SAE Algorithmique 1.02

PARTIE 1 :

Structure des données / Les différents types :

- Villes stockées dans un tableau “tiut” de pointeur vers structure(“VilleIUT”)
- Type VilleIUT est un char(31) avec le nom de la ville(“nom”) et (commencement de la liste type “ListeDept” nommée “lDept”) un pointeur sur le type “MaillonDept” (liste de départements)
- Le type “MaillonDept” est constitué d’un département “d” (type “Departement”) et d’un pointeur sur le maillon suivant “suiv”.
- Le type “Departement” est constitué d’un char(31) “dept”, nombre de places ou capacité d’accueil en première année “nbP” (entier), nom du responsable des admissions du département “respAd” (char(31)).

Un **utilisateur** en consultation qui peut être par exemple un futur étudiant cherchant :

- les villes où il y a un IUT,
- les départements dans chaque IUT,
- le nombre de places en première année,
- les IUT possédant un département bien particulier

Un **administrateur** qui a le droit de mettre à jour des données des IUT par exemple :

- modifier le nombre de places dans un département,
- créer un département dans un IUT,
- supprimer un département d’un IUT,
- lancer et arrêter la phase de candidature (cf Partie II)
- modifier le nom du responsable d’un département

Descriptions des fonctions:

int login(void) : Maël
1er écran qu'on voit, saisit de si on est un administrateur ou utilisateur, dans le cas ou la personne est un administrateur, un mot de passe est demandé.
retourne -1 en cas d'erreur ou 0 pour l'utilisateur ou 1 pour l'administrateur.

mdp : mettez20svp

void afficherPlace(MaillonDept m) : Maël
affiche les places en 1 année pour un département dans un IUT.En paramètre est présent le maillon que l'on doit traiter.

int chargement(VilleIUT *tiut[], int *tLog, int tMax) : Johnny
La fonction chargement commence évidemment par l'ouverture d'un fichier via une variable flot. Si l'ouverture du fichier se passe mal, la fonction renvoie -1. Au début de la fonction, on déclare également des pointeurs vers des structures "VilleIUT" et "MaillonDept" qui serviront de variables temporaires lors de la lecture du fichier.
Les pointeurs pointent vers des variables allouées dynamiquement ; en cas de problème d'allocation mémoire, une erreur est affichée et la fonction retourne -2 pour le premier pointeur et -3 pour le second. Pour la lecture, on commence par lire le nom de la ville. Ensuite on vérifie que la ville n'est pas présente dans le tableau ; si elle est présente, on appelle la fonction insererDept pour insérer le département dans la liste de la ville lDept. Si la ville n'est pas présente, on l'insère dans le tableau et après avoir inséré le département dans la liste lDept.

int insererDept(VilleIUT *v, MaillonDept m) : Johnny, Maël
Insert le département mis en paramètre dans l'iut passé en paramètre, et cela dans l'ordre alphabétique. Retourne un code de retour qui indique si l'insertion s'est bien passé.

MaillonDept* rechercheDept(ListeDept lDept, int *trouve, char nom[ ]) : Johnny, Maël
Recherche si un département précis existe dans une ville avec sa liste passée en paramètre, à partir d'un nom de département, l'indique dans "trouve" et retourne l'adresse de celui-ci ou l'adresse du maillon précédent sa position d'insertion.

int rechercheIUT(VilleIUT *tiut[], int tLog, char ville[], int *trouve); Sami (test/correction Maël)

Permet d'effectuer une recherche dans le tableau tiut[], plus précisément une recherche dichotomique qui coupe le tableau en plusieurs parties pour pouvoir comparer les valeurs et trouver celle recherchée.

void afficherVille(VilleIUT v) : Sami , Johnny
Permet l'affichage d'une ville.

void menuCandidat(VilleIUT *tiut[], int *tLog, int tmax) : Sami, Johnny,Maël
Affichage du menu dédié à l'utilisateur.

void menuAdmin(VilleIUT *tiut[], int tLog, int tmax) : Sami, Johnny,Maël
Affichage du menu dédié à l'Admin.

int insérerVille(VilleIUT *tiut[], int *tLog, int *tMax, char *nomV): Malak,Johnny
Permet l'insertion d'une ville dans le tableau tiut[], dans le cas où la ville existe déjà un message d'erreur est renvoyé.Si le tableau tiut[] est plein on fait appel à la fonction reallocation.

void afficherTIUT(VilleIUT *tiut[], int tLog) : Johnny
Affiche toutes les villes présentes dans le tableau tiut[] en appelant la fonction afficherVille.

ListeDept listenouv(void) : Johnny
Permet la création d'une nouvelle liste vierge.

ListeDept insérerEntête(ListeDept lDept, Departement d) : Johnny
Permet l'insertion d'une nouvelle donnée en début de liste en allouant la place nécessaire grâce au malloc.

ListeDept insérerDept(ListeDept lDept, Departement d) : Johnny
Permet d'insérer un département en fonction de plusieurs vérifications:
-Si la liste de département est vide alors on appelle insérerEntête.
-Si on compare le département voulu avec le précédent et que celui-ci retourne >0 alors on retourne insérerEntête.

-Si on compare le département voulu aux autres et qu'on retrouve le même donc =0 alors le département est déjà présent .
Sinon on ajoute le département aux maillon.

ListeDept supprimerEntete(ListeDept lDept) : Johnny
Supprime l'en-tête d'une liste donnée en paramètre.

ListeDept supprimerDept(ListeDept lDept, Departement d) : Johnny
Parcours une liste à la recherche d'un élément et appelle la fonction supprimerEntete pour supprimer celui-ci.

void creerDept(VilleIUT *tiut[],int tLog) Johnny
Demande à l'utilisateur la ville dans laquelle insérer un nouveau département, vérifie que celle-ci existe puis lui demande le nom du département à insérer et vérifie que celui-ci n'existe pas déjà. Elle demande ensuite le nombre de places disponibles et le nom du responsable du département. Enfin elle insère le département dans la liste en appelant la fonction insérerDept.

void retirerDept(VilleIUT *tiut[], int *tLog);
sert à retirer un département dans une ville en appelant les fonctions supprimerDept

int longueur(ListeDept lDept) : Johnny
Calcule la longueur d'une liste de départements à l'aide d'une boucle.

bool vide(ListeDept lDept) : Johnny
Vérifie si une liste est vide et renvoie un booléen.

void miseAJourGlobale(VilleIUT *tiut[], int tLog);Maël
Affiche les IUT, demande dans quel IUT faire une modification, puis affiche les départements de l'IUT et demande le département sur lequel faire la modification, Ensuite affiche un menu des mises à jour possibles et appelle les fonctions de mise à jour correspondante à la saisie

void miseAJourNomDept(VilleIUT *tiut[], int tLog);Maël
Affiche le département puis Permet de modifier le nom d'un département et demande confirmation, il dit si le nom apparaît déjà dans la liste des départements de l'IUT et affiche le département après la modification

void miseAJourResp(VilleIUT *tiut[], int tLog);Maël

Affiche le département puis Permet de modifier le nom du responsable d'un département et demande confirmation et affiche le département après la modification

void miseAJourPlaces(VilleIUT *tiut[], int tLog) : Maël
Affiche le département puis Permet de modifier le nombre de places en 1ère année d'un département et demande confirmation et affiche le département après la modification

void afficherVilleDep(VilleIUT v) : Johnny
Affiche le nom d'une ville passée en paramètre ainsi que tous ses départements.

void afficheDeptDesIUT(VilleIUT *tiut[], int tLog); Maël
Recherche tous les IUT qui possèdent un département saisie en début de fonction, et les affiche.

void menuCandidat(VilleIUT *tiut[], int *tLog, int tMax); Sami
Permet un affichage d'un menu pour un candidat lui permettant d'accéder à différentes options qui seront des appels de différentes fonctions.
Cela permet donc d'afficher tous les IUT, d'afficher tous les département d'un IUT, d'afficher le nombre de places d'un département en 1ere année et d'afficher tout les IUT comportant un département donnée.

void enregistrement(VilleIUT *tiut[], int tLog); Johnny
Cette fonction permet la sauvegarde à la fin de l'utilisation du programme.

void afficheDeptDesIUT(VilleIUT *tiut[], int tLog); Maël
Permet l'affichage des IUT comportant un département après la saisie de l'utilisateur.

void afficherDep(Departement d); Johnny
Permet d'afficher les caractéristiques d'un département donnée en paramètre dans un IUT.

int supprimerVille(VilleIUT *tiut[], char nomV[], int *tLog, int pos); Johnny
Permet la suppression d'un IUT donnée en paramètre dont tous les département liées à cette IUT et libère l'espace nécessaire.
void clearpage(void); Sami
Permet la validation d'un system("clear").

void globale(void); Johnny
Permet le fonctionnement globale du programme.

ListeDept ListeDeptnouv(void); Maël
Permet la création d'une liste vide.

3) Structure des fichiers:

-Dans cette première partie nous avons opter pour un seul fichier texte regroupant toutes les informations qui concerne chaque IUT, où on retrouve respectivement la ville où se situe l'IUT, les départements qu'il propose ainsi que le nombre de places et le nom du responsable.

Partie II :

1) Structures

Nous avons donc 2 nouvelles structures :

Typedef struct (Choix) :

-Cette structure comporte 2 variables de "char" qui concernent le nom de de la ville (**ville**) correspondant au choix du Candidat et le nom du département(**dep**) qui lui aussi correspondant au même choix.

-Elle comporte aussi 2 variables d'entiers "int" qui représente les décisions prises par le responsable des admissions(**decisionResp**) et la décision prise par le Candidat concernant son entrée dans l'IUT(**decisionCand**).

Typedef struct (Candidat) :

-Cette structure comporte une variable d'entiers "int" représentant le numéro du candidat(**numC**) et 2 variables "char" pour le nom(**nom**) et prénom(**prenom**) du Candidat.

-On retrouve aussi un "float" de longueur 4 pour y stocker les notes(**notes**) et de même pour la moyenne(**moyenne**) de cette étudiant.

-Le Candidat donne son nombre de choix de candidature(**nombreChoix**) et on y retrouve aussi un tableaux de pointeur(****tChoix**) où les choix sont regroupés.

Nouveau maillon ajoutée “MaillonCandidat” et création d’une nouvelle liste ListeCand, Ajout d’une file “FileCand”.

2) Description des fonction:

Candidat ** chargementCandidats(int *tmax); Johnny
Permet de charger un fichier dans un tableau de pointeur à l’aide de la fonction lireCandidat.

Candidat * lireCandidat(FILE *flot); Johnny
Cette fonction permet de lire un candidat dans un fichier et de renvoyer un pointeur vers la structure candidat lue. Il faut d’abord allouer dynamiquement la place du candidat et lire ses informations dedans. Ensuite il faut allouer dynamiquement les choix dans le tableau de pointeurs vers des structures Choix ensuite il faut lire les choix.

Choix * lireChoix(FILE *flot); Johnny
Cette fonction permet de lire le choix d’un candidat dans un fichier et de renvoyer un pointeur vers la structure Choix *.

void sauvegarder(Candidat *tCand[], int tMax); Johnny
Cette fonction permet la sauvegarde des candidats et des choix qui leurs sont liés.

void afficherChoix(Choix *c); Johnny
Cette fonction affiche les informations concernant un choix donné en paramètre.

void afficherCandidat(Candidat *c); Johnny
Cette fonction affiche les informations concernant un Candidat donné en paramètre.

void afficherCandChoix(Candidat *tCand[], int tMax); Johnny
Cette fonction affiche tous les candidats ainsi que leurs choix respectifs.

void afficherCandDep(Candidat *tCand[], int tMax); Johnny
Cette fonction permet d’afficher tous les candidats qui ont fait une demande pour un département.

Candidat ** reallocationCand(Candidat *tCand[], int tMax); Johnny
Cette fonction permet de re-allouer le tableau des candidats avec un espace en plus.

Choix ** reallocationChoix(Choix *tChoix[], int nbChoix); Johnny
Cette fonction permet de re-allouer le tableau des choix avec un espace en plus.

void triCandidats(Candidat *tCand[], int tMax); Johnny
Cette fonction tri un tableau de candidats à l'aide d'un tri par échange et appel donc la fonction "plusGrandCand()" et "echangerCand()".

int plusGrandCand(Candidat *tCand[], int tMax); Johnny
Fonction parcourant le tableau candidat et comparant ses éléments afin de trouver l'élément le plus grand.

void echangerCand(Candidat *tCand[], int i, int j); Johnny
Fonction qui permet l'échange entre deux éléments dans un tableau candidat.

void triChoix(Choix *tChoix[], int nombreChoix); Johnny
Cette fonction tri un tableau de choix à l'aide d'un tri par échange et appel donc la fonction "plusGrandChoix()" et "echangerChoix()".

int plusGrandChoix(Choix *tChoix[], int nombreChoix); Johnny
Fonction parcourant le tableau choix et comparant ses éléments afin de trouver l'élément le plus grand.

void echangerChoix(Choix *tChoix[], int i, int j); Johnny
Fonction qui permet l'échange entre deux éléments dans un tableau choix .

int rechercherChoix(Choix *tChoix[], int nombreChoix, char ville[], char dep[], int *trouve);
Cette fonction permet d'effectuer une recherche dichotomique en comparant les villes ainsi que les départements dans un tableau choix, si le choix est trouvé la variable trouve passée par pointeur retourne la valeur 1 et renvoie sa position. Dans le cas où le choix n'est pas trouvé la variable trouve retourne la valeur 0 et retourne la position d'insertion.

int rechercherCandidat(Candidat *tCand[], int tMax, int numeroC, int *trouve);
Cette fonction permet d'effectuer une recherche dichotomique en comparant les numéros de candidats dans un tableau Candidat , si le numéro de candidat est trouvé la variable trouve passée par pointeur retourne la valeur 1 et renvoie sa position. Dans le cas où le numéro de candidat n'est pas trouvé la variable trouve retourne la valeur 0 et retourne la position d'insertion.

FileCand filenouv(void);
Fonction qui permet de créer une nouvelle file d'attente vide.

FileCand adjq(FileCand f , Candidat *c);
Fonction qui permet d'ajouter un candidat en fin de file d'attente.

FileCand supt(FileCand f);

Fonction qui permet de supprimer un candidat de la file d'attente.

Candidat * tete(FileCand f);
Fonction qui retourne le candidat en tête de la file d'attente.
int longueurFile(FileCand f);
Fonction qui retourne la longueur de la file d'attente.

int positionFileAttente(FileCand f, int numeroC);
Fonction qui retourne la position d'un candidat dans la file d'attente.

bool videFile(FileCand f);Maël
Fonction booléenne qui retourne la valeur true si la liste d'attente est vide, dans le cas contraire elle retourne la valeur false.

void afficher(FileCand f);
Fonction qui va permettre l'affichage de la file d'attente.

ListeCand listeCandNouv(void);
Fonction qui permet de créer une liste vide.

ListeCand insererCandEntete(void);
Fonction qui permet d'inserer un MaillonCand en début d'une liste passée en paramètre et puis renvoie cette même liste.

ListeCand insererCand(ListeCand lCand ,Candidat *c);
Fonction qui permet d'insérer un Maillon dans une liste donnée en paramètre dans un ordre croissant/alphabétique. Si la liste est vide elle insère le nouveau Maillon en tête;Si le numéro de candidat est inférieur à celui de la lise testée , le maillon est inséré en tête;Si le maillon existe déjà la fonction retourne la liste sans la modifier.Si aucun cas précédent n'est respecté la fonction recommence avec le maillon suivant de la liste.

ListeCand supprimerCandEntete(ListeCand lCand);
Fonction qui permet de supprimer un Maillon en tête de la liste donnée en paramètre. Lorsque le Maillon est supprimé l'espace mémoire de ce dernier est supprimé aussi .Dand le cas où la liste est vide on quitte le programme.

ListeCand supprimerCand(ListeCand lCand, Candidat *c);
Fonction qui permet de supprimer un Maillon d'une liste lDept passée en paramètre à partir de son attribut nom de département. Si la liste est vide ou si le maillon à supprimer n'existe pas, la liste est retournée sans changements.Si le maillon à supprimer est trouvé il sera alors supprimé.Dans le cas où aucune des conditions précédentes n'a été respectée on passe au maillon suivant.

int longueur (ListeCand lCand);
Fonction qui permet d'obtenir la longueur de la liste et retourne le nombre de maillons.

bool videListe (ListeCand lCand);
Fonction booléenne qui permet de savoir si une liste est vide.

ListeCand rechercherCandListe (ListeCand lCand, int numeroC, int *trouve);
Fonction qui permet de rechercher un maillon dans une liste de départements passée en paramètre pour ensuite renvoyer son adresse. Si la liste est vide où si le département recherché n'est pas trouvé, une variable trouve passée en pointeur va prendre la valeur 0 et la liste sera retournée. Si le département recherché est trouvé la variable trouve va prendre la valeur 1 et la liste sera retournée. Dans le cas où aucune des conditions précédentes n'est respectée on passe au maillon suivant.

void menuResp(Candidat *tCand[],int tMax, int phaseCandidature, FileCand *f,ListeCand *ladmis,float *mini);Maël
menu du responsable d'admission depuis lequel il peut voir la file d'attente et la liste des admissions, et gérer les admissions

FileCand insertFile(Candidat *tCand[],int tMax);Maël

FileCand fileToAdmis(FileCand f, Liste *l,int nbAdmis, float mini);Maël
Fait passer les candidats,avec une moyenne supérieure à"mini", de la file d'attente à la liste des admis "nbAdmis" fois.

void gererAdmis(FileCand *f,ListeCand *l, float mini);Maël
sous-menu de "menuResp" spécialement pour gérer les admissions

int comptClermont(Candidat *tCand[],int tMax);Maël
Compte et retourne le nombre de Candidat ayant choisi Clermont-Ferrand et son département Informatique.

3) Structure des fichiers:

-Nous avons décidé de rester sur un fichier texte et avons gardé la même organisation que le fichier montré pour la partie II.

4) Structure de données:

-Afin de stocker les informations concernant les candidatures nous avons un tableau dynamique de pointeurs vers des structures candidat dynamiques comportant une liste chaînée “ListeCand” et une file , la structure candidat contient aussi un tableau de taille 4 pour stocker les notes.

-Les informations concernant les choix sont stockés différemment car nous avons un autre tableau dynamique de pointeurs vers des structures cette fois ci Choix dynamiques.

Comparaison des algorithmes de tris et recherche Partie I et II

Partie I :

Nous avons d’abord une récursive permettant de rechercher un maillon dans une liste de département qui nécessite aucune autre fonction.

Ensuite nous avons effectué une recherche dichotomique dans le tableau “tiut” d’une ville qui nécessite aucune autre fonction .

Cette partie n’as pas besoin de fonction de tri la liste est déjà trié.

Partie II :

Cette partie nécessite d’avoir une ou plusieurs fonctions de tris :

-**triCandidats** : on effectue un tri par échange sur un tableau de candidat
Pour cela nous avons besoin de deux nouvelles fonctions :
-**plusGrandCand** permet de trouver le plus grand éléments du tableau.
-**echangerCand** permet d’échanger deux éléments du tableau

-**triChoix :** on effectue un tri par échange sur un tableau de choix
Encore une fois on se sert de deux fonctions :
-plusGrandChoix permet de trouver le plus grand éléments du tableau.
-**echangerChoix** permet d’échanger deux éléments du tableau.

-**triTemp :** Tout comme **triCand** cette fonction tri les candidats mais cette fois en fonction de leur moyennes donc on réutilise **echangerCand** et on crée une fonction **plusGrandTemp** pour pouvoir trouver la meilleur moyenne parmi les candidats.

Nous avons deux recherches dichotomiques afin de rechercher les choix et les candidats.

-**rechercherChoix** et **rechercherCandidat.**

Comparaison :

On peut voir que les seules différences se font entre les différentes recherches malgré la majorité des recherches dichotomiques effectuées on retrouve une recherche dans une liste qui implique une récursive.