

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.

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

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

typedef struct {
    char course_name[50]; int credits;
    int grade;
} Course;

typedef struct {
    int roll_no; char name[50]; char dept[50];
    Course courses[MAX_COURSES]; int num_courses;
    float gpa;
} Student;

Student students[MAX_STUDENTS];
int student_count = 0;
int gpa_column_created = 0;
```

```

void save_to_file() {
    FILE *file = fopen(FILE_NAME, "w");
    if (file == NULL) {
        printf("Error opening file for writing.\n"); return;
    }

    fprintf(file, "%d %d\n", student_count, gpa_column_created); for (int
i = 0; i < student_count; i++) {
        fprintf(file, "%d %s %s %d", students[i].roll_no, students[i].name,
students[i].dept, students[i].num_courses);
        if (gpa_column_created) {
            fprintf(file, " %.2f", students[i].gpa);
        }
        fprintf(file, "\n");
        for (int j = 0; j < students[i].num_courses; j++) {
            fprintf(file, "%s %d %d\n", students[i].courses[j].course_name,
students[i].courses[j].credits, students[i].courses[j].grade);
        }
    }
    fclose(file);
}

```

```

void load_from_file() {
    FILE *file = fopen(FILE_NAME, "r");
    if (file == NULL) {
        printf("Error opening file for reading.\n"); return;
    }

    fscanf(file, "%d %d", &student_count, &gpa_column_created);
    for (int i = 0; i < student_count; i++) {
        if(gpa_column_created) {
            fscanf(file, "%d %s %s %d %f", &students[i].roll_no,
students[i].name, students[i].dept, &students[i].num_courses,
&students[i].gpa);
        }else {
            fscanf(file, "%d %s %s %d", &students[i].roll_no,
students[i].name, students[i].dept, &students[i].num_courses);
        }
        for (int j = 0; j < students[i].num_courses; j++) {
            fscanf(file, "%s %d %d", students[i].courses[j].course_name,
&students[i].courses[j].credits, &students[i].courses[j].grade);
        }
    }
}

```

```

    }
    fclose(file);
}

void insert_student() {
    if (student_count >= MAX_STUDENTS) { printf("Maximum student limit
reached.\n"); return;
    }

    Student s;
    printf("Enter roll number: "); scanf("%d", &s.roll_no); printf("Enter
name: "); scanf("%s", s.name); printf("Enter department: "); scanf("%s",
s.dept);
    printf("Enter number of courses (3 or 4): "); scanf("%d",
&s.num_courses);
    for (int i = 0; i < s.num_courses; i++) {
        printf("Enter course %d name: ", i + 1);
        scanf("%s", s.courses[i].course_name);
        printf("Enter course %d credits: ", i + 1);
        scanf("%d", &s.courses[i].credits);
        printf("Enter course %d grade: ", i + 1);
        scanf("%d", &s.courses[i].grade);
    }
    s.gpa = 0.0;
    students[student_count++] = s;
    save_to_file();
}

void calculate_gpa(Student *s) {
    int total_credits = 0;
    int total_points = 0;
    for (int i = 0; i < s->num_courses; i++) {
        total_credits += s->courses[i].credits;
        total_points += s->courses[i].credits * s->courses[i].grade;
    }
    s->gpa = (float)total_points / total_credits;
}

void create_gpa_column() {
    gpa_column_created = 1; save_to_file();
}

```

```

}

void delete_course(int roll_no, char *course_name) {
    for (int i = 0; i < student_count; i++) {
        if (students[i].roll_no == roll_no) {
            for (int j = 0; j < students[i].num_courses; j++) {
                if (strcmp(students[i].courses[j].course_name,
course_name) == 0) {
                    for (int k = j; k < students[i].num_courses - 1; k++)
{
                        students[i].courses[k] = students[i].courses[k +
1];
                    }
                    students[i].num_courses--; save_to_file();
                    return;
                }
            }
        }
    }
    printf("Course not found for the given roll number.\n");
}

void insert_new_course(int roll_no, Course new_course) {
    for (int i = 0; i < student_count; i++) {
        if (students[i].roll_no == roll_no && students[i].num_courses ==
3) {
            students[i].courses[students[i].num_courses++] = new_course;
            save_to_file();
            break;
        }
    }
}

void update_course_name(int roll_no, char *old_name, char *new_name) {
    for (int i = 0; i < student_count; i++) {
        if (students[i].roll_no == roll_no) {
            for (int j = 0; j < students[i].num_courses; j++) {
                if (strcmp(students[i].courses[j].course_name, old_name)
== 0) {
                    strcpy(students[i].courses[j].course_name, new_name);

```

```

        save_to_file();
        break;
    }
}

}

}

}

void upgrade_grade_point(int roll_no, int old_grade, int new_grade) {
    for (int i = 0; i < student_count; i++) {
        if (students[i].roll_no == roll_no) {
            for (int j = 0; j < students[i].num_courses; j++) {
                if (students[i].courses[j].grade == old_grade) {
                    students[i].courses[j].grade = new_grade;
                }
            }
            save_to_file();
            break;
        }
    }
}

void generate_grade_report(int roll_no) {
    for (int i = 0; i < student_count; i++) {
        if (students[i].roll_no == roll_no) {
            printf("Roll No: %d\n", students[i].roll_no);
            printf("Name: %s\n", students[i].name);

            printf("Department: %s\n", students[i].dept);
            if (gpa_column_created) {
                printf("GPA: %.2f\n", students[i].gpa);
            }
            for (int j = 0; j < students[i].num_courses; j++) {
                printf("Course: %s, Credits: %d, Grade: %d\n",
students[i].courses[j].course_name, students[i].courses[j].credits,
students[i].courses[j].grade);
            }
            break;
        }
    }
}

```

```
}
```

```
void menu() {
    int choice, roll_no, old_grade, new_grade;
    Course new_course;
    char old_name[50], new_name[50]; load_from_file();
    while (1) {
        printf("1. Insert Student\n");
        printf("2. Create GPA Column\n");
        printf("3. Delete Course\n");
        printf("4. Insert New Course\n");
        printf("5. Update Course Name\n");
        printf("6. Upgrade Grade Point\n");
        printf("7. Generate Grade Report\n");
        printf("8. Calculate GPA\n");
        printf("9. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                insert_student(); break;
            case 2:
                create_gpa_column(); break;
            case 3:
                printf("Enter roll number: ");
                scanf("%d", &roll_no);
                printf("Enter course name to delete: ");
                char course_name[50];
                scanf("%s", course_name);
                delete_course(roll_no, course_name);
                break;
            case 4:
                printf("Enter roll number: ");
                scanf("%d", &roll_no);
                printf("Enter new course name: ");
                scanf("%s", new_course.course_name);
                printf("Enter new course credits: ");
                scanf("%d", &new_course.credits);
                printf("Enter new course grade: ");
                scanf("%d", &new_course.grade);
```

```

        insert_new_course(roll_no, new_course);
        break;
case 5:
    printf("Enter roll number: ");
    scanf("%d", &roll_no);
    printf("Enter old course name: ");
    scanf("%s", old_name);
    printf("Enter new course name: ");
    scanf("%s", new_name);
    update_course_name(roll_no, old_name, new_name); break;
case 6:
    printf("Enter roll number: ");
    scanf("%d", &roll_no);
    printf("Enter old grade: ");
    scanf("%d", &old_grade);
    printf("Enter new grade: ");
    scanf("%d", &new_grade);
    upgrade_grade_point(roll_no, old_grade, new_grade);
    break;
case 7:
    printf("Enter roll number: ");
    scanf("%d", &roll_no);
    generate_grade_report(roll_no);
    break;
case 8:
    printf("Enter roll number: ");
    scanf("%d", &roll_no);
    for (int i = 0; i < student_count; i++) {
        if (students[i].roll_no == roll_no) {
            calculate_gpa(&students[i]); save_to_file();
            break;
        }
    }
    break;
case 9:
    exit(0);
default:
    printf("Invalid choice.\n");
}
}

```

```
}
```

```
int main() {  
    menu();  
    return 0;  
}
```


2. 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@nift.edu.

QUERY :

```
CREATE TABLE Student (  
    Std_rollno INT PRIMARY KEY,  
    Std_name VARCHAR(50),  
    Dept VARCHAR(50),  
    C1 CHAR(10),  
    C2 CHAR(10),  
    C3 CHAR(10)  
);
```

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

QUERY:

```
INSERT INTO Student (Std_rollno, Std_name, Dept, C1, C2, C3) VALUES  
  
(1, 'Naveen', 'Computer Science and Engineering', 'DBMS', 'CN', 'CA'),  
(2, 'Soorya', 'Computer Science and Engineering', 'DBMS', 'CN', 'CA'),  
(3, 'Kishore', 'Computer Science and Engineering', 'DBMS', 'CN', 'CA'),  
(4, 'Lohesh', 'Computer Science and Engineering', 'DBMS', 'CN', 'CA'),  
(5, 'Japeth', 'Computer Science and Engineering', 'DBMS', 'CN', 'CA');
```

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

QUERY:

```
ALTER TABLE  
  
Student DROP COLUMN C2, DROP COLUMN C3;
```

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

QUERY:

```
ALTER TABLE Student
ADD COLUMN DoB
VARCHAR(10), ADD
COLUMN Email
VARCHAR(50);
```

```
UPDATE Student SET DoB = '01/01/2004', Email = '106122001@nitt.edu' WHERE
Std_rollno = 1;
UPDATE Student SET DoB = '02/02/2004', Email = '106122002@nitt.edu' WHERE
Std_rollno = 2;
UPDATE Student SET DoB = '03/03/2004', Email = '106122003@nitt.edu' WHERE
Std_rollno = 3;
UPDATE Student SET DoB = '04/04/2004', Email = '106122004@nitt.edu' WHERE
Std_rollno = 4;
UPDATE Student SET DoB = '05/05/2004', Email = '106122005@nitt.edu' WHERE
Std_rollno = 5;
```

```
ALTER TABLE Student

MODIFY COLUMN DoB VARCHAR(10)
NOT NULL, ADD CONSTRAINT
CHECK (
    DoB REGEXP '^[0-3][0-9]/[0-1][0-9]/[0-9]{4}$'
    AND SUBSTRING(DoB, 1, 2) BETWEEN 1 AND
    31 AND SUBSTRING(DoB, 4, 2) BETWEEN 1
    AND 12
),
ADD CONSTRAINT CHECK (Email LIKE '%@nitt.edu');
```

d. Change Course1 datatype to varchar.

Query:

```
ALTER TABLE Student MODIFY COLUMN C1 VARCHAR(20);
```

e. Update the column name 'Std_rollno' to 'Std_rno'.

Query:

```
ALTER TABLE Student CHANGE COLUMN Std_rollno Std_rno INT;
```

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

Query:

```
UPDATE Student SET C1 = 'OS'
```

```
WHERE C1 = 'DBMS';
```

g. Delete a student record with student name starting with letter 'S'.

Query:

```
DELETE FROM  
Student WHERE  
Std_name LIKE  
'S%';
```

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

Query:

```
SELECT * FROM Student  
  
WHERE SUBSTRING(DoB, 7, 4) > 2005;
```

g. Simulate RENAME, COMMENT, TRUNCATE and DROP.

Queries:

```
RENAME TABLE Student TO NewStudent;  
ALTER TABLE NewStudent COMMENT = 'This is a student table';  
TRUNCATE TABLE NewStudent;  
DROP TABLE NewStudent;
```

