IUT de Paris - Rives de Seine
Université Paris Cité

Melouk Zakary 1ère année
Nguyen Phugon 1ère année
BUT informatique FI
Groupe 85

SAE D'INITIATION AU DÉVELOPPEMENT : IMPLÉMENTATION D'UN BESOIN CLIENT

IUT Paris Rives de Seine
Année universitaire 2022-2023

Sommaire

I. Présentation de l'application

a. Rôle fonctionnel

L'application que nous devons concevoir a pour but de gérer une formation. Elle doit pouvoir recevoir deux semestres, et pour chaque semestre un maximum de 10 matières dans lesquelles il peut y avoir au maximum 5 épreuves. Chaque épreuve est dotée d'un coefficient par UE. Cette formation comprend un nombre d'UE et un nombre d'élèves. Le nombre d'UE doit être compris entre 3 et 6 et un maximum de 100 étudiants peut être enregistré. Pour chaque élève présent dans la formation, une note doit pouvoir être attribuée à une épreuve et à la fin, un relevé semestriel et un relevé annuel avec les moyennes finales et la décision du jury doit être affiché.

b. Entrée et sorties de l'application

L'application doit recevoir des informations par une seule entrée qui est le clavier, et renvoyer des informations par une seule sortie qui est l'écran. Les informations tapées au clavier sont enregistrées, renvoient un message, et sont réutilisées à bon escient pour afficher des résultats. Pour faire fonctionner l'application, une formation avec son nombre d'UE doit d'abord être définie, en tapant « formation » suivie d'un nombre d'UE. L'application vérifiera si le nombre d'UE est bien compris entre 3 et 6 et s'il n'a pas été déjà défini. Elle affichera un message d'erreur ou « Le nombre d'UE est defini ». Aucune autre commande ne peut être exécutée si le nombre d'UE n'est pas défini. Puis, suit la création de matière et d'épreuve avec la commande « epreuve » suivie du numéro de semestre, du nom de la matière et du nom de l'épreuve ainsi qu'un coefficient par UE. Cette commande vérifiera la validité du numéro de semestre (1 ou 2) et si la matière existe. Si elle n'existe pas, elle affichera « Matiere ajoutee a la formation » puis « Epreuve ajoutee a la formation ». Si la matière existe, elle cherchera l'épreuve en affichant soit « Epreuve existante » ou « Epreuve ajoutee a la formation » et vérifiera également si tous les coefficients sont positifs et s'il y en a au moins un qui est supérieur à 0 en renvoyant un message d'erreur si cela n'est pas le cas. La commande coefficients les vérifiera pour chaque UE en tapant « coefficient », elle calculera leur somme par UE et si-celle-ci est nulle, elle renverra un message d'erreur ou sinon « Coefficients corrects ». Ensuite, la commande note avec « note » suivie du numéro de semestre, du nom de l'élève, de la matière, de l'épreuve et de la note. Elle vérifiera l'existence de l'étudiant, de la matière et de l'épreuve et, que la note est bien comprise entre 0 et 20. Elle affichera alors des messages d'erreur, ou « Etudiant ajoute a la formation », « note ajoutee à l'etudiant ». Si l'étudiant existe déjà, elle affichera seulement « note ajoutée à l'étudiant ». La commandes notes permettra elle, de vérifier les notes par étudiant en écrivant « notes » suivies d'un numéro de semestre et du nom d'un étudiant. Elle affichera des messages d'erreur si l'étudiant n'existe pas ou s'il lui manque des notes, et dans le cas contraire, elle affichera « Notes correctes ». Il existe, une 7ème commande qui permet de calculer les moyennes par matières et par UE dans un semestre donné, il faut taper « releve » suivi d'un numéro de semestre et du nom d'un étudiant. La commande vérifiera le numéro de semestre, l'existence de l'étudiant et s'il a comptabilisé des notes dans toutes les épreuves. Elle affichera des messages d'erreurs pour les problèmes concernés, ou elle affichera, si tout va bien, un relevé pour un semestre donné, avec les moyennes par matière pour les différentes UE, et les moyennes de toutes les matières pour chaque UE. Enfin, la 8ème commande affichera une décision en tapant « decision » suivie d'un nom d'étudiant. La commande vérifiera l'existence de l'étudiant, la validité de ses notes dans les 2 semestres et la validité des coefficients. Elle affichera un message d'erreur ou, si toutes les vérifications sont concluantes, ses moyennes dans les 2 semestres par UE ainsi que les UE acquises et la décision du jury. La commande exit en tapant « exit » permet de sortir du programme.

II. Les tests

a. Organisation des Tests

Pour avancer dans notre projet, à chaque commande que nous faisions, nous effectuions une série de tests pour pouvoir passer à la commande suivante. Nous avons créé nos propres tests en vérifiant d'abord que ce sont les bons messages que notre exécuteur de commande renvoie, en vérifiant toutes les erreurs potentielles précisées dans le sujet, mais aussi en vérifiant si les valeurs sont réellement stockées et si elles le sont dans les bonnes variables. Pour cela, nous utilisions les points d'arrêts qui nous permettaient de vérifier la contenance d'une variable durant les tests. À chaque sprint atteint, nous appliquions cette série de tests puis nous confrontions notre programme au fichier in-out mis à disposition en fonction du sprint atteint. Les fichiers in-out base permettait de vérifier si les variables étaient bien stockées et que les bons messages s'affichaient, et aussi de tester les vérifications faites par le programme et les messages d'erreurs associés. Cela nous a permis de ne pas continuer l'élaboration de notre programme sans savoir si celui-ci fonctionne ou pas.

b. Bilan de validation des tests.

Nous avons finalement réussi à finir l'application demandée. Nous avons validé les sprint numéros 1, 2, 3, 4 et même le sprint bonus numéro 5 qui testait notre programme avec toutes les valeurs maximums. Nous avons donc fini la dernière commande, que nous avons d'abord testée avec une série de test personnels, en testant chaque variable, chaque message d'erreur et chaque scénario possible. Nous avons utilisé ce fonctionnement sur chaque commande et cela nous a permis d'avancer beaucoup plus rapidement puisque les erreurs étaient facilement détectables, donc possibles à corriger également. La trace d'exécution du sprint 5 étant trop longue, nous mettons en annexe celle du sprint 4 que ce soit celui de base ou celui qui teste les erreurs de l'utilisateur.

III. Bilan du projet

a. Difficultés rencontrées

La réalisation de cette application a été laborieuse. Nous avons rencontré de très nombreuses difficultés que nous avons su surmonter.
Comprendre le fonctionnement de l'application a été le plus difficile. Sachant que la structure de base a été donnée, il fallait comprendre cette structure et surtout comment l'utiliser, comment y stocker les informations que l'on souhaite. Le tableau expliquant la formation, fourni dans le sujet a été à cet effet d'une grande aide, puisqu'il montrait comment était construite la formation, comment s'imbriquait les épreuves dans les matières dans les semestres, l'attribution d'un coefficient par UE à chaque épreuve, etc… Lorsque nous avions compris cela, nous avons dû faire face aux pointeurs.
Comprendre leur fonctionnement nous a également pris du temps, surtout dans le cadre des structures pour réussir à stocker des informations à l'intérieur sans les perdre durant tout le long de l'exécution du programme. Nous avons dû, ensuite, modifier la structure pour y rajouter des variables pour éviter de les déclarer dans le « main » et d'utiliser des pointeurs dans chaque commande.
Le travail en groupe a permis l'entraide, puisque nous étions deux à réfléchir à un problème, nous pouvions donc avoir l'avis de l'autre pour compléter notre compréhension et élargir notre vision afin de concevoir et développer plus rapidement et efficacement des solutions fiables qui répondent aux problèmes que nous avions rencontrées. Un autre point de vue que le sien s'avère toujours utile dans la résolution d'un problème. Chaque difficulté rencontrée nous a permis de mieux comprendre le langage C ainsi que ses subtilités.

b. Réussite et Amélioration

Nous avons réussi à terminer l'application en validant le dernier sprint. Toutes les commandes fonctionnent. Cependant, certains aspects pourraient être améliorés :

1. La taille de la formation qui pourrait être agrandie, car limitée à 100 étudiants
2. La taille du programme qui fait dans les 680 lignes peut être réduite.
3. Le principe de classe ou de groupe pourrait également s'ajouter, avec une moyenne générale sur toute la promotion par épreuve, par matière, par semestre et par UE mais également par groupe. Cela permettrait à l'utilisateur, en l'occurrence un professeur, d'avoir une vue d'ensemble sur ses étudiants, mais également de vérifier si les groupes sont homogènes, suivre leur progression et mesurer les écarts de niveau entre les groupes par matière, par UE, par semestre, etc., identifier à temps les groupes qui rencontrent des difficultés pour mettre en œuvre des actions pédagogiques correctives.
4. Notre main utilise beaucoup d'octets de pile ce qui est également à réduire en trouvant une solution.
5. Nous aurions pu améliorer l'affichage des relevés semestriels et annuels pour passer les sprints sans le « /w » mais notre affichage et quand même réussit et esthétique.
6. Nous aurions pu améliorer l'affichage avec les accents.

IV. Annexes

a. Annexe 1: Trace d'exécution du sprint 4 de base

```
formation 3
Le nombre d'UE est defini
epreuve 1 Programmation Projet 1 2 0
Matiere ajoutee a la formation
Epreuve ajoutee a la formation
epreuve 1 Programmation DST 2 3 0
Epreuve ajoutee a la formation
epreuve 1 SGBD Participation 0.5 0 0.5
Matiere ajoutee a la formation
Epreuve ajoutee a la formation
epreuve 1 SGBD Rapport 1.5 0 1.5
Epreuve ajoutee a la formation
epreuve 2 Architecture Interrogation 1 0 2
Matiere ajoutee a la formation
Epreuve ajoutee a la formation
epreuve 2 Architecture DST 0 1 4
Epreuve ajoutee a la formation
epreuve 2 Systeme QCM 2 3 0.5
Matiere ajoutee a la formation
Epreuve ajoutee a la formation
epreuve 2 Systeme Expose 3 2 0.5
Epreuve ajoutee a la formation
note 1 Paul Programmation Projet 12
Etudiant ajoute a la formation
Note ajoutee a l'etudiant
note 1 Paul Programmation DST 9
Note ajoutee a l'etudiant
note 1 Paul SGBD Participation 16
Note ajoutee a l'etudiant
note 1 Paul SGBD Rapport 12
Note ajoutee a l'etudiant
note 2 Paul Architecture Interrogation 18
Note ajoutee a l'etudiant
note 2 Paul Architecture DST 12
Note ajoutee a l'etudiant
note 2 Paul Systeme QCM 7
Note ajoutee a l'etudiant
note 2 Paul Systeme Expose 8
Note ajoutee a l'etudiant
note 1 Paule Programmation Projet 8
Etudiant ajoute a la formation
Note ajoutee a l'etudiant
note 1 Paule Programmation DST 11
Note ajoutee a l'etudiant
note 1 Paule SGBD Participation 20
Note ajoutee a l'etudiant
note 1 Paule SGBD Rapport 0
Note ajoutee a l'etudiant
note 2 Paule Architecture Interrogation 18
Note ajoutee a l'etudiant
note 2 Paule Architecture DST 12
Note ajoutee a l'etudiant
note 2 Paule Systeme QCM 7
Note ajoutee a l'etudiant
note 2 Paule Systeme Expose 8
Note ajoutee a l'etudiant
note 1 Paulo Programmation Projet 12
Etudiant ajoute a la formation
Note ajoutee a l'etudiant
note 1 Paulo Programmation DST 9
Note ajoutee a l'etudiant
note 1 Paulo SGBD Participation 16
Note ajoutee a l'etudiant
note 1 Paulo SGBD Rapport 12
Note ajoutee a l'etudiant
note 2 Paulo Architecture Interrogation 17
Note ajoutee a l'etudiant
note 2 Paulo Architecture DST 15
Note ajoutee a l'etudiant
note 2 Paulo Systeme QCM 16
Note ajoutee a l'etudiant
note 2 Paulo Systeme Expose 19
Note ajoutee a l'etudiant
decision Paul
                    UE1     UE2     UE3
S1                  11.2    10.2    13.0
S2                  9.3     8.1     13.0
--
Moyennes annuelles  10.2    9.1     13.0
Acquisition         UE1, UE3
Devenir             Passage
decision Paule
                    UE1     UE2     UE3
S1                  8.0     9.8     5.0
S2                  9.3     8.1     13.0
--
Moyennes annuelles  8.6     8.9     9.0
Acquisition         Aucune
Devenir             Redoublement
decision Paulo
                    UE1     UE2     UE3
S1                  11.2    10.2    13.0
S2                  17.6    16.8    15.9
--
Moyennes annuelles  14.4    13.5    14.4
Acquisition         UE1, UE2, UE3
Devenir             Passage
exit
```

b. Annexe 2: Trace d'exécution du sprint 4 d'erreur

```
formation 3
Le nombre d'UE est defini
epreuve 1 Programmation Projet 1 2 0
Matiere ajoutee a la formation
Epreuve ajoutee a la formation
epreuve 1 Programmation DST 2 3 0
Epreuve ajoutee a la formation
epreuve 2 Architecture Interrogation 1 0 2
Matiere ajoutee a la formation
Epreuve ajoutee a la formation
epreuve 2 Architecture DST 0 1 4
Epreuve ajoutee a la formation
note 1 Paul Programmation Projet 12
Etudiant ajoute a la formation
note 1 Paul Programmation DST 9
Note ajoutee a l'etudiant
note 2 Paul Architecture Interrogation 18
Note ajoutee a l'etudiant
note 2 Paul Architecture DST 12
Note ajoutee a l'etudiant
decision Paul
Les coefficients d'au moins un semestre sont incorrects
epreuve 1 SGBD Participation 0.5 0 0.5
Matiere ajoutee a la formation
Epreuve ajoutee a la formation
epreuve 1 SGBD Rapport 1.5 0 1.5
Epreuve ajoutee a la formation
note 1 Paul SGBD Participation 16
Note ajoutee a l'etudiant
decision Paul
Il manque au moins une note pour cet etudiant
note 1 Paul SGBD Rapport 12
Note ajoutee a l'etudiant
decision Paulo
Etudiant inconnu
decision Paul
                        UE1     UE2     UE3
S1                      11.2    10.2    13.0
S2                      18.0    12.0    14.0
--
Moyennes annuelles      14.6    11.1    13.5
Acquisition             UE1, UE2, UE3
Devenir                 Passage
exit
```

c. Annexe 3: Listing des sources

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <stdbool.h>
#include <math.h>
#pragma warning(disable:4996)
#pragma warning(disable:6031)

enum {
    NB_SEMESTRES = 2,
    MIN_UE = 3,
    MAX_UE = 6,
    MAX_MATIERES = 10,
    MAX_EPREUVES = 5,
    MAX_ETUDIANTS = 100,
    MAX_CHAR = 30,
    MAX_ESPACE = 20
};
const float MIN_NOTE = 0.f, MAX_NOTE = 20.0f;

/* Declarer les constantes*/

typedef char CH30[30];
typedef struct {
    CH30 nom; /* Nom épreuve */
    float coef[MAX_UE]; /* Coeff*/
    float note[MAX_ETUDIANTS];
}Epreuve;

typedef struct {
    CH30 nom; /*  Nom de de matière */
    unsigned int nBEpreuves;
    unsigned int indice_epreuve;
    unsigned int indice_next_epreuve;
    Epreuve e[MAX_EPREUVES];
}Matiere;

typedef struct {
    float Moyennes[MAX_ETUDIANTS];
    float moyenne_annuelle[MAX_ETUDIANTS];
}Moyenne;

typedef struct {
```

```
    float Moyenne_matiere[MAX_ETUDIANTS];
    Moyenne mo1[MAX_MATIERES];
}Tableau_moyenne;

typedef struct {
    unsigned int nbMatieres;
    unsigned int indice_matiere;
    unsigned int indice_next_matiere;
    Tableau_moyenne tm[MAX_UE];
    Matiere m[MAX_MATIERES];
    float Moyenne_par_semestre[MAX_ETUDIANTS];
} Semestre;

typedef struct {
    unsigned int nbUE;
    Moyenne mo[MAX_UE];
    unsigned int NBETUDIANT;
    Semestre s[NB_SEMESTRES]; /* Numéro semestre*/
    CH30 nom[MAX_ETUDIANTS];
    unsigned int indice_etudiant;
    unsigned int indice_next_etudiant;
}Formation;

// Fonction pour remplir toutes les notes de toutes la matieres de toutes les
epreuves à 21 ainsi que tout les indices  de toutes les matières de toutes
// les epreuves a 0. Elle prends en paramètre un pointeur vers une structure
formation et ne renvoit rien.

void initialiser_formation(Formation* f) {
    f->indice_etudiant = 0;
    f->indice_next_etudiant = 0;
    f->NBETUDIANT = 0;
    for (unsigned int i = 0; i < NB_SEMESTRES; ++i) {
        for (unsigned int u = 0; u < MAX_MATIERES; ++u) {
            f->s[i].indice_matiere = 0;
            f->s[i].indice_matiere = 0;
            for (unsigned int p = 0; p < MAX_EPREUVES; ++p) {
                f->s[i].m[p].indice_next_epreuve = 0;
                for (unsigned int x = 0; x < MAX_ETUDIANTS; ++x) {
                    f->s[i].m[u].e[p].note[x] = MAX_NOTE + 1;

                }
            }
        }
    }
}

// Fonction permettant de rechercher une matière dans la structure. Elle prend
en paramètre un pointeur vers une structure formation, un entier étant un
```

```
// numéro de semestre, une chaine de caractère correspondant au nom de la
matière et un bool. Elle renvoit un bool, qui sera vrai ou faux.

bool trouve_matiere(Formation* f, const unsigned int numero_semestre, const
CH30 NOM_MATIERE, bool trouvematiere) {
    for (unsigned int j = 0; j < MAX_MATIERES; ++j) {
        if (strcmp(NOM_MATIERE, f->s[numero_semestre - 1].m[j].nom) == 0) { //
Compare le nom de la matière avec le nom de chaque matière de la structure
            trouvematiere = true;
            f->s[numero_semestre - 1].indice_matiere = j; // Stock l'indice de
la matière trouvé
            return trouvematiere;
        }
    }
    return trouvematiere;
}

// Fonction permettant de rechercher une epreuve dans la matière
correspondante  dans la structure. Elle prend en paramètre une pointeur vers
une strucutre
// formation, un entier étant un numéro de semestre, une chaine de caractère
correspondant au nom de l'épreuve et un bool. Elle renvoit un bool, qui sera
// vrai ou faux.

bool trouve_epreuve(Formation* f, const unsigned int numero_semestre, const
CH30 NOM_EPREUVE, bool trouveepreuve) {
    for (unsigned int j = 0; j < MAX_EPREUVES; ++j) {
        if (strcmp(NOM_EPREUVE, f->s[numero_semestre - 1].m[f-
>s[numero_semestre - 1].indice_matiere].e[j].nom) == 0) { // Compare le nom de
l'èpreuve avec le nom de chaque epreuve de la matière trouvé avant
            trouveepreuve = true;
            f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_matiere].indice_epreuve = j; // Stock l'indice de l'épreuve trouvé
            return trouveepreuve;
        }
    }
    return trouveepreuve;
}

// Fonction permettant de rechercher un étudiant dans la structure. Elle prend
en paramètre une pointeur vers une structure formation,
// une chaine de caractère correspondant au nom de l'étudiant et un bool. Elle
renvoit un bool, qui sera vrai ou faux.

bool trouve_etudiant(Formation* f, const CH30 nom_etu, bool trouveetudiant) {
    for (unsigned int y = 0; y < f->NBETUDIANT; ++y) {
        if (strcmp(nom_etu, f->nom[y]) == 0) { // Compare le nom de l'étudiant
avec le nom de chaque étudiant de la structure
            trouveetudiant = true;
```

```
            f->indice_etudiant = y; // Recuperation de l'indice de l'étudiant
            return trouveetudiant;
        }
    }
    return trouveetudiant;
}

//Fonction qui permet d'initaliser les notes de chaque étudiant de la
formation dans chaque epreuve de chaque matière a MAX-NOTE+1. Elle prends en
paramètre
//un pointeur vers une structure formation et un entier correspondant au
numéro de semestre. Elle renvoit un entier qui est 1 ou 0.


int verif_note(Formation* f, const unsigned int numero_semestre) {
    unsigned int nombre_note = 0;
    unsigned int nombre_epreuve = 0;
    for (unsigned int u = 0; u < f->s[numero_semestre - 1].nbMatieres; ++u) {
        for (unsigned int i = 0; i < f->s[numero_semestre -
1].m[u].nBEpreuves; ++i) {
            nombre_epreuve += 1;
            if (f->s[numero_semestre - 1].m[u].e[i].note[f->indice_etudiant]
!= MAX_NOTE + 1) // Les notes sont intialisés a MAX_NOTE+1, donc chaque note
qui lui sera égal sera donc manquante
                nombre_note += 1;
        }
    }
    if (nombre_note != nombre_epreuve) {
        return 1;
    }
    else {
        return 0;
    }
}

//Fonction qui permet de calculer les moyennes d'un étudiant dans une
structure formation. Elle prend en paramètre un pointeur vers une structure
formation et
// un entier correspondant a l'indice d'un étudiant et ne renvoi rien.

void calcul_moyenne(Formation* f, const unsigned int indice_etudiant) {
    for (int y = 0; y < NB_SEMESTRES; ++y) {
        for (unsigned int u = 0; u < f->nbUE; ++u) {
            float somme_coeff2 = 0; //Comptabilise les coeff de toutes les
matières d'une UE d'un semestre
            for (unsigned int i = 0; i < f->s[y].nbMatieres; ) {
                float somme_coeff = 0; //Comptabilise les coeff de toutes les
epreuves d'une matière d'une UE d'un semestre
                for (unsigned int x = 0; x < f->s[y].m[i].nBEpreuves; ) {
```

```
                    f->s[y].tm[u].mo1[i].Moyennes[indice_etudiant] += f-
>s[y].m[i].e[x].note[indice_etudiant] * f->s[y].m[i].e[x].coef[u];
                    f->s[y].tm[u].Moyenne_matiere[indice_etudiant] += f-
>s[y].m[i].e[x].note[indice_etudiant] * f->s[y].m[i].e[x].coef[u];
                    somme_coeff += f->s[y].m[i].e[x].coef[u];
                    somme_coeff2 += f->s[y].m[i].e[x].coef[u];
                    ++x;
                    if (x == f->s[y].m[i].nBEpreuves) {
                        f->s[y].tm[u].mo1[i].Moyennes[indice_etudiant] = f-
>s[y].tm[u].mo1[i].Moyennes[indice_etudiant] / somme_coeff;
                    }
                }
                ++i;
                if (i == f->s[y].nbMatieres) {
                    f->s[y].tm[u].Moyenne_matiere[indice_etudiant] = f-
>s[y].tm[u].Moyenne_matiere[indice_etudiant] / somme_coeff2;
                }
            }
        }
    }
}

// Fonction qui permet d'initialiser toutes les moyennes d'un etudiant a 0
avant de les calculer. Elle reçoit un pointeur vers une structure formation et
un
// entier correspondant à l'indice de l'etudiant. Elle ne renvoit rien.

void moyenne_in(Formation* f, const unsigned int indice_etudiant) {
    for (int y = 0; y < NB_SEMESTRES; ++y) {
        for (unsigned int u = 0; u < f->nbUE; ++u) {
            for (unsigned int i = 0; i < f->s[y].nbMatieres; ++i) {
                f->s[y].tm[u].mo1[i].Moyennes[indice_etudiant] = 0;
            }
            f->s[y].tm[u].Moyenne_matiere[indice_etudiant] = 0;
            f->mo[u].moyenne_annuelle[indice_etudiant] = 0;

        }
    }
}

/**********************************************************************Debut
de la commande
C2**********************************************************************
********/

//Fonction qui permet de definir le nombre d'UE d'une structure. Elle reçoit
un pointeur vers une structure et ne renvoit rien.

void formation(Formation* g) {
```

```
    unsigned int Nombre_UE;
    scanf("%d", &Nombre_UE);
    if (MIN_UE > Nombre_UE || Nombre_UE > MAX_UE) {
        printf("Le nombre d'UE est incorrect\n");
    }
    else if (g->nbUE > 0) {
        printf("Le nombre d'UE est deja defini\n");
    }
    else {
        g->nbUE = Nombre_UE;
        printf("Le nombre d'UE est defini\n");
    }
}
/**************************************************************************Fin de
la commande
C2**************************************************************************
********/
/**************************************************************************Debut
de la commande
C3**************************************************************************
********/

//Fonction qui permet de definir une epreuve dans la formation, elle reçoit un
pointeur vers une structure formation et ne renvoit rien.

void epreuve(Formation* f) {

    if (f->nbUE > MAX_UE || f->nbUE < MIN_UE) {
        printf("Le nombre d'UE n'est pas defini\n");
        return;
    }
    unsigned int numero_semestre, coeffnul = 0;
    CH30 NOM_EPREUVE = "", NOM_MATIERE = "";
    bool trouvematiere = false, trouveepreuve = false;
    float tab[6];
    scanf("%d", &numero_semestre);
    scanf("%s", NOM_MATIERE);
    scanf("%s", NOM_EPREUVE);
    for (unsigned int p = 0; p < f->nbUE; p++) {  // Recuperation des
coefficent
        scanf("%f", &tab[p]);
        if (tab[p] == 0)
            coeffnul += 1;  // Comptabilisation des coefficient nul
    }

    if (numero_semestre != 1 && numero_semestre != 2) {
        printf("Le numero de semestre est incorrect\n");
        return;
    }
```

```
    trouvematiere = trouve_matiere(f, numero_semestre, NOM_MATIERE,
trouvematiere);

    if (trouvematiere == true) {
        trouveepreuve = trouve_epreuve(f, numero_semestre, NOM_EPREUVE,
trouveepreuve);
        if (trouveepreuve == true) {
            printf("Une meme epreuve existe deja\n");
            return;
        }
        else {
            if (coeffnul == f->nbUE) { // Verification coefficients tous nul
                printf("Au moins un des coefficients est incorrect\n");
                return;
            }
            else {
                for (unsigned int i = 0; i < f->nbUE; ++i) {
                    if (tab[i] < 0) { // Verification coefficients tous
positifs
                        printf("Au moins un des coefficients est
incorrect\n");
                        return;
                    }
                }
            }
            Epreuve epreuve = { 0 };
            strcpy(f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_matiere].e[f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_matiere].indice_next_epreuve].nom, NOM_EPREUVE); //Ajouer une
epreuve
            printf("Epreuve ajoutee a la formation\n");
            f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_matiere].nBEpreuves += 1; // Comptabiliser les épreuves ajoutees
            for (unsigned int i = 0; i < f->nbUE; ++i) {
                f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_matiere].e[f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_matiere].indice_next_epreuve].coef[i] = tab[i]; //Copier les
coefficients du tableau dans la structure
            }
            f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_matiere].indice_next_epreuve += 1;
        }
    }
    else {
        if (coeffnul == f->nbUE) { // Verification coefficients tous nul
            printf("Au moins un des coefficients est incorrect\n");
            return;
        }
```

```
        else {
            for (unsigned int i = 0; i < f->nbUE; ++i) {
                if (tab[i] < 0) { // Verification coefficients tous positifs
                    printf("Au moins un des coefficients est incorrect\n");
                    return;
                }
            }
        }
        strcpy(f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_next_matiere].nom, NOM_MATIERE); // copier la matiere dans la
structure

        printf("Matiere ajoutee a la formation\n");

        f->s[numero_semestre - 1].nbMatieres += 1; // Comptabiliser les
matières ajoutees

        strcpy(f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_next_matiere].e[f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_next_matiere].indice_next_epreuve].nom, NOM_EPREUVE); // Ajouter
l'épreuve dans la structure

        printf("Epreuve ajoutee a la formation\n");

        f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_next_matiere].nBEpreuves += 1; //Comptabiliser les epreuves ajoutees

        for (unsigned int i = 0; i < f->nbUE; ++i) {
            f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_next_matiere].e[f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_next_matiere].indice_next_epreuve].coef[i] = tab[i]; // Ajouter les
coefficients dans la structure
        }
        f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_next_matiere].indice_next_epreuve += 1;

        f->s[numero_semestre - 1].indice_next_matiere += 1;
    }
}
/**********************************************************************Fin de
la commande
C3**********************************************************************
********/
/**********************************************************************Debut
de la commande
C4**********************************************************************
********/
```

```
// Fonction qui permet de verifier les coefficients d'un trimestre donnee par
l'utilisateur. Elle reçoit un pointeur vers une structure formation et ne
renvoit rien.

void coefficients(Formation* f) {

    if (f->nbUE > MAX_UE || f->nbUE < MIN_UE) {
        printf("Le nombre d'UE n'est pas defini\n");
        return;
    }
    unsigned int numero_semestres, nombre_epreuve_vide = 0, nombre_ue_valide =
0;
    scanf_s("%d", &numero_semestres);
    if (numero_semestres != 1 && numero_semestres != 2) {
        printf("Le numero de semestre est incorrect\n");
        return;
    }
    if (f->s[numero_semestres - 1].nbMatieres == 0) {   //Verifier le nombre
de matiere
        printf("Le semestre ne contient aucune epreuve\n");
        return;
    }
    else {
        for (unsigned int z = 0; z < MAX_MATIERES; ++z) {
            if (f->s[numero_semestres - 1].m[z].nBEpreuves == 0) { // Verifier
le nombre d'épreuves
                nombre_epreuve_vide += 1;
                if (nombre_epreuve_vide == MAX_MATIERES) {
                    printf("Le semestre ne contient aucune epreuve\n");
                    return;
                }
            }
        }
    }

    for (unsigned int p = 0; p < f->nbUE; ++p) {
        unsigned int nombre_coeff_nul = 0;
        unsigned int nombre_coeff = 0;
        for (unsigned int q = 0; q < f->s[numero_semestres - 1].nbMatieres;
++q) {
            for (unsigned int r = 0; r < f->s[numero_semestres -
1].m[q].nBEpreuves; ++r) {
                nombre_coeff += 1;
                if (f->s[numero_semestres - 1].m[q].e[r].coef[p] == 0) //
Comptabiliser Coeff null
                    nombre_coeff_nul += 1;
            }
        }
```

```
        if (nombre_coeff_nul == nombre_coeff) { // Comparer le nombre de coeff
nul, au nombre de coeff, par UE
            printf("Les coefficients d'au moins une UE de ce semestre sont
tous nuls\n");
            return;
        }
        else {
            nombre_ue_valide += 1;
        }
    }
    if (nombre_ue_valide == f->nbUE) {
        printf("Coefficients corrects\n");

    }
}
/**************************************************************************Fin de
la commande
C4**************************************************************************
********/
/**************************************************************************Debut
de la commande
C5**************************************************************************
********/

//Fonction qui permet la saisi de note pour un etudiant par epreuve d'une
matiere. Elle reçoit un pointeur vers une structure formation et ne renvoit
rien.

void note(Formation* f) {
    if (f->nbUE > MAX_UE || f->nbUE < MIN_UE) {
        printf("Le nombre d'UE n'est pas defini\n");
        return;
    }

    unsigned int numero_semestre;
    CH30 nom_etu = "", nom_matiere = "", nom_epreuve = "";
    float note;
    bool trouvematiere = false, trouveepreuve = false, trouveetudiant = false;
    scanf("%d", &numero_semestre);
    scanf("%s", nom_etu);
    scanf("%s", nom_matiere);
    scanf("%s", nom_epreuve);
    scanf("%f", &note);

    if (numero_semestre != 1 && numero_semestre != 2) {
        printf("Le numero de semestre est incorrect\n");
        return;
    }
```

```
    trouvematiere = trouve_matiere(f, numero_semestre, nom_matiere,
trouvematiere); // Cherche la matière dans la formation

    if (trouvematiere == true) {
        trouveepreuve = trouve_epreuve(f, numero_semestre, nom_epreuve,
trouveepreuve); // Cherche l'épreuve dans la formation

        if (trouveepreuve == true) {
            if (note < MIN_NOTE || note>MAX_NOTE) {
                printf("Note incorrecte\n");
                return;
            }
            else {
                trouveetudiant = trouve_etudiant(f, nom_etu,
trouveetudiant);// Cherche l'étudiant dans la formation

            }
            if (trouveetudiant == true) {
                if (f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_matiere].e[f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_matiere].indice_epreuve].note[f->indice_etudiant] != MAX_NOTE + 1)
// Comparaison avec la note initialisée
                    printf("Une note est deja definie pour cet etudiant\n");
                else {
                    f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_matiere].e[f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_matiere].indice_epreuve].note[f->indice_etudiant] = note; //
Attribution de la note à l'étudiant
                    printf("Note ajoutee a l'etudiant\n");
                }
            }
            else {
                strcpy(f->nom[f->indice_next_etudiant], nom_etu); // Creation
de l'étudiant dans la formation
                printf("Etudiant ajoute a la formation\n");
                f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_matiere].e[f->s[numero_semestre - 1].m[f->s[numero_semestre -
1].indice_matiere].indice_epreuve].note[f->indice_next_etudiant] = note; //
Attribution de la note à l'étudiant
                f->NBETUDIANT += 1;
                printf("Note ajoutee a l'etudiant\n");
                f->indice_next_etudiant += 1;
            }
        }
        else {
            printf("Epreuve inconnue\n");
            return;
        }
    }
```

```
    else {
        printf("Matiere inconnue\n");
        return;
    }
}
/**************************************************************************Fin
de la commande
C5**************************************************************************
********/
/**************************************************************************Début
de la commande
C6**************************************************************************
********/

//Fonction qui permet de verifier la validité des notes pour un eleve. Elle
reçoit un pointeur vers une structure formation et ne renvoit rien.

void notes(Formation* f) {
    if (f->nbUE > MAX_UE || f->nbUE < MIN_UE) {
        printf("Le nombre d'UE n'est pas defini\n");
        return;
    }

    unsigned int numero_semestre, nombre_note = 0, nombre_epreuve = 0;
    CH30 nom_etudiant = "";
    bool trouveetudiant = false;

    scanf("%d", &numero_semestre);
    scanf("%s", nom_etudiant);

    if (numero_semestre != 1 && numero_semestre != 2) {
        printf("Le numero de semestre est incorrect\n");
        return;
    }
    trouveetudiant = trouve_etudiant(f, nom_etudiant, trouveetudiant); //
Cherche l'étudiant dans la formation

    if (trouveetudiant == true) {
        if (verif_note(f, numero_semestre) == 1) { // Recherche de la note de
l'étudiant
            printf("Il manque au moins une note pour cet etudiant\n");
            return;
        }
        else
            printf("Notes correctes\n");
    }
    else {
        printf("Etudiant inconnu\n");
        return;
```

```
    }
}
/*************************************************************************Fin de
la commande
C6*************************************************************************
********/
/*************************************************************************Début
de la commande
C7*************************************************************************
********/

//Fonction qui permet d'afficher le releve semestrielle d'un eleve. Elle
reçoit un pointeur vers une structure formation et ne renvoit rien.

void releve(Formation* f) {
    if (f->nbUE > MAX_UE || f->nbUE < MIN_UE) {
        printf("Le nombre d'UE n'est pas defini\n");
        return;
    }

    unsigned int nombre_ue_valide = 0, numero_semestres;
    CH30 nom_etu = "";
    bool trouveetudiant = false;
    scanf("%d", &numero_semestres);
    scanf("%s", nom_etu);

    if (numero_semestres != 1 && numero_semestres != 2) {
        printf("Le numero de semestre est incorrect\n");
        return;
    }
    trouveetudiant = trouve_etudiant(f, nom_etu, trouveetudiant); // Cherche
l'étudiant dans la formation

    moyenne_in(f, f->indice_etudiant); // Permet d'initaliser les moyenne de
l'étudiant a 0 pour eviter les erreurs de calcul

    if (trouveetudiant == false) {
        printf("Etudiant inconnu\n");
        return;
    }
    for (unsigned int p = 0; p < f->nbUE; ++p) {
        unsigned int nombre_coeff_nul = 0;
        unsigned int coeff = 0;
        for (unsigned int q = 0; q < f->s[numero_semestres - 1].nbMatieres;
++q) {
            for (unsigned int r = 0; r < f->s[numero_semestres -
1].m[q].nBEpreuves; ++r) {
                coeff += 1;
```

```
                if (f->s[numero_semestres - 1].m[q].e[r].coef[p] == 0) //
Verification coeff nul
                    nombre_coeff_nul += 1;
            }
        }
        if (nombre_coeff_nul == coeff) {
            printf("Les coefficients de ce semestre sont incorrects\n");
            return;
        }
    }

    if (verif_note(f, numero_semestres) == 1) {
        printf("Il manque au moins une note pour cet etudiant\n");
        return;
    }

    calcul_moyenne(f, f->indice_etudiant); // Permet de calculer les moyennes
d'un etudiants

    printf("\t\t\t");

    for (unsigned int u = 0; u < f->nbUE; ++u)
        printf("UE%d\t", u + 1);

    for (unsigned int i = 0; i < f->s[numero_semestres - 1].nbMatieres; ++i) {
        printf("\n%s", f->s[numero_semestres - 1].m[i].nom);
        for (int x = 0; x < MAX_ESPACE; ++x) {
            if (strlen(f->s[numero_semestres - 1].m[i].nom) == x) { //
Parametrage de l'affichage pour avoir des espaces égaux entre les matières et
les notes,
                for (int p = 0; p < MAX_ESPACE - x; ++p)           //entre
autre, pour aligner les notes
                    printf(" ");
            }
        }

        for (unsigned int u = 0; u < f->nbUE; ++u) {
            if (f->s[numero_semestres - 1].tm[u].mo1[i].Moyennes[f-
>indice_etudiant] >= MIN_NOTE || f->s[numero_semestres -
1].tm[u].mo1[i].Moyennes[f->indice_etudiant] <= MAX_NOTE)
                printf("\t%2.1f", floorf(f->s[numero_semestres -
1].tm[u].mo1[i].Moyennes[f->indice_etudiant] * 10) / 10); // Affichage des
notes arrondis
            else
                printf("\tND");
        }
    }

    printf("\n");
```

```
    printf("--\n");
    printf("Moyennes\t\t");

    for (unsigned int i = 0; i < f->nbUE; ++i)
        printf("%2.1f\t", floorf(f->s[numero_semestres -
1].tm[i].Moyenne_matiere[f->indice_etudiant] * 10) / 10); // Affichage des
moyennes arrondis.

    printf("\n");
}
/**************************************************************************Fin de
la commande
C7**************************************************************************
********/
/**************************************************************************Debut
de la commande
C8**************************************************************************
********/

//Fonction qui permet d'afficher le releve annuelle d'un eleve ainsi que la
decision du jury. Elle reçoit un pointeur vers une structure formation et ne
renvoit rien.

void decision(Formation* f) {
    if (f->nbUE > MAX_UE || f->nbUE < MIN_UE) {
        printf("Le nombre d'UE n'est pas defini\n");
        return;
    }

    unsigned int UE_acquise = 0, nombre_epreuve = 0, nombre_note = 0, virgule
= 0;
    CH30 nom_etu = "";
    scanf("%s", nom_etu);
    bool trouveetudiant = false;

    trouveetudiant = trouve_etudiant(f, nom_etu, trouveetudiant);
    if (trouveetudiant == false) {
        printf("Etudiant inconnu\n");
        return;
    }

    moyenne_in(f, f->indice_etudiant); //Remet toutes les moyennes de
l'étudiant à 0 avant de les calculer

    for (int i = 0; i < NB_SEMESTRES; ++i) {
        for (unsigned int x = 0; x < f->nbUE; ++x) {
            int coeff_null = 0;
            int tous_les_coeff = 0;
            for (unsigned int u = 0; u < f->s[i].nbMatieres; ++u) {
```

```
                for (unsigned int p = 0; p < f->s[i].m[u].nBEpreuves; ++p) {
                    tous_les_coeff += 1;
                    if (f->s[i].m[u].e[p].coef[x] == 0)
                        coeff_null += 1;
                }
            }
            if (coeff_null == tous_les_coeff) {
                printf("Les coefficients d'au moins un semestre sont
incorrects\n");
                return;
            }
        }
    }
    for (unsigned i = 1; i < NB_SEMESTRES + 1; ++i) {
        if (verif_note(f, i) == 1) {
            printf("Il manque au moins une note pour cet etudiant\n");
            return;
        }
    }

    calcul_moyenne(f, f->indice_etudiant); //Calcul des moyennes de l'étudiant

    for (unsigned int x = 0; x < f->nbUE; ++x) {
        for (unsigned int y = 0; y < NB_SEMESTRES; ) {
            f->mo[x].moyenne_annuelle[f->indice_etudiant] += f-
>s[y].tm[x].Moyenne_matiere[f->indice_etudiant];
            ++y;
            if (y == NB_SEMESTRES)
                f->mo[x].moyenne_annuelle[f->indice_etudiant] = f-
>mo[x].moyenne_annuelle[f->indice_etudiant] / NB_SEMESTRES;
        }
    }

    printf("\t\t\t");
    for (unsigned int u = 0; u < f->nbUE; ++u) {
        printf("UE%d\t", u + 1);
    }
    for (int i = 0; i < NB_SEMESTRES; ++i) {
        printf("\nS%d\t\t\t", i + 1);
        for (unsigned int u = 0; u < f->nbUE; ++u)
            printf("%2.1f\t", floorf(f->s[i].tm[u].Moyenne_matiere[f-
>indice_etudiant] * 10) / 10);
    }

    printf("\n--\n");
    printf("Moyennes annuelles\t");

    for (unsigned int y = 0; y < f->nbUE; ++y) {
```

```
        printf("%2.1f\t", floorf(f->mo[y].moyenne_annuelle[f->indice_etudiant]
* 10) / 10);
    }
    printf("\n");
    printf("Acquisition\t\t");
    for (unsigned int y = 0; y < f->nbUE; ++y) {
        if (f->mo[y].moyenne_annuelle[f->indice_etudiant] >= 10)
            UE_acquise += 1;
    }
    for (unsigned int y = 0; y < f->nbUE; ++y) {
        if (f->mo[y].moyenne_annuelle[f->indice_etudiant] >= 10) {
            printf("UE%d", y + 1);
            virgule += 1;
            if (UE_acquise != virgule)
                printf(", ");
        }
    }
    if (UE_acquise == 0)
        printf("Aucune");

    printf("\n");

    printf("Devenir\t");
    if (UE_acquise > f->nbUE / 2)
        printf("\t\tPassage\n");
    else
        printf("\t\tRedoublement\n");
}
/**************************************************************************Fin de
la commande
C8**************************************************************************
********/
/**************************************************************************Debut
du
main**************************************************************************
**********/

int main() {
    CH30 cde = "";
    Formation g = { 0 };
    initialiser_formation(&g);

    do {
        scanf("%s", cde);
        if (strcmp(cde, "formation") == 0)
            formation(&g);
        else if (strcmp(cde, "epreuve") == 0)
            epreuve(&g);
        else if (strcmp(cde, "coefficients") == 0)
```

```
            coefficients(&g);
        else if (strcmp(cde, "note") == 0)
            note(&g);
        else if (strcmp(cde, "notes") == 0)
            notes(&g);
        else if (strcmp(cde, "releve") == 0)
            releve(&g);
        else if (strcmp(cde, "decision") == 0)
            decision(&g);
    } while (strcmp(cde, "exit") != 0);

    return 0;
}
/**********************************************************************Fin du
main**********************************************************************
**********/
```