Leveque Julia et Martins Paulo

Classe 108, groupe 9 sur Moodle

Dossier – Initiation au développement – Projet (S1.01)

Ce dossier présente une application de gestion de formation, codée en C.

IUT de Paris - Rives de Seine
Université Paris Cité

2022-2023

Tables des matières:

Présentation du projet

Rôle fonctionnel de l'application

Cette application est un interpréteur de commande, en langage C, permettant de gérer une formation universitaire.

Il permet de définir une formation et lui associer un certain nombre d'UE. Il est aussi possible d'ajouter des matières et, à chacune, des épreuves avec leur coefficient par UE. Enfin, il peut aussi ajouter des étudiants et leur attribuer des notes.

Cet interpréteur peut aussi vérifier les coefficients entrés, les notes attribuées aux étudiants.

A la fin d'un semestre, il est possible d'afficher les relevés de notes d'un étudiant et, à la fin de l'année, la note annuelle de celui-ci ainsi que la décision de passage du jury.

Les entrées et sorties de l'application

L'application prend en entrée des commandes de l'utilisateur (chaîne de caractère) ainsi que des paramètres propres à chacune (pouvant être des entiers naturels ou encore des chaînes de caractère).

Exemple: *epreuve [NuméroSemestre] [NomMatière] [NomEpreuve] [coefficients]*

Elle renvoie des réponses adéquates à l'utilisateur, en cas de réussite ou échec de la procédure.

- La fonction principale (main): Elle prend en entrée la commande de l'utilisateur et stoppe l'exécution du programme s'il écrit "exit".

- La fonction formation: Prend en entrée le nombre d'UE souhaité pour la formation et lui attribue.

- La fonction epreuve: Prend en entrée le semestre où ajouter l'épreuve, à quelle matière elle appartient et ses coefficients par UE. Elle crée au besoin la matière et l'épreuve si celles-ci n'existent pas déjà.

- La fonction coefficients: Prend en paramètre un numéro de semestre et vérifie si les coefficients de toutes les épreuves sont corrects. Elle renvoie un message informant de l'état des coefficients.

- La fonction note: Elle prend en entrée le semestre correspondant, le nom de l'étudiant, de la matière, l'épreuve puis la note à lui attribuer. Elle crée si besoin une

entrée pour l'étudiant, initialise toutes ses notes à -1 (non attribuée) et les modifie en fonction des entrées de l'utilisateur. Elle affiche aussi les éventuels problèmes rencontrés (Matière inconnue, note incorrecte ...)

- La fonction notes: Prend en entrée le numéro de semestre correspondant et le nom de l'étudiant. Elle renvoie un message quant à l'état de ses notes (Il manque une note à cet étudiant, notes correctes ...)

- La fonction releve: Prend en entrée le numéro de semestre choisi et le nom de l'étudiant. Affiche le relevé de notes de celui-ci, ses moyennes ou bien un message d'erreur.

- La fonction decision: Elle prend en entrée le nom d'un étudiant et affiche la décision du jury pour son passage de l'année, si ce n'est pas possible, elle affiche un message d'erreur.

Organisation des tests de l'application

Les tests se sont fait à partir des différents sprints fournis sur Moodle, ils étaient composés d'un fichier texte contenant un jeu de données et d'un second fichier texte contenant les résultats attendus pour ceux-ci.

Après avoir pris connaissance des commandes nécessaires à la validation des différents sprints, nous avons développé et corrigé ces dernières jusqu'à la validation dudit sprint.

Pour les premières commandes nous avons privilégié la sortie standard pour des tests plus rapides et vérifier les variations plus facilement. Pour les suivantes, nous avons créé nos propres jeux de test et avons aussi vérifié les cas qui n'étaient pas forcément abordés dans les sprints.

Enfin, nous avons testé les préconditions de chaque fonction grâce à des assertions. Elles nous ont aussi permis de supprimer des avertissements. Par exemple, nous avons vérifié le bon fonctionnement des fonctions *scanf* en comparant leur valeur de sortie au nombre de valeurs que chaque fonction est censée recevoir.

Bilan de validation

Nous avons réussi à valider tous les sprints donnés, ainsi que le sprint 5 bonus lors de la séance de TD, le tout en prenant en compte les alignements à faire.

Bilan de projet

Le projet est une réussite notamment lors du développement des 2 premières commandes qui ont été finalisées le jour du sujet.

Cependant, le développement de la commande 3 a été plus complexe car nous avons eu des difficultés à comprendre son fonctionnement et à exploiter les structures imbriquées, ce qui nous a rendu la tâche difficile.

Suite à celle-ci le développement des commandes suivantes a été fait assez rapidement grâce à notre assiduité et persévérance. Nous avons aussi été capables de mettre en place des alignements lors de l'affichage du relevé et des décisions.

De plus, nous avons essayé d'améliorer la lisibilité du code grâce à des commentaires, de la documentation pour chaque fonction et nous avons réduit toutes les lignes en dessous de 120 caractères et sous 80 caractères quand cela était possible.

Afin d'améliorer cet interpréteur nous pouvons peut-être optimiser le code afin d'avoir une complexité algorithmique réduite.

Annexes

Listing du code source

```c
/*****************************************************************************
Nom ......... : main.c
Rôle ........ : Interpréteur de commande gérant une formation universitaire
Auteurs ..... : Julia Leveque (108) et Paulo Martins (108) groupe 9 sur Moodle
Version ..... : 11/11/2022
*****************************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <assert.h>
#pragma warning(disable:4996 6262)

// Déclaration des types énumérés
enum {
    NB_SEMESTRES = 2,
    MIN_UE = 3,
    MAX_UE = 6,
    MAX_MATIERES = 10,
    MAX_EPREUVES = 5,
    MAX_ETUDIANTS = 100,
    MAX_CHAR = 30
};

// Puis des constantes
const float MIN_NOTE = 0.f, MAX_NOTE = 20.f;

// Et enfin des types définis
typedef char CH30[MAX_CHAR + 1];
typedef unsigned int uint;

typedef struct {
    // Pour chaque matière, l'étudiant à une note par épreuve
    float note[MAX_EPREUVES];
}NoteMat;

typedef struct {
```

```c
    // Pour chaque semestre, l'étudiant à plusieurs matières
    NoteMat noteMat[MAX_MATIERES];
} NotesSem;

typedef struct {
    CH30 nom; // Chaque étudiant à un nom
    NotesSem noteSem[NB_SEMESTRES]; // Et des notes dans le semestre
} Etudiant;

typedef struct {
    CH30 nom; // Une épreuve à un nom
    float coef[MAX_UE]; // Et un coefficient
} Epreuve;

typedef struct {
    CH30 nom; // Une matière a un nom
    uint nbEpreuves; // Un certain nombre d'épreuve
    Epreuve epreuves[MAX_EPREUVES]; // Un tableau contenant ces épreuves
} Matiere;

typedef struct {
    uint nbMatieres; // Chaque semestre a un certain nombre de matières
    Matiere matieres[MAX_MATIERES]; // Un tableau contenant ces matières
} Semestre;

typedef struct {
    uint nbUE; // nombre de coef, commun à toutes les épreuves
    uint nbEtudiants; // Nombre max d'étudiant dans la formation
    Semestre semestres[NB_SEMESTRES]; // Tableau contenant les semestres
    Etudiant etudiants[MAX_ETUDIANTS]; // Tableau contenant les étudiants
} Formation;


/*@brief Vérifie que le nombre d'UE d'une formation est défini.
* @param[in] forma, une structure Formation.
* @return 1 si le nombre d'UE est défini, 0 sinon.
*/
int verifUE(const Formation* forma) {
    if (!(forma->nbUE)) {
        printf("Le nombre d'UE n'est pas defini\n");
        return 0;
    }
    return 1;
}
```

```
/*@brief Vérifie qu'un étudiant existe dans la formation.
* @param[in] forma, une structure Formation.
* @param[in] nomEtu, le nom de l'étudiant.
* @return l'indice de l'étudiant s'il existe, le nombre max d'étudiant + 1 sinon.
*/
int verifEtu(const Formation* forma, const CH30 nomEtu) {
    for (uint nbEtu = 0; nbEtu < forma->nbEtudiants; ++nbEtu) {
        if (!(strcmp(forma->etudiants[nbEtu].nom, nomEtu))) {
            return nbEtu;
        }
    }
    printf("Etudiant inconnu\n");
    return MAX_ETUDIANTS + 1;
}


/*@brief Vérifie qu'une matière existe dans un semestre donnée de la formation.
* @param[in] forma, une structure Formation.
* @param[in] noSem, le numéro de semestre.
* @param[in] le nom de la matière.
* @pre noSem doit être égal à 0 ou 1.
* @return l'indice de la matière si elle existe, le nombre max de matière + 1 sinon.
*/
int verifMat(const Formation* forma, const uint noSem, const CH30 mat) {
    assert(!(noSem) || noSem == 1);
    for (uint nbMat = 0; nbMat < forma->semestres[noSem].nbMatieres; ++nbMat) {
        if (!(strcmp(forma->semestres[noSem].matieres[nbMat].nom, mat))) {
            return nbMat;
        }
    }
    printf("Matiere inconnue\n");
    return MAX_MATIERES + 1;
}


/*@brief Vérifie qu'une épreuve existe dans une matière donnée de la formation lors un semestre donné.
* @param[in] forma, une structure Formation
* @param[in] noSem, le numéro de semestre.
* @param[in] idMat, l'indice de la matière
* @param[in] epr, le nom de l'épreuve.
* @pre noSem doit être égal à 0 ou 1.
* @pre idMat de doit pas dépasser le nombre total de matière du semestre.
* @return l'indice de l'épreuve si elle existe, le nombre max d'épreuve + 1 sinon.
*/
```

```
int verifEpr(const Formation* forma, const uint noSem, const uint idMat, const CH30 epr) {
    assert(!(noSem) || noSem == 1);
    assert(idMat <= forma->semestres[noSem].nbMatieres);
    uint nbEprForma = forma->semestres[noSem].matieres[idMat].nbEpreuves;
    CH30 nom; // On mettra le nom de l'épreuve à comparer avec la chaîne epr ici
    for (uint indEpr = 0; indEpr < nbEprForma; ++indEpr) {
        strcpy(nom, forma->semestres[noSem].matieres[idMat].epreuves[indEpr].nom);
        if (!(strcmp(nom, epr))) {
            return indEpr;
        }
    }
    printf("Epreuve inconnue\n");
    return MAX_EPREUVES + 1;
}


/*@brief Défini un nombre d'UE à une formation donnée.
* @param[in, out] forma, une structure Formation.
*/
void definirFormation(Formation* forma) {
    int ue;
    const int retour = scanf("%d", &ue);
    assert(retour == 1);
    if (forma->nbUE) {
        printf("Le nombre d'UE est deja defini\n");
    }
    else if (ue >= MIN_UE && ue <= MAX_UE) {
        forma->nbUE = ue;
        printf("Le nombre d'UE est defini\n");
    }
    else {
        printf("Le nombre d'UE est incorrect\n");
    }
}


/*@brief Ajoute une épreuve à la formation, et la matière correspondante si besoin.
* @param[in] forma, une structure Formation.
*/
void ajoutEpreuve(Formation* forma) {
    uint noSem, nbMatForma = 0, nbEprForma = 0;
    uint totMat, totEpr;
    float ue_tab[MAX_UE], sum_ue = 0.;
    CH30 mat = {'\0'}, epr = {'\0'}; // Afin d'éviter le warning C6054
    if (!(verifUE(forma))) {
```

```
    return;
}
// Numéro de semestre, nom de la matière et nom de l'épreuve
const int retour = scanf("%d %s %s", &noSem, &mat, &epr);
assert(retour == 3);
noSem -= 1;
for (uint ue = 0; ue < forma->nbUE; ++ue) {
    const int retour_b = scanf("%f", &ue_tab[ue]);
    assert(retour_b == 1);
}
if (noSem != 0 && noSem != 1) {
    printf("Le numero de semestre est incorrect\n");
}
else {
    totMat = forma->semestres[noSem].nbMatieres;
    for (uint nMat = 0; nMat < totMat; ++nMat) {
        // Si une matière existante est la même que celle donnée
        if (!(strcmp(forma->semestres[noSem].matieres[nMat].nom, mat))) {
            totEpr = forma->semestres[noSem].matieres[nMat].nbEpreuves;
            for (uint nbEpr = 0; nbEpr < totEpr; ++nbEpr) {
                // Si une épreuve existante est la même que celle donnée
                if (!(strcmp(forma->semestres[noSem].matieres[nMat].epreuves[nbEpr].nom, epr))) {
                    printf("Une meme epreuve existe deja\n");
                    return;
                }
            }
        }
    }
    for (uint coef = 0; coef < forma->nbUE; ++coef) {
        if (ue_tab[coef] < 0) {
            printf("Au moins un des coefficients est incorrect\n");
            return;
        }
        sum_ue += ue_tab[coef];
        // Si la somme des coefficients est égale à 0
        if (coef == (forma->nbUE - 1) && (!(sum_ue))) {
            printf("Au moins un des coefficients est incorrect\n");
            return;
        }
    }
    // Tant qu'on n'a pas parcouru toutes les matières
    // ni trouvé de matière ayant le même nom
    while ((nbMatForma < forma->semestres[noSem].nbMatieres) &&
            strcmp(mat, forma->semestres[noSem].matieres[nbMatForma].nom)){
        ++nbMatForma;
```

```
        }
        // On ajoute la matière si elle n'existe pas
        if (!(nbMatForma < forma->semestres[noSem].nbMatieres)){
            strcpy(forma->semestres[noSem].matieres[nbMatForma].nom, mat);
            forma->semestres[noSem].nbMatieres += 1;
            forma->semestres[noSem].matieres[nbMatForma].nbEpreuves = 0;
            printf("Matiere ajoutee a la formation\n");
        }
        // Tant qu'on n'a pas parcouru toutes les épreuves
        // ni trouvé d'épreuve ayant le même nom
        while ((nbEprForma < forma->semestres[noSem].matieres[nbMatForma].nbEpreuves) &&
                strcmp(epr, forma->semestres[noSem].matieres[nbMatForma].epreuves[nbEprForma].nom)){
            ++nbEprForma;
        }
        // On ajoute l'épreuve si elle n'existe pas
        if(!(nbEprForma < forma->semestres[noSem].matieres[nbMatForma].nbEpreuves)){
            strcpy(forma->semestres[noSem].matieres[nbMatForma].epreuves[nbEprForma].nom, epr);
            forma->semestres[noSem].matieres[nbMatForma].nbEpreuves += 1;
            printf("Epreuve ajoutee a la formation\n");
            // On lui ajoute les coefficients entrés
            for (uint nb_coef = 0; nb_coef < forma->nbUE; ++nb_coef) {
                forma->semestres[noSem].matieres[nbMatForma].epreuves[nbEprForma].coef[nb_coef] =
                ue_tab[nb_coef];
            }
        }
    }
}


/*@brief Calcule la somme des coefficients de chaque UE d'un semestre donné.
* @param[in] forma, une structure Formation.
* @param[in] noSem, le numéro de semestre.
* @pre noSem doit être égal à 0 ou 1.
* @return 0 si une somme est nulle, 1 sinon.
*/
int verifCoefUE(const Formation* forma, const uint noSem) {
    assert(!(noSem) || noSem == 1);
    uint totMat, totEpr;
    float sumUE = 0;
    for (uint ue = 0; ue < forma->nbUE; ++ue) {
        sumUE = 0; // On remet la somme à 0 à chaque nouvelle UE
        totMat = forma->semestres[noSem].nbMatieres;
        for (uint nbMat = 0; nbMat < totMat; ++nbMat) {
            totEpr = forma->semestres[noSem].matieres[nbMat].nbEpreuves;
            for (uint nbEpr = 0; nbEpr < totEpr; ++nbEpr) {
```

```
                sumUE += forma->semestres[noSem].matieres[nbMat].epreuves[nbEpr].coef[ue];
            }
        }
        // Si la somme des coefficients est nulle
        if (!(sumUE)) {
            return 0;
        }
    }
    return 1;
}


/*@brief Vérifie que les coefficients d'un semestre sont tous corrects.
* @param[in] forma, une structure Formation.
*/
void verifCoef(const Formation* forma) {
    uint noSem, verifSomme;
    float sumUE = 0;
    if (!(verifUE(forma))) {
        return;
    }
    const int retour = scanf("%d", &noSem);
    assert(retour == 1);
    noSem -= 1;
    if (noSem >=0 && noSem < NB_SEMESTRES) {
        // On part du principe qu'il ne peut pas y avoir de matière vide
        // Si la matière n'existe pas alors elle n'a pas d'épreuve.
        if (!(forma->semestres[noSem].nbMatieres)) {
            printf("Le semestre ne contient aucune epreuve\n");
            return;
        }
        // Pour toutes les UEs on vérifie si la somme de leur coefficients est nulle
        verifSomme = verifCoefUE(forma, noSem);
        if (verifSomme) {
            printf("Coefficients corrects\n");
        }
        else {
            printf("Les coefficients d'au moins une UE de ce semestre sont tous nuls\n");
        }
    }
    else {
        printf("Le numero de semestre est incorrect\n");
    }
}
```

```
/*@brief Trouve ou ajoute un étudiant à la formation.
* @param[in, out] forma, une structure Formation.
* @param[in] etu, le nom de l'étudiant.
* @return l'indice où l'étudiant est enregistré.
*/
int ajoutEtudiant(Formation* forma, const CH30 etu) {
    uint cpt = 0;
    // Tant qu'on n'a pas parcouru tous les étudiants
    // ni trouvé d'étudiant ayant le même nom
    while ((cpt < forma->nbEtudiants) && strcmp(etu, forma->etudiants[cpt].nom)) {
        ++cpt;
    }
    // Si l'etudiant n'existe pas, on l'ajoute
    if (!(cpt < forma->nbEtudiants)) {
        strcpy(forma->etudiants[cpt].nom, etu);
        forma->nbEtudiants += 1;
        printf("Etudiant ajoute a la formation\n");
        // On initialise toutes ses notes à -1 (non attribuée)
        for (uint nbSem = 0; nbSem < NB_SEMESTRES; ++nbSem) {
            for (uint nbMat = 0; nbMat < MAX_MATIERES; ++nbMat) {
                for (uint nbEpr = 0; nbEpr < MAX_EPREUVES; ++nbEpr) {
                    forma->etudiants[cpt].noteSem[nbSem].noteMat[nbMat].note[nbEpr] = -1;
                }
            }
        }
    }
    return cpt;
}


/*@brief Ajoute une note à un étudiant.
* @param[in, out] forma, une structure Formation.
*/
void ajoutNote(Formation* forma) {
    uint noSem, numMat, numEpr, numEtu;
    float note;
    CH30 etu, mat, epr;
    if (!(verifUE(forma)))
        return;
    const int retour = scanf("%d %s %s %s %f", &noSem, &etu, &mat, &epr, &note);
    assert(retour == 5);
    noSem -= 1;
    if (noSem >= 0 && noSem < NB_SEMESTRES) {
        if (note >= MIN_NOTE && note <= MAX_NOTE) {
```

```
            // On détermine l'indice de la matière
            numMat = verifMat(forma, noSem, mat);
            if (numMat != MAX_MATIERES + 1) {
                // On détermine l'indice de l'épreuve
                numEpr = verifEpr(forma, noSem, numMat, epr);
                if (numEpr != MAX_EPREUVES + 1) {
                    // On détermine l'indice de l'étudiant
                    numEtu = ajoutEtudiant(forma, etu);
                    // Si sa note n'est pas déjà attribuée on la change
                    if (forma->etudiants[numEtu].noteSem[noSem].noteMat[numMat].note[numEpr] == -1) {
                        forma->etudiants[numEtu].noteSem[noSem].noteMat[numMat].note[numEpr] = note;
                        printf("Note ajoutee a l'etudiant\n");
                    }
                    else {
                        printf("Une note est deja definie pour cet etudiant\n");
                    }
                }
            }
        }
        else {
            printf("Note incorrecte\n");
        }
    }
    else {
        printf("Le numero de semestre est incorrect\n");
    }
}


/*@brief Vérifie si, dans un semestre donné, il manque une note à un étudiant.
* @param[in] forma, une structure Formation.
* @param[in] noSem, le numéro du semestre.
* @param[in] idEtu, l'indice de l'étudiant.
* @pre noSem doit être égal à 0 ou 1.
* @pre idEtu ne doit pas dépasser le nombre total d'étudiant de la formation.
* @return 1 si l'étudiant a toutes ses notes attribuées, 0 sinon.
*/
int parcoursNote(const Formation* forma, const uint noSem, const uint idEtu) {
    assert(!(noSem) || noSem == 1);
    assert(idEtu <= forma->nbEtudiants);
    uint totMat = forma->semestres[noSem].nbMatieres, totEpr;
    for (uint nbMat = 0; nbMat < totMat; ++nbMat){
        totEpr = forma->semestres[noSem].matieres[nbMat].nbEpreuves;
        for (uint nbEpr = 0; nbEpr < totEpr; ++nbEpr) {
            if (forma->etudiants[idEtu].noteSem[noSem].noteMat[nbMat].note[nbEpr] == -1) {
```

```
                return 0;
            }
        }
    }
    return 1;
}


/*@brief Vérifie qu'un étudiant à une note pour chaque épreuve d'un semestre.
* @param[in, out] forma, une structure Formation.
*/
void verifNote(Formation* forma) {
    uint noSem, idEtu = MAX_ETUDIANTS + 1, parcNote;
    CH30 nomEtu;
    if (!(verifUE(forma))) {
        return;
    }
    const int retour = scanf("%d %s", &noSem, &nomEtu);
    assert(retour == 2);
    noSem -= 1;
    if (noSem < NB_SEMESTRES && noSem >= 0) {
        idEtu = verifEtu(forma, nomEtu); // On détermine l'indice de l'étudiant
        if (idEtu != MAX_ETUDIANTS + 1) {
            parcNote = parcoursNote(forma, noSem, idEtu);
            if (parcNote) {
                printf("Notes correctes\n");
            }
            else {
                printf("Il manque au moins une note pour cet etudiant\n");
            }
        }
    }
    else {
        printf("Le numero de semestre est incorrect\n");
    }
}


/*@brief Calcule la moyenne d'une UE d'un étudiant pour un semestre donné.
* @param[in] forma, une structure Formation.
* @param[in] noSem, le numéro du semestre.
* @param[in] idEtu, l'indice de l'étudiant.
* @param[in] ue, l'UE de la moyenne calculée.
* @pre noSem doit être égal à 0 ou 1.
* @pre idEtu ne doit pas dépasser le nombre total d'étudiant de la formation.
```

```
* @pre ue ne doit pas dépasser le nombre total d'UE de la formation.
* @return la moyenne de l'UE.
*/
float moyenneEtu(const Formation* forma, const uint noSem, const uint idEtu, const uint ue){
    assert(!(noSem) || noSem == 1);
    assert(idEtu <= forma->nbEtudiants);
    assert(ue <= forma->nbUE);
    uint totMat = forma->semestres[noSem].nbMatieres, totEpr;
    float moyenneUE = 0., sumCoefUE = 0., coef = 0., note = 0., moy = 0.;
    for(uint nbMat= 0; nbMat < totMat; ++nbMat){
        totEpr = forma->semestres[noSem].matieres[nbMat].nbEpreuves;
        for(uint nbEpr = 0; nbEpr < totEpr; ++nbEpr){
            coef = forma->semestres[noSem].matieres[nbMat].epreuves[nbEpr].coef[ue];
            note = forma->etudiants[idEtu].noteSem[noSem].noteMat[nbMat].note[nbEpr];
            moyenneUE += (coef * note);
            sumCoefUE += forma->semestres[noSem].matieres[nbMat].epreuves[nbEpr].coef[ue];
        }
    }
    // On fait la moyenne
    moy = moyenneUE / sumCoefUE;
    if (moy < 10) {
        printf(" ");
    }
    printf("%.1f ", floorf(10 * (moy)) / 10);
    return moy;
}


/*@brief Affiche le relevé de note d'un étudiant pour un semestre donné.
* @param[in] forma, une structure Formation.
* @param[in] noSem, le numéro du semestre
* @param[in] idEtu, l'indice de l'étudiant.
* @pre noSem doit être égal à 0 ou 1.
* @pre idEtu ne doit pas dépasser le nombre total d'étudiant de la formation
*/
void affichageReleve(const Formation* forma, const uint noSem, const uint idEtu){
    assert(!(noSem) || noSem == 1);
    assert(idEtu <= forma->nbEtudiants);
    uint totMat = forma->semestres[noSem].nbMatieres, totEpr;
    size_t maxMat = 0;
    float noteEtu = 0., sumCoef = 0., sumNote = 0., moy = 0.;
    float noteActu = 0., coefActu = 0.;
    // Détermine le nom de matière le plus long pour l'affichage
    for (uint mat = 0; mat < totMat; ++mat) {
        if (strlen(forma->semestres[noSem].matieres[mat].nom) > maxMat) {
```

```
            maxMat = strlen(forma->semestres[noSem].matieres[mat].nom);
        }
    }
    for (uint esp = 0; esp <= maxMat; ++esp) {
        printf(" ");
    }
    for (uint ueForma = 0; ueForma < forma->nbUE; ++ueForma) {
        printf(" UE%d ", ueForma + 1);
    }
    // On affiche le nom de la matière et les notes en fonction des UEs
    // Pour chaque matière
    for (uint nbMat = 0; nbMat < totMat; ++nbMat){
        printf("\n%s ", forma->semestres[noSem].matieres[nbMat].nom);
        for (size_t len = strlen(forma->semestres[noSem].matieres[nbMat].nom); len < maxMat; ++len) {
            printf(" ");
        }
        for (uint ue = 0; ue < forma->nbUE; ++ue) {
            noteEtu = 0.; // On remet la note de l'étudiant à 0
            sumCoef = 0.; // On remet la somme des coefficients à 0
            totEpr = forma->semestres[noSem].matieres[nbMat].nbEpreuves;
            for (uint nbEpr = 0; nbEpr < totEpr; ++nbEpr) {
                noteActu = forma->etudiants[idEtu].noteSem[noSem].noteMat[nbMat].note[nbEpr];
                coefActu = forma->semestres[noSem].matieres[nbMat].epreuves[nbEpr].coef[ue];
                noteEtu += (coefActu* noteActu);
                sumCoef += forma->semestres[noSem].matieres[nbMat].epreuves[nbEpr].coef[ue];
            }
            if (!(sumCoef)) {
                printf("  ND ");
            }
            else {
                moy = floorf(10 * (noteEtu / sumCoef)) / 10; // Arrondi
                if (moy < 10) {
                    printf(" ");
                }
                printf("%.1f ", moy);
            }
        }
    }
    printf("\n--\nMoyennes");
    for (size_t space = strlen("Moyennes"); space <= maxMat; ++space) {
        printf(" ");
    }
    for (uint nbUE = 0; nbUE < forma->nbUE; ++nbUE) {
        moyenneEtu(forma, noSem, idEtu, nbUE);
    }
```

```
    printf("\n");
}


/*@brief Détermine s'il est possible d'afficher le relevé de note d'un étudiant.
* @param[in] forma, une structure Formation.
*/
void releve(const Formation* forma) {
    uint noSem, idEtu, coefsCorrects, notesCorrectes;
    CH30 nomEtu;
    if (!(verifUE(forma))) {
        return;
    }
    const int retour = scanf("%d %s", &noSem, &nomEtu);
    assert(retour == 2);
    noSem -= 1;
    if (noSem >= 0 && noSem < NB_SEMESTRES) {
        idEtu = verifEtu(forma, nomEtu);
        if (idEtu != MAX_ETUDIANTS + 1) {
            coefsCorrects = verifCoefUE(forma, noSem);
            if (coefsCorrects) {
                notesCorrectes = parcoursNote(forma, noSem, idEtu);
                if (notesCorrectes) {
                    affichageReleve(forma, noSem, idEtu); // Affiche le relevé
                }
                else {
                    printf("Il manque au moins une note pour cet etudiant\n");
                }
            }
            else {
                printf("Les coefficients de ce semestre sont incorrects\n");
            }
        }
    }
    else {
        printf("Le numero de semestre est incorrect\n");
    }
}


/*@brief Calcule la moyenne annuelle d'une UE d'un étudiant.
* @param[in] forma, une structure Formation.
* @param[in] tabMoy, un pointeur vers un tableau contenant les moyennes de des UE par semestre.
* @param[in] ue, le numéro de l'UE.
* @param[in] ind, le nombre de moyennes d'UE calculées.
```

```
* @pre ue ne doit pas dépasser le nombre total d'UE de la formation
* @pre ind ne doit pas dépasser le nombre d'UE de la formation * NB_SEMESTRES.
* @return la moyenne annuelle de l'étudiant pour l'UE donnée.
*/
float moyenneAnnee(const Formation* forma, const float* tabMoy, uint ue, const uint ind) {
    assert(ue <= forma->nbUE);
    assert(ind <= (NB_SEMESTRES * forma->nbUE));
    float sumMoy = 0.;
    // Pour chaque semestre
    for (uint nbSem = 0; nbSem < NB_SEMESTRES; ++nbSem) {
        for (ue; ue < ind; ue += forma->nbUE) { // On prend la moyenne de l'UE
            sumMoy += tabMoy[ue];
        }
    }
    return sumMoy / NB_SEMESTRES;
}


/*@brief Affiche la décision du jury pour le passage d'un étudiant donné.
* @param[in] forma, une structure Formation.
* @param[in] idEtu, l'indice de l'étudiant.
* @pre idEtu ne doit pas dépasser le nombre total d'étudiant de la formation.
*/
void afficheDecision(const Formation* forma, const uint idEtu){
    assert(idEtu <= forma->nbEtudiants);
    float moy = 0., moyUESem[NB_SEMESTRES * MAX_UE];
    uint ind = 0, premier = 1, ue_passee[MAX_UE], decision = 0;
    uint nmbUE = forma->nbUE, uePassage;
    // Pour chaque UE, on initialise l'ue_passe[UE] à 0 (non obtenue)
    for (uint ue = 0; ue < MAX_UE; ++ue) {
        ue_passee[ue] = 0;
    }
    printf("\t\t   ");
    // On affiche le nombre d'UE
    for (uint ueForma = 0; ueForma < forma->nbUE; ++ueForma) {
        printf(" UE%d ", ueForma + 1);
    }
    for (uint nbSem = 0; nbSem < NB_SEMESTRES; ++nbSem) {
        printf("\nS%d\t\t   ", nbSem + 1);
        for (uint ueSem = 0; ueSem < forma->nbUE; ++ueSem) {
            // On récupère les moyennes de l'étudiant
            moyUESem[ind] = moyenneEtu(forma, nbSem, idEtu, ueSem);
            ++ind; // On incrémente le nombre de moyennes
        }
```

```
    }
    printf("\n--\nMoyennes annuelles ");
    // On fait les moyenne annuelles et on mémorise lesquelles sont >= 10
    for (uint nbUE = 0; nbUE < forma->nbUE; ++nbUE) {
        moy = floorf(10 * (moyenneAnnee(forma, moyUESem, nbUE, ind))) / 10;
        if (moy < 10) {
            printf(" ");
        }
        printf("%.1f ", moy);
        if (moy >= 10.0) {
            ue_passee[nbUE] = 1;
        }
    }
    printf("\nAcquisition\t   ");
    // On affiche les possibles UEs où moyenne >= 10
    for (uint ue = 0; ue < forma->nbUE; ++ue) {
        if (ue_passee[ue]) {
            if (premier) { // Si c'est la première UE
                printf("UE%d", ue + 1);
                premier = 0;
            }
            else {
                printf(", UE%d", ue + 1);
            }
            decision += 1; // On ajoute 1 au nombre d'UE passées
        }
    }
    if (!(decision)) { // Si aucune UE n'est passée
        printf("Aucune");
    }
    printf("\nDevenir\t\t   ");
    uePassage = nmbUE / 2;
    if (decision > uePassage) {
        printf("Passage\n");
    }
    else {
        printf("Redoublement\n");
    }
}


/*@brief Vérifie s'il est possible d'afficher la décision du jury pour un étudiant.
* @param[in] forma, une structure Formation.
*/
void decision(const Formation* forma){
```

```
    uint idEtu, coefsCorrects, notesCorrectes, semCorrect = 0;
    CH30 nomEtu;
    const int retour = scanf("%s", &nomEtu);
    assert(retour == 1);
    idEtu = verifEtu(forma, nomEtu);
    if (idEtu != MAX_ETUDIANTS + 1) {
        for (uint nbSem = 0; nbSem < NB_SEMESTRES; ++nbSem) {
            coefsCorrects = verifCoefUE(forma, nbSem);
            if (coefsCorrects) {
                notesCorrectes = parcoursNote(forma, nbSem, idEtu);
                if (notesCorrectes) {
                    semCorrect += 1;
                }
                else {
                    printf("Il manque au moins une note pour cet etudiant\n");
                    return;
                }
            }
            else {
                printf("Les coefficients d'au moins un semestre sont incorrects\n");
                return;
            }
        }
        if (semCorrect == 2){ // Si les deux semestres sont corrects
            afficheDecision(forma, idEtu);
        }
    }
}


/*@brief Fonction principale du programme.
* @return 0 si tout s'est bien passé.
*/
int main() {
    char cde[31] = "";
    Formation f;
    f.nbUE = 0;
    f.nbEtudiants = 0;
    for (int i = 0; i < NB_SEMESTRES; ++i) {
        f.semestres[i].nbMatieres = 0;
    }
    do {
        const int retour = scanf("%s", &cde);
        assert(retour == 1);
        if (!(strcmp(cde, "formation"))) {
```

```
            definirFormation(&f);
        }
        else if (!(strcmp(cde, "epreuve"))) {
            ajoutEpreuve(&f);
        }
        else if (!(strcmp(cde, "coefficients"))) {
            verifCoef(&f);
        }
        else if (!(strcmp(cde, "note"))) {
            ajoutNote(&f);
        }
        else if (!(strcmp(cde, "notes"))) {
            verifNote(&f);
        }
        else if (!(strcmp(cde, "releve"))) {
            releve(&f);
        }
        else if (!(strcmp(cde, "decision"))) {
            decision(&f);
        }
    } while (strcmp(cde, "exit"));
}
```

Trace d'exécution du test

```
G:\Mon Drive\BUT 1\WorkSpace\VS\PeriodeA\Sae\Sae\x64\Debug>Sae.exe <in-sp4-base.txt> res-base.txt

G:\Mon Drive\BUT 1\WorkSpace\VS\PeriodeA\Sae\Sae\x64\Debug>fc res-base.txt out-sp4-base.txt
Comparaison des fichiers res-base.txt et OUT-SP4-BASE.TXT
FC : aucune différence trouvée
```

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

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

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

Figure 1: Trace d'exécution du sprint 4 – en rouge, les données saisies par l'utilisateur, en bleu, les messages affichés par le programme.