Gestion d’une formation

ZHENG Mickaël
DOGRU Erwan
Groupe 104

Notre projet consiste à développer, c'est-à-dire concevoir, coder, tester et intégrer un interpréteur de commande pour la gestion d’une formation universitaire. La problématique professionnelle est la création d'une application avec des commandes précises à prendre en compte, qui vont permettre de fixer la structure de la formation et ainsi que les étudiants de celle-ci. Comme le fait d'affecter des notes à un étudiant précis et de réaliser des fonctions de fin de semestre et d'année.

Tables des matières

Présentation du projet

1. Le rôle fonctionnel de l'application

L'application est conçue pour permettre une gestion complète de la formation universitaire, celle-ci est composé donc plusieurs commandes pour répondre à ce besoin. Ces commandes sont saisies par l'utilisateur sur le clavier par le biais de l'interpréteur de commandes dans la fonction main.

2. Les entrées et sorties de l'application

- Le programme doit permettre la définition d'un nombre d'Unités d'Enseignement (U.E.) pour la formation, le critère est la suivante : minimum 3 et maximum 6 U.E. Une commande « formation » intervient pour ce besoin le nombre saisi pour éviter les erreurs de l'utilisateur. Sa sortie est constituée de réponse positif ou d'erreurs qui suivent l'entrée au clavier du client.
- L'enseignement universitaire est composé de matières et les matières elles-mêmes sont constituées d'épreuves. Ces matières font partie d'un semestre qui sont saisie par l'utilisateur, et chaque épreuve à des coefficients associé pour chaque U.E.
  La commande « épreuve » ajoute une matière seulement si celle-ci n'a pas été ajouté. Elle vérifie de même les coefficients saisis avec un critère précis. Sa sortie si tout est respecté affiche que l'épreuve est ajoutée, au contraire elle contient le cas échéant, un message d'erreur par rapport à la saisie de l'utilisateur.
- Le besoin du client comprend le fait de prendre le soin de vérifier les coefficients pour chaque U.E. Il s'agit ici de vérifier si les coefficients d'une U.E pour un semestre saisie ne sont pas tous nuls. Sa sortie est un message qui affiche que tout est correct, le cas échéant, un message d'erreur par rapport au saisie du semestre ou bien que les coefficients d'une U.E sont tous nuls.
- L'interpréteur de commande doit avoir la capacité de la gestion des étudiants d'une formation. Si lors de la saisie du nom de l'étudiant n'est pas reconnu, celle-ci doit l'intégrer à la formation, la commande « note » doit également être capable d'ajouter des à l'étudiant souhaité. Tout en vérifiant que la note saisie soit correcte (>0 et 20<) ou que l'étudiant n'ait pas déjà une note pour une épreuve saisie.
- La commande « notes » permet à l'utilisateur de vérifier si un étudiant à une note pour chaque épreuve d'un semestre saisie. Sa sortie affiche que toutes les notes sont correctes, au contraire, un message d'erreur par rapport au semestre saisie, l'étudiant, ou le manque de note.
- Le client peut également afficher un relevé de notes pour la fin d'un semestre saisie. Le programme doit ainsi calculer une moyenne pondérée pour chaque matière à chaque U.E. Les moyennes spécifiques et les arrondis sur une décimale demandés par le client sont respectés par le programme. Celle-ci affiche un tableau avec les U.E, matière et note saisie, puis pour chaque U.E sa moyenne. Le cas échéant, un message d'erreur par rapport à un saisie de l'utilisateur.
- A la fin de la deuxième semestre, l'utilisateur peut ajouter la décision du jury suivant les moyennes d'un étudiant donné. Sa sortie est un tableau affichant les moyennes pour chaque U.E de la formation entière. Puis compte les moyennes annuelles, ainsi que les U.E acquis et le résultat de l'étudiant qui peut prendre la forme de « passage » ou bien le « redoublement ».

Organisation des test de l'application et le bilan

- Pour vérifier la fiabilité de notre application, des jeux d'essais ont étaient organisé et mis à notre disposition sur moddle. Un jeu de données, chaque sprint donné contient un ensemble de commandes devant être supporté par notre programme (in) et les résultats attendus (out). En modifiant quelque fonction ou en créant de nouvelles, les jeux d'essais (in & out) se sont avéré efficace pour vérifier si elles fonctionnaient comme nous l'entendions.
  Cependant, nous sommes parvenus à développer les différents sprints dans les temps, en respectant le cahier des charges.

Sprint	Commandes
SP1	➢ Formation ➢ Epreuve ➢ Coefficient ➢ Exit
SP2	➢ SP1 ➢ Note ➢ Notes
SP3	➢ SP2 ➢ Relevé
SP4	➢ SP3 ➢ Décision
SP5	➢ SP4 ➢ Alignements respectés

Grâce aux structures fournie, tout ce qui étais à faire était de réutiliser les fonctions de ces structures pour les utiliser à **bonnes escient**.

Bilan du projet

- ## Les difficultés rencontrées

  Lorsque la Saé a été donné, l'idée d'un parseur (ou Analyseur syntaxique, interpréteur) était ma première idée. C'est-à-dire analyser chaque mot un par un et déterminer la commande qu'il faut appeler selon le premier mot entrer. Par exemple : si on entre formation 3 le parseur aurait mis dans un tableau de chaînes de caractères « formation » dans au première indice, puis « 3 » dans le deuxième indice du tableau, puis notre programme aurait appelé aurait appelé la fonction correspondant à formation, etc... Mais l'idée a été abandonnée lorsqu'une solution plus simple a été donné dans le cours, tout simplement utiliser des scanf.

```
void determine_commande(char commande[], int taille, char arg[][30],
unsigned int* nb_arg){
  unsigned int nb_char_arg = 0;
  for (int i = 0; i < taille; ++i){
    if(commande[i]==' '){
      *nb_arg += 1;
      nb_char_arg = 0;
    }
    else{
      arg[*nb_arg][nb_char_arg]=commande[i];
      ++nb_char_arg;
    }
  }
  *nb_arg += 1;
}
```

  Quelque dans cette idée, mais l'extrait ci dessus ne fonctionne pas.

  Le fait de lier plusieurs fonctions entre elles, peut s'avérer compliquer à certain moment. Certains calculs de traitement pour l'affichage de note ou de décision de jury ont pris assez du temps à se les approprier et les mettre en œuvre. L'ajout d'une épreuve était compliqué à réaliser, mais par la suite grâce à l'aide au projet, elle s'est avérée plus facile à interpréter, à imaginer une solution.

- ## Ce qui est réussi

  La collaboration et l'organisation en binôme a été une réussite.
  Le projet répond aux besoins du client.

- ## Ce qui peut être amélioré

  Une meilleure appropriation du sujet et des besoins du client pour les exploiter à son maximum.

Annexes

- Source du programme :

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#pragma warning(disable: 4996)
#pragma warning(disable: 6031)
#pragma warning(disable: 6202)

/*
Lorsque j'écris [in] Formation ou [in] Etudiants dans les documentations, il
faut entendre
"un pointeur vers une structure Formation/Etudiants"
*/

enum {
NB_SEMESTRES = 2,
MIN_UE = 3,
MAX_UE = 6,
MAX_MATIERES = 10,
MAX_EPREUVES = 5,
MAX_ETUDIANTS = 100,
MAX_CHAR = 31
};

typedef enum {
    False,
    True
}BOOL;

const float MIN_NOTE = 0.f, MAX_NOTE = 20.f, NOTE_PASSAGE = 10.f;

/*Epreuve contient le nom de l'épreuve, la valeur des coefficients
dans chaque UE ainsi que les notes de chaque étudiants*/
typedef struct {
char nomEpreuve[MAX_CHAR];
float coef[MAX_UE];
float notes[MAX_ETUDIANTS];
}Epreuve;

/*Matiere contient le nom de la matière, le nombre d'épreuve, les moyennes de
chaque
étudiants dans chaque UE, le total des coefficients dans chaque UE
ainsi qu'un tableau d'épreuves*/
typedef struct {
```

```
char nomMatiere[MAX_CHAR];
unsigned int nbEpreuves;
float moyennes[MAX_ETUDIANTS][MAX_UE];
float total_coef[MAX_UE];
Epreuve epreuves[MAX_EPREUVES];
}Matiere;

/*Semestre contien le nombre de matière ainsi qu'un tableau de matières*/
typedef struct {
unsigned int nbMatieres;
Matiere matieres[MAX_MATIERES];
}Semestre;

/*Foramtion contient le nombre d'UE, un BOOL permetant de savoir si le nombre
d'UE
a été défini ou non ainsi qu'un tableau de semestres*/
typedef struct {
unsigned int nbUE;
BOOL formation_cree;
Semestre semestres[NB_SEMESTRES];
}Formation;

/*Etudiants contient les nombre d'étudiant, un tableau contenant le nombre
notes de
chaque étudiants dans les deux semestres et un autre tableau contenant le
prenom
de chaque étudiant*/
typedef struct {
unsigned int nb_etu;
int nb_note[NB_SEMESTRES][MAX_ETUDIANTS];
char etudiant[MAX_ETUDIANTS][MAX_CHAR];
}Etudiants;




//fournir un pointuer vers les bonnes variables plutot que formation +
numVariable
/*[in] Formation: f
[out] BOOL True si la formation est crée et False sinon*/
BOOL verif_formation(const Formation* f);

/*[in] int: num_sem
[out] BOOL True si num_sem est compris entre MIN_UE et MAX_UE et False
Affiche un message d'erreur dans le cas où la fonction renvoie False
*/
BOOL verif_semestre(int num_sem);

/**/
```

```c
int determine_indice_mat(const Formation* f, int nu_se, const char* nom_mat,
int nb_mat);

/*[in] Pointeur: tab pointant vers un tableau de taille "taille"
[out] BOOL: True si au moins un des coefficients est supérieur ou égal à 0,
False dans le cas contraire*/
BOOL verif_coef(const float* tab, int taille);

/*[in] Formation: f la formation, int: num_se numéro du semestre
[out] BOOL False si tous les coefficients d'une UE du semestre num_se sont
nuls, True dans le cas contraire
Permet de verifier les coefficients d'un semestre*/
BOOL verification_coefficients_ue(const Formation* f, int num_se);

/*[in] Etudiants: etu les étudiants, chaine de caractères: prenom prenom de
l'étudiant
[out] int -1 si l'étudiant au prenom "prenom" n'existe pas dans le tableau
etu.etudiant et
son indice dans le cas contraire*/
int determine_indice_etu(const Etudiants* etu, const char prenom[]);

/*[in] Formation: f la formation, int: num_se numéro du semestre, num_mat
numéro de la matière recherché,
nb_epr nombre d'épreuve(utile pour la C3 mais pas pour la C5),
chaine de caractères: nom_epr nom de l'epreuve recherché
[out] int -1 si l'epreuve n'existe pas et son indice dans le cas contraire*/
int determine_indice_epr(const Formation* f, int num_se, int num_mat, const
char* nom_epr, int nb_epr);

/*[in] float: note
[out] False si note n'est pas compris entre 0. et 20. compris*/
BOOL verif_note(float note);

/*[in] Semestre: sem le semestre concerné
[out] int le nombre total d'epreuve du semestre*/
int nombre_epr(const Semestre* s);

/*[in] Semestre: s
[out] int le nombre de symbole dans la matière ayant le nom le plus long*/
int determine_max_char(const Semestre* s);

/*[in] Matiere: mat la matiere, int: num_etu le numéro de l'étudiant
num_ue le numéro de l'UE
[out] float la moyenne de l'étudiant dans l'UE num_ue et la matiere à l'indice
num_mat
et -1 si la somme des coefficients de cette matiere est égale à 0
La fonction permet également de modifier le tableau total_coef dans mat*/
float moyenne_matiere(Matiere* mat, int num_etu, int num_ue);
```

```
/*[in] Matiere: mat la matière, int: num_etu le numéro de l'étudiant, num_ue
le numéro de l'ue
La fonction permet de mettre à jour les moyennes d'un étudiant dans la
matière*/
void update_moyenne_mat(Matiere* mat, int num_etu, int num_ue);

/*[in] Formation: f la formation
La fonction permet d'afficher correctement des UE dans la première ligne dans
les affichages de releve
ainsi que decision*/
void affiche_ue(const Formation* f);

/*[in] Formation: f la formation, int: num_se le numéro du semestre, num_etu
le numéro
de l'étudiant, tableau de float: moyennes_ue tableau de moyennes
La fonction permet de modifier le tableau moyennes_ue et
permet d'obtenir la moyenne d'une UE pour un éudiant*/
void moyenne_ue(const Formation* f, int num_se, float* moyennes_ue, int
num_etu);

/*[in] Formation la formation, int: num_se le numéro du semestre, num_etu le
numéro de l'étudiant
La fonction permet d'afficher les moyennes de chaque UE sur un semestre pour
un étudiant*/
void affiche_moyenne_ue(const Formation* f, int num_se, int num_etu);
/*[in] Formation: f la formation
La fonction permet d'initialiser le nombre d'épreuve, de matière à 0
et initialiser les notes de tous les élèves à -1*/
void init_forma(Formation* f);

/*[in] Etudiants: etu contient tous les étudiants
La fonction permet d'initialiser le nombre de note de tous les étudiants à 0*/
void init_etu(Etudiants* etu);


/*Les 8 Commandes*/

/*C1
La fonction permet de fermer le programme*/
void fermer();

/*C2
[in] Formation: f la formation
La fonction permet d'initialiser le nombre d'UE, le nombre UE est compris
entre 3 et 6
et affiche des messages d'erreur ou de succes dans les cas définies dans le
sujet*/
void formation(Formation* f);
```

```
/*C3
[in] Formation: f la formation
La fonction permet d'ajoute une nouvelle matiere et une nouvelle epreuve si
elles n'existent
pas encore et affiche des messages d'erreur ou de succes dans les cas définies
dans le sujet*/
void epreuve(Formation* f);

/*C4
[in] Formation: f la formation
La fonction permet de verifier si les coefficients d'un semestre sont
correctes, c'est
à dire que pour chaque UE de ce semestre, la somme des coefficients est
supérieur à 0
et affiche des messages d'erreur ou de succes dans les cas définies dans le
sujet*/
void coefficients(const Formation *f);

/*C5
[in] Formation: f la formation, Etudiant: etu les étudiants
La fonction permet d'ajouter une note à un étudiant dans une matière et une
épreuve, si l'étudiant existe déjà et sinon rajoute l'étudiant
et affiche des messages d'erreur ou de succes dans les cas définies dans le
sujet*/
void note(Formation* f, Etudiants* etu);

/*C6
[in] Formation: f la formation, Etudiant: les étudiants
La fonction permet de verifier si un étudiant a bien toutes les notes
et affiche des messages d'erreur ou de succes dans les cas définies dans le
sujet*/
void notes(Formation* f, Etudiants* etu);

/*C7
[in] Formation: f la formation, Etudiants: etu les étudiants
La fonction permet d'afficher un relevé des notes d'un semestre pour un
étudiant.
Ce relevé contient la moyenne pondéré de chaque matière d'un semestre dans
chaque UE
ainsi que la moyenne pondéré de chaque UE dans un semestre.
Et affiche des messages d'erreur dans les cas définies dans le sujet*/
void affichage_releve(Formation* f, Etudiants* etu);

/*C8
[in]Formation: f la formation, Etudians: etu les étudiants
La fonction permet d'afficher la décision du jury pour un étudiant.
Cette affichage est composé des moyennes des semestres pour chaque UE,
les moyennes annuelles dans chaque UE, l'acquisition de chaque UE (aucune
si aucune UE n'a été acquis)et la décision du jury: Passage si
```

```
l'étudiant a au moins la moyennedans la moitié (arrondi au supérieur)
et Redoublement dans le cascontraire.
Et affiche des messages d'erreur dans les cas définies dans le sujet*/
void decision(Formation* f, const Etudiants* etu);

BOOL verif_formation(const Formation* f){
    if (f->formation_cree == False){
        printf("Le nombre d'UE n'est pas defini\n");
    }
    return f->formation_cree;
}

// C2
void formation(Formation* f){
    int nb_eu;
    scanf("%d", &nb_eu);
    if(f->formation_cree){
        printf("Le nombre d'UE est deja defini\n");
        return;
    }
    if (nb_eu >= MIN_UE && nb_eu <= MAX_UE){
        printf("Le nombre d'UE est defini\n");
        f->nbUE = nb_eu;
        f->formation_cree = True;
        return;
    }
    printf("Le nombre d'UE est incorect\n");
    return;
}

BOOL verif_semestre(int num_sem){
    if (num_sem > NB_SEMESTRES || num_sem < 1){
        printf("Le numero de semestre est incorrect\n");
        return False;
    }
    return True;
}

//C3
void epreuve(Formation* f){
    int nu_se, nu_mat; // Numero du semestre, Numero de la matiere
    char nom_mat[MAX_CHAR], nom_epr[MAX_CHAR];
    float coefs[MAX_UE];
    unsigned int nb_epr;
    scanf("%d", &nu_se);

    unsigned int nb_mat = f->semestres[nu_se-1].nbMatieres;
    scanf("%s %s", nom_mat, nom_epr);
    for (int i = 0; i < f->nbUE; ++i){
```

```
        scanf("%f", &coefs[i]);
    }

    if(verif_formation(f) == False)
        return;

    if (verif_semestre(nu_se) == False)
        return;
    nu_mat = determine_indice_mat(f, nu_se, nom_mat, nb_mat);
    if (nu_mat == -1)
        nb_epr = f->semestres[nu_se-1].matieres[nb_mat].nbEpreuves;  //nb_mat
    else
        nb_epr = f->semestres[nu_se-1].matieres[nu_mat].nbEpreuves;  //nu_mat

    if (determine_indice_epr(f, nu_se, nu_mat, nom_epr, nb_epr) >= 0){
        printf("Une meme epreuve existe deja\n");
        return;
    }
    if (verif_coef(coefs, f->nbUE) == False){
        printf("Au moins un des coefficients est incorrect\n");
        return;
    }
    if (nu_mat == -1){
        strcpy(f->semestres[nu_se-1].matieres[nb_mat].nomMatiere, nom_mat); //
cas ou la matiere n'existe pas encore
        printf("Matiere ajoutee a la formation\n");
        nu_mat = nb_mat;
        f->semestres[nu_se-1].nbMatieres += 1;
    }

    strcpy(f->semestres[nu_se-1].matieres[nu_mat].epreuves[f->semestres[nu_se-
1].matieres[nu_mat].nbEpreuves].nomEpreuve, nom_epr);
    printf("Epreuve ajoutee a la formation\n");
    f->semestres[nu_se-1].matieres[nu_mat].nbEpreuves += 1;

    for (int i = 0; i < f->nbUE; ++i){
        f->semestres[nu_se-1].matieres[nu_mat].epreuves[nb_epr].coef[i] =
coefs[i];
    }
    return;
}
int determine_indice_mat(const Formation* f, int nu_se, const char* nom_mat,
int nb_mat){
    for (int j = 0; j < nb_mat; ++j){
        if (strcmp(nom_mat, f->semestres[nu_se-1].matieres[j].nomMatiere) ==
0){
            return j;
        }
    }
```

```
    return -1;
}

BOOL verif_coef(const float* tab, int taille){
    BOOL flag = False;
    for (int i = 0; i < taille; ++i){
        if (tab[i] != 0.) flag = True;
        if (tab[i] < 0.) return False;
    }
    return flag;
}

//C4
void coefficients(const Formation *f){
    unsigned int nu_se;
    BOOL coef_valid; // au moins un coefficient d'une ue est valid
    scanf("%d", &nu_se);

    if(verif_formation(f) == False)
        return;

    if(nu_se > NB_SEMESTRES || nu_se < 1) {
        printf("Le numero de semestre est incorrect\n");
        return;
    }

    if(f->semestres[nu_se- 1].nbMatieres == 0) {
        printf("Le semestre ne contient aucune epreuve\n");
        return;
    }
    coef_valid = verification_coefficients_ue(f, nu_se);
    if (coef_valid == False){
        printf("Les coefficients d'au moins une UE de ce semestre sont tous
nuls\n");
        return;
    }
    printf("Coefficients corrects\n");
    return;
}

BOOL verification_coefficients_ue(const Formation* f, int num_se){
    BOOL coef_valid;
    for (int i = 0; i < f->nbUE; ++i){
        coef_valid = False;
        for (int j = 0; j < f->semestres[num_se-1].nbMatieres; ++j){
            for (int k = 0; k < f->semestres[num_se-1].matieres[j].nbEpreuves;
++k){
                if (f->semestres[num_se-1].matieres[j].epreuves[k].coef[i] !=
0.) coef_valid = True;
```

```
                if (coef_valid) break;
            }
            if (coef_valid) break;
        }
        if (coef_valid == False){
            return coef_valid;
        }
    }
    return True;
}

//C5
void note(Formation* f, Etudiants* etu){
    int num_se;
    float note_etu;
    char prenom[MAX_CHAR], nom_mat[MAX_CHAR], nom_epr[MAX_CHAR];
    scanf("%d %s %s %s %f", &num_se, prenom, nom_mat, nom_epr, &note_etu);

    if(verif_formation(f) == False)
        return;

    if (verif_semestre(num_se) == False)
        return;

    int num_mat = determine_indice_mat(f, num_se, nom_mat, f-
>semestres[num_se-1].nbMatieres);
    if (num_mat == -1){
        printf("Matiere inconnue\n");
        return;
    }
    int nb_epr = f->semestres[num_se-1].matieres[num_mat].nbEpreuves;
    int num_epr = determine_indice_epr(f, num_se, num_mat, nom_epr, nb_epr);
    if (num_epr == -1){
        printf("Epreuve inconnue\n");
        return;
    }

    if (verif_note(note_etu) == False){
        printf("Note incorrecte\n");
        return;
    }
    int num_etu = determine_indice_etu(etu, prenom);
    if (num_etu == -1){
        int nb_etu = etu->nb_etu;
        strcpy(etu->etudiant[nb_etu], prenom);
        printf("Etudiant ajoute a la formation\n");
        num_etu = etu->nb_etu;
        etu->nb_etu += 1;
    }
```

```
    if (f->semestres[num_se-
1].matieres[num_mat].epreuves[num_epr].notes[num_etu] != -1){
        printf("Une note est deja definie pour cet etudiant\n");
        return;
        }

    f->semestres[num_se-1].matieres[num_mat].epreuves[num_epr].notes[num_etu]
= note_etu;
    printf("Note ajoutee a l'etudiant\n");
    etu->nb_note[num_se-1][num_etu] += 1;
    return;
}

int determine_indice_etu(const Etudiants* etu, const char prenom[]){
    for (int i = 0; i < etu->nb_etu; ++i){
        if (strcmp(prenom, etu->etudiant[i]) == 0){
            return i;
        }
    }
    return -1;
}

int determine_indice_epr(const Formation* f, int num_se, int num_mat, const
char* nom_epr, int nb_epr){
    for (int i = 0; i < nb_epr; ++i){
        if (strcmp(nom_epr, f->semestres[num_se-
1].matieres[num_mat].epreuves[i].nomEpreuve) == 0){
            return i;
        }
    }
    return -1;
}

BOOL verif_note(float note){
    BOOL flag = True;
    if (note > MAX_NOTE || note < MIN_NOTE)
        flag = False;
    return flag;
}


//C6
void notes(Formation* f, Etudiants* etu){
    int num_se;
    char prenom[MAX_CHAR];
    scanf("%d %s", &num_se, prenom);

    if(verif_formation(f) == False)
        return;
```

```
    if (verif_semestre(num_se) == False)
        return;
    int num_etu = determine_indice_etu(etu, prenom);
    if (num_etu == -1){
        printf("Etudiant inconnu\n");
        return;
    }
    int somme_epr = nombre_epr(&f->semestres[num_se-1]); //Revient au nombre
total de note
    if (etu->nb_note[num_se-1][num_etu] != somme_epr){  //on pourrait cree une
fonction car on fait ca 3fois
        printf("Il manque au moins une note pour cet etudiant\n");
        return;
    }
    printf("Notes correctes\n");
    return;
}

int nombre_epr(const Semestre* s){
    int somme_epr = 0;
    for (int i = 0; i < s->nbMatieres; ++i){
        somme_epr += s->matieres[i].nbEpreuves;
    }
    return somme_epr;
}

//C7
void affichage_releve(Formation* f, Etudiants* etu){
    int num_se;
    char prenom[CHAR_MAX];
    scanf("%d %s", &num_se, prenom);

    if(verif_formation(f) == False)
        return;

    if(verif_semestre(num_se) == False){
        return;
    }
    int num_etu = determine_indice_etu(etu, prenom);
    if (num_etu == -1){
        printf("Etudiant inconnu\n");
        return;
    }

    if (verification_coefficients_ue(f, num_se) == False){
        printf("Les coefficients de ce semestre sont incorrects\n");
        return;
    }
```

```
    if (etu->nb_note[num_se-1][num_etu] != nombre_epr(&f->semestres[num_se-
1])){
        printf("Il manque au moins une note pour cet etudiant\n");
        return;
    }

    int max_char_mat = determine_max_char(&f->semestres[num_se-1]);
    if (max_char_mat < strlen("Moyennes")) max_char_mat = strlen("Moyennes");
    printf("%*c ", max_char_mat, ' ');
    affiche_ue(f);
    float moyenne_mat;
    char nom_mat[MAX_CHAR];
    for (int i = 0; i < f->semestres[num_se-1].nbMatieres; ++i){
        strcpy(nom_mat, f->semestres[num_se-1].matieres[i].nomMatiere);
        printf("%-*s ", max_char_mat, nom_mat);
        for (int j = 0; j < f->nbUE; ++j){
            update_moyenne_mat(&f->semestres[num_se-1].matieres[i], num_etu,
j);
            moyenne_mat = f->semestres[num_se-
1].matieres[i].moyennes[num_etu][j];
            if (moyenne_mat != -1.){
                printf("%4.1f ", floorf(moyenne_mat*10)/10);
            }
            else{
                printf("%4s ", "ND");
            }
        }
        printf("\n");
    }
    printf("--\n%-*s ", max_char_mat, "Moyennes");
    affiche_moyenne_ue(f, num_se, num_etu);
    return;
}

void update_moyenne_mat(Matiere* mat, int num_etu, int num_ue){
    float moyenne_mat;
    moyenne_mat = moyenne_matiere(mat, num_etu, num_ue);
    mat->moyennes[num_etu][num_ue] = moyenne_mat;
}

void affiche_ue(const Formation* f){
    for (int i = 1; i <= f->nbUE; ++i){
        printf("%3s%-2d", "UE", i);
    }
    printf("\n");
    return;
}
```

```
int determine_max_char(const Semestre* s){
    int max = 0;
    int nb_char;
    for (int i = 0; i < s->nbMatieres; ++i){
        nb_char = strlen(s->matieres[i].nomMatiere);
        if (nb_char > max)
            max = nb_char;
    }
    return max;
}

float moyenne_matiere(Matiere* mat, int num_etu, int num_ue){
    float somme_notes = 0., somme_coefs = 0., note, coef;
    for (int i = 0; i < mat->nbEpreuves; ++i){
        note = mat->epreuves[i].notes[num_etu];
        coef = mat->epreuves[i].coef[num_ue];
        somme_notes += note*coef;
        somme_coefs += coef;
    }
    mat->total_coef[num_ue] = somme_coefs;
    if (somme_coefs > 0.){
        return somme_notes/somme_coefs;
    }
    return -1.;
}

void moyenne_ue(const Formation* f, int num_se, float* moyennes_ue, int
num_etu){
    float somme_moyenne, somme_coefs, moyenne, coef;
    for (int i = 0; i < f->nbUE; ++i){
        somme_moyenne = 0.;
        somme_coefs = 0.;
        for (int j = 0; j < f->semestres[num_se-1].nbMatieres; ++j){
            moyenne = f->semestres[num_se-1].matieres[j].moyennes[num_etu][i];
            coef = f->semestres[num_se-1].matieres[j].total_coef[i];
            if (moyenne != -1)
                somme_moyenne += moyenne*coef;
            somme_coefs += coef;
        }
        moyennes_ue[i] = somme_moyenne/somme_coefs;
    }
    return;
}

void affiche_moyenne_ue(const Formation* f, int num_se, int num_etu){
    float moyennes_ue[MAX_UE];
    moyenne_ue(f, num_se, moyennes_ue, num_etu);
    for (int i = 0; i < f->nbUE; ++i){
        printf("%4.1f ", floorf(moyennes_ue[i]*10)/10);
```

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

//C8
void decision(Formation* f, const Etudiants* etu){
    char prenom[MAX_CHAR];
    scanf("%s", prenom);

    if (verif_formation(f) == False)
        return;
    int num_etu = determine_indice_etu(etu, prenom);
    if (num_etu == -1){
        printf("Etudiant inconnu\n");
        return;
    }
    if (verification_coefficients_ue(f, 1) == False ||
verification_coefficients_ue(f, 2) == False){
        printf("Les coefficients d'au moins un semestre sont incorrects\n");
        return;
    }
    if (etu->nb_note[0][num_etu] != nombre_epr(&f->semestres[0]) || etu-
>nb_note[1][num_etu] != nombre_epr(&f->semestres[1])){
        printf("Il manque au moins une note pour cet etudiant\n");
        return;
    }

    float moyennes_ue1[MAX_UE], moyennes_ue2[MAX_UE];
    float moyenne_sem_ue; // La moyenne entre les deux semestres pour une ue
    int nb_ue_acquis = 0.f;

    for (int i = 0; i < NB_SEMESTRES; ++i){
        for (int j = 0; j < f->semestres[i].nbMatieres; ++j){
            for (int k = 0; k < f->nbUE; ++k){
                update_moyenne_mat(&f->semestres[i].matieres[j], num_etu, k);
            }
        }
    }
    moyenne_ue(f, 1, moyennes_ue1, num_etu);
    moyenne_ue(f, 2, moyennes_ue2, num_etu);
    int nb_char = strlen("Moyennes annuelles");
    printf("%-*c ", nb_char, ' ');
    affiche_ue(f);
    for (int i = 1; i <= NB_SEMESTRES; ++i){
        printf("S%-*d", nb_char, i);
        affiche_moyenne_ue(f, i, num_etu);
    }
    printf("--\n%s ", "Moyennes annuelles");
```

```
    for (int i = 0; i < f->nbUE; ++i){
        moyenne_sem_ue = (moyennes_ue1[i] + moyennes_ue2[i]) / NB_SEMESTRES;
        printf("%4.1f ", floorf(moyenne_sem_ue*10)/10);
    }
    printf("\n%-*s ", nb_char, "Acquisition");
    for (int i = 0; i < f->nbUE; ++i){
        moyenne_sem_ue = (moyennes_ue1[i] + moyennes_ue2[i]) / NB_SEMESTRES;
        if (moyenne_sem_ue >= NOTE_PASSAGE){
            if(nb_ue_acquis != 0) printf(", ");
            printf("UE%d", i+1);
            ++nb_ue_acquis;
        }
    }
    if (nb_ue_acquis == 0){
        printf("Aucune");
    }
    printf("\n%-*s ", nb_char, "Devenir");
    if (nb_ue_acquis > (f->nbUE/2)){
        printf("Passage\n");
        return;
    }
    printf("Redoublement\n");
    return;
}

void init_forma(Formation* f){
    f->formation_cree = False;
    for (int i = 0; i < NB_SEMESTRES; ++i){
        f->semestres[i].nbMatieres = 0;
        for (int j = 0; j < MAX_MATIERES; ++j){
            f->semestres[i].matieres[j].nbEpreuves = 0;
            for (int k = 0; k < MAX_EPREUVES; ++k){
                for (int l = 0; l < MAX_ETUDIANTS; ++l){
                    f->semestres[i].matieres[j].epreuves[k].notes[l] = -1;
                }
            }
        }
    }
}

void init_etu(Etudiants* etu){
    etu->nb_etu = 0;
    for (int i = 0; i < NB_SEMESTRES; ++i){
        for (int j = 0; j < MAX_ETUDIANTS; ++j){
            etu->nb_note[i][j] = 0;
        }
    }
}
```

```c
void fermer(){
    exit(EXIT_SUCCESS);
}

int main(int argc, char *argv[])
{
    Formation f;
    init_forma(&f);
    Etudiants etudiants;
    char cde[MAX_CHAR] = "";
    do {
        scanf("%s", cde);
        if (strcmp(cde, "formation") == 0){
            formation(&f);
        }
        else if (strcmp(cde, "epreuve") == 0){
            epreuve(&f);
        }
        else if (strcmp(cde, "coefficients") == 0){
            coefficients(&f);
        }
        else if (strcmp(cde, "note") == 0){
            note(&f, &etudiants);
        }
        else if (strcmp(cde, "notes") == 0){
            notes(&f, &etudiants);
        }
        else if (strcmp(cde, "releve") == 0){
            affichage_releve(&f, &etudiants);
        }
        else if (strcmp(cde, "decision") == 0){
            decision(&f, &etudiants);
        }
    }
    while(strcmp(cde,"exit") != 0);
        fermer();
    return 0;
}
```

- Trace d’exécution des sprints :

Résultats des sprints 4, le sprint le plus haut, sans compter le sprint des recettes (l’affichage n’est pas respecté sur Word **mais il est respecté** sur un bloc-notes classique).

res-sp4-base.txt :

Le nombre d'UE est defini

Matiere ajoutee a la formation

Epreuve ajoutee a la formation

Epreuve ajoutee a la formation

Matiere ajoutee a la formation

Epreuve ajoutee a la formation

Epreuve ajoutee a la formation

Matiere ajoutee a la formation

Epreuve ajoutee a la formation

Epreuve ajoutee a la formation

Matiere ajoutee a la formation

Epreuve ajoutee a la formation

Epreuve ajoutee a la formation

Etudiant ajoute a la formation

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Etudiant ajoute a la formation

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Etudiant ajoute a la formation

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

	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

	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

	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

res-sp4-erreur.txt :

Le nombre d'UE est defini

Matiere ajoutee a la formation

Epreuve ajoutee a la formation

Epreuve ajoutee a la formation

Matiere ajoutee a la formation

Epreuve ajoutee a la formation

Epreuve ajoutee a la formation

Etudiant ajoute a la formation

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Note ajoutee a l'etudiant

Les coefficients d'au moins un semestre sont incorrects

Matiere ajoutee a la formation

Epreuve ajoutee a la formation

Epreuve ajoutee a la formation

Note ajoutee a l'etudiant

Il manque au moins une note pour cet etudiant

Note ajoutee a l'etudiant

Etudiant inconnu

	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