# Assignment Description

You are creating a program that reads grade information from a file named grades.txt. There are at most 20 student records in a file; each student will have 5 test scores.  You will create a 2d array of doubles to hold the test score information.  It will have a max size of 20 rows and 5 columns.  After reading the data into the array you will need to calculate statistics for each column and display them to the user.

# GitHub URL (optional)

<https://github.com/wesleyhixon/Programming-Assignments/tree/38445a44dcfc2eb797c12473fbbefd5520298f80/M06%20Part%20B%20Programming%20Assignment%202>

# Readme Documentation

Input Information: A list of up to 20 rows of 5 columns of numbers in the form of test grades

Output Information: Output is the highest score, lowest score, average score, and median score for each test.

# Flowchart Screen Shots (optional)

Screen shot(s) here

# UML and Use Case Diagrams (optional)

Screen shot(s) here

# Source Code of All files (.h, .cpp)

1. #include <iostream>
2. #include <fstream>
3. #include <sstream>
4. #include <iomanip>
5. using namespace std;
6. /\*
7. Program Name: Grade Analysis
8. Author: Wesley Hixon
9. Date Last Updated: 7/11/2024
10. Purpose: Given a .txt with up to 5 test scores for up to 20 students,
11. Calculate high, low, average, and median scores for each test.
12. \*/
13. // Function prototypes
14. void getHighScoreIndex(double scores[][5], int numRows, int colToAnalyze, int& highScoreIndex);
15. void getLowScoreIndex(double scores[][5], int numRows, int colToAnalyze, int& lowScoreIndex);
16. void getAverageScore(double scores[][5], int numRows, int colToAnalyze, double& average);
17. void getMedianScore(double scores[][5], int numRows, int colToAnalyze, double& median);
18. void selectionSort(double scoreColumn[], int numItems);
19. void getColumn(double scores[][5], double column[], int columnIndex, int numRows);
20. void getScores(double scores[20][5], string fileName, int& numRows);
21. int main(){
22. int numRows = 0;
23. double scores[20][5] = {  0  };
24. getScores(scores, "grades.txt", numRows); // Getting test scores and storing in scores[][]
26. cout << "Grade Statistics:" << endl;        // Statistics table
27. for(int column = 0; column < 5; column++){
28. int highScoreIndex, lowScoreIndex;      // Defining variables for statistics
29. double average, median;
30. getHighScoreIndex(scores, numRows, column, highScoreIndex); // Getting statistics
31. getLowScoreIndex(scores, numRows, column, lowScoreIndex);
32. getAverageScore(scores, numRows, column, average);
33. getMedianScore(scores, numRows, column, median);
34. cout << "Test " << column + 1 << endl; // Outputting table values
35. cout << fixed << setprecision(2);
36. cout << '\t' << "Highest Score: " << scores[highScoreIndex][column] << endl;
37. cout << '\t' << "Lowest Score: " << scores[lowScoreIndex][column] << endl;
38. cout << '\t' << "Average Score: " << average << endl;
39. cout << '\t' << "Median Score: " << median << endl;
40. cout << endl << endl;
41. }
42. return 0;
43. }
44. // This function gets the scores from the .txt and stores them in 2d array scores[][]
45. void getScores(double scores[20][5], string fileName, int& numRows){
46. ifstream gradesFile;
47. gradesFile.open("grades.txt");
48. string line;
49. for(int row = 0; row < 20; row++ ){                         // For each row,
50. if(gradesFile.eof()){                                   // If eof reached, break;
51. break;
52. }
53. getline(gradesFile, line);                              // Get the next line,
55. stringstream s(line);                                   // Create a stringstream to extract data
57. for(int column = 0; column < 5; column++){              // For each column,
58. string score;
59. s >> score;                                         // Extract 5 lines
60. scores[row][column] = stod(score);                  // And store them in scores[][]
61. }
62. numRows++;
63. }
64. gradesFile.close();
65. }
66. void selectionSort(double scoreColumn[], int numRows){
68. double temp;
69. bool sorted = false;
70. int notSorted = numRows - 1;
71. while(notSorted != 0){                      // Repeat this until every score is sorted
72. double largestScore = 0;                // Set the largest score to 0
73. int largestScoreIndex;                  // And define largestScoreIndex
74. for(int i = notSorted; i >= 0; i--){    // Decrement through the items which are not sorted
75. if(scoreColumn[i] > largestScore){  // If a new largest score is found,
76. largestScore = scoreColumn[i];  // replace largestScore
77. largestScoreIndex = i;          // and record the index
78. }
79. }
80. temp = scoreColumn[notSorted];                              // Store the value of the last index which is not sorted temporarily,
81. scoreColumn[notSorted] = scoreColumn[largestScoreIndex];    // Move the largest score to the end,
82. scoreColumn[largestScoreIndex] = temp;                      // And put the temporary value where the largest score used to be
83. notSorted--;                                                // Finally, decrement notSorted
84. }
86. }
87. // This function grabs the column we need from the 2d function
88. void getColumn(double scores[][5], double column[], int columnIndex, int numRows){
89. for(int row = 0; row < numRows; row++){       // For each row,
90. column[row] = scores[row][columnIndex];   // Grab the value at the column we are looking for
91. }
92. }
93. // This function finds the index of the highest score in a column
94. void getHighScoreIndex(double scores[][5], int numRows, int colToAnalyze, int& highScoreIndex){
95. double highestScore = 0;                            // Initializing highestScore for comparison
97. for(int i = 0; i < numRows; i++){
98. if(scores[i][colToAnalyze] > highestScore){     // If a new highest score is found,
99. highScoreIndex = i;                         // Store the index and record the new highest score
100. highestScore = scores[i][colToAnalyze];
101. }
102. }
103. }
104. // This function finds the index of the lowest score in a column
105. void getLowScoreIndex(double scores[][5], int numRows, int colToAnalyze, int& lowScoreIndex){
106. double lowestScore = 100;                           // Highest possible score on a test is 100
107. for(int i = 0; i < numRows; i++){
108. if(scores[i][colToAnalyze] < lowestScore){      // If a new lowest score is found,
109. lowScoreIndex = i;                          // Store the index and record the new lowest score
110. lowestScore = scores[i][colToAnalyze];
111. }
112. }
113. }
114. // This function finds the average score in a column
115. void getAverageScore(double scores[][5], int numRows, int colToAnalyze, double& average){
116. double sum = 0;                                     // Sum for later calculation
117. for(int i = 0; i < numRows; i++){                   // Calculate the sum of the column
118. sum += scores[i][colToAnalyze];
119. }
120. average = sum / numRows;                            // Calculate the average
121. }
122. // This function finds the median score in a column
123. void getMedianScore(double scores[][5], int numRows, int colToAnalyze, double& median){
124. double column[20] = {0};                            // Defining a column for future use
125. int middleIndex;                                    // Middle index to get the median score
126. getColumn(scores, column, colToAnalyze, numRows);   // Getting a column so that it can be sorted
127. selectionSort(column, numRows);                     // Sorting the column
128. if(numRows % 2 != 0){
129. numRows++;                                      // If there is an odd number of rows, add one
130. }
131. middleIndex = numRows / 2;                          // Find the middle row
132. median = column[middleIndex];                       // Find the median
133. }

# Three Use Case Screen Shots

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated