DOSSIER DE DÉVELOPPEMENT SAÉ 1.01 Implémentation d'un besoin client

Paul PERIGAULT | Mouhamed Nadir BSIKRI
Groupe 105 Groupe 105

Table des matières

I – Présentation du Projet

Le principe de ce projet a été de développer un interpréteur de commandes qui permet à l'utilisateur d'entrer différentes commandes au clavier afin de définir un nombre d'UE, d'entrer des épreuves pour des matières, d'ajouter un étudiant et une note pour cette étudiant, de consulter le relevé de cette étudiant (ses moyennes pondérées pour chaque matière de chaque UE selon le semestre voulu) et enfin de savoir si l'étudiant à réussi son année ou non. En bref, le projet consiste à permettre à l'utilisateur de faire la gestion d'une formation d'étudiants grâce à cet interpréteur de commandes.

Son fonctionnement est le suivant : lorsque l'utilisateur entre la commande 2 pour définir un nombre d'UE, il doit veiller à ne pas avoir déjà défini le nombre d'UE et à entrer un nombre compris entre 3 et 6. Lorsqu'il souhaite ajouter une nouvelle épreuve, il indique le nom de l'épreuve, la matière et le semestre ainsi que les coefficients pour chaque UE en veillant à ne pas ajouter une épreuve déjà existante dans une matière (pour le semestre). L'utilisateur peut vérifier si les coefficients sont valides pour chaque UE d'un semestre donné. Une fois que l'utilisateur a entré les épreuves voulues, en partant du principe qu'il pourra toujours en entrer d'autres lorsqu'il le souhaite, l'utilisateur peut ajouter des étudiants à la formation en leur assignant des notes (une à la fois) pour une épreuve. Il doit veiller à ne pas rajouter une nouvelle fois une note pour la même épreuve (de la même matière pour un semestre donné) à l'étudiant. L'épreuve et la matière doivent nécessairement existées dans la formation. Ensuite, il peut vérifier pour un étudiant existant dans la formation s'il a une note pour chaque épreuve de chaque matière d'un semestre donné. Enfin, il peut consulter le relevé des moyennes d'un élève. Il y retrouvera, pour l'étudiant donné, la moyenne pondérée des notes épreuves d'une matière pour chaque UE ainsi que la moyenne pondérée de l'ensemble des épreuves (toute matière confondue) pour chaque UE. Aussi, et pour conclure, l'utilisateur peut consulter la décision de passage à l'année suivante d'un étudiant existant dans la formation. Est figuré dessus la moyenne pondérée de l'ensemble des épreuves pour chaque UE et pour chaque semestre ainsi que la moyenne annuelle pour chaque UE. Pour les deux dernières possibilités, l'utilisateur veille à avoir affecter une note pour chaque épreuve du semestre (/des semestres) à l'étudiant existant dans la formation.

II – Organisation des tests de l'application et bilan de validation

ORGANISATION DES TESTS DE L'APPLICATION :

Tout d'abord, au début du développement de notre application, pour les premières commandes, nous avons réalisé les tests « à la main » avec le clavier dans lequel nous entrions des in pour avoir les out. Nous pouvions tester rapidement les premières commandes car les contraintes n'étaient pas très nombreuses et les différents cas possibles non plus. Cela nous permettait d'aller provoquer une erreur nous-même pour repérer celle-ci et la corriger rapidement. Cependant, au fur et à mesure de l'avancement du développement de l'application, nous devions tester des cas de plus en plus conséquents et divers, et il devenait impossible de tout tester à la main.

Nous procédions alors ainsi : Pour chaque commande réalisée, nous testions nous-même à la main le bon fonctionnement de la commande ainsi que les différents cas d'erreur possibles entrés par l'utilisateur. Ensuite, lorsque nous voulions tester de manière globale notre application pour le sprint développé avec un large nombre d'entrées pour un grand nombre de sorties, nous testions notre fichier exécutable (.exe) avec les in out (sprints) fournies par l'enseignant référent du projet. De cette manière, on était sûr qu'aucune erreur ne s'est glissée dans notre programme. Nous comparions alors notre out avec le out de l'enseignant ; si aucune différence n'a été trouvée, alors le sprint est validé et nous passions à la validation des sprints suivants. Cependant lorsqu'on avait des erreurs qu'on ne comprenait pas, nous avions recours au débogueur de Visual Studio et nous faisions des tests à la main pour un grand nombre de cas afin de trouver quelle est la cause de l'erreur. De plus, les sprints fournis par l'enseignant étaient composés d'un sprint de base et d'un sprint d'erreur. Nous testions d'abord le sprint de base qui ne contenait aucune erreur faite par l'utilisateur, puis le sprint erreur (faites par l'utilisateur) pour avoir une validation complète du sprint.

Exemple concret pour le sprint 1 : Nous avons testé à la main la commande 1 à la fin de son développent, puis la commande 2, toujours à la main. Ensuite, pour la commande 3, nous n'avons pas testé en détails les commandes 1 et 2 pour nous concentrer sur les tests de la commande 3. De même pour la commande 4. Les commandes 1 à 4 forment le sprint 1. Nous avons alors testé le jeu de test in-sp1-base.txt pour comparer notre résultat avec le out-sp1-base.txt, de même pour le in-sp1-erreur.txt. S'il n'y a aucune différence, on commence le développement des commandes suivantes, sinon, on corrige notre programme en notant les erreurs perceptibles jusqu'à valider ce sprint en le testant régulièrement.

BILAN DE VALIDATION :

De notre côté, nous avons validé les sprint 1 (base+ erreur), 2(base + erreur), 3(base+ erreur) fournies par l’enseignant référent du projet sur Moodle. Nous avons validé auprès de notre enseignant, lors des recettes, le sprint 3 sans le /w car les alignements sont corrects. Nous n’avons pas
eu à modifier notre code, le sprint est passé directement et le professeur ne nous a pas fait de remarques particulières.

III – Bilan

A – Difficultés rencontrées

En ce qui me concerne (Paul), j'ai eu beaucoup de mal à comprendre les structures de données Struct et leurs manipulations ainsi que trouver les bons chemins d'accès à certaines données. Mon binôme a pris le temps de m'expliquer et j'ai pu comprendre l'une des notions les plus importantes de ce projet.

Pour ma part (Mouhamed), Trouver une manière rendre le code moins lourd avec des petites fonctions auxiliaires n'a pas été simple. En effet, il fallait réfléchir à ce qu'il faut alléger dans notre code pour pouvoir extraire de petites fonctions réalisant de manière général les actions répétitives. Ainsi, j'ai appris à travailler ma logique et à poser ma réflexion sur un papier avant même de programmer ma fonction de manière concrète. Aussi, définir le type de données pour créer nos variables selon la situation n'a pas été chose facile. Là encore, il faut prendre en compte les variables et leur utilité selon nos besoins. Enfin, pour la commande 7, il a été compliqué de faire les alignements pour avoir un tableau correct. En effet, il fallait trouver le bon nombre d'espaces pour séparer les différents affichages. Nous avons pris un certain temps pour comprendre comment définir le nombre d'espaces nécessaires, mais nous y sommes parvenus malgré tout. Enfin, il a été difficile de s'organiser en binôme pour trouver le temps nécessaire de travailler sur le projet. En effet, au-delà des créneaux SAE, il est compliqué de trouver un temps où nous pouvions débriefer des difficultés rencontrées, des choses que nous avons réalisées et faire la mise en commun de nos idées. Nous avons donc dû redoubler d'efforts pour travailler et nous concentrer sur cette SAE en plus des autres que nous avions déjà.

B – Améliorations suggérées

Nous pensons que créer des fonctions plus courtes et faire une fonction main plus courte pourrait être plus lisible et compréhensible. Nous aurions pu aussi faire en sorte de simplifier le placement des notes de l'étudiant dans son tableau de notes. En effet, nous n'avons trouvé qu'un moyen assez étrange, mais qui fonctionne, pour stocker les notes de l'étudiant. Nous aurions dû réfléchir à plusieurs petites choses pour améliorer notre code telles que l'utilisation de structures de données plus faciles que d'autres à certains endroits, la définition de types pour faciliter la compréhension de certaines données (à quoi elles servent concrètement).

C – Réussites et Progression

Pour un premier projet nous avons réussi à comprendre les notions essentielles du C en général. Nous avons appris à implémenter dans un programme un besoin client assez long et assez détaillé écrit en langage naturel en développant les besoins du client sans ne rien oublier. Nous avons aussi appliqué les enseignements du cours de manière concrète et

explicite. Il est assez satisfaisant de ne pas avoir abandonné le projet et de l'avoir presque réalisé dans son entièreté. Il est aussi satisfaisant de remarquer une progression de nos compétences en langage C au fil de l'avancement du développement de l'application. Aussi, nous avons pu remarquer l'aisance et la maîtrise que nous avons acquises pour dorénavant pouvoir travailler l'entièreté du temps en autonomie pour nos projets. En bref, ce cadre de travail avec lequel nous n'étions pas du tout familier fait maintenant partie de manière permanente de notre quotidien en tant qu'étudiant en informatique. Enfin, nous avons réussis à programmer un code entier assez long et précis alors que nous avions l'habitude de petits programmes à écrire.

IV - ANNEXES

La trace d'exécution du test du sprint de plus haut niveau atteint.

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

note 1 Paul Programmation Projet 12

Etudiant ajoute a la formation

Note ajoutee a l'etudiant

epreuve 1 Programmation DST 2 3 0

Epreuve ajoutee a la formation

note 1 Paul Programmation DST 9

Note ajoutee a l'etudiant

epreuve 1 SGBD Participation 0.5 0 0.5

Matiere ajoutee a la formation

Epreuve ajoutee a la formation

note 1 Paul SGBD Participation 16

Note ajoutee a l'etudiant

epreuve 1 SGBD Rapport 1.5 0 1.5

Epreuve ajoutee a la formation

releve 1 Paul

Il manque au moins une note pour cet etudiant

note 1 Paul SGBD Rapport 12

Note ajoutee a l'etudiant

epreuve 2 Architecture Interrogation 1 0 2

Matiere ajoutee a la formation

Epreuve ajoutee a la formation

note 2 Paul Architecture Interrogation 18

Note ajoutee a l'etudiant

epreuve 2 Architecture DST 0 1 4

Epreuve ajoutee a la formation

note 2 Paul Architecture DST 12

Note ajoutee a l'etudiant

epreuve 2 Systeme QCM 2 3 0.5

Matiere ajoutee a la formation

Epreuve ajoutee a la formation

note 2 Paul Systeme QCM 7

Note ajoutee a l'etudiant

epreuve 2 Systeme Expose 3 2 0.5

Epreuve ajoutee a la formation

releve 2 Paul

Il manque au moins une note pour cet etudiant

note 2 Paul Systeme Expose 8

Note ajoutee a l'etudiant

releve 1 Paul

	UE1	UE2	UE3
Programmation	10.0	10.2	ND
SGBD	13.0	ND	13.0

--

Moyennes	11.2	10.2	13.0

releve 2 Paul

UE1 UE2 UE3

Architecture 18.0 12.0 14.0

Systeme 7.6 7.4 7.5

--

Moyennes 9.3 8.1 13.0

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

releve 1 Paule

UE1 UE2 UE3

Programmation 10.0 9.8 ND

SGBD 5.0 ND 5.0

--

Moyennes 8.0 9.8 5.0

note 2 Paulo Architecture Interrogation 17

Etudiant ajoute a la formation

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

releve 2 Paulo

	UE1	UE2	UE3
Architecture	17.0	15.0	15.6
Systeme	17.8	17.2	17.5
--			
Moyennes	17.6	16.8	15.9

Exit

Le listing complet de nos sources.

```
/*
************************************************************************
********************** */
/*
                          */
/*                                                     :::++++++++::    :::
     :::   :::+++::+++:::   */
/*    SAE_S1_01.c                                      :::        :::   :::
     :::         ::         */
/*                                                     :::        :::   :::
     :::         ++         */
/*    By: Mouhamed Nadir and
Paul                        :::++++++++::    :::        :::         ++         *
/
/*                                                     :::++++++++::    :::
     :::         ++         */
/*  Created: 2022/10/01 by Bsikri with
perigault       :::        :::   :::        :::         ++         */
/*                                                     :::        :::   :::
     :::         ::         */
/*                                                     :::++++++++::    :::+++
+++++:::         ::         */
/*   Impl?mentation d'un besoin client: Gestion d'une
formation                                          */
/*
************************************************************************
********************** */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#pragma warning(disable: 4996)
#pragma warning(disable: 6031)
#pragma warning(disable: 6202)

/*NB_SEMESTRES represente le nombre de semestres de la formation*/
/*MIN_UE et MAX_UE representent le nombre d'UE minimal et maximal de la
formation*/
/*MAX_MATIERES est le nombre de mati?res possible ? ins?rer dans la
formation*/
/*MAX_EPREUVES est le nombre d'?preuves possible ? ins?rer dans la
formation*/
```

```c
/* MAX_ETUDIANTS = 100 est le nombre d'�tudiants possible � ins�rer dans la
formation*/
/*NOVALUE est la valeur par d�faut lorsqu'il n'y a pas de note pour une
�preuve et pour mati�re donn�e*/
enum {
    NB_SEMESTRES = 2,
    MIN_UE = 3,
    MAX_UE = 6,
    MAX_MATIERES = 10,
    MAX_EPREUVES = 5,
    MAX_ETUDIANTS = 100,
    MAX_CHAR = 30,
    NOVALUE = -1,
    CONSTANTE = 50,
    CONSTANTE2 = 5,
};

/*cr�ation d'un type uint pour faciliter l'�criture de "unsigned int" par la
suite.*/
typedef unsigned int uint;
/*definition d'un type enum Bool avec true et false qui prennent
respectivement les valeurs 0 et 1*/
typedef enum { true = 1, false = 0 }Bool;

/*type struct Etudiant
* nom est un char qui repr�sente le nom de l'�tudiant
* notes est un tableau qui stock l'ensemble des notes de l'�tudiant*/
typedef struct {
    char nom[MAX_CHAR + 1];
    float notes[NB_SEMESTRES * (MAX_MATIERES * MAX_EPREUVES)];
}Etudiant;

/*type struct Epreuve
* nom est un char qui comporte le nom de l'�preuve
*coef est un tableau de float comportant les coefficients de l'�preuve pour
chaque UE
*/
typedef struct {
    char nom[MAX_CHAR + 1];
    float coef[MAX_UE];
} Epreuve;

/*type struct Matiere
* nom est un char qui comporte le nom de la mati�re
```

```
* nbEpreuves, uint, est le nombre d'epreuves de la mati□re
* epreuve est un tableau de struct Epreuve(5max)
*/
typedef struct {
    char nom[MAX_CHAR + 1];
    uint nbEpreuves;
    Epreuve epreuve[MAX_EPREUVES];
}Matiere;

/*type struct Semestre
*nbMatieres, uint, represente le nombre de mati□re pour un semestre
* matiere est un tableau de struct Matiere (10 max)*/
typedef struct {
    uint nbMatieres;
    Matiere matieres[MAX_MATIERES];
} Semestre;

/*type struct Formation
semestres est un tableau  de struc Semestre (2max)
etudiants est un tableau de struct Etudiant(100max)
 nbrEtudiants est le nombre d'□tudiants actuel dans la formation.
*/
typedef struct {
    uint nbUE; // nombre de coef, commun □ toutes les □preuvesSemestre
semestres[NB_SEMESTRES];
    Semestre semestres[NB_SEMESTRES];
    Etudiant etudiants[MAX_ETUDIANTS];
    uint nbrEtudiants;
} Formation;

/*La fonction Formation1 prend en param□tre un pointeur Formation et v□rifie
que
le nombre d'UE de la formation et bien situ□ entre 3 et 6. Elle ne renvoie
rien.*/
void Formation1(Formation* numberue);

/*la fonction VerifCoeffs prend en parm□tres une structure Formation et un
tableau coeff non modifiable.
Elle retourne la somme des valeurs du tableau coeffs dans une variable de type
float.*/
float VerifCoeffs(const Formation* Form, const float coeffs[]);

/*La fonction epreuve prend en param□tres un pointeur Formation, un entier,
deux variables char et un tableau de float.
```

```
elle int�gre le nom d'une mati�re, le nom d'une epreuve et ses coeffs, sous
certaines conditions, � la sturcture formation. Elle ne renvoie rien*/
void epreuve(Formation* formation, uint numsem, const char nommat[], const
char nomeprv[], const float coeffs[]);

/*La fonction Coefficients prend en param�tres un pointeur vers une structure
Formation et un pointeur vers un entier non sign� numerosem.
elle verifie pour chaque UE que tous les coefficients de l'UE sont non nuls.
Elle ne renvoie rien.*/
void Coefficients(const Formation* formation, uint numerosem);

/*La fonction PrepaForm prend en argument un pointeur de structure
Formation.Elle initialise
le nombre de matieres et d'epreuve � zero afin qu'on puisse s'en servir comme
indice
et placer les mati�res,�preuves,�tudiants et notes futurs au bon endroit.
Elle ne renvoie rien.*/
void PreparForm(Formation* form);

/*La fonction IndNote prends en param�tres trois uint numerosem, j et k
et renvoie l'indice du tableau de notes dans lequel on doit placer la note de
l'�tudiant
ou l'indice du tableau de notes dans lequel est plac� une note*/
int IndNote(uint numerosem, uint j, uint k);

/*La fonction AjoutNote prend en param�tres un pointeur Formation, un uint
numerosem, un const char nometudiant
,un const char nommat, un const char nomeprv et un float noteetud. Elle ajoute
la note noteetud � l'etudiant nometudiant
pour le semestre numerosem-1 pour l'�preuve nomeprv de la mati�re nommat.
Elle ne renvoie rien*/
void AjoutNote(Formation* form, uint numerosem, const char nometudiant[],
const char nommat[], const char nomeprv[], float noteetud);

/*La fonction SemestreValid prends en param�tre un pointeur Formation (non
modifi�), un uint numerosem et renvoie un Bool
selon si le numerosem est valide ou non,c'est � dire s'il n'est pas �gal �
NB_SEMESTRES ou � NB_SEMESTRES-1*/
Bool SemestreValid(const Formation* form, const uint numerosem);

/*la fonction Note prend en param�tre un pointeur Formation form, un uint
numerosem et un const char nometudiant
et v�rifie si l'�tudiant nometudiant a une note pour l'ensemble des �preuves
du semestre numerosem*/
```

```
void Note(const Formation* form, uint numerosem, const char nometudiant[]);

/*La fonction VerifsCoeffsUE prend en param�tres un Pointeur Formtation
qu'elle ne modifie pas, ainsi qu'un uint numerosem
,un numero de semestre, et v�rifie si les coefficients de chaque UE du
semestre numerosem sont corrects ou non en renvoyant
un Bool*/
Bool VerifsCoeffsUE(const Formation* form, uint numerosem);

/*La fonction PlusLongNom prend en param�tres un Pointeur Formtation qu'elle
ne modifie pas
, ainsi qu'un uint numerosem,un numero de semestre,
et renvoie la longueur du nom de la mati�re du semestre de la formation qui a
le nom le plus long.
Le mot le plus long par d�faut est "Moyennes"*/
uint PlusLongNom(const Formation* form, uint numerosem);

/*La fonction Releve prends en param�tres un pointeur Formation, un uint
numerosem, et un nometudiant.
Elle affiche le relev� de moyenne pour l'�tudiant nometudiant et pour le
semestre numerosem*/
void Releve(const Formation* formation, uint numerosem, const char
nometudiant[]);

void main() {
    char command[MAX_CHAR + 1];

    /*declaration de notre structure formation*/
    Formation formation;
    formation.nbUE = 0;

    /*variable servant � savoir si le nombre d'UE est d�j� d�fini ou non*/
    uint nbdef = 0;

    /*variables pour r�cup�rer les donn�es saisies par l'utilisateur*/
    uint numerosem;
    char nommat[MAX_CHAR + 1];
    char nomeprv[MAX_CHAR + 1];
    float coeffs[MAX_UE];
    char nometudiant[MAX_CHAR];
    float noteetud;

    /*appel de la fonction prepaform afin de mettre le nombre de mati�res et
d'�preuves � z�ro
```

```
    et de s'en servir comme indices par la suite pour savoir o placer nos
epreuves et nos mati res*/
    PreparForm(&formation);

    do {
        scanf("%s", command);
        if (strcmp(command, "formation") == 0) {
            //commande 2
            nbdef = formation.nbUE;
            if (nbdef != 0) {
                printf("Le nombre d'UE est deja defini\n");
            }
            else {
                scanf("%u", &formation.nbUE);
                Formation1(&formation);
            }
        }
        else if (strcmp(command, "epreuve") == 0) {
            //commande3
            if (formation.nbUE == 0) {
                printf("Le nombre d'UE n'est pas defini\n");
            }
            else {
                //variables pour la commande3

                scanf("%u %s %s", &numerosem, &nommat, &nomeprv);
                for (unsigned int i = 0; i < formation.nbUE; ++i) {
                    scanf("%f", &coeffs[i]);
                }
                epreuve(&formation, numerosem, nommat, nomeprv, coeffs);
            }
        }
        else if (strcmp(command, "coefficients") == 0) {
            //commande4
            if (formation.nbUE == 0) {
                printf("Le nombre d'UE n'est pas defini\n");
            }
            else {
                uint numerosem;
                scanf("%u", &numerosem);
                Coefficients(&formation, numerosem);
            }
        }//fin commande 4
        else if (strcmp(command, "note") == 0) {
```

```
            //commande 5
            if (formation.nbUE == 0) {
                printf("Le nombre d'UE n'est pas defini\n");
            }
            else {
                scanf("%u %s %s %s %f", &numerosem, &nometudiant, &nommat,
&nomeprv, &noteetud);
                AjoutNote(&formation, numerosem, nometudiant, nommat, nomeprv,
noteetud);
            }
        }
        else if (strcmp(command, "notes") == 0) {
            //commande 6
            if (formation.nbUE == 0) {
                printf("Le nombre d'UE n'est pas defini\n");
            }
            else {
                scanf("%u", &numerosem);
                scanf("%s", &nometudiant);

                Note(&formation, numerosem, nometudiant);
            }
        }
        else if (strcmp(command, "releve") == 0) {
            //commande 7
            if (formation.nbUE == 0) {
                printf("Le nombre d'UE n'est pas defini\n");
            }
            else {
                scanf("%u", &numerosem);
                scanf("%s", &nometudiant);

                Releve(&formation, numerosem, nometudiant);
            }
        }
    } while (strcmp(command, "exit") != 0);/* Commande 1, fait quitter le
programme.*/
}

Bool SemestreValid(const Formation* form, const uint numerosem) {
    Bool bool = true;
    if (numerosem < NB_SEMESTRES - 1 || numerosem > NB_SEMESTRES) {
        bool = false;
    }
```

```
    return bool;
}

void PreparForm(Formation* form) {
    for (uint i = 0; i < NB_SEMESTRES; ++i) {
        form->semestres[i].nbMatieres = 0;
    }

    for (uint j = 0; j < NB_SEMESTRES; ++j) {
        for (uint k = 0; k < MAX_MATIERES; ++k) {
            form->semestres[j].matieres[k].nbEpreuves = 0;
        }
    }
    form->nbrEtudiants = 0;

    for (uint l = 0; l < MAX_ETUDIANTS; ++l) {
        for (uint m = 0; m < NB_SEMESTRES * (MAX_MATIERES * MAX_EPREUVES);
++m) {
            form->etudiants[l].notes[m] = NOVALUE;
        }
    }
}

void Formation1(Formation* formation) {
    if (formation->nbUE < MIN_UE || formation->nbUE > MAX_UE) {
        printf("Le nombre d'UE est incorrect\n");
        formation->nbUE = 0;
    }
    else {
        printf("Le nombre d'UE est defini\n");
    }
}

float VerifCoeffs(const Formation* Form, const float coeffs[]) {
    float somcoeffs = 0.0;
    for (uint l = 0; l < Form->nbUE; ++l) {
        somcoeffs += coeffs[l];
    }
    return somcoeffs;
}

void epreuve(Formation* f, uint numsem, const char nommat[], const char
nomeprv[], const float coeffs[])
{
```

```
    Semestre* s = &(f->semestres[numsem - 1]);
    uint i;

    /*verification de la validit? du num?ro de semestre*/
    if (SemestreValid(f,numsem) == false)
    {
        printf("Le numero de semestre est incorrect\n");
        return;
    }

    /*parcours de toutes les mati?res acquises jusqu'? en trouver une
    qui porte le m?me nom que celle entr?e par l'utilisateur.
    */
    for (i = 0; i < s->nbMatieres; ++i) {
        Matiere* m = &(s->matieres[i]);
        if (strcmp(nommat, m->nom) == 0) {

            uint j = 0;

            /*On verifie que notre epreuve n'existe pas deja dans la mati?re,
            sinon, on l'ajoute dans notre mati?re
            */
            while (j < m->nbEpreuves && strcmp(nomeprv, m->epreuve[j].nom) !=
0)
                ++j;

            /*Si j est inferieur ? notre nombre d'epreuve, c'est que l'autre
condition du while s'est arret?e.
            Cela veut dire que nous avons trouv? notre epreuve.*/
            if (j < m->nbEpreuves) {
                printf("Une meme epreuve existe deja\n");
                return;
            }
            /*on commence par v?rifier nos coefficients.
            Si le coefficient est n?gatif, on affiche un message et on quitte
la fonction.*/
            for (uint k = 0; k < f->nbUE; ++k) {
                if (coeffs[k] < 0) {
                    printf("Au moins un des coefficients est incorrect\n");
                    return;
                }
            }
            //appel ? verifcoeffs pour v?rifier qu'au moins un des
coefficients est sup?rieur ? z?ro.
```

```
            if (VerifCoeffs(f, coeffs) == 0) {
                printf("Au moins un des coefficients est incorrect\n");
                return;
            }
            /*conditions verifi�es,
            on ajoute notre epreuve  notre formation pour la mati�re
concern�e.*/
            strcpy(m->epreuve[j].nom, nomeprv);
            /*on ajoute 1  notre nombre d'epreuve pour la mati�re
concern�e.*/
            ++(m->nbEpreuves);
            for (uint l = 0; l < f->nbUE; ++l) {
                m->epreuve[j].coef[l] = coeffs[l];
            }
            printf("Epreuve ajoutee a la formation\n");
            return;
        }
    }

    /** mati�re entr�e par l'utilisateur ne porte le nom d'aucune des
mati�res d�j acquises.
    on l'ajoute alors  notre formation.
    on ajoute l'�preuve dans cette mati�re m�me.*/
    if (i == s->nbMatieres) {
        if (VerifCoeffs(f, coeffs) == 0) {
            printf("Au moins un des coefficients est incorrect\n");
            return;
        }
        for (uint k = 0; k < f->nbUE; ++k) {
            if (coeffs[k] < 0) {
                printf("Au moins un des coefficients est incorrect\n");
                return;
            }
        }
        /*Sinon, les coeffs sont valides, on ajoute notre premi�re preuve 
notre premi�re mati�re,
        elle-m�me ajout�e  notre formation.*/
        strcpy(s->matieres[i].nom, nommat);
        ++(s->nbMatieres);
        printf("Matiere ajoutee a la formation\n");

        strcpy(s->matieres[i].epreuve[0].nom, nomeprv);
        (s->matieres[i].nbEpreuves) = 1;
```

```
        /*on integre les coefficients  la premi re epreuve de la premi re
mati re de la formation.*/
        for (uint l = 0; l < f->nbUE; ++l) {
            s->matieres[i].epreuve[0].coef[l] = coeffs[l];
        }
        printf("Epreuve ajoutee a la formation\n");
        return;
    }
}

void Coefficients(const Formation* f, uint numerosem) {
    Semestre* s = &(f->semestres[numerosem - 1]);
    float somcoefue = 0.;
    //verification de la validit  du num ro de semestre
    if (SemestreValid(f,numerosem)==false) {
        printf("Le numero de semestre est incorrect\n");
        return;
    }
    //S'il n'y a aucune mati re, c'est qu'il n'y a aucune epreuve.
    if (s->nbMatieres == 0) {
        printf("Le semestre ne contient aucune epreuve\n");
        return;
    }
    if (VerifsCoeffsUE(f, numerosem - 1) == false) {
        printf("Les coefficients de ce semestre sont incorrects\n");
        return;
    }
}

int IndNote(uint numerosem, uint j, uint k) {
    int indtabnote = CONSTANTE * numerosem + CONSTANTE2 * j + k;
    return indtabnote;
}

void AjoutNote(Formation* form, uint numerosem, const char nometudiant[],
const char nommat[], const char nomeprv[], float noteetud) {

    Semestre* s = &(form->semestres[numerosem - 1]);
    const float MIN_NOTE = 0.f, MAX_NOTE = 20.f;

    /*verification de la validit  du num ro de semestre*/
    if (SemestreValid(form, numerosem) == false){
        printf("Le numero de semestre est incorrect\n");
        return;
```

```
    }
    /*verification de la note*/
    if (noteetud < MIN_NOTE || noteetud > MAX_NOTE) {
        printf("Note incorrecte\n");
        return;
    }

    /*declaration de variables utiles afin de
    le nombre de notes ou le nombre d'etudiant par exemple*/
    uint a = 0;
    uint i = 0;
    uint j = 0;
    uint indnbretudiant = form->nbrEtudiants;

    for (a = 0; a < indnbretudiant; ++a) {
        /*on cherche l'�tudiant parmis nos �tudiants*/
        Etudiant* etud = &(form->etudiants[a]);
        if (strcmp(nometudiant, etud->nom) == 0) {

            /*parcours des mati�res*/
            for (i = 0; i < s->nbMatieres; ++i) {
                Matiere* m = &(s->matieres[i]);
                if (strcmp(nommat, m->nom) == 0) {

                    while (j < m->nbEpreuves && strcmp(nomeprv, m-
>epreuve[j].nom) != 0)
                        ++j;
                    if (j < m->nbEpreuves) {
                        if (etud->notes[IndNote(numerosem - 1, i, j)] !=
NOVALUE) {
                            printf("Une note est deja definie pour cet
etudiant\n");
                            return;
                        }
                        else {
                            etud->notes[IndNote(numerosem - 1, i, j)] =
noteetud;
                            printf("Note ajoutee a l'etudiant\n");
                            return;
                        }
                    }
                    printf("Epreuve inconnue\n");
                    return;
                }
```

```
            }
            printf("Matiere inconnue\n");
            return;
        }
    }

    /*Si l'tudiant n'existe pas, on l'ajoute  la formation*/
    if (a == indnbretudiant) {
        Etudiant* etudi = &(form->etudiants[a]);
        uint k;
        for (k = 0; k < s->nbMatieres; ++k) {
            Matiere* mat = &(s->matieres[k]);
            if (strcmp(nommat, mat->nom) == 0) {

                uint l = 0;
                while (l < mat->nbEpreuves && strcmp(nomeprv, mat-
>epreuve[l].nom) != 0)
                    ++l;
                if (l < mat->nbEpreuves) {
                    strcpy(etudi->nom, nometudiant);
                    ++(form->nbrEtudiants);
                    printf("Etudiant ajoute a la formation\n");

                    etudi->notes[IndNote(numerosem - 1, k, l)] = noteetud;
                    printf("Note ajoutee a l'etudiant\n");
                    return;
                }
                printf("Epreuve inconnue\n");
                return;
            }
        }
        printf("Matiere inconnue\n");
        return;
    }
}

void Note(const Formation* form, uint numerosem, const char nometudiant[]) {

    Semestre* s = &(form->semestres[numerosem - 1]);

    /*validit du numero de semestre*/
    if (SemestreValid(form, numerosem) == false) {
        printf("Le numero de semestre est incorrect\n");
        return;
```

```
    }

    uint indnbretudiant = form->nbrEtudiants;
    uint i = 0;

    /*parcours des notes de l'tudiant*/
    for (i = 0; i < indnbretudiant; ++i) {
        Etudiant* etud = &(form->etudiants[i]);
        if (strcmp(nometudiant, etud->nom) == 0) {

            for (uint j = 0; j < s->nbMatieres; ++j) {
                Matiere* m = &(s->matieres[j]);

                for (uint k = 0; k < m->nbEpreuves; ++k) {
                    /*si l'tudiant  une note NOVALUE, alors il lui manque
une note*/
                    if (etud->notes[IndNote(numerosem - 1, j, k)] == NOVALUE)
{
                        printf("Il manque au moins une note pour cet
etudiant\n");
                        return;
                    }

                }
            }
            printf("Notes correctes\n");
            return;
        }
    }
    /*l'tudiant n'exsite pas aprs le parcours de tous les tudiants*/
    if (i == indnbretudiant) {
        printf("Etudiant inconnu\n");
        return;
    }
}

Bool VerifsCoeffsUE(const Formation* form, uint numerosem) {
    Bool bool = true;
    Semestre* s = &(form->semestres[numerosem]);
    float somcoefue = 0.0;
    for (uint i = 0; i < form->nbUE; ++i) {
        for (uint j = 0; j < s->nbMatieres; ++j) {
            Matiere* mat = &(s->matieres[j]);
            for (uint k = 0; k < mat->nbEpreuves; ++k) {
```

```
                Epreuve* epr = &(mat->epreuve[k]);
                somcoefue += epr->coef[i];
            }
        }
        if (somcoefue == 0) {
            bool = false;
            return bool;
        }
        somcoefue = 0.0;


    }
    return bool;
}

uint PlusLongNom(const Formation* form, uint numerosem) {
    const char* Moyenne = "Moyennes";
    uint lenmax = strlen(Moyenne);
    Semestre* s = &(form->semestres[numerosem]);
    for (uint i = 0; i < s->nbMatieres; ++i) {
        Matiere* mat = &(s->matieres[i]);
        if (strlen(mat->nom) >= lenmax) {
            lenmax = strlen(mat->nom);
        }

    }
    return lenmax;
}

void Releve(const Formation* formation, uint numerosem, const char
nometudiant[]) {
    Semestre* s = &(formation->semestres[numerosem - 1]);

    //verification de la validit? du num?ro de semestre
    if (SemestreValid(formation, numerosem) == false) {
        printf("Le numero de semestre est incorrect\n");
        return;
    }

    uint a = 0;
    uint indnbretudiant;
    indnbretudiant = formation->nbrEtudiants;
    Etudiant* etudiant;
```

```
    /*on verifie l'existance de l'�tuidiant*/
    for (a = 0; a < indnbretudiant; ++a) {
        if (strcmp(nometudiant, formation->etudiants[a].nom) == 0) {
            break;

        }
    }
    if (a == indnbretudiant) {
        printf("Etudiant inconnu\n");
        return;
    }
    etudiant = &(formation->etudiants[a]);

    /*on verifie si les coefficients du semestre sont corrects*/
    if (VerifsCoeffsUE(formation, numerosem - 1) == false) {
        printf("Les coefficients de ce semestre sont incorrects\n");
        return;
    }

    /*verifiaction si l'�tudiant  toutes les notes*/
    for (uint j = 0; j < s->nbMatieres; ++j) {
        Matiere* m = &(s->matieres[j]);
        for (uint k = 0; k < m->nbEpreuves; ++k) {
            if (etudiant->notes[IndNote(numerosem - 1, j, k)] == NOVALUE) {
                printf("Il manque au moins une note pour cet etudiant\n");
                return;
            }
        }
    }

    // recherche de la longueur du nom de la mati�re ayant le nom le plus nom
dans le semestre pour les alignements du tableau
    uint pluslongnom = PlusLongNom(formation, numerosem - 1);

    /*affichages des UE avec un espacement du d�but de l'affichage �gal  la
longueur du nom le plus long +1 */
    printf("%*s", pluslongnom + 1, "");
    for (int u = 0; u < formation->nbUE; ++u) {
        if (u == 0) {
            printf(" UE%d  ", u + 1);
        }
        else if (u == (formation->nbUE) - 1) {
            printf("UE%d ", u + 1);
        }
```

```
        else
            printf("UE%d  ", u + 1);
    }
    printf("\n");

    float moypond = 0.0, totalnotes = 0.0, sommecoeffs = 0.0;

    /*On parcours les mati□res*/
    for (uint j = 0; j < s->nbMatieres; ++j) {
        Matiere* matr = &(s->matieres[j]);

        /*l'espace entre le nom de l'□preuve et la prmi□re note et □gal au
nombre de caract□res de la mati□re ayant le nom le plus long
        soustrait au nom de la mati□re affich□e +1*/
        uint strmatr = strlen(matr->nom);
        uint espace = pluslongnom - strmatr + 1;
        printf("%s%*s", matr->nom, espace, "");

        for (uint k = 0; k < formation->nbUE; ++k) {
            for (uint l = 0; l < matr->nbEpreuves; l++)
            {
                totalnotes += (etudiant->notes[IndNote(numerosem - 1, j, l)] *
matr->epreuve[l].coef[k]);
                sommecoeffs += matr->epreuve[l].coef[k];
            }
            if (sommecoeffs == 0) {
                printf("  ND ");
            }
            else {
                /*affichage de la moyenne pond□r□e de la mati□re pour une
UE*/
                moypond = totalnotes / sommecoeffs;
                if (moypond >= 10.0)
                    printf("%.1f ", floorf(moypond * 10) / 10);
                else
                    printf(" %.1f ", floorf(moypond * 10) / 10);
                totalnotes = 0.0;
                sommecoeffs = 0.0;
            }
        }
        printf("\n");
    }

    float moyUEtotal = 0.0, notesUEtotal = 0.0, somcoefUEtotal = 0.0;
```

```
    printf("--\n");
    const char* Moyenne = "Moyennes";
    uint strmoy = strlen(Moyenne);
    uint espace2 = (pluslongnom - strmoy) + 1;
    printf("%s%*s", Moyenne, espace2, "");

    for (uint k = 0; k < formation->nbUE; ++k) {
        for (uint j = 0; j < s->nbMatieres; ++j) {
            Matiere* matr = &(s->matieres[j]);
            for (uint l = 0; l < matr->nbEpreuves; l++) {
                notesUEtotal += (etudiant->notes[IndNote(numerosem - 1, j, l)]
* matr->epreuve[l].coef[k]);
                somcoefUEtotal += matr->epreuve[l].coef[k];


            }
        }
        moyUEtotal = notesUEtotal / somcoefUEtotal;
        if (moyUEtotal >= 10.0) {
            printf("%.1f ", floorf(moyUEtotal * 10) / 10);
        }
        else {
            printf(" %.1f ", floorf(moyUEtotal * 10) / 10);
        }
        notesUEtotal = 0.0;
        somcoefUEtotal = 0.0;
    }
    printf("\n");
}
```