1. Develop an implementation package using 'C' program to process a FILE containing student details for the given queries.
A student record has the following format:

Std_rollno, Std_name, Dept, C1, C1_c, C1_g, C2, C2_c, C2_g, C3, C3_c, C3_g

Note: C1 refers to Course1, C1_c refers to credit of the course, C1_g refers to the grade in that course and so on.

Every student should have a unique rollno.

A student should have at least 3 courses and maximum four.

A grade point is in integer: S - 10; A - 9; B - 8; C - 7; D - 6; E - 5; F – 0.

Create a file and develop a menu driven system for the following queries.

a. Insert at least 5 student records.

b. Create a column 'GPA' for all the students.

c. For a student with four courses, delete(deregister) a course name.

d. For the same student you deleted in 'c', insert a new course name.

e. Update the name of a course for two different students.

f. Calculate GPA of all students using the GPA formula. Refer the following:

https://www.nitt.edu/home/academics/rules/BTech_Regulations_2019.pdf

g. Upgrade the grade point of a student who has secured '7' in a course.

h. Calculate the updated GPA of the student in 'g'.

i. Generate a Grade report of a student given the roll no. or name.

Code:

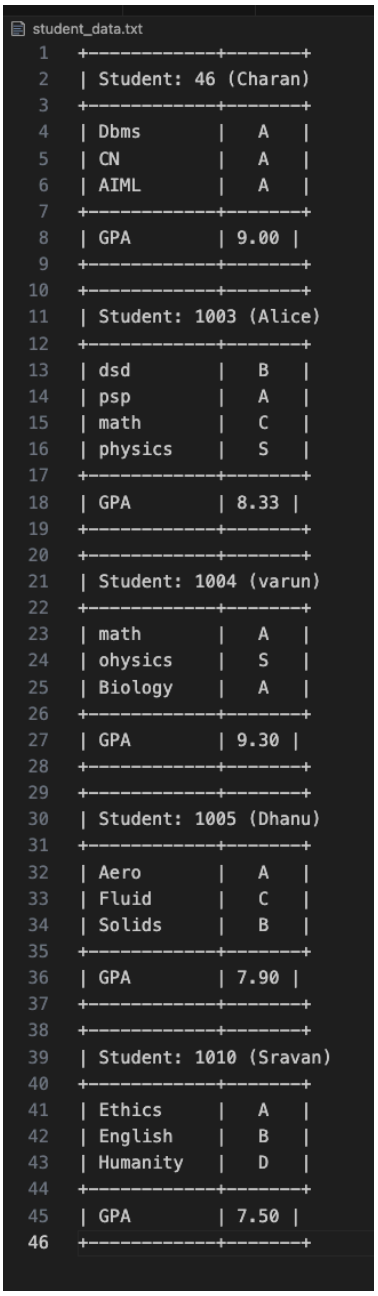
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_STUDENTS 100
#define MAX_COURSES 4
#define FILE_NAME "student_data.txt"

typedef struct {
   char course_name[10];
   int credit_hours;
   char grade;
} Course;

typedef struct {
   int roll_number;
```

```
    char student_name[50];
    char department[10];
    Course enrolled_courses[MAX_COURSES];
    int course_count;
    float gpa;
} Student;

Student student_records[MAX_STUDENTS];
int student_count = 0;

int grade_to_points(char grade) {
    switch (grade) {
        case 'S': return 10;
        case 'A': return 9;
        case 'B': return 8;
        case 'C': return 7;
        case 'D': return 6;
        case 'E': return 5;
        case 'F': return 0;
        default: return 0;
    }
}

void calculate_gpa(Student *student) {
    int total_points = 0;
    int total_credit_hours = 0;
    for (int i = 0; i < student->course_count; i++) {
        total_points += grade_to_points(student->enrolled_courses[i].grade) * student-
>enrolled_courses[i].credit_hours;
        total_credit_hours += student->enrolled_courses[i].credit_hours;
    }
    if (total_credit_hours > 0) {
        student->gpa = (float) total_points / total_credit_hours;
    } else {
        student->gpa = 0.0;
    }
}

void add_student_record() {
    if (student_count >= MAX_STUDENTS) {
        printf("Cannot add more students.\n");
        return;
    }
    Student *student = &student_records[student_count++];
    printf("Enter roll number: ");
    scanf("%d", &student->roll_number);
    printf("Enter name: ");
    scanf("%s", student->student_name);
    printf("Enter department: ");
    scanf("%s", student->department);
    printf("Enter number of courses (3 to 4): ");
    scanf("%d", &student->course_count);
    for (int i = 0; i < student->course_count; i++) {
        printf("Enter course %d name: ", i + 1);
        scanf("%s", student->enrolled_courses[i].course_name);
        printf("Enter course %d credit hours: ", i + 1);
        scanf("%d", &student->enrolled_courses[i].credit_hours);
        printf("Enter course %d grade: ", i + 1);
        scanf(" %c", &student->enrolled_courses[i].grade);
    }
    calculate_gpa(student);
}

void create_gpa_column() {
    for (int i = 0; i < student_count; i++) {
        calculate_gpa(&student_records[i]);
    }
    printf("GPA column created for all students.\n");
}

void remove_course(int roll_number, const char *course_name) {
    for (int i = 0; i < student_count; i++) {
        if (student_records[i].roll_number == roll_number) {
            for (int j = 0; j < student_records[i].course_count; j++) {
                if (strcmp(student_records[i].enrolled_courses[j].course_name, course_name) == 0) {
                    for (int k = j; k < student_records[i].course_count - 1; k++) {
                        student_records[i].enrolled_courses[k] = student_records[i].enrolled_courses[k + 1];
                    }
                    student_records[i].course_count--;
                    calculate_gpa(&student_records[i]);
                    printf("Course %s deleted for student %d.\n", course_name, roll_number);
                    return;
                }
            }
        }
    }
    printf("Course not found for the student.\n");
}

void add_course(int roll_number, const char *course_name, int credit_hours, char grade) {
    for (int i = 0; i < student_count; i++) {
        if (student_records[i].roll_number == roll_number) {
            if (student_records[i].course_count >= MAX_COURSES) {
                printf("Cannot add more courses for this student.\n");
                return;
            }
            Course *course = &student_records[i].enrolled_courses[student_records[i].course_count+
+];
            strcpy(course->course_name, course_name);
            course->credit_hours = credit_hours;
            course->grade = grade;
            calculate_gpa(&student_records[i]);
            printf("Course %s added for student %d.\n", course_name, roll_number);
            return;
        }
    }
    printf("Student not found.\n");
}

void update_course_name(int roll_number, const char *old_name, const char *new_name) {
    for (int i = 0; i < student_count; i++) {
        if (student_records[i].roll_number == roll_number) {
            for (int j = 0; j < student_records[i].course_count; j++) {
                if (strcmp(student_records[i].enrolled_courses[j].course_name, old_name) == 0) {
                    strcpy(student_records[i].enrolled_courses[j].course_name, new_name);
                    printf("Course name updated from %s to %s for student %d.\n", old_name,
new_name, roll_number);
                    return;
                }
            }
        }
    }
    printf("Course not found for the student.\n");
}

void calculate_all_gpas() {
    create_gpa_column();
}

void upgrade_student_grade(char grade, int new_points) {
    for (int i = 0; i < student_count; i++) {
        for (int j = 0; j < student_records[i].course_count; j++) {
            if (student_records[i].enrolled_courses[j].grade == grade) {
                student_records[i].enrolled_courses[j].grade = new_points;
            }
        }
        calculate_gpa(&student_records[i]);
    }
    printf("Grades upgraded for all students.\n");
}

void recalculate_student_gpa(int roll_number) {
    for (int i = 0; i < student_count; i++) {
        if (student_records[i].roll_number == roll_number) {
            calculate_gpa(&student_records[i]);
            printf("GPA recalculated for student %d.\n", roll_number);
            return;
        }
    }
    printf("Student not found.\n");
}

void generate_grade_report(int roll_number) {
    for (int i = 0; i < student_count; i++) {
        if (student_records[i].roll_number == roll_number) {
            printf("Grade report for student %d:\n", roll_number);
            printf("+------------+-------+\n");
            printf("| Course     | Grade |\n");
            printf("+------------+-------+\n");
            for (int j = 0; j < student_records[i].course_count; j++) {
                printf("| %-10s |   %c   |\n", student_records[i].enrolled_courses[j].course_name,
student_records[i].enrolled_courses[j].grade);
            }
            printf("+------------+-------+\n");
            printf("| GPA        | %.2f |\n", student_records[i].gpa);
            printf("+------------+-------+\n");
            return;
        }
    }
    printf("Student not found.\n");
}

void display_menu() {
    printf("1. Insert student record\n");
    printf("2. Create GPA column\n");
    printf("3. Remove course\n");
    printf("4. Add course\n");
    printf("5. Update course name\n");
    printf("6. Calculate GPA for all students\n");
    printf("7. Upgrade grade\n");
    printf("8. Recalculate GPA for a student\n");
    printf("9. Generate grade report\n");
    printf("10. Exit\n");
}

void read_student_data_from_file(const char *filename) {
    FILE *fp = fopen(filename, "r");
    if (fp == NULL) {
        printf("Error opening file %s.\n", filename);
        return;
    }
    student_count = 0;
    while (fscanf(fp, "%d %s %s %d", &student_records[student_count].roll_number,
student_records[student_count].student_name,
                 student_records[student_count].department,
&student_records[student_count].course_count) == 4) {
        for (int i = 0; i < student_records[student_count].course_count; i++) {
            fscanf(fp, "%s %d %c", student_records[student_count].enrolled_courses[i].course_name,
                   &student_records[student_count].enrolled_courses[i].credit_hours,
&student_records[student_count].enrolled_courses[i].grade);
        }
        calculate_gpa(&student_records[student_count]);
        student_count++;
        if (student_count >= MAX_STUDENTS) {
            printf("Maximum student limit reached.\n");
            break;
        }
    }
    fclose(fp);
}

void write_student_data_to_file(const char *filename) {
    FILE *fp = fopen(filename, "w");
    if (fp == NULL) {
        printf("Error opening file %s for writing.\n", filename);
        return;
    }
    for (int i = 0; i < student_count; i++) {
        fprintf(fp, "+------------+-------+\n");
        fprintf(fp, "| Student: %d (%s)\n", student_records[i].roll_number,
student_records[i].student_name);
        fprintf(fp, "+------------+-------+\n");
        for (int j = 0; j < student_records[i].course_count; j++) {
            fprintf(fp, "| %-10s |   %c   |\n", student_records[i].enrolled_courses[j].course_name,
student_records[i].enrolled_courses[j].grade);
        }
        fprintf(fp, "+------------+-------+\n");
        fprintf(fp, "| GPA        | %.2f |\n", student_records[i].gpa);
        fprintf(fp, "+------------+-------+\n");
    }
    fclose(fp);
    printf("Student data saved to file %s.\n", filename);
}

void add_student_to_file(const char *filename, Student *student) {
    FILE *fp = fopen(filename, "a");
    if (fp == NULL) {
        printf("Error opening file %s for appending.\n");
        return;
    }
    fprintf(fp, "+------------+-------+\n");
    fprintf(fp, "| Student: %d (%s)\n", student->roll_number, student->student_name);
    fprintf(fp, "+------------+-------+\n");
    for (int i = 0; i < student->course_count; i++) {
        fprintf(fp, "| %-10s |   %c   |\n", student->enrolled_courses[i].course_name, student-
>enrolled_courses[i].grade);
    }
    fprintf(fp, "+------------+-------+\n");
    fprintf(fp, "| GPA        | %.2f |\n", student->gpa);
    fprintf(fp, "+------------+-------+\n");
    fclose(fp);
    printf("Student data added to file %s.\n", filename);
}

void delete_student_from_file(const char *filename, int roll_number) {
    FILE *fp = fopen(filename, "r");
    if (fp == NULL) {
        printf("Error opening file %s.\n", filename);
        return;
    }

    FILE *temp_fp = fopen("temp.txt", "w");
    if (temp_fp == NULL) {
        fclose(fp);
        printf("Error creating temporary file.\n");
        return;
    }

    int found = 0;
    char line[256];

    while (fgets(line, sizeof(line), fp)) {
        int current_roll_number;
        sscanf(line, "%d", &current_roll_number);
        if (current_roll_number == roll_number) {
            found = 1;
            continue;
        }
        fputs(line, temp_fp);
    }

    fclose(fp);
    fclose(temp_fp);

    if (found) {
        remove(filename);
        rename("temp.txt", filename);
        printf("Student with roll number %d deleted from file.\n", roll_number);
    } else {
        remove("temp.txt");
        printf("Student with roll number %d not found in file.\n", roll_number);
    }
}

int main() {
    int choice;
    const char *filename = "student_data.txt";

    read_student_data_from_file(filename);

    do {
        display_menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                add_student_record();
                add_student_to_file(filename, &student_records[student_count - 1]);
                break;
            case 2:
                create_gpa_column();
                break;
            case 3: {
                int roll_number;
                char course_name[10];
                printf("Enter roll number: ");
                scanf("%d", &roll_number);
                printf("Enter course name: ");
                scanf("%s", course_name);
                remove_course(roll_number, course_name);
                write_student_data_to_file(filename);
                break;
            }
            case 4: {
                int roll_number;
                char course_name[10];
                int credit_hours;
                char grade;
                printf("Enter roll number: ");
                scanf("%d", &roll_number);
                printf("Enter course name: ");
                scanf("%s", course_name);
                printf("Enter credit hours: ");
                scanf("%d", &credit_hours);
                printf("Enter grade: ");
                scanf(" %c", &grade);
                add_course(roll_number, course_name, credit_hours, grade);
                write_student_data_to_file(filename);
                break;
            }
            case 5: {
                int roll_number;
                char old_name[10], new_name[10];
                printf("Enter roll number: ");
                scanf("%d", &roll_number);
                printf("Enter old course name: ");
                scanf("%s", old_name);
                printf("Enter new course name: ");
                scanf("%s", new_name);
                update_course_name(roll_number, old_name, new_name);
                write_student_data_to_file(filename);
                break;
            }
            case 6:
                calculate_all_gpas();
                break;
            case 7: {
                char grade;
                int new_points;
                printf("Enter grade to upgrade: ");
                scanf(" %c", &grade);
                printf("Enter new points: ");
                scanf("%d", &new_points);
                upgrade_student_grade(grade, new_points);
                write_student_data_to_file(filename);
                break;
            }
            case 8: {
                int roll_number;
                printf("Enter roll number: ");
                scanf("%d", &roll_number);
                recalculate_student_gpa(roll_number);
                write_student_data_to_file(filename);
                break;
            }
            case 9: {
                int roll_number;
                printf("Enter roll number: ");
                scanf("%d", &roll_number);
                generate_grade_report(roll_number);
                break;
            }
            case 10:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 10);
    return 0;
}
```

TERMINAL VIEW:

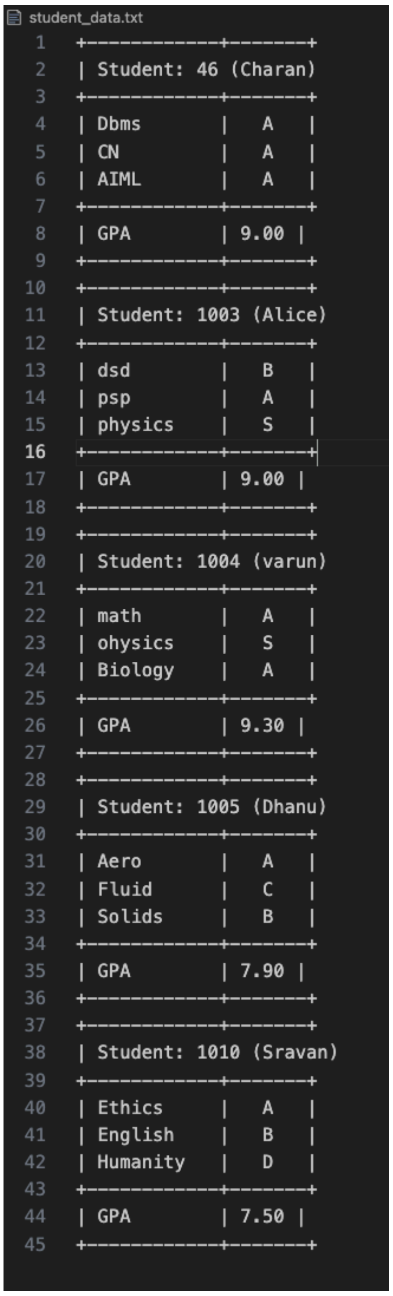
```
1. Insert student record
2. Create GPA column
3. Remove course
4. Add course
5. Update course name
6. Calculate GPA for all students
7. Upgrade grade
8. Recalculate GPA for a student
9. Generate grade report
10. Exit
Enter your choice: 1
Enter roll number: 46
Enter name: Charan
Enter department: CSE
Enter number of courses (3 to 4): 3
Enter course 1 name: Dbms
Enter course 1 credit hours: 3
Enter course 1 grade: A
Enter course 2 name: CN
Enter course 2 credit hours: 3
Enter course 2 grade: A
Enter course 3 name: AIML
Enter course 3 credit hours: 4
Enter course 3 grade: A
Student data added to file student_data.txt.
1. Insert student record
2. Create GPA column
3. Remove course
4. Add course
5. Update course name
6. Calculate GPA for all students
7. Upgrade grade
8. Recalculate GPA for a student
9. Generate grade report
10. Exit
Enter your choice: 1
Enter roll number: 1003
Enter name: Alice
Enter department: ECE
Enter number of courses (3 to 4): 4
Enter course 1 name: dsd
Enter course 1 credit hours: 3
Enter course 1 grade: B
Enter course 2 name: psp
Enter course 2 credit hours: 2
Enter course 2 grade: A
Enter course 3 name: math
Enter course 3 credit hours: 4
Enter course 3 grade: C
Enter course 4 name: physics
Enter course 4 credit hours: 3
Enter course 4 grade: S
```

File after Inserting Data:



```
student_data.txt
+------------+-------+
| Student: 46 (Charan)
+------------+-------+
| Dbms       |   A   |
| CN         |   A   |
| AIML       |   A   |
+------------+-------+
| GPA        | 9.00 |
+------------+-------+
+------------+-------+
| Student: 1003 (Alice)
+------------+-------+
| dsd        |   B   |
| psp        |   A   |
| math       |   C   |
| physics    |   S   |
+------------+-------+
| GPA        | 8.33 |
+------------+-------+
+------------+-------+
| Student: 1004 (varun)
+------------+-------+
| math       |   A   |
| ohysics    |   S   |
| Biology    |   A   |
+------------+-------+
| GPA        | 9.30 |
+------------+-------+
+------------+-------+
| Student: 1005 (Dhanu)
+------------+-------+
| Aero       |   A   |
| Fluid      |   C   |
| Solids     |   B   |
+------------+-------+
| GPA        | 7.90 |
+------------+-------+
+------------+-------+
| Student: 1010 (Sravan)
+------------+-------+
| Ethics     |   A   |
| English    |   B   |
| Humanity   |   D   |
+------------+-------+
| GPA        | 7.50 |
+------------+-------+
```

After Deregistration:



```
student_data.txt
+------------+-------+
| Student: 46 (Charan)
+------------+-------+
| Dbms       |   A   |
| CN         |   A   |
| AIML       |   A   |
+------------+-------+
| GPA        | 9.00 |
+------------+-------+
+------------+-------+
| Student: 1003 (Alice)
+------------+-------+
| dsd        |   B   |
| psp        |   A   |
| physics    |   S   |
+------------+-------+
| GPA        | 9.00 |
+------------+-------+
+------------+-------+
| Student: 1004 (varun)
+------------+-------+
| math       |   A   |
| ohysics    |   S   |
| Biology    |   A   |
+------------+-------+
| GPA        | 9.30 |
+------------+-------+
+------------+-------+
| Student: 1005 (Dhanu)
+------------+-------+
| Aero       |   A   |
| Fluid      |   C   |
| Solids     |   B   |
+------------+-------+
| GPA        | 7.90 |
+------------+-------+
+------------+-------+
| Student: 1010 (Sravan)
+------------+-------+
| Ethics     |   A   |
| English    |   B   |
| Humanity   |   D   |
+------------+-------+
| GPA        | 7.50 |
+------------+-------+
```

##Thus we performed different operations given and implemented given instructions ##

Q2)Create a Student schema using the student details given in Q.No.1 and execute the following basic queries.

Note: When defining the schema, exclude the following columns: Course_credit and Course_grade for all the courses.

Make sure you have the following constraints: Course is declared in char datatype.

DoB should be in date (dd/mm/yyyy) format. Provide a not-null constraint for dob.

Email should have the following format: xxx@nitt.edu

a. Insert at least 5 student records into the Student table.

b. Delete Course2 and Course3 attributes from the Student table.

c. Insert two new columns DoB and email into the Student table.

d. Change Course1 datatype to varchar2.

e. Update the column name ‘Std_rollno’ to ‘Std_rno’.

f. Update all student records who pursue a course named “DBMS” to “OS”.

g. Delete a student record with student name starting with letter ‘S’.

h. Display all records in which a student has born after the year 2005.

i. Simulate DROP and TRUNATE commands with the database you created.

Step 1: Create the Student Schema

```
CREATE TABLE Student (
   Std_rollno INT PRIMARY KEY,
   Std_name VARCHAR(50),
   Dept VARCHAR(10),
   Course1 CHAR(10),
   Course2 CHAR(10),
   Course3 CHAR(10),
   Course4 CHAR(10),
   dob DATE NOT NULL,
   email VARCHAR(50) CHECK (email LIKE '%@nitt.edu')
);
```

Step 2: Insert at least 5 student records into the Student table

```
/INSERT INTO Student (Std_rollno, Std_name, Dept, Course1, Course2, Course3, Course4,
dob, email) VALUES
(1, 'Charan', 'CSE', 'DBMS', 'OS', 'Math', 'Physics', '2004-03-29', '001@nitt.edu'),
(2, 'Alice', 'ECE', 'Circuits', 'Signals', 'Math', 'Physics', '1999-02-02', '002@nitt.edu'),
(3, 'varun', 'EEE', 'Power', 'Machines', 'Math', 'Physics', '2001-03-03', '003@nitt.edu'),
(4, 'Dhanu', 'MECH', 'Thermo', 'Mechanics', 'Math', 'Physics', '2002-04-04', '004@nitt.edu'),
(5, 'Sravan', 'CIVIL', 'Structures', 'Materials', 'Math', 'Physics', '1998-05-05', ‘005@nitt.edu');
```

Step 3: Delete Course2 and Course3 attributes from the Student table

```
ALTER TABLE Student DROP COLUMN course2;
ALTER TABLE Student DROP COLUMN course3;
```

Step 4: Insert two new columns dob and email into the Student table

It is inserted in Step 2

Step 5: Change Course1 datatype to VARCHAR(2)

ALTER TABLE Student MODIFY COLUMN course1 VARCHAR(2);

Step 6: Update the column name Std_rollno to Std_rno

ALTER TABLE Student CHANGE Std_rollno Std_rno INT;

Step 7: Update all student records who pursue a course named "DBMS" to “OS"

UPDATE Student SET Course1 = 'OS' WHERE Course1 = ‘DBMS';

Step 8: Delete a student record with a student name starting with the letter ’S'

DELETE FROM Student WHERE Std_name LIKE ’S%';

Step 9: Display all records in which a student has born after the year 2005

SELECT * FROM Student WHERE YEAR(dob) > 2005;

Step 10: Simulate DROP and TRUNCATE commands with the database you create

To drop the table:

DROP TABLE Student;
To truncate the table:

TRUNCATE TABLE Student;