CSLR 51

DBMS LAB - SESSION 2

Bunni Ranadeesh
106122026

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.

PROGRAM :

```
#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;
char grade;
} Course;

typedef struct {
int rollno;
char name[50];
char dept[10];
Course courses[MAX_COURSES];
int course_count;
float gpa;
} Student;

Student students[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_credits = 0;
for (int i = 0; i < student->course_count; i++) {
total_points += grade_to_points(student->courses[i].grade) * student-
>courses[i].credit;
total_credits += student->courses[i].credit;
}
if (total_credits > 0) {
student->gpa = (float) total_points / total_credits;
}
```

```
else {
student->gpa = 0.0;
}
}

void add_student() {
if (student_count >= MAX_STUDENTS) {
printf("Cannot add more students.\n");
return;
}
Student *student = &students[student_count++];
printf("Enter roll number: ");
scanf("%d", &student->rollno);
printf("Enter name: ");
scanf("%s", student->name);
printf("Enter department: ");
scanf("%s", student->dept);
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->courses[i].course_name);
printf("Enter course %d credit: ", i + 1);
scanf("%d", &student->courses[i].credit);
printf("Enter course %d grade: ", i + 1);
scanf(" %c", &student->courses[i].grade);
}
calculate_gpa(student);
}


void create_gpa_column() {
for (int i = 0; i < student_count; i++) {
calculate_gpa(&students[i]);
}

printf("GPA column created for all students.\n");
}

void delete_course(int rollno, const char *course_name) {

for (int i = 0; i < student_count; i++) {
if (students[i].rollno == rollno) {
for (int j = 0; j < students[i].course_count; j++) {
if (strcmp(students[i].courses[j].course_name, course_name) == 0) {
for (int k = j; k < students[i].course_count - 1; k++) {
students[i].courses[k] = students[i].courses[k + 1];
}
students[i].course_count--;
calculate_gpa(&students[i]);
printf("Course %s deleted for student %d.\n", course_name, rollno);
return;
}
}
}
}
printf("Course not found for the student.\n");
}

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


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

void calculate_all_gpa() {
create_gpa_column();
}

void upgrade_grade(char grade, int new_points) {
for (int i = 0; i < student_count; i++) {
for (int j = 0; j < students[i].course_count; j++) {
if (students[i].courses[j].grade == grade) {
students[i].courses[j].grade = new_points;
}
}
```

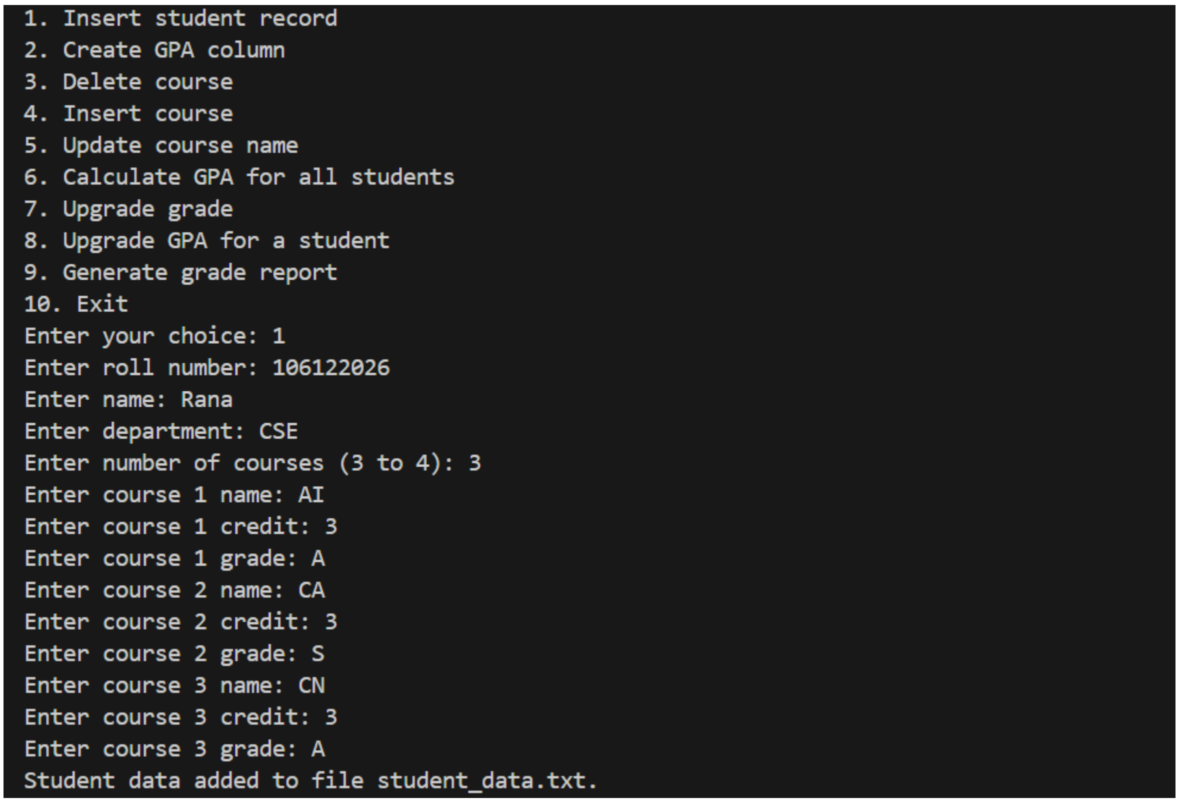
```
calculate_gpa(&students[i]);
}
printf("Grades upgraded for all students.\n");
}

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

void generate_grade_report(int rollno) {
for (int i = 0; i < student_count; i++) {
if (students[i].rollno == rollno) {
printf("Grade report for student %d:\n", rollno);
printf("+-----------+-------+\n");
printf("| Course     | Grade |\n");
printf("+-----------+-------+\n");
for (int j = 0; j < students[i].course_count; j++) {
printf("| %-10s |   %c   |\n", students[i].courses[j].course_name,
students[i].courses[j].grade);
}
printf("+-----------+-------+\n");
printf("| GPA        | %.2f |\n", students[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. Delete course\n");
printf("4. Insert course\n");
printf("5. Update course name\n");
printf("6. Calculate GPA for all students\n");
printf("7. Upgrade grade\n");
printf("8. Upgrade 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", &students[student_count].rollno,
students[student_count].name,
students[student_count].dept, &students[student_count].course_count) == 4)
{
for (int i = 0; i < students[student_count].course_count; i++) {
fscanf(fp, "%s %d %c", students[student_count].courses[i].course_name,
&students[student_count].courses[i].credit,
&students[student_count].courses[i].grade);
}
calculate_gpa(&students[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", students[i].rollno, students[i].name);
fprintf(fp, "+-----------+-------+\n");
for (int j = 0; j < students[i].course_count; j++) {
fprintf(fp, "| %-10s |   %c   |\n", students[i].courses[j].course_name,
students[i].courses[j].grade);
}
fprintf(fp, "+-----------+-------+\n");
fprintf(fp, "| GPA        | %.2f |\n", students[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", filename);
return;
}
fprintf(fp, "+-----------+-------+\n");
fprintf(fp, "| Student: %d (%s)\n", student->rollno, student->name);
fprintf(fp, "+-----------+-------+\n");
for (int i = 0; i < student->course_count; i++) {
```

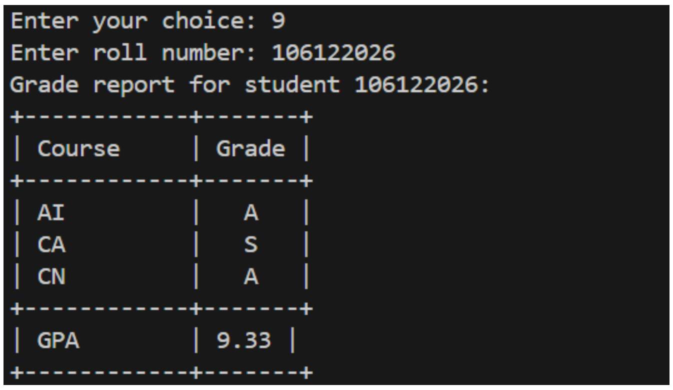
```
fprintf(fp, "| %-10s |   %c   |\n", student->courses[i].course_name,
student->courses[i].grade);
}
fprintf(fp, "+------------+-------+\n");
fprintf(fp, "| GPA        | %.2f |\n", student->gpa);
fprintf(fp, "+------------+-------+\n");
fclose(fp);
printf("Student data added to file %s.\n", filename);
}


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

// Create a temporary file to store data except the student to be deleted
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_rollno;
sscanf(line, "%d", &current_rollno);
if (current_rollno == rollno) {
found = 1;
continue; // skip this line
}
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", rollno);
} else {
remove("temp.txt");
printf("Student with roll number %d not found in file.\n", rollno);
}
}

int main() {
int choice;
```

```
const char *filename = "student_data.txt"; // File name for student data

read_student_data_from_file(filename); // Read existing data from file

do {
display_menu();
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
add_student();
add_student_to_file(filename, &students[student_count - 1]); // Add the
last added student to file
break;
case 2:
create_gpa_column();
break;
case 3: {
int rollno;
char course_name[10];
printf("Enter roll number: ");
scanf("%d", &rollno);
printf("Enter course name: ");
scanf("%s", course_name);
delete_course(rollno, course_name);
write_student_data_to_file(filename); // Update file after deletion
break;
}
case 4: {
int rollno;
char course_name[10];
int credit;
char grade;
printf("Enter roll number: ");
scanf("%d", &rollno);
printf("Enter course name: ");
scanf("%s", course_name);
printf("Enter credit: ");
scanf("%d", &credit);
printf("Enter grade: ");
scanf(" %c", &grade);
insert_course(rollno, course_name, credit, grade);
write_student_data_to_file(filename); // Update file after insertion
break;
}
case 5: {
int rollno;
char old_name[10], new_name[10];
printf("Enter roll number: ");
scanf("%d", &rollno);
printf("Enter old course name: ");
scanf("%s", old_name);
printf("Enter new course name: ");
scanf("%s", new_name);
update_course_name(rollno, old_name, new_name);
write_student_data_to_file(filename); // Update file after course name
update
break;
}
case 6:
calculate_all_gpa();
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_grade(grade, new_points);
write_student_data_to_file(filename); // Update file after grade upgrade
break;
}
case 8: {
int rollno;
printf("Enter roll number: ");
scanf("%d", &rollno);
upgrade_gpa(rollno);
write_student_data_to_file(filename); // Update file after GPA upgrade
break;
}
case 9: {
int rollno;
printf("Enter roll number: ");
scanf("%d", &rollno);
generate_grade_report(rollno);
break;
}
case 10:
printf("Exiting...\n");
break;
default:
printf("Invalid choice. Please try again.\n");
}
} while (choice != 10);

return 0;
}
```

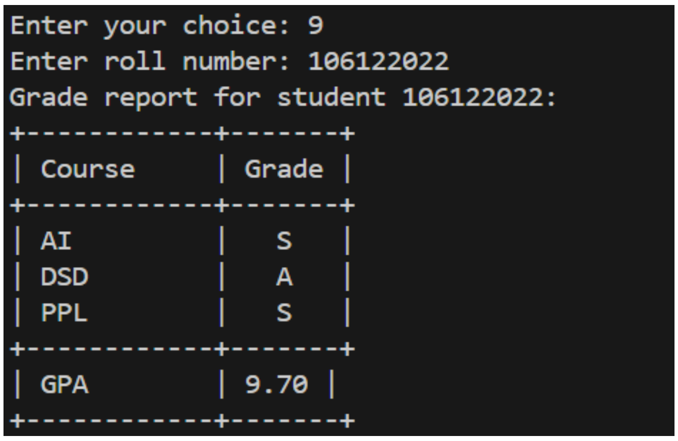
OUTPUT :



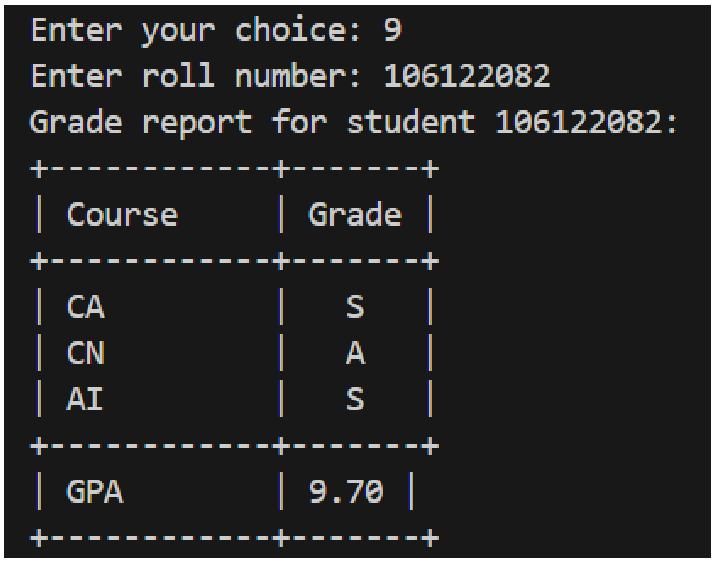
```
1. Insert student record
2. Create GPA column
3. Delete course
4. Insert course
5. Update course name
6. Calculate GPA for all students
7. Upgrade grade
8. Upgrade GPA for a student
9. Generate grade report
10. Exit
Enter your choice: 1
Enter roll number: 106122026
Enter name: Rana
Enter department: CSE
Enter number of courses (3 to 4): 3
Enter course 1 name: AI
Enter course 1 credit: 3
Enter course 1 grade: A
Enter course 2 name: CA
Enter course 2 credit: 3
Enter course 2 grade: S
Enter course 3 name: CN
Enter course 3 credit: 3
Enter course 3 grade: A
Student data added to file student_data.txt.
```



```
Enter your choice: 9
Enter roll number: 106122026
Grade report for student 106122026:
+------------+-------+
| Course     | Grade |
+------------+-------+
| AI         |   A   |
| CA         |   S   |
| CN         |   A   |
+------------+-------+
| GPA        | 9.33 |
+------------+-------+
```



```
Enter your choice: 9
Enter roll number: 106122022
Grade report for student 106122022:
+------------+-------+
| Course     | Grade |
+------------+-------+
| AI         |   S   |
| DSD        |   A   |
| PPL        |   S   |
+------------+-------+
| GPA        | 9.70 |
+------------+-------+
```



```
Enter your choice: 9
Enter roll number: 106122082
Grade report for student 106122082:
+------------+-------+
| Course     | Grade |
+------------+-------+
| CA         |   S   |
| CN         |   A   |
| AI         |   S   |
+------------+-------+
| GPA        | 9.70 |
+------------+-------+
```

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

```
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')
);
```

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.

```
INSERT INTO Student (Std_rollno, Std_name, Dept, Course1, Course2, Course3, Course4, dob, email)
VALUES
(1, 'Rana', 'CSE', 'DBMS', 'OS', 'Math', 'Physics', '2000-01-01', 'rana@nitt.edu'),
(2, 'Amrut’, 'CSE', 'Networks, 'Math', 'Physics', '1999-02-02', 'amrut@nitt.edu'),
(3, 'Raj', 'CSE', 'Computer Architecture', 'Machines', 'Math', 'Physics', '2001-03-03', 'raj@nitt.edu'),
(4, 'Charan', 'CSE', ‘Design’, 'Big data', 'Math', 'Physics', '2002-04-04', 'charan@nitt.edu'),
(5, 'Neel', 'CSE', 'Structures', 'Materials', 'Math', 'Physics', '1998-05-05', 'neel@nitt.edu');
```

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

```
ALTER TABLE Student DROP COLUMN Course2;
ALTER TABLE Student DROP COLUMN Course3;
```

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

The columns dob and email are already added in the initial schema creation.

d. Change Course1 datatype to varchar2.

```
ALTER TABLE Student MODIFY COLUMN Course1 VARCHAR(2);
```

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

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

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

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

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

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

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

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

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

To drop the table:
```
DROP TABLE Student;
```
To truncate the table:
```
TRUNCATE TABLE Student;
```

--- * * THANK YOU * * ---