**Easy Version**

**This task requires each group of students to design and implement a C program. More specifically, the program should read and store student registration numbers, five units that the student is taking and the score that the student has in each unit. The program should then grade the student according to DeKUT grading criteria, summarise the units and the grades attained and provide a recommendation (e.g Very Good, Good, Fair, Poor) based on the grade attained. Use of functions in this work is highly encouraged**

1. #include <stdio.h>
2. // Function to calculate the grade based on the score
3. char calculateGrade(float score) {
4. // The function takes a floating-point 'score' as input and returns a character representing the grade.
5. // It uses if-else statements to determine the grade based on the score and returns the appropriate grade character.
6. if (score >= 70.0) {
7. return 'A';  // If the score is 70 or above, return 'A'.
8. } else if (score >= 60.0) {
9. return 'B';  // If the score is between 60 and 69.99, return 'B'.
10. } else if (score >= 50.0) {
11. return 'C';  // If the score is between 50 and 59.99, return 'C'.
12. } else if (score >= 40.0) {
13. return 'D';  // If the score is between 40 and 49.99, return 'D'.
14. } else {
15. return 'E';  // For any score below 40, return 'E'.
16. }
17. }
18. int main() {
19. // Constants
20. const int MAX\_STUDENTS = 5;  // Maximum number of students allowed (adjustable if needed).
21. const int MAX\_UNITS = 5;     // Maximum number of units per student (adjustable if needed).
22. // Arrays to store student data
23. int regNumbers[MAX\_STUDENTS];                         // Array to store registration numbers of students.
24. char unitNames[MAX\_UNITS][50];                        // 2D array to store names of units for each student.
25. float unitScores[MAX\_STUDENTS][MAX\_UNITS];            // 2D array to store scores of each unit for each student.
26. // Read student data
27. printf("Enter data for %d students:\n", MAX\_STUDENTS);
28. for (int i = 0; i < MAX\_STUDENTS; i++) {
29. printf("\nStudent %d:\n", i + 1);
30. printf("Registration Number: ");
31. scanf("%d", &regNumbers[i]);                      // Read the registration number of the current student.
32. getchar();  // Clear newline character from the input buffer (necessary after using scanf with %d).
33. for (int j = 0; j < MAX\_UNITS; j++) {
34. printf("Unit %d Name: ", j + 1);
35. fgets(unitNames[j], sizeof(unitNames[j]), stdin);      // Read the name of the unit for the current student.
36. unitNames[j][strcspn(unitNames[j], "\n")] = '\0';      // Remove trailing newline character from the input.
37. printf("Unit %d Score: ", j + 1);
38. scanf("%f", &unitScores[i][j]);                 // Read the score of the current unit for the current student.
39. getchar();  // Clear newline character from the input buffer (necessary after using scanf with %f).
40. }
41. }
42. // Print summary
43. printf("\nSummary:\n");
44. for (int i = 0; i < MAX\_STUDENTS; i++) {
45. printf("\nStudent %d (Reg. No. %d):\n", i + 1, regNumbers[i]);
46. for (int j = 0; j < MAX\_UNITS; j++) {
47. printf("Unit %d: %s\n", j + 1, unitNames[j]);          // Print the name of the unit for the current student.
48. printf("Score: %.2f\n", unitScores[i][j]);            // Print the score of the current unit for the current student.
49. printf("Grade: %c\n", calculateGrade(unitScores[i][j]));  // Call the 'calculateGrade' function to get the grade and print it.
50. }
51. }
52. return 0;
53. }

**Complex version**

1. #include <stdio.h>
2. #include <string.h>
3. // Function to calculate the grade based on the score
4. char getGrade(float score) {
5. // This function takes a floating-point 'score' as input and returns the corresponding grade as a character.
6. // It uses a series of if-else statements to determine the grade based on the input score.
7. // The grading system is as follows:
8. // - A: score >= 90.0
9. // - B: score >= 80.0 and score < 90.0
10. // - C: score >= 70.0 and score < 80.0
11. // - D: score >= 60.0 and score < 70.0
12. // - F: score < 60.0
13. if (score >= 90.0) {
14. return 'A';
15. } else if (score >= 80.0) {
16. return 'B';
17. } else if (score >= 70.0) {
18. return 'C';
19. } else if (score >= 60.0) {
20. return 'D';
21. } else {
22. return 'F';
23. }
24. }
25. // Function to swap two student's data
26. void swapStudents(int\* a, int\* b, char name[][50], float scores[][5], int index1, int index2) {
27. // This function swaps the data of two students (IDs, unit names, and unit scores) in the provided arrays.
28. // 'a' and 'b' are dummy arrays for the student IDs, but we can use the same array since the data we want to swap is at the same index.
29. // The 'name' array is a 2D array to store the names of units for each student.
30. // The 'scores' array is a 2D array to store the scores of each unit for each student.
31. // 'index1' and 'index2' are the indices of the two students' data to be swapped.
32. // Swap student IDs
33. int tempInt = a[index1];
34. a[index1] = a[index2];
35. a[index2] = tempInt;
36. // Swap unit names
37. char tempName[50];
38. strcpy(tempName, name[index1]);
39. strcpy(name[index1], name[index2]);
40. strcpy(name[index2], tempName);
41. // Swap unit scores for all 5 units
42. for (int i = 0; i < 5; i++) {
43. float tempFloat = scores[index1][i];
44. scores[index1][i] = scores[index2][i];
45. scores[index2][i] = tempFloat;
46. }
47. }
48. // Function to sort students based on their average scores
49. void sortStudents(int studentIDs[], char unitNames[][50], float unitScores[][5], int numStudents) {
50. // This function sorts the students based on their average scores (calculated from their unit scores).
51. // The function uses the bubble sort algorithm to rearrange the student data in ascending order of average scores.
52. // 'studentIDs' is the array containing the IDs of each student.
53. // 'unitNames' is the 2D array containing the names of units for each student.
54. // 'unitScores' is the 2D array containing the scores of each unit for each student.
55. // 'numStudents' is the total number of students.
56. for (int i = 0; i < numStudents - 1; i++) {
57. for (int j = 0; j < numStudents - i - 1; j++) {
58. // Calculate average scores for the current student (j) and the next student (j + 1)
59. float avgScore1 = 0, avgScore2 = 0;
60. for (int k = 0; k < 5; k++) {
61. avgScore1 += unitScores[j][k];  // Accumulate the scores for the current student.
62. avgScore2 += unitScores[j + 1][k];  // Accumulate the scores for the next student.
63. }
64. avgScore1 /= 5;  // Calculate the average score for the current student.
65. avgScore2 /= 5;  // Calculate the average score for the next student.
66. // Compare the average scores and swap the students' data if the next student's average score is smaller
67. if (avgScore1 > avgScore2) {
68. swapStudents(studentIDs, studentIDs, unitNames, unitScores, j, j + 1);  // Swap the data of the two students.
69. }
70. }
71. }
72. }
73. int main() {
74. // Constants
75. const int NUM\_STUDENTS = 5;  // Maximum number of students.
76. const int NUM\_UNITS = 5;     // Maximum number of units per student.
77. // Arrays to store student data
78. int studentIDs[NUM\_STUDENTS];                  // Array to store student IDs.
79. char unitNames[NUM\_UNITS][50];                 // 2D array to store names of units for each student.
80. float unitScores[NUM\_STUDENTS][NUM\_UNITS];     // 2D array to store scores of each unit for each student.
81. // Read student data
82. printf("Enter data for %d students:\n", NUM\_STUDENTS);
83. for (int i = 0; i < NUM\_STUDENTS; i++) {
84. printf("\nStudent %d:\n", i + 1);
85. printf("Student ID: ");
86. scanf("%d", &studentIDs[i]);                      // Read the student ID of the current student.
87. getchar(); // Clear newline character from the input buffer (necessary after using scanf with %d).
88. for (int j = 0; j < NUM\_UNITS; j++) {
89. printf("Unit %d Name: ", j + 1);
90. fgets(unitNames[j], sizeof(unitNames[j]), stdin);      // Read the name of the unit for the current student.
91. unitNames[j][strcspn(unitNames[j], "\n")] = '\0';      // Remove trailing newline character from the input.
92. printf("Unit %d Score: ", j + 1);
93. scanf("%f", &unitScores[i][j]);                 // Read the score of the current unit for the current student.
94. getchar(); // Clear newline character from the input buffer (necessary after using scanf with %f).
95. }
96. // Sort students based on their average scores (from bottom to top)
97. sortStudents(studentIDs, unitNames, unitScores, NUM\_STUDENTS);
98. // Print summary in reverse order (from the bottom)
99. printf("\nSummary (from the bottom):\n");
100. for (int i = NUM\_STUDENTS - 1; i >= 0; i--) {
101. printf("\nStudent %d (ID: %d):\n", NUM\_STUDENTS - i, studentIDs[i]);
102. for (int j = 0; j < NUM\_UNITS; j++) {
103. printf("Unit %d: %s\n", j + 1, unitNames[j]);
104. printf("Score: %.2f\n", unitScores[i][j]);
105. printf("Grade: %c\n", getGrade(unitScores[i][j]));
106. }
107. }
108. return 0;
109. }