2022-2023

BUT1 – SAE S1.01

Implémentation d'un besoin client

– Gestion d'une formation –

UNIVERSITE PARIS CITE

OLIVIER DA SILVA & NICOLAS TASSIN - 104

Table des matières

Présentation du projet

Rôle fonctionnel

Ce projet a pour objectif de réaliser un interpréteur de commande afin de gérer une formation universitaire. Le projet est composé de huit commande, chacune jouant un rôle précis pour mener à bien les relevés de notes qui sont le but de notre projet.

Voici la liste des commandes :

1. Commande « exit »

Cette commande permet tout simplement de quitter l'application à la saisie du mot exit.

2. Définition de la formation

L'utilisateur doit saisir la chaîne de caractère « formation » suivi du nombre de formation qui doit être compris entre 3 et 6.

3. Ajout d'une épreuve

L'utilisateur saisie la chaîne de caractère « épreuve » suivi du numéro de semestre égal à 1 ou 2, le nom de la matière, nom de l'épreuve et d'un coefficient par UE.

4. Vérification des coefficients

Ici on vérifie la validité de nos coefficients

5. Ajout d'une note

L'utilisateur doit saisir une note pour un étudiant à une épreuve dans un semestre.

6. Vérifications notes

Cette commande vérifie si la saisie des notes de l'étudiant sont correctes.

7. Affichage d'un relevé de note

Une fois les actions précédentes effectuée on affiche un relevé de note pour l'étudiant

8. Affichage d'une décision du jury

L'utilisateur doit saisir la chaîne de caractère « décision », c'est ici qu'on pourra savoir si l'étudiant à validé son année ou non.

Les entrées/sorties de l'application

Notre programme sera confronté à différentes entrées et sorties :

Entrée : formation *nombre d'UE*

Sortie attendue : Le nombre d'UE est défini

Sortie possible : Le nombre d'UE est déjà défini/Le nombre d'UE est incorrect/Le nombre d'UE n'est pas défini

Entrée : épreuve *numéro de semestre* *nom de la matière* *nom de l'épreuve* *coefficient par UE*

Sortie attendue : Epreuve ajouté à la formation -> Matière ajouté à la formation

Sortie possible : Le numéro de semestre est incorrect/Une même épreuve existe déjà/Au moins un des coefficients est incorrect

Entrée : coefficients *numéro de semestre*

Sortie attendue : Coefficients corrects

Sortie possible : Le numéro de semestre est incorrect/Le semestre ne contient aucune épreuve/ Les coefficients d'au moins une UE de ce semestre sont tous nuls

Entrée : note *numéro de semestre* *nom de l'étudiant* *nom de la matière* *nom d'une épreuve* *note/20*

Sortie attendue : Note ajouté à l'étudiant

Sortie possible : Le numéro de semestre est incorrect/Matière inconnue/Epreuve inconnue/Note incorrecte/Une note est déjà définie pour cet étudiant

Entrée : notes *numéro de semestre* *nom de l'étudiant*

Sortie attendue : Notes correctes

Sortie possible : Le numéro de semestre est incorrect/Etudiant inconnu/Il manque au moins une note pour cet étudiant

Entrée : relevés *numéro de semestre* *nom de l'étudiant*

Sortie possible : Le numéro de semestre est incorrect/Etudiant inconnu/Les coefficients de ce semestre sont incorrects/Il manque au moins une note pour cet étudiant

Entrée : décision *nom de l'étudiant*

Sortie possible : Le numéro de semestre est incorrect/Etudiant inconnu/Les coefficients d'au moins un semestre sont incorrects/Il manque au moins une note pour cet étudiant

Les recettes

Organisation des tests

Pour savoir si notre application fonctionne correctement nous devons lui faire passer des Sprints, des fichiers qui nous permettrons de savoir si oui ou non notre application est conforme à nos attentes.

Ici nous devons passer 4 Sprints.

Sprint 1 : De la commande 1 à la commande 4

Sprint 2 : De la commande 1 à la commande 6

Sprint 3 : De la commande 1 à la commande 7

Sprint 4 : De la commande 1 à la commande 8

Durant les recettes nous devons indiquer à notre chargé de TP quel est la commande la plus haute que nous avons atteint, afin qu'il puisse savoir jusqu'à quel sprint cela correspond.

Le bilan de validation

Concernant nos recettes nous avons validé jusqu'au Sprint 5, nous avons terminé les huit commande cependant notre affichage concernant les moyennes annuelles avait rencontré un léger problème :

Nos notes s'arrondissaient parfois au supérieur et parfois à l'inférieur, heureusement ce problème a été résolu.

Bilan de projet

Les difficultés rencontrées

Il faut l'avouer ce projet était quand même assez difficile, pour le réussir avec facilité ou sans difficulté il faut avoir de grosses connaissances concernant ce langage ce qui n'est pas forcément le cas pour tous. Nous étions arrivés jusqu'à la C5 avec quelques soucis, mais sans problème particulier, c'est à partir de la C6 qu'on a eu des blocages. Nous avons donc décidé de demander de l'aide à des amis présent dans l'IUT ou encore quelques proches à nous. Nous avions réussi à atteindre la dernière commande, mais malheureusement, Nicolas a été confronté à problème très grave qui nous a empêché d'avoir l'accès à notre code. D'autant plus que moi-même Olivier je n'avais pas les dernières versions de notre code avec moi. Nous avons donc dû reprendre notre code depuis ses débuts.

Nicolas a donc pris la décision de prévenir Mr. Fessy pour qu'il soit averti de la situation et que nous ne nous retrouvons pas pénaliser. Nous avons pu être dispensés de recette la semaine du 7 novembre et avoir la chance d'un léger temps supplémentaire pour finaliser notre projet.

Nous avons actuellement réussi à finir le projet, cependant le seul problème comme dit précédemment se trouvait à l'arrondit des moyennes annuelles, qui parfois s'arrondissait au point supérieur et d'autres fois au point inférieur.

Mme Caraty nous avait donné du temps en plus pour régler notre erreur et repasser les recettes.

Ce qui est réussi

Nous avons eu la chance de refaire notre code en quelques jours alors que le projet a été donné pour un mois. Nous pouvons être fier de nous d'autant plus que notre passage au Sprint 3 a été validé.

Ce qui peut être amélioré

Nous devrons songer à être plus organisé, dans le sens à avoir chacun les dernières versions du code pour éviter de se retrouver dans une situation délicate en cas de problème.

Annexes

```c
#include <stdio.h>
#include <string.h>
#include <math.h>
#pragma warning(disable : 4996)

// === START TYPEDEF & CONST === //

//Defining limits & "magical" numbers
enum {
    MAX_SENTENCES = 30,
    MAX_SEMESTERS = 2,
    MIN_UNITS = 3,
    MAX_UNITS = 6,
    MAX_TESTS = 5,
    MAX_COURSES = 10,
    MAX_STUDENTS = 100,
    MIN_GRADE = 0,
    MAX_GRADE = 20,
    NONEXISTENT_VALUE = -1,
};

/**
 * Boolean type, return 0 to False, 1 to True
 */
typedef enum {
    FALSE = 0,
    TRUE = 1,
} Boolean;

typedef struct {
    char name[MAX_SENTENCES];
    float coef[MAX_UNITS];
    float grade[MAX_STUDENTS];
} Test;

typedef struct {
    char name[MAX_SENTENCES];
    int testIdx;
    Test test[MAX_TESTS];
} Course;

typedef struct {
    int courseIdx;
    Course course[MAX_COURSES];
} Semester;

typedef struct {
    char name[MAX_SENTENCES];
} Student;

typedef struct {
    Student student[MAX_STUDENTS];
    Semester semester[MAX_SEMESTERS];
```

```
    int studentIdx;
    int units;
} Formation;

// === UTILS Funcitons === //

/**
 * Reset/setup all start formation value to zero
 * [in-out] formation actual formation.
 */
void initFormation(Formation* formation) {
    formation->units = NONEXISTENT_VALUE;
    formation->studentIdx = 0;
    formation->semester[0].courseIdx = 0;
    formation->semester[1].courseIdx = 0;
}

/**
 * Check if the semester is valid, and reduce by one if it's true
 * [in-out] semesterIdx actual Semester index.
 */
Boolean isValidSemester(int* semesterIdx) {
    Boolean result = (*semesterIdx == 1 || *semesterIdx == 2);
    result ? (*semesterIdx)-- : printf("Le numero de semestre est
incorrect\n");
    return result;
}

/**
 * Verif if all coef are more than zero and at least one superior to zero
 * [in] coefList List of the coefs
 * [in] coefCount Number of the coefs
 */
Boolean checkCoef(const float* coefList, int coefCount) {
    Boolean hasCorrectValues = FALSE;

    for (int i = 0; i < coefCount; ++i) {
        if (coefList[i] < 0.f) return FALSE;
        if (coefList[i] > 0.f) hasCorrectValues = TRUE;
    }

    return hasCorrectValues;
}

/**
 * Check all coef for one semester and return TRUE if it's ok, FALSE
otherwise
 * [in] formation the actual formation
 * [in] the actual semester
 */
Boolean checkCoefForSemester(Formation* formation, Semester* semester) {
    for (int i = 0; i < semester->courseIdx; ++i) {
        for (int j = 0; j < semester->course[i].testIdx; ++j) {
            if (!checkCoef(semester->course[i].test[j].coef, formation-
>units)) {
                printf("Les coefficients d'au moins une UE de ce semestre
sont tous nuls\n");
                return FALSE;
            }
        }
    }
```

```
    return TRUE;
}

/**
 * Check if student have all his grades for a selected semester
 * [in] semester the selected semester
 * [in] studentIdx the wanted student index
 * Return true if all his grades are OK, false otherwise
 */
Boolean checkGradesForStudent(Semester* semester, int studentIdx) {
    for (int i = 0; i < semester->courseIdx; ++i) {

        Course* course = &semester->course[i];
        for (int j = 0; j < course->testIdx; ++j) {

            Test* test = &course->test[j];
            if (test->grade[studentIdx] == NONEXISTENT_VALUE) {
                printf("Il manque au moins une note pour cet etudiant\n");
                return FALSE;
            }
        }
    }

    return TRUE;
}

/**
 * Check if the student doesn't have already a grade
 * [in] test the actual test
 * [in] studentIdx the actual student index
 */
Boolean canAddGrade(Test* test, int studentIdx) {
    return test->grade[studentIdx] == NONEXISTENT_VALUE;
}

/**
 * Check and return if semester has test
 * [in] semester the actual semester
 */
Boolean semesterHasTest(Semester* semester) {
    if (semester->courseIdx == 0) return FALSE;

    for (int i = 0; i < semester->courseIdx; ++i) {
        if (semester->course[i].testIdx == 0) continue;
        return TRUE;
    }

    printf("Le semestre ne contient aucune épreuve\n");
    return FALSE;
}


// === GETTERS FUNCTIONS === //

/**
 * Get course index by name and return it (or -1 if not found)
 * [in] semester the actual semester
 * [in] courseName the course name researched
 */
int getCourseIndex(Semester* semester, char* courseName) {
```

```
    for (int i = 0; i < MAX_COURSES; ++i) {
        if (strcmp(semester->course[i].name, courseName) == 0) {
            return i;
        }
    }

    return NONEXISTENT_VALUE;
}

/**
 * Get test index by course and test name and return it (or -1 if not
found)
 * [in] semester The actual semester
 * [in] courseName The course name researched
 * [in] testName the test name researched
 */
int getTestIndex(Semester* semester, char* courseName, char* testName) {

    int courseIndex = getCourseIndex(semester, courseName);

    if (courseIndex == NONEXISTENT_VALUE) return NONEXISTENT_VALUE;

    Course course = semester->course[courseIndex];

    for (int i = 0; i < MAX_TESTS; ++i) {
        if (strcmp(course.test[i].name, testName) == 0) {
            return i;
        }
    }

    return NONEXISTENT_VALUE;
}

/**
 * Get student index by his name and return it (or -1 if not found)
 * @param formation the actual formation
 * @param studentName the researched student name
 */
int getStudentIndex(Formation* formation, char* studentName) {

    for (int i = 0; i < formation->studentIdx; ++i) {
        if (!strcmp(formation->student[i].name, studentName)) {
            return i;
        }
    }

    return NONEXISTENT_VALUE;
}

/**
 * Count mean for a selected semester
 * [in] formation the actual formation
 * [in] semester the wanted semester
 * [in-out] means the mean storage
 * [in] studentIdx Wanted student index
 */
void countMean(Formation* formation, Semester* semester, float* means, int
studentIdx) {

    for (int ue = 0; ue < formation->units; ue++) {
        float grades = 0.f;
```

```
        float coef = 0.f;

        for (int c = 0; c < semester->courseIdx; c++) {
            Course course = semester->course[c];

            for (int t = 0; t < course.testIdx; t++) {
                Test test = course.test[t];

                grades += test.grade[studentIdx] * test.coef[ue];
                coef += test.coef[ue];
            }
        }

        means[ue] = floorf((grades / coef) * 10) / 10;
    }

}

// === DEFINE RESOURCES FUNCTIONS === //

/**
 * setup course name and his test index.
 * [out] course The actual created course
 * [in] name The wanted name
 */
void defineCourse(Course* course, char* name) {
    strcpy(course->name, name);
    course->testIdx = 0;
}

/**
 * Setup a test name, hist name and  coefs
 * [out] test The actual created test
 * [in] name the wanted name
 * [in] coefs the wanted coef array
 * [in] coefCount array count of coefs
 */
void defineTest(Test* test, char* name, const float* coefs, int coefCount)
{
    strcpy(test->name, name);

    for (int i = 0; i < coefCount; ++i) {
        test->coef[i] = coefs[i];
    }

    for (int i = 0; i < MAX_STUDENTS; i++) {
        test->grade[i] = NONEXISTENT_VALUE;
    }
}

/**
 * Setup a student name
 * [out] student the actual created student
 * [in] name the wanted name
 */
void defineStudent(Student* student, char* name) {
    strcpy(student->name, name);
}

/**
 * Setup a test grade for a selected student
```

```
 * [out] test the actual modified test
 * [in] studentIdx the selected student
 * [in] grade the student grade
 */
void defineGrade(Test* test, int studentIdx, float grade) {
    test->grade[studentIdx] = grade;
}

// === CREATE RESOURCES FUNCTIONS === //

/**
 * Add a new course with a selected name
 * [in] semester Actual selected semester
 * [in] courseName Wanted name
 */
int addNewCourse(Semester* semester, char* courseName) {
    int newCourseIndex = semester->courseIdx;

    Course* course = &(semester->course[newCourseIndex]);

    defineCourse(course, courseName);

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

    return (semester->courseIdx)++;
}

/**
 * Add a new test with a selected name in a selected course
 * [in-out] Formation Actual formation
 * [in] courseName Wanted name
 * [in] courseName Wanted name
 * [in] semesterIndex The semester index
 * [in] coefs coef list
 */
void addNewTest(Formation* formation, char* courseName, char* testName, int
semesterIndex, float* coefs) {
    Semester* semester = &formation->semester[semesterIndex];

    int courseIndex = getCourseIndex(semester, courseName);
    if (courseIndex == NONEXISTENT_VALUE) {
        courseIndex = addNewCourse(semester, courseName);
    }

    Course* course = &semester->course[courseIndex];

    int newTestIndex = course->testIdx;

    Test* test = &(course->test[newTestIndex]);

    defineTest(test, testName, coefs, formation->units);

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

    ++(course->testIdx);
}

/**
 * Add a new student with a selected name
 * [in-out] formation the actual formation
 * [in] studentName wanted student name
```

```
 */
void addNewStudent(Formation* formation, char* studentName) {
    int studentIdx = formation->studentIdx;
    defineStudent(&(formation->student[studentIdx]), studentName);

    printf("Etudiant ajoute a la formation\n");

    (formation->studentIdx)++;
}

/**
 * Add a new test with a selected grade for a selected student name
 * [in] formation the actual formation
 * [in-out] test the created test
 * [in] studentName the researched student name
 * [in] grade the selected grade
 */
void addNewGrade(Formation* formation, Test* test, char* studentName, float
grade) {
    int studentIdx = getStudentIndex(formation, studentName);

    if (test->grade[studentIdx] != NONEXISTENT_VALUE) {
        printf("Une note est deja definie pour cet etudiant\n");
        return;
    }

    defineGrade(test, studentIdx, grade);
    printf("Note ajoutee a l'etudiant\n");
}

// === PRINT ARRAYS FUNCTIONS === //

/**
 * Print header with units name (UE1, UE2...)
 * [in] formation the actual formation
 * [in] nameLength the length to let before print the header
 */
void printHeader(Formation* formation, int nameLength) {
    printf("%*s", nameLength + 1, " ");
    for (int i = 0; i < formation->units; ++i) {
        printf(" %s%d ", "UE", i + 1);
    }
    printf("\n");
}

/**
 * Print courses and his grade for a selected student
 * [in] formation The actual formation
 * [in] course The selected course
 * [in] studentIdx The wanted student index
 * [in] nameLength the space to let before print grades
 * [in-out] grades the grade data
 * [in-out] countGrades coef count
 */
void printStatementCourse(Formation* formation, Course* course, int
studentIdx, int nameLength, float* grades,
    float* countGrades) {
    float values;
    float coef;

    printf("%s %*s", course->name, nameLength - strlen(course->name), "");
```

```
    for (int ue = 0; ue < formation->units; ue++) {
        values = 0.f;
        coef = 0.f;

        for (int i = 0; i < course->testIdx; i++) {
            Test* test = &course->test[i];

            values += test->grade[studentIdx] * test->coef[ue];
            coef += test->coef[ue];
        }

        if (coef == 0) {
            printf("  ND ");
        }
        else {
            float mean = values / coef;
            grades[ue] += values;
            countGrades[ue] += coef;
            printf("%4.1f ", floorf(mean * 10) / 10);
        }


    }

    printf("\n");
}

/**
 * Print means for units for the formation
 * [in] formation the actual formation
 * [in] nameLength space to let before showing means
 * [in] grades the calculated grades
 * [in] countedGrades the sum of coefs
 */
void printMeans(Formation* formation, int nameLength, const float* grades,
const float* countedGrades) {

    printf("--\n");
    printf("%s %*s", "Moyennes", nameLength - strlen("Moyennes"), "");

    for (int i = 0; i < formation->units; ++i) {
        float mean = grades[i] / countedGrades[i];
        float roundedMean = floorf(mean * 10) / 10;
        printf("%4.1f ", roundedMean);
    }

    printf("\n");
}

/**
 * Print all decision (means of each semesters, decision and if he can pass
his year)
 * [in] formation the actual formation
 * [in] means the means of each semester and twice in mean
 */
void printDecision(Formation* formation, float
means[MAX_SEMESTERS][MAX_UNITS]) {

    int minLength = strlen("Moyennes annuelles");
    printHeader(formation, minLength);
```

```
    for (int s = 0; s < MAX_SEMESTERS; s++) {
        printf("S%i %*s", s + 1, minLength - 2, "");

        for (int ue = 0; ue < formation->units; ue++) {
            printf("%4.1f ", means[s][ue]);
        }
        printf("\n");
    }

    printf("--\n");

    int achievementCount = 0, actualAchievement = 0;

    printf("Moyennes annuelles ");
    for (int ue = 0; ue < formation->units; ue++) {
        float mean = means[MAX_SEMESTERS][ue];
        printf("%4.1f ", mean);
        if (mean >= 10.f) {
            achievementCount++;
        }
    }
    printf("\n");
    printf("Acquisition %*s", minLength - strlen("Acquisition"), "");
    for (int ue = 0; ue < formation->units; ue++) {
        float mean = means[MAX_SEMESTERS][ue];
        if (mean >= 10.f) {
            printf("UE%i%s", ue + 1, (achievementCount - 1 ==
actualAchievement) ? "" : ", ");
            actualAchievement++;
        }
    }
    printf("\n");

    printf("Devenir %*s", minLength - strlen("Devenir"), "");
    printf("%s", (formation->units <= (achievementCount * 2) - 1) ?
"Passage" : "Redoublement");
    printf("\n");

}

// ------ //

/**
 * Define units and verif if the wanted units are between the limits
 * [in] formation The global formation
 */
void defineUnits(Formation* formation) {
    int unitReaded;
    scanf("%d", &unitReaded);

    if (formation->units != NONEXISTENT_VALUE) {
        printf("Le nombre d'UE est deja defini\n");
        return;
    }

    if (unitReaded > MAX_UNITS || unitReaded < MIN_UNITS) {
        printf("Le nombre d'UE est incorrect\n");
        return;
    }
```

```
    formation->units = unitReaded;
    printf("Le nombre d'UE est defini\n");
}

/**
 * Add a test with an entered semester index, course Name, test Name and a
list of coefs
 * [in] formation The global formation
 */
void addTest(Formation* formation) {
    int semesterIdx;
    char courseName[MAX_SENTENCES];
    char testName[MAX_SENTENCES];
    float coefList[MAX_UNITS];

    scanf("%d %s %s", &semesterIdx, courseName, testName);

    if (!isValidSemester(&semesterIdx)) return;

    for (int i = 0; i < formation->units; i++) {
        scanf("%f", &coefList[i]);
    }

    if (!checkCoef(coefList, formation->units)) {
        printf("Au moins un des coefficients est incorrect\n");
        return;
    }

    Semester* semester = &formation->semester[semesterIdx];

    if (getTestIndex(semester, courseName, testName) != NONEXISTENT_VALUE)
{
        printf("Une meme epreuve existe deja\n");
        return;
    }

    addNewTest(formation, courseName, testName, semesterIdx, coefList);
}

/**
 * Check coef for a semester with an entered semester index
 * [in] formation The global formation
 */
void checkCoefSemester(Formation* formation) {
    int semesterIdx;

    scanf("%d", &semesterIdx);

    if (!isValidSemester(&semesterIdx)) return;

    Semester* semester = &(formation->semester[semesterIdx]);

    if (!semesterHasTest(semester)) return;

    if (!checkCoefForSemester(formation, semester)) return;

    printf("Coefficients corrects\n");
}

/**
```

```
 * Add a grade with an entered semester index, student Name, course Name,
test Name and a list of grade
 * [in] formation The global formation
 */
void addGrade(Formation* formation) {

    int semesterIdx, courseIdx, testIdx;;
    char studentName[MAX_SENTENCES], courseName[MAX_SENTENCES],
testName[MAX_SENTENCES];
    float grade;

    scanf("%d %s %s %s %f", &semesterIdx, studentName, courseName,
testName, &grade);

    if (!isValidSemester(&semesterIdx)) return;

    Semester* semester = &formation->semester[semesterIdx];

    courseIdx = getCourseIndex(semester, courseName);
    if (courseIdx == NONEXISTENT_VALUE) {
        printf("Matiere inconnue\n");
        return;
    }

    Course* course = &semester->course[courseIdx];

    testIdx = getTestIndex(semester, courseName, testName);
    if (testIdx == NONEXISTENT_VALUE) {
        printf("Epreuve inconnue\n");
        return;
    }

    Test* test = &course->test[testIdx];

    if (grade < MIN_GRADE || grade > MAX_GRADE) {
        printf("Note incorrecte\n");
        return;
    }

    int studentIdx = getStudentIndex(formation, studentName);

    if (studentIdx == NONEXISTENT_VALUE) {
        addNewStudent(formation, studentName);
    }
    else if (!canAddGrade(test, studentIdx)) {
        printf("Une note est deja definie pour cet etudiant\n");
        return;
    }

    addNewGrade(formation, test, studentName, grade);
}

/**
 * Check if student have grades for a selected semester, with an entered
semester and student Name
 * [in] formation The global formation
 */
void checkNotes(Formation* formation) {
    int semesterIdx;
    char studentName[MAX_SENTENCES];
```

```
    scanf("%d %s", &semesterIdx, studentName);

    if (!isValidSemester(&semesterIdx)) {
        printf("Le numero de semestre est incorrect\n");
        return;
    }
    semesterIdx--;

    int studentIdx = getStudentIndex(formation, studentName);

    if (studentIdx == NONEXISTENT_VALUE) {
        printf("Etudiant inconnu\n");
        return;
    }

    Semester* semester = &(formation->semester[semesterIdx]);

    if (!checkGradesForStudent(semester, studentIdx)) return;

    printf("Notes correctes\n");
}

/**
 * Show an array who show a statement of semester, with an entered semester
index and student Name
 * [in] formation The global formation
 */
void getStatement(Formation* formation) {
    int semesterIdx;
    char studentName[MAX_SENTENCES];

    scanf("%d %s", &semesterIdx, studentName);

    if (!isValidSemester(&semesterIdx)) {
        printf("Le numero de semestre est incorrect\n");
        return;
    }

    int studentIdx = getStudentIndex(formation, studentName);

    if (studentIdx == NONEXISTENT_VALUE) {
        printf("Etudiant inconnu\n");
        return;
    }

    Semester semester = formation->semester[semesterIdx];

    if (!checkGradesForStudent(&semester, studentIdx)) return;

    int strLen = 0;

    for (int i = 0; i < semester.courseIdx; ++i) {
        Course course = semester.course[i];
        int courseLength = (int)strlen(course.name);
        if (strLen < courseLength) {
            strLen = courseLength;
        }
    }

    float grades[MAX_UNITS];
    float countedGrades[MAX_UNITS];
```

```
    for (int i = 0; i < formation->units; ++i) {
        grades[i] = 0.f;
        countedGrades[i] = 0.f;
    }

    printHeader(formation, strLen);
    for (int i = 0; i < semester.courseIdx; ++i) {
        Course course = semester.course[i];
        printStatementCourse(formation, &course, studentIdx, strLen,
grades, countedGrades);
    }


    printMeans(formation, strLen, grades, countedGrades);
}

/**
 * Show a array who show a decision of all the year, with an entered
student Name
 * [in] formation The global formation
 */
void getDecision(Formation* formation) {
    char studentName[MAX_SENTENCES];

    scanf("%s", studentName);

    int studentIdx = getStudentIndex(formation, studentName);

    if (studentIdx == NONEXISTENT_VALUE) {
        printf("Etudiant inconnu\n");
        return;
    }

    //Check if all semesters are ok
    for (int i = 0; i < MAX_SEMESTERS; i++) {
        Semester* semester = &(formation->semester[i]);

        if (!checkGradesForStudent(semester, studentIdx)) return;
        if (!checkCoefForSemester(formation, semester)) return;
    }

    float means[MAX_SEMESTERS + 1][MAX_UNITS];

    for (int i = 0; i < MAX_SEMESTERS; i++) {
        Semester* semester = &(formation->semester[i]);

        countMean(formation, semester, means[i], studentIdx);
    }

    for (int ue = 0; ue < formation->units; ue++) {
        float meanFirstSemester = floorf(means[0][ue] * 10) / 10;
        float meanSecondSemester = floorf(means[1][ue] * 10) / 10;
        float sumOfMeans = roundf((meanFirstSemester + meanSecondSemester)
* 100) / 100;
        float globalMean = floorf(sumOfMeans * 10.f / (float)2.f) / 10;
        means[MAX_SEMESTERS][ue] = globalMean;
    }

    printDecision(formation, means);
}
```

```c
int main(void) {

    char command[MAX_SENTENCES + 1];
    Formation formation;
    initFormation(&formation);

    do {
        scanf("%s", command);

        if (!strcmp(command, "formation")) defineUnits(&formation);
        else if (!strcmp(command, "epreuve")) addTest(&formation);
        else if (!strcmp(command, "coefficients"))
checkCoefSemester(&formation);
        else if (!strcmp(command, "note")) addGrade(&formation);
        else if (!strcmp(command, "notes")) checkNotes(&formation);
        else if (!strcmp(command, "releve")) getStatement(&formation);
        else if (!strcmp(command, "decision")) getDecision(&formation);
    } while (strcmp(command, "exit") != 0);

}
```