Arbre\_N\_aire(new Noeud\_N(CEO));

arbre.InsererSuccesseur(arbre.Racine, DirectComN);

arbre.InsererFrere(DirectComN, DirectOpN);

arbre.InsererSuccesseur(DirectComN, Com1N);

arbre.InsererFrere(Com1N, Com2N);

arbre.InsererFrere(DirectOpN, DirectRHN);

arbre.InsererSuccesseur(DirectOpN, ChefEq1N);

arbre.InsererFrere(ChefEq1N, ChefEq2N);

arbre.InsererSuccesseur(ChefEq1N, Chauff11N);

arbre.InsererFrere(Chauff11N, Chauff12N);

arbre.InsererFrere(Chauff12N, Chauff13N);

arbre.InsererSuccesseur(ChefEq2N, Chauff21N);

arbre.InsererFrere(Chauff21N, Chauff22N);

arbre.InsererFrere(DirectRHN, DirectFinanceN);

arbre.InsererSuccesseur(DirectRHN, RH1N);

arbre.InsererFrere(RH1N, RH2N);

arbre.InsererSuccesseur(DirectFinanceN, DirectComptN);

arbre.InsererSuccesseur(DirectComptN, Compt1N);

arbre.InsererFrere(Compt1N, Compt2N);

arbre.InsererFrere(DirectComptN, ControlGestN);

#endregion

return arbre; // On retroune l'arbre n-aires

}

#endregion

#region SAUVEGARDE FICHIER SALARIE

static void SauvegardeSalarie(Salarie[] salaries, string nomFich) // Permet de sauvegarder les modifications réalisés dans le fichier txt

{

StreamWriter fichEc = new StreamWriter(nomFich); // on réecrit le fichier

for (int i = 0; i < salaries.Length; i++) // réécrit fichier 1 à 1

{

if (salaries[i] != null) // Regarde si salariés est null ou non

{

fichEc.WriteLine(salaries[i].ToString()); // Ecriture du salarié avec le séparateur nécéssaire pour la lecture du fichier txt, qui est ";"

}

}

fichEc.Close(); // ferme le fichier

}

#endregion

#region CREATION SALARIE

public static Salarie CreationSalarie() // Permet de créer un salarié en ajoutant en tapant ces attributs

{

#region Partie création d'un salarié

// Création des attributs du salarié un par un

Console.WriteLine("Numéro de Sécurité Sociale :");

string numeroSS = Console.ReadLine();

Console.WriteLine("Nom :");

string nom = Console.ReadLine();

Console.WriteLine("Prenom :");

string prenom = Console.ReadLine();

Console.WriteLine("Date de naissance :");

int annee = Annee();

int mois = Mois();

int jour = Jour(mois, annee);

string naissance = jour + "/" + mois + "/" + annee;

Console.WriteLine("Adresse :");

string adresse = Console.ReadLine();

Console.WriteLine("Mail :");

string mail = Console.ReadLine();

Console.WriteLine("Numéro de téléphone :");

string num = Console.ReadLine();

Console.WriteLine("Sexe : (M si Homme / F si Femme)");

string sex = Console.ReadLine();

Console.WriteLine("Date d'embauche :");

int annee2 = Annee();

int mois2 = Mois();

int jour2 = Jour(mois2, annee2);

string embauche = jour2 + "/" + mois2 + "/" + annee2;

Console.WriteLine("Poste au sein de l'entreprise :");

string poste = Console.ReadLine();

Console.WriteLine("Salaire : ");

string salaire = Console.ReadLine();

#endregion

Salarie newSalarie = new Salarie(numeroSS, nom, prenom, naissance, adresse, mail, num, sex, embauche, poste, salaire); // On initialise le nouveau salarié créé

return newSalarie; // Retourne le salarié créé

}

#endregion

#region AJOUT SALARIE

static void AjoutSalarie(Salarie[] salaries, Chauffeur[] chauffeurs, string nomFichSalaries, string nomFichChauffeurs) // Permet d'ajouter un salarié au fichier txt qui lui est lié

{

int n = salaries.Length; // longueur du tableau

Salarie newSalarie = CreationSalarie(); // On créé un nouveau salarié à ajouter au fichier txt

int res = 1;

for (int i = 0; i < n; i++) // On parcourt le fichier txt 1 par 1

{

if (salaries[i] != null) // Vérifie que salarié est bien différent de null

{

if (salaries[i].Nom == newSalarie.Nom && salaries[i].Prenom == newSalarie.Prenom) // On checrhe à savoir si le salarié existe déjà

{

Console.WriteLine("\nSalarié déjà existant : ERROR\n");

res = 0;

break;

}

}

}

// Si le salarié n'existe pas, ce qui se vérifie par : res = 1

if (res == 1)

{

StreamWriter fichEc = new StreamWriter(nomFichSalaries); // On écrit le fichier pour le modifier

if (n == 0) // Si le fichier est vide

{

fichEc.WriteLine(newSalarie.ToString()); // On écrit le nouveau salarié créé dans le fichier

fichEc.Close(); // On ferme le fichier pour stopper l'écriture

}

else

{

Salarie[] salaries2 = new Salarie[n]; // On créé un tableau de longueur des nb salarié du fichier

for (int j = 0; j < salaries.Length; j++)

{

salaries2[j] = salaries[j]; // On Initialise les valeurs du tableau un par un

fichEc.WriteLine(salaries2[j].ToString()); // On écrit ces valeurs dans le fichier

}

salaries2[n - 1] = newSalarie; // On rajoute à la dernière case du tableau le nouveau salarié

fichEc.WriteLine(salaries2[n - 1].ToString()); // On l'écrit dans le fichier

fichEc.Close(); // On ferme le fichier pour stopper l'écriture

}

}

SauvegardeSalarie(LectureFichierSalaries(nomFichSalaries, false), nomFichSalaries); // On sauvegrade les modifications

if (res == 1)

{

if ((newSalarie.Poste.ToUpper() == "CHAUFFEUR") || (newSalarie.Poste.ToUpper() == "CHAUFFEUSE")) // On doit l'ajouter dans le fichier chauffeurs

{

// On initialise les frais du chauffeur

Console.WriteLine("\nSaississez les frais kilométriques et honoraires du chauffeur " + newSalarie.Nom + " " + newSalarie.Prenom + " : ");

Console.WriteLine("\nFrais honoraires du chauffeur : ");

int fraisHonoraires = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("\nFrais kilométriques du chauffeur : ");

int fraisKilométrique = Convert.ToInt32(Console.ReadLine());

Chauffeur newChauff = new Chauffeur(newSalarie.NumeroSS, newSalarie.Nom, newSalarie.Prenom, newSalarie.Naissance, newSalarie.Adresse, newSalarie.Mail, newSalarie.Tel, newSalarie.Sex, newSalarie.Embauche, newSalarie.Poste, newSalarie.Salaire, "Libre", Convert.ToString(fraisHonoraires), Convert.ToString(fraisKilométrique));

StreamWriter fichEc = new StreamWriter(nomFichChauffeurs); // On écrit le fichier pour le modifier

if (n == 0) // Si le fichier est vide

{

fichEc.WriteLine(newChauff.ToString()); // On écrit le nouveau salarié créé dans le fichier

fichEc.Close(); // On ferme le fichier pour stopper l'écriture

}

else

{

Chauffeur[] chauffeurs2 = new Chauffeur[n]; // On créé un tableau de longueur des nb salarié du fichier

for (int j = 0; j < chauffeurs.Length; j++)

{

chauffeurs2[j] = chauffeurs[j]; // On Initialise les valeurs du tableau un par un

fichEc.WriteLine(chauffeurs2[j].ToString()); // On écrit ces valeurs dans le fichier

}

chauffeurs2[n - 1] = newChauff; // On rajoute à la dernière case du tableau le nouveau salarié

fichEc.WriteLine(chauffeurs2[n - 1].ToString()); // On l'écrit dans le fichier

fichEc.Close(); // On ferme le fichier pour stopper l'écriture

}

}

}

SauvegardeChauffeur(LectureFichierChauffeurs(nomFichChauffeurs, false), nomFichChauffeurs); // On sauvegrade les modifications

}

#endregion

#region RECHERCHE SALARIE

public static Salarie rechercheSalarie(Salarie[] salaries) // Ce code permet de rechercher un salarié au sein du fichier texte associé aux salariés

{

Salarie sal = new Salarie(); // initialise un salarié à null

// On initialise le Nom et Prénom du salarié qu'on souhaite rechercher pour le trouver au sein du fichier

Console.WriteLine("\nDonnez un salarié de la liste : ");

Console.WriteLine("\nNom :");

string nom = Convert.ToString(Console.ReadLine());

Console.WriteLine("\nPrénom :");

string prenom = Convert.ToString(Console.ReadLine());

for (int i = 0; i <= salaries.Length - 1; i++) // On parcourt le fichier un par un

{

if (salaries[i] != null) // vérifie que salaries[i] est bien différent de null

{

if ((salaries[i].Nom.ToUpper() == nom.ToUpper()) && (salaries[i].Prenom.ToUpper() == prenom.ToUpper())) // On compare le nom et prénom de chaque salariés, voir si ils correspondent au nom et prénom donnés

{

Console.WriteLine("\nLe salarié " + nom + " " + prenom + " a été sélectionné.\n");

sal = salaries[i]; // On affecte le salarié correspondant au salarié null

Console.WriteLine(sal.AfficheSalarie()); // On affiche le client si il est trouvé de manière propre

break; // On met fin à la boucle for si trouvé

}

}

}

return sal; // On retourne le salarié si il est trouvé, sinon on retourne le salarié null

}

#endregion

#region SUPPRIMER SALARIE

static void SupprimerSalaries(Salarie[] salaries, string nomFich) // Permet de supprimer un salarié du fichier texte

{

// On initialise le Nom et Prénom du salarié qu'on souhaite supprimer pour le trouver au sein du fichier

Console.WriteLine("\nDonnez un salarié à viré : ");

Console.WriteLine("\nNom :");

string nom = Convert.ToString(Console.ReadLine());

Console.WriteLine("\nPrénom :");

string prenom = Convert.ToString(Console.ReadLine());

for (int i = 0; i <= salaries.Length - 1; i++) // On parcourt le fichier un par un

{

if (salaries[i] != null) // vérifie que salaries[i] est bien différent de null

{

if ((salaries[i].Nom.ToUpper() == nom.ToUpper()) && (salaries[i].Prenom.ToUpper() == prenom.ToUpper())) // On compare le nom et prénom de chaque salarié, voir si ils correspondent au nom et prénom donnés

{

Console.WriteLine("Le Salarie " + nom + " " + prenom + " a été viré.");

salaries[i] = null; // On initialise le salarié à supprimer à null

}

}

}

SauvegardeSalarie(salaries, nomFich); // On sauvegarde la modification au sein du fichier texte

}

#endregion

#endregion

#region MODULE SALARIE

static void ModSalarie(Arbre\_N\_aire arbre, string nomFichSalarie, string nomFichChauffeurs) // Fonction avec SWITCH permettant de choisir l'action du module salarié qu'on veut

{

string n;

int s = 0;

while (s < 1)

{

Console.WriteLine("\n" +

"\n ~~~~~~~~~~~~~~~~~~~~ Module SALARIE ~~~~~~~~~~~~~~~~~~~~" +

"\n1. Embaucher un nouveau salarié " +

"\n2. Licencier un salarié " +

"\n3. Inclure un salarié à l'organigramme " +

"\n4. Exclure un salarié de l'organigramme" +

"\n5. Affichage " +

"\n6. QUITTER ");

do

{

Console.WriteLine("\nQue souhaitez-vous faire ? (Donnez un nombre entre 1 et 6)");

n = Console.ReadLine();

} while (Convert.ToInt32(n) < 1 || Convert.ToInt32(n) > 6);

switch (n)

{

case "1": { AjoutSalarie(LectureFichierSalaries(nomFichSalarie, false), LectureFichierChauffeurs(nomFichChauffeurs, false), nomFichSalarie, nomFichChauffeurs); break; }

case "2": { SupprimerSalaries(LectureFichierSalaries(nomFichSalarie, false), nomFichSalarie); break; }

case "3":

{

arbre.AjoutArbre(arbre.Racine, rechercheSalarie(LectureFichierSalaries(nomFichSalarie, false)), SaisirNom());

arbre.AffichageArborescence(arbre.Racine, 0);

break;

}

case "4":

{

arbre.SuppressionArbre(arbre.Racine, SaisirNom());

arbre.AffichageArborescence(arbre.Racine, 0);

break;

}

case "5": { arbre.AffichageArborescence(arbre.Racine, 0); break; }

case "6": { Console.WriteLine("\nRetour au MENU\n"); s++; break; }

}

}

}

#endregion

#region COMMANDE

#region VEHICULE

#region SAISIE VEHICULE

public static string SaisieVeh() // Permet de saisir un véhicule

{

Console.WriteLine("\nVehicule (Voiture, Camionnette, Camion) :");

string veh = Console.ReadLine();

if ((veh.ToUpper() == "VOITURE") || (veh.ToUpper() == "CAMIONETTE") || (veh.ToUpper() == "CAMION")) // Correspond bien à un véhicule existant

{

return veh; // retourne le véhicule concerné

}

else

{

return SaisieVeh(); // Si saisie n'est pas valide, on recommence

}

}

#endregion

#region INFORMATION VEHICULE

public static string InfoVehicule(string veh) // Permet de saisir les informations en fonction du véhicule sélectionnés

{

if (veh.ToUpper() == "VOITURE")

{

Console.WriteLine("Nombre de passager : ");

string s = Console.ReadLine();

return s + " passager(s)";

}

if (veh.ToUpper() == "CAMIONNETTE")

{

Console.WriteLine("Usage du véhicule : ");

string s = Console.ReadLine();

return s;

}

if (veh.ToUpper() == "CAMION")

{

Console.WriteLine("Volume : ");

string vol = Console.ReadLine();

Console.WriteLine("Matière : ");

string mat = Console.ReadLine();

string s = vol + " / " + mat;

return s;

}

return InfoVehicule(veh);

}

#endregion

#region PRIX VEHICULE

public static int PrixVehicule(string veh) // Permet d'obtenir le prix du véhicule en fonction de notre sélection

{

// En effet, prix doit évoluer en fonction du véhicule désigné

if (veh.ToUpper() == "VOITURE")

{

Console.WriteLine(500 + " euro pour la VOITURE");

return 500;

}

if (veh.ToUpper() == "CAMIONNETTE")

{

Console.WriteLine(750 + " euro pour la CAMIONNETTE");

return 750;

}

if (veh.ToUpper() == "CAMION")

{

Console.WriteLine(1000 + " euro pour le CAMION");

return 1000;

}

Console.WriteLine("\nLe véhicule défini n'est pas valide : ERROR\n");

return 0;

}

#endregion

#endregion

#region SAUVEGARDE FICHIER CHAUFFEUR

static void SauvegardeChauffeur(Chauffeur[] chauffeurs, string nomFich) // Permet de sauvegarder les modifications réalisés dans le fichier txt

{

StreamWriter fichEc = new StreamWriter(nomFich); // on réecrit le fichier

for (int i = 0; i < chauffeurs.Length; i++) // réécrit fichier 1 à 1

{

if (chauffeurs[i] != null) // Regarde si chauffeur est null ou non

{

fichEc.WriteLine(chauffeurs[i].ToString()); // Ecriture du chauffeur avec le séparateur nécéssaire pour la lecture du fichier txt, qui est ";"

}

}

fichEc.Close(); // ferme le fichier

}

#endregion

#region SAUVEGARDE FICHIER COMMANDE

static void SauvegardeCommandes(Commande[] commandes, string nomFich) // Permet de sauvegarder les modifications réalisés dans le fichier txt

{

StreamWriter fichEc = new StreamWriter(nomFich); // on réecrit le fichier

for (int i = 0; i < commandes.Length; i++) // réécrit fichier 1 à 1

{

if (commandes[i] != null) // Regarde si commande est null ou non

{

fichEc.WriteLine(commandes[i].ToString()); // Ecriture des commandes avec le séparateur nécéssaire pour la lecture du fichier txt, qui est ";"

}

}

fichEc.Close(); // ferme le fichier

}

#endregion

#region RECHERCHE COMMANDE

public static Commande RechercheCommande(Commande[] commandes) // Ce code permet de rechercher une commande au sein du fichier texte associé aux clients

{

// On initialise le numéro de référence de la commande qu'on souhaite rechercher pour la trouver au sein du fichier

Console.WriteLine("\nDonnez le numéro de la commande : ");

string numero = Console.ReadLine();

for (int i = 0; i < commandes.Length; i++) // On parcourt le fichier un par un

{

if (commandes[i] != null) // vérifie que commandes[i] est bien différent de null

{

if (commandes[i].Numero == numero) // On compare le numéro de référence chaque commandes, voir si ils correspondent au numéro donné

{

Console.WriteLine("\nTrouvé\n");

Console.Write("La commande n°" + numero + " est la suivante : \n");

Console.WriteLine(commandes[i].AfficheCommande()); // On affiche la commande si il est trouvé de manière propre

return commandes[i]; // On retourne la commande trouvé

}

}

}

Console.Write("\nLa commande n°" + numero + " n'existe pas.\n");

return null; // On retourne null car aucune commande n'existe

}

#endregion

#region RECHERCHE CLIENT ASSOCIE A UNE COMMANDE

public static Client ClientCommande(string nomFichClients) // Recherche client associé à une commande

{

Client[] clients = LectureFichierClients(nomFichClients, false); // On crée notre tableau de clients par rapport au fichier donné

if (RechercheClientRetourner(clients) != null) // On cherche à savoir si le client existe ou non

{

Client client1 = RechercheClientRetourner(clients); // on cherche le client qu'on veut

return client1; // On retourne le client qu'on cherche

}

else // si il n'existe pas

{

Client client2 = CreationClient(); // On crée un nouveau client

AjoutAvecClientConnu(clients, nomFichClients, client2); // On ajoute ce nouveau client au fichier

return client2; // On retorune le client créer

}

}

#endregion

#region RECHERCHE CHAUFFEURS LIBRE

public static Chauffeur RechercheChauffeurLibre(string nomFichChauffeurs) // Ce code permet de rechercher un chauffeur libre au sein du fichier texte associé aux chauffeurs

{

Chauffeur[] chauffeurs = LectureFichierChauffeurs(nomFichChauffeurs, false); // On crée notre tableau de chauffeurs par rapport au fichier donné

int cpt = 0; // initialise un compteur

for (int i = 0; i < chauffeurs.Length; i++) // On parcourt le fichier un par un

{

if (chauffeurs[i].Statut.ToUpper() == "LIBRE") // On cherche les salariés qui ont leur statut en "libre"

{

Console.WriteLine(chauffeurs[i].ToString()); // On affiche les chauffeurs libres pour pouvoir le choisir

cpt += 1; // augmente de 1 à chaque client

}

}

if (cpt == 0) // Si compteur = 0, alors pas de chauffeur libre

{

Console.WriteLine("\nAucun chauffeur n'est libre. ");

return null; // Retourne null

}

else

{

// On initialise le numéro de référence de la commande qu'on souhaite rechercher pour la trouver au sein du fichier

Console.WriteLine("\nDonnez le nom du chauffeur libre à affecter : ");

string nom = Console.ReadLine();

for (int i = 0; i < chauffeurs.Length; i++)

{

if (chauffeurs[i].Nom.ToUpper() == nom.ToUpper())

{

Console.WriteLine("\nTrouvé");

chauffeurs[i].Statut = "Occupe"; // Change le statut du chauffeur

Console.WriteLine(chauffeurs[i].AfficheChauffeur()); // Affiche proprement le chauffeur sélectionné

SauvegardeChauffeur(chauffeurs, nomFichChauffeurs); // Sauvegarde les modifications dans le fichier

return chauffeurs[i]; // Retourne le chauffeur voulu

}

}

Console.WriteLine("\nVous n'avez pas saisi un chauffeur valide");

return RechercheChauffeurLibre(nomFichChauffeurs); // Rappelle la fonction

}

}

#endregion

#region MODIFIER MONTANT D'UN CLIENT

public static void modifMontant(Client client, int prix, string nomFichClients) // Ce code permet de modifier le montant d'un client donné

{

// En effet, à chaque commande passée, on doit modifier le montant du client associé à la commande, en ajoutant à son montant le prix de la commande

//

Client[] clients = LectureFichierClients(nomFichClients, false); // On crée un tableau de client associé au fichier txt

for (int i = 0; i < clients.Length; i++) // On parcourt le fichier un par un

{

if (clients[i] != null) // vérifie que clients[i] est bien différent de null

{

if ((clients[i].Nom.ToUpper() == client.Nom.ToUpper()) && (clients[i].Prenom.ToUpper() == client.Prenom.ToUpper())) // On compare le nom et prénom de chaque client, voir si ils correspondent au nom et prénom de la commande

{

double prixCalc = Convert.ToInt32(clients[i].Montant) + prix; // Initialise le nouveaux prix

clients[i].Montant = Convert.ToString(prixCalc); // Modifie le montant du clients[i]

Console.Write(client.AfficheClient());

}

}

}

SauvegardeClients(clients, nomFichClients); // On sauvegrade les modifications

}

#endregion

#region CREATION COMMANDE

public static Commande CreationCommande(string nomFichClients, string nomFichChauffeurs, int[,] graphHeures, int[,] graphKilometres) // Permet de créer une commande en tapant ces attributs

{

#region Partie création d'une commande

// Création des attributs Commande un par un

Console.WriteLine("\nCREATION D'UNE COMMANDE\n");

Console.WriteLine("\nNuméro de la commande :");

string numero = Console.ReadLine();

Console.WriteLine("\nClient associé :");

Client client = ClientCommande(nomFichClients);

string clientS = client.Nom + " " + client.Prenom;

Console.WriteLine("\nChauffeur :");

Console.WriteLine("\nChoississez un chauffeur de la société :");

Chauffeur chauffeur = RechercheChauffeurLibre(nomFichChauffeurs);

string chauf = "";

if (chauffeur == null) // si le chauffeur n'existe pas

{

chauf = "En attente d'un chauffeur";

}

else // sinon

{

chauf = chauffeur.Nom + " " + chauffeur.Prenom;

}

Console.WriteLine("\nLivraison (POINT A) :");

string lA = Console.ReadLine().ToUpper();

Console.WriteLine("\nLivraison (POINT B) :");

string lB = Console.ReadLine().ToUpper();

string veh = SaisieVeh();

Console.WriteLine("\nInfo sur le véhicule :");

string info = InfoVehicule(veh.ToUpper());

Console.WriteLine("\nPrix de la livraison :");

int prix = 0;

if (chauffeur == null) // si le chauffeur n'existe pas

{

Console.WriteLine("Le prix ne peut encore être défini");

prix = -1;

}

else // sinon

{

// Le prix comprend le prix du véhicule, mais égalment les frais kilométriques et les honoraires du chauffeur

prix = PrixVehicule(veh.ToUpper()) + DijkstraCalc(graphHeures, TransformerEnNumero(lA) + 1, TransformerEnNumero(lB) + 1) \* Convert.ToInt32(chauffeur.fraisHonoraires) + (DijkstraCalc(graphKilometres, TransformerEnNumero(lA) + 1, TransformerEnNumero(lB) + 1) \* Convert.ToInt32(chauffeur.fraisKilométrique)) / 60;

int montant = Convert.ToInt32(client.Montant) + prix;

modifMontant(client, montant, nomFichClients);

}

Console.WriteLine("\n\nDate :");

int annee = Annee();

int mois = Mois();

int jour = Jour(mois, annee);

string date = jour + "/" + mois + "/" + annee;

#endregion

Commande newCommande = new Commande(numero, clientS, lA, lB, Convert.ToString(prix), veh, info, chauf, date); // On initialise le nouveau client créé

return newCommande; // Retourne la commande créé

}

#endregion

#region AJOUT COMMANDE

static void AjoutCommande(string nomFichClients, string nomFichCommandes, string nomFichChauffeurs, int[,] graphHeures, int[,] graphKilometres) // Permet d'ajouter une commande au fichier txt qui lui est lié

{

Commande[] commandes = LectureFichierCommandes(nomFichCommandes, false); // Initialise un tableau de Commande lié au fichier

Commande newCommande = CreationCommande(nomFichClients, nomFichChauffeurs, graphHeures, graphKilometres); // Création d'une nouvelle commande

StreamWriter fichEc = new StreamWriter(nomFichCommandes); // On écrit le fichier associé

if (commandes.Length == 0) // Si le fichier ne contient aucune commande

{

fichEc.WriteLine(newCommande.ToString()); // On écrit la nouvelle commande créé dans le fichier

fichEc.Close(); // On ferme le fichier pour stopper l'écriture

}

else // sinon

{

Commande[] commandes2 = new Commande[commandes.Length + 1]; // On créé un tableau de longueur des nb commandes du fichier

for (int j = 0; j < commandes.Length; j++)

{

commandes2[j] = commandes[j]; // On Initialise les valeurs du tableau un par un

fichEc.WriteLine(commandes2[j].ToString()); // On écrit ces valeurs dans le fichier

}

commandes2[commandes.Length] = newCommande; // On rajoute à la dernière case du tableau le nouveau client

fichEc.WriteLine(commandes2[commandes.Length].ToString()); // On l'écrit dans le fichier

fichEc.Close(); // On ferme le fichier pour stopper l'écriture

}

Console.WriteLine("\nLa commande numéro n°" + newCommande.Numero + " a été modifié.");

Console.WriteLine(newCommande.AfficheCommande());

}

#endregion

#region SUPPRIMER COMMANDE

static void SupprimerCommande(string nomFichCommandes) // Permet de supprimer une commande du fichier texte

{

Commande[] commandes = LectureFichierCommandes(nomFichCommandes, false); // Initialise un tableau de Commande lié au fichier

// On initialise le numéro de référence de la commande qu'on souhaite suuprimer au sein du fichier

Console.WriteLine("\nDonnez le numéro d'une commande à supprimer : ");

string numero = Console.ReadLine();

for (int i = 0; i <= commandes.Length - 1; i++) // On parcourt le fichier un par un

{

if (commandes[i] != null) // vérifie que commandes[i] est bien différent de null

{

if (commandes[i].Numero == numero) // On compare le numéro de référence chaque commandes, voir si ils correspondent au numéro donné

{

Console.WriteLine("\nLa commande numéro n°" + numero + " a été supprimé.");

commandes[i] = null; // On initialise la commande à supprimer à null

}

}

}

SauvegardeCommandes(commandes, nomFichCommandes); // On sauvegarde la modification au sein du fichier texte

}

#endregion

#region MODIFIER COMMANDE

static void ModifierCommande(Commande[] commandes, string nomFichCommandes, string nomFichClients, string nomFichChauffeurs, int[,] graphHeures, int[,] graphKilometres) // Ce code permet de modifier une commande au sein du fichier

{

// On initialise le numéro de référence de la commande qu'on souhaite modifier au sein du fichier

Console.WriteLine("\nDonnez une commande à modifier : ");

Console.WriteLine("\nNuméro de la commande :");

string numero = Console.ReadLine();

for (int i = 0; i <= commandes.Length - 1; i++) // On parcourt le fichier un par un

{

if (commandes[i] != null) // vérifie que commandes[i] est bien différent de null

{

if (commandes[i].Numero == numero) // On compare le numéro de référence chaque commandes, voir si ils correspondent au numéro donné

{

commandes[i] = CreationCommande(nomFichClients, nomFichChauffeurs, graphHeures, graphKilometres); // On modifie la commande[i] en créant une nouvelle commande directement grâce à la fonction CreationCommande

Console.WriteLine("\nLa commande numéro n°" + numero + " a été modifié.");

Console.WriteLine(commandes[i].AfficheCommande());

}

}

}

SauvegardeCommandes(commandes, nomFichCommandes); // On sauvegrade les modifications

}

#endregion

#region CHEMIN LE PLUS COURT - Algorithme de Dijkstra

public static string[] DijkstraChemin(int[,] graph, int depart, int arrive) // Ce code permet d'appliquer l'Algorithme de Dijkstra

{

if (depart > arrive)

{

int t = arrive;

arrive = depart;

depart = t;

}

int cpt = 0; // initisalise un compteur

int[] distance = new int[arrive]; // Création d'un tableau avec les distances

bool[] chemin = new bool[arrive]; // Création d'un tableau permettant de ne pas repasser par un même chemin

string[] villes = new string[arrive + 1]; // Création d'un tableau permettant de stocker les villes dans lesquelles on passe et qui enregistre en dernier la valeur du chemin le plus court

for (int i = 0; i < arrive; i++) // remplit les tableaux distance et chemin

{

distance[i] = int.MaxValue;

chemin[i] = false;

}

distance[depart] = 0; // Car lieux de départ donc correspond aux km 0

for (int j = 0; j < arrive - 1; j++)

{

int u = Minimum(distance, chemin, arrive); // Cherche minimun distance qui renvoit l'indice

chemin[u] = true; // on passe par le chemin u donc met en TRUE

for (int v = 0; v < arrive; v++)

{

if ((Convert.ToBoolean(graph[u, v])) && (distance[u] != int.MaxValue) && (distance[u] + graph[u, v] < distance[v])) // Vrai ssi distance[u] est différente de la distance max

{

distance[v] = distance[u] + graph[u, v]; // Actualise la valeur de distance[v]

villes[u] = TransformerEnVille(u); // On Enregistre la ville du passage

cpt = u;

}

}

}

villes[cpt + 1] = TransformerEnVille(arrive); // Permet d'avoir le derneir lieu de passage

return villes; // Retourne le tableau du chemin et de la valeur

}

public static int DijkstraCalc(int[,] graph, int depart, int arrive) // Ce code permet d'appliquer l'Algorithme de Dijkstra

{

if (depart > arrive)

{

int t = arrive;

arrive = depart;

depart = t;

}

int[] distance = new int[arrive]; // Création d'un tableau avec les distances

bool[] chemin = new bool[arrive]; // Création d'un tableau permettant de ne pas repasser par un même chemin

for (int i = 0; i < arrive; i++) // remplit les tableaux distance et chemin

{

distance[i] = int.MaxValue;

chemin[i] = false;

}

distance[depart] = 0; // Car lieux de départ donc correspond aux km 0

for (int j = 0; j < arrive - 1; j++)

{

int u = Minimum(distance, chemin, arrive); // Cherche minimun distance qui renvoit l'indice

chemin[u] = true; // on passe par le chemin u donc met en TRUE

for (int v = 0; v < arrive; v++)

{

if ((Convert.ToBoolean(graph[u, v])) && (distance[u] != int.MaxValue) && (distance[u] + graph[u, v] < distance[v])) // Vrai ssi distance[u] est différente de la distance max

{

distance[v] = distance[u] + graph[u, v]; // Actualise la valeur de distance[v]

Console.Write(TransformerEnVille(u) + " => ");

}

}

}

Console.Write(TransformerEnVille(arrive));

return distance[distance.Length - 1]; // Retourne le tableau du chemin et de la valeur

}

#region Affichage du chemin

static void Chemin(string[] tab) // Ce code permet de renvoyer le chemin à emprunter comme un GPS

{

Console.WriteLine("\nChemin à emprunter : ");

for (int i = 0; i < tab.Length - 1; i++)

{

if (i == tab.Length - 2)

{

Console.Write(tab[i] + "\n");

}

else

{

if (tab[i] == null)

{

Console.Write("");

}

else

{

Console.Write(tab[i] + " => ");

}

}

}

}

#endregion

#region Minimun distance

public static int Minimum(int[] distance, bool[] chemin, int arrive) // Permet de trouver la distance minimal

{

int minI = 0; // Initialise l'index minimun

int min = int.MaxValue; // Associe une valeur maximun

for (int i = 0; i < arrive; i++)

{

if ((chemin[i] == false) && (distance[i] <= min)) // Verifie si chemin n'a jamais été emprunté (si chemin[i] == false) ET si la distance entre deux points est bien la minimun (distance[i] <= min)

{

min = distance[i]; // Initialise le minimun qui est inférieur au minimun précédent

minI = i; // Modifie index

}

}

return minI; // Renvoit l'index qui a la plus petite distance

}

#endregion

#region Transformer ville en nombre (et inversement)

static string TransformerEnVille(int n) // Permet de passer d'un nombre à une ville

{

if (n == 0)

{

return "PARIS";

}

if (n == 1)

{

return "ROUEN";

}

if (n == 2)

{

return "LYON";

}

if (n == 3)

{

return "ANGERS";

}

if (n == 4)

{

return "LA ROCHELLE";

}

if (n == 5)

{

return "BORDEAUX";

}

if (n == 6)

{

return "BIARRITZ";

}

if (n == 7)

{

return "TOULOUSE";

}

if (n == 8)

{

return "PAU";

}

if (n == 9)

{

return "NIMES";

}

if (n == 10)

{

return "MONTPELLIER";

}

if (n == 11)

{

return "MARSEILLE";

}

if (n == 12)

{

return "AVIGNON";

}

if (n == 13)

{

return "MONACO";

}

if (n == 14)

{

return "TOULON";

}

return null;

}

static int TransformerEnNumero(string n) // Permet de passer d'une ville à un nombre

{

if (n == "PARIS")

{

return 0;

}

if (n == "ROUEN")

{

return 1;

}

if (n == "LYON")

{

return 2;

}

if (n == "ANGERS")

{

return 3;

}

if (n == "LA ROCHELLE")

{

return 4;

}

if (n == "BORDEAUX")

{

return 5;

}

if (n == "BIARRITZ")

{

return 6;

}

if (n == "TOULOUSE")

{

return 7;

}

if (n == "PAU")

{

return 8;

}

if (n == "NIMES")

{

return 9;

}

if (n == "MONTPELLIER")

{

return 10;

}

if (n == "MARSEILLE")

{

return 11;

}

if (n == "AVIGNON")

{

return 12;

}

if (n == "MONACO")

{

return 13;

}

if (n == "TOULON")

{

return 14;

}

Console.WriteLine("La ville écrite n'existe pas");

return -1;

}

#endregion

#endregion

#endregion

#region MODULE COMMANDE

static void ModCommande(Arbre\_N\_aire arbre, int[,] graphHeures, int[,] graphKilometres, string nomFichClients, string nomFichCommandes, string nomFichChauffeurs) // Fonction avec SWITCH permettant de choisir l'action du module commande qu'on veut

{

string n;

int s = 0;

while (s < 1)

{

Console.WriteLine("\n" +

"\n ~~~~~~~~~~~~~~~~~~~~ Module COMMANDE ~~~~~~~~~~~~~~~~~~~~" +

"\n1. Ajout d'une commande " +

"\n2. Supprimer une commande " +

"\n3. Rechercher une commande " +

"\n4. Modifier une commande " +

"\n5. Affichage " +

"\n6. Affichage du chemin à emprunter pour une commande " +

"\n7. QUITTER ");

do

{

Console.WriteLine("\nQue souhaitez-vous faire ? (Donnez un nombre entre 1 et 7)");

n = Console.ReadLine();

} while (Convert.ToInt32(n) < 1 || Convert.ToInt32(n) > 7);

switch (n)

{

case "1": { AjoutCommande(nomFichClients, nomFichCommandes, nomFichChauffeurs, graphHeures, graphKilometres); break; }

case "2": { SupprimerCommande(nomFichCommandes); break; }

case "3": { RechercheCommande(LectureFichierCommandes(nomFichCommandes, false)); break; }

case "4": { ModifierCommande(LectureFichierCommandes(nomFichCommandes, false), nomFichCommandes, nomFichClients, nomFichChauffeurs, graphHeures, graphKilometres); break; }

case "5": { LectureFichierCommandes(nomFichCommandes, true); break; }

case "6":

{

Commande commande = RechercheCommande(LectureFichierCommandes(nomFichCommandes, true));

if (commande == null)

{

Console.WriteLine("\nIl est impossible d'afficher le chemin à emprunter pour cette commande.\n");

}

else

{

Chemin(DijkstraChemin(graphKilometres, TransformerEnNumero(commande.LivraisonA), TransformerEnNumero(commande.LivraisonB)));

}

break;

}

case "7": { Console.WriteLine("\nRetour au MENU\n"); s++; break; }

}

}

}

#endregion

#region STATISTIQUE

#region Afficher par chauffeur le nombre de livraisons effectuées

public static void AfficherChauffeursLivraisons(string nomFichCommandes, string nomFichChauffeurs) // Ce code permet d'afficher le nombre de livraisons par chauffeur

{

Commande[] commandes = LectureFichierCommandes(nomFichCommandes, false);

Chauffeur[] chauffeurs = LectureFichierChauffeurs(nomFichChauffeurs, false);

string[] nomChauff = new string[chauffeurs.Length]; // Permet de stocker les noms des Chauffeurs dans le tableau

int[] nbLivraisons = new int[chauffeurs.Length]; // Permet de compter et d'actualiser les livraisons pour un chauffeurs[j]

for (int i = 0; i < chauffeurs.Length; i++) // parcours tableau chauffeurs

{

if (chauffeurs[i] != null) // Regarde si le chauffeur est null ou non

{

nomChauff[i] = chauffeurs[i].Nom + " " + chauffeurs[i].Prenom; // écrit le nom et prénom du chauffeurs[i] dans nomChauff[i]

}

}

for (int i = 0; i < commandes.Length; i++) // parcours tableau commandes

{

if (commandes[i] != null) // Regarde si la commande est null ou non

{

for (int j = 0; j < nomChauff.Length; j++) // // parcours tableau nomChauff

{

if (nomChauff[j] == commandes[i].Chauffeur) // Si nomChauff correspond au chauffeur de la commande

{

nbLivraisons[j] += 1; // ajout de plus 1

}

}

}

}

for (int j = 0; j < nomChauff.Length; j++) // parcourt nomChauff et nbLivraisons pour l'affichage

{

Console.WriteLine("Nombre de livraison du chauffeur " + nomChauff[j] + " : " + nbLivraisons[j] + " livraison(s)."); // Affichage du chauffeur et du nombre de livraisons qu'il a effectué

}

}

#endregion

#region Afficher les commandes selon une période de temps

public static void CommandesTemps(string nomFichCommandes) // Ce code permet d'afficher l'ensemble des dates se trouvant dans un intervalle précis

{

#region Date minimun de la période

// Initialise la date minimun de la comparaison

Console.WriteLine("\nSaississez la date minimun de la période : ");

int anneeMin = Annee();

int moisMin = Mois();

int jourMin = Jour(moisMin, anneeMin);

//Utilisation des propriétes DateTime()

DateTime minDate = new DateTime(anneeMin, moisMin, jourMin);

Console.WriteLine("\nDate minimun : " + minDate.ToShortDateString() + "\n");

#endregion

#region Date maximun de la période

// Initialise la date maximun de la comparaison

Console.WriteLine("\nSaississez la date maximun de la période : ");

int anneeMax = Annee();

int moisMax = Mois();

int jourMax = Jour(moisMax, anneeMax);

//Utilisation des propriétes DateTime()

DateTime maxDate = new DateTime(anneeMax, moisMax, jourMax);

Console.WriteLine("\nDate maximun : " + maxDate.ToShortDateString() + "\n");

#endregion

Commande[] commandes = LectureFichierCommandes(nomFichCommandes, false);

for (int i = 0; i < commandes.Length; i++)

{

DateTime date = new DateTime(SplitDate(commandes[i].Date)[2], SplitDate(commandes[i].Date)[1], SplitDate(commandes[i].Date)[0]); // Comme ma date est une chaine de caractère dont les jours/mois/années sont séparés par "/", on utilise la fonction SplitDate que j'ai créé, puis on utilise les propriétés de DateTime

if ((DateTime.Compare(minDate, date) < 0) && (DateTime.Compare(maxDate, date) > 0)) // Utilisation des fonctionnalités de DateTime pour les compraraisons

{

Console.WriteLine(commandes[i].AfficheCommande()); // Affichage propre des commandes situés dans cette intervalle

}

}

}

#endregion

#region Afficher la moyenne des prix des commandes

public static void MoyennePrixCommande(string nomFichCommandes) // Ce code permet de renvoyer le moyenne des prix d'une commande

{

Commande[] commandes = LectureFichierCommandes(nomFichCommandes, false);

int n = commandes.Length; // Longueur du tableau qui permettra d'obtenir la moyenne par la suite

int somme = 0; // Initialise une somme correspondant à la somme totale de fin

for (int i = 0; i < n; i++) // Parcourt le tableau de commandes

{

somme += Convert.ToInt32(commandes[i].Prix); // Actualise la somme totale à chaque commande

}

Console.WriteLine("Moyenne des prix des commandes : " + (double)somme / n + " en euros"); // Affichage du résultat souhaité

}

#endregion

#region Afficher la moyenne des comptes clients

public static void MoyenneCompteClients(string nomFichClients) // Ce code permet de renvoyer le moyenne des comptes clients

{

Client[] clients = LectureFichierClients(nomFichClients, false);

int n = clients.Length; // Longueur du tableau qui permettra d'obtenir la moyenne par la suite

int somme = 0; // Initialise une somme correspondant à la somme totale de fin

for (int i = 0; i < n; i++) // Parcourt le tableau de clients

{

somme += Convert.ToInt32(clients[i].Montant); // Actualise la somme totale à chaque montant

}

Console.WriteLine("Moyenne des comptes des clients : " + (double)somme / n + " en euros");

}

#endregion

#region Afficher la liste des commandes pour un client

public static void ListeCommandesClient(string nomFichCommandes)

{

Commande[] commandes = LectureFichierCommandes(nomFichCommandes, false);

int n = commandes.Length;

// On initialise le Nom et Prénom du client dont on souhaite listé les commandes

Console.WriteLine("\nDonnez le nom et prénom du client dont vous voulez listé les commandes : ");

Console.WriteLine("\nNom : ");

string nom = Console.ReadLine();

Console.WriteLine("\nPrénom : ");

string prenom = Console.ReadLine();

string client = nom + " " + prenom;

for (int i = 0; i < n; i++) // Parcourt le tableau de commandes

{

if (client.ToUpper() == commandes[i].Clients.ToUpper()) // Si le client de la commande correspond au client sélectionné

{

Console.WriteLine(commandes[i].AfficheCommande()); // Affichage propre des commandes

}

}

}

#endregion

#endregion

#region MODULE STATISTIQUE

static void ModStatistique(string nomFichClients, string nomFichCommandes, string nomFichChauffeurs) // Fonction avec SWITCH permettant de choisir l'action du module statistique qu'on veut

{

string n;

int s = 0;

while (s < 1)

{

Console.WriteLine("\n" +

"\n ~~~~~~~~~~~~~~~~~~~~ Module STATISTIQUE ~~~~~~~~~~~~~~~~~~~~" +

"\n1. Affichage du nombre de livraisons effectuées par chauffeurs " +

"\n2. Affichage des commandes selon une période de temps " +

"\n3. Affichage de la moyenne des prix de l'ensembles des commandes " +

"\n4. Affichage de la moyenne des comptes clients " +

"\n5. Afficher de la liste des commandes pour un client " +

"\n6. QUITTER ");

do

{

Console.WriteLine("\nQue souhaitez-vous faire ? (Donnez un nombre entre 1 et 6)");

n = Console.ReadLine();

} while (Convert.ToInt32(n) < 1 || Convert.ToInt32(n) > 6);

switch (n)

{

case "1": { AfficherChauffeursLivraisons(nomFichCommandes, nomFichChauffeurs); break; }

case "2": { CommandesTemps(nomFichCommandes); break; }

case "3": { MoyennePrixCommande(nomFichCommandes); break; }

case "4": { MoyenneCompteClients(nomFichClients); break; }

case "5": { ListeCommandesClient(nomFichCommandes); break; }

case "6": { Console.WriteLine("\nRetour au MENU\n"); s++; break; }

}

}

}

#endregion

#region AUTRES

#region Réinitialiser statut des chauffeurs

public static void ReinitiliserStatutChauffeur(Chauffeur[] chauffeurs, string nomFichChauffeurs)

{

int cpt = 0;

for (int i = 0; i < chauffeurs.Length; i++)

{

if (chauffeurs[i].Statut.ToUpper() == "OCCUPE")

{

Console.WriteLine(chauffeurs[i].ToString());

cpt++;

}

}

Console.WriteLine("\nVoulez-vous modifier l'ensemble des statues (taper Oui) ou seulement celui d'un individu (taper Non) ? " +

"\nSinon taper Exit");

string choix = Console.ReadLine();

if (choix.ToUpper() == "OUI")

{

for (int i = 0; i < chauffeurs.Length; i++)

{

if (chauffeurs[i] != null)

{

if (chauffeurs[i].Statut.ToUpper() == "OCCUPE")

{

chauffeurs[i].Statut = "Libre";

}

}

}

}

if (choix.ToUpper() == "NON")

{

Console.WriteLine("\nDonnez le nom et prénom du chauffeur : ");

Console.WriteLine("\nNom : ");

string nom = Console.ReadLine();

Console.WriteLine("\nPrénom : ");

string prenom = Console.ReadLine();

for (int i = 0; i < chauffeurs.Length; i++)

{

if (chauffeurs[i] != null)

{

if ((chauffeurs[i].Nom == nom) && (chauffeurs[i].Prenom == prenom))

{

if (chauffeurs[i].Statut.ToUpper() == "OCCUPE")

{

chauffeurs[i].Statut = "Libre";

Console.WriteLine(chauffeurs[i].ToString());

break;

}

else

{

Console.WriteLine("\nLe statut est déjà LIBRE\n");

break;

}

}

else

{

Console.WriteLine("\nCe chauffeur n'existe pas.\n");

ReinitiliserStatutChauffeur(chauffeurs, nomFichChauffeurs);

}

}

}

}

SauvegardeChauffeur(chauffeurs, nomFichChauffeurs);

}

#endregion

#region Bilan Comptable

public static void Bilan(string nomFichSalaries, string nomFichCommandes)

{

Salarie[] salaries = LectureFichierSalaries(nomFichSalaries, false);

Commande[] commandes = LectureFichierCommandes(nomFichCommandes, false);

double somme1 = 0;

double somme2 = 0;

for (int i = 0; i < salaries.Length; i++)

{

somme1 += Convert.ToInt32(salaries[i].Salaire);

}

Console.WriteLine("\nTotal des salaires des employés : " + somme1 + " en euros\n");

for (int i = 0; i < commandes.Length; i++)

{

somme2 += Convert.ToInt32(commandes[i].Prix);

}

Console.WriteLine("\nPrix de l'ensemble des commandes : " + somme2 + " en euros\n");

Console.WriteLine("\nSaississez le montant des charges théoriques à payés par rapport à l'année passé : ");

double charges = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("\nCharge à payé : " + charges + " en euros\n");

Console.WriteLine("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

Console.WriteLine("\nBILAN COMPTABLE\n");

double bilan = somme2 - somme1 - charges;

if (bilan < 0)

{

Console.WriteLine("\nL'entreprise TransConnect est en déficit de : " + bilan + " euros.\n" +

"Une reconstruction de la société est nécéssaire.\n");

}

if (bilan == 0)

{

Console.WriteLine("\nL'entreprise TransConnect n'à réalisé aucun bénéfice.\n" +

"Une reconstruction de la société est nécéssaire.\n");

}

if (bilan > 0)

{

Console.WriteLine("\nL'entreprise TransConnect à réalisé un bénéfice de : " + bilan + " euros\n");

}

}

#endregion

#region Tri par numéro de commandes

#region Transformation

public static List<Commande> TransformerListCommande(Commande[] commandes) // Collections génériques

{

List<Commande> listeClient = new List<Commande>();

int n = commandes.Length;

for (int i = 0; i < n; i++)

{

listeClient.Add(commandes[i]);

}

return listeClient;

}

public static Commande[] TransformerTabCommande(List<Commande> commandes) // Collections génériques

{

int n = commandes.Count;

Commande[] tabCommandes = new Commande[n];

for (int i = 0; i < n; i++)

{

tabCommandes[i] = commandes[i];

}

return tabCommandes;

}

#endregion

public static void TrierNumero(List<Commande> mylist, string nomFich) //Tri avec interface

{

Console.WriteLine("\nTri selon le NUMERO de la commande\n");

mylist.Sort(mylist[0].Compare);

mylist.ForEach(x => Console.WriteLine(x.ToString()));

Commande[] commandes = mylist.ToArray();

SauvegardeCommandes(TransformerTabCommande(mylist), nomFich);

}

#endregion

#region Moyenne d'âge de l'entreprise

public static void Age(string nomFichSalaries, string nomFichClients)

{

double somme = 0;

int age = 0;

int[] tranches = new int[7];

Salarie[] salaries = LectureFichierSalaries(nomFichSalaries, false);

Client[] clients = LectureFichierClients(nomFichClients, false);

for (int i = 0; i < salaries.Length; i++)

{

if (salaries[i] != null)

{

somme += DateTime.Now.Year - SplitDate(salaries[i].Naissance)[2];

}

}

Console.WriteLine("\nLa moyenne d'âge de la société est : " + (double)somme / salaries.Length + " ans.\n");

for (int i = 0; i < clients.Length; i++)

{

if (clients[i] != null)

{

age = DateTime.Now.Year - SplitDate(clients[i].Naissance)[2]; // âge du client[i}

#region Tranches d'âge

if (10 < age && 20 > age)

{

tranches[0] += 1;

}

if (20 < age && 30 > age)

{

tranches[1] += 1;

}

if (30 < age && 40 > age)

{

tranches[2] += 1;

}

if (40 < age && 50 > age)

{

tranches[3] += 1;

}

if (50 < age && 60 > age)

{

tranches[4] += 1;

}

if (60 < age && 70 > age)

{

tranches[5] += 1;

}

if (70 < age)

{

tranches[6] += 1;

}

#endregion

}

}

for (int i = 0; i<tranches.Length; i++)

{

Console.WriteLine("Tranches d'âge [" + (i + 1) \* 10 + ";" + (i + 2) \* 10 + "] : " + tranches[i] + " client(s).");

}

}

#endregion

#endregion

#region MODULE AUTRES

static void ModAutres(string nomFichClients, string nomFichCommandes, string nomFichChauffeurs, string nomFichSalaries) // Fonction avec SWITCH permettant de choisir l'action du module autres qu'on veut

{

string n;

int s = 0;

while (s < 1)

{

Console.WriteLine("\n" +

"\n ~~~~~~~~~~~~~~~~~~~~ Module AUTRES ~~~~~~~~~~~~~~~~~~~~" +

"\n1. Réinitialiser l'ensemble des statues des chauffeurs " +

"\n2. Bilan Comptable" +

"\n3. Trier par numéro de commandes" +

"\n4. Moyenne d'âge de l'entreprise" +

"\n5. QUITTER ");

do

{

Console.WriteLine("\nQue souhaitez-vous faire ? (Donnez un nombre entre 1 et 5)");

n = Console.ReadLine();

} while (Convert.ToInt32(n) < 1 || Convert.ToInt32(n) > 5);

switch (n)

{

case "1": { ReinitiliserStatutChauffeur(LectureFichierChauffeurs(nomFichChauffeurs, false), nomFichChauffeurs); break; }

case "2": { Bilan(nomFichSalaries, nomFichCommandes); break; }

case "3": { TrierNumero(TransformerListCommande(LectureFichierCommandes(nomFichCommandes, false)), nomFichCommandes); break; }

case "4": { Age(nomFichSalaries, nomFichClients); break; }

case "5": { Console.WriteLine("\nRetour au MENU\n"); s++; break; }

}

}

}

#endregion

#region MAIN

static void Main(string[] args)

{

Arbre\_N\_aire arbre = ArbreEntreprise();

string n;

#region Ensemble des fichiers parcourues (A MODIFIER SI BESOIN)

string nomFichClients = @"C:\Users\thoma\Desktop\Client.txt";

string nomFichCommandes = @"C:\Users\thoma\Desktop\Commandes.txt";

string nomFichSalaries = @"C:\Users\thoma\Desktop\Salaries.txt";

string nomFichChauffeurs = @"C:\Users\thoma\Desktop\Chauffeurs.txt";

#endregion

#region Graphique Djikstra - Heures de routes

// PARIS - ROUEN - LYON - ANGERS - LA ROCHELLE - BORDEAUX - BIARRITZ - TOULOUSE - PAU - NIMES - MONTPELLIER - MARSEILLE - AVIGNON - MONACO - TOULON

int[,] graphHeures = {

{ 0, 105, 295, 191, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 105, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 295, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 191, 0, 0, 0, 140, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 140, 0, 98, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 98, 0, 107, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 107, 0, 159, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 159, 0, 101, 146, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 101, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 146, 0, 0, 35, 73, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 35, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 73, 0, 0, 60, 123, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 60, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 123, 0, 0, 95 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 95, 0 }

};

#endregion

#region Graphique Djikstra - Kilomètres de routes

// PARIS - ROUEN - LYON - ANGERS - LA ROCHELLE - BORDEAUX - BIARRITZ - TOULOUSE - PAU - NIMES - MONTPELLIER - MARSEILLE - AVIGNON - MONACO - TOULON

int[,] graphKilometres = {

{ 0, 133, 464, 294, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 133, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 464, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 294, 0, 0, 0, 187, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 187, 0, 183, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 183, 0, 202, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 202, 0, 309, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 309, 0, 193, 289, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 193, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 289, 0, 0, 52, 126, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 52, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 126, 0, 0, 99, 224, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 99, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 224, 0, 0, 169 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 169, 0 }

};

#endregion

// Main avec SWITCH permettant de choisir le module qu'on veut

int s = 0;

while (s < 1)

{

Console.WriteLine("\n" +

"\n ~~~~~~~~~~~~~~~~~~~~ MENU ~~~~~~~~~~~~~~~~~~~~" +

"\n1. Module CLIENT " +

"\n2. Module SALARIE " +

"\n3. Module COMMANDE " +

"\n4. Module STATISTIQUES " +

"\n5. Module AUTRES " +

"\n6. QUITTER ");

do

{

Console.WriteLine("\nQue souhaitez-vous faire ? (Donnez un nombre entre 1 et 6)");

n = Console.ReadLine();

} while (Convert.ToInt32(n) < 1 || Convert.ToInt32(n) > 6);

switch (n)

{

case "1": { ModClient(nomFichClients); break; }

case "2": { ModSalarie(arbre, nomFichSalaries, nomFichChauffeurs); break; }

case "3": { ModCommande(arbre, graphHeures, graphKilometres, nomFichClients, nomFichCommandes, nomFichChauffeurs); break; }

case "4": { ModStatistique(nomFichClients, nomFichCommandes, nomFichChauffeurs); break; }

case "5": { ModAutres(nomFichClients, nomFichCommandes, nomFichChauffeurs, nomFichSalaries); break; }

case "6": { Console.WriteLine("\nAu revoir !\n"); s++; break; }

}

}

}

#endregion

}

}