Jacob Hartt

CS2060.002

Professor Deborah Harding

February 2, 2023

HW04 Semester Grades and Statistics (Arrays)

Purpose: To demonstrate how to modularize your program using function and implement arrays.

Effort: Individual (Academic Integrity)

As a CS department we want to make sure we follow due process if we feel a code violation occurred. See UCCS Student Academic Ethics Code. This means all people involved in the violation, whether you took someone else's work or you gave your work to someone else. **These violations can lead to expulsion**. If a Code violation has occurred (due either to my own observation or due to a report by a third party) the following will happen

- Receive a grade of zero for the assignment.
- Have their names placed on the CS department's list for academic violations.
- Have their names sent to the Dean of Students.

Points: 100 (See rubric in canvas)

Deliverables: lastnameFirstnameHW04.c file and document with reflection/learning. Do not upload a zip file.

Description:

Write a program to simulate how canvas calculates your grades using weights and reports the class average. You do not need to submit a requirements analysis and design document but I suggest you approach this problem that way. If you need help with your code I will ask you to show me your design.

Follow SDLC to design and implement a solution that is

1. Solution Functionally Complete and Correct
2. Quality Readable Code
3. Maintainable and Secure Code

Requirements

The program calculates the final grade for each student based on the final grades for each category. The program calculates the final grade based on the category weights. Then it displays the final grade to the nearest tenth along with the letter grade and final class average. Canvas allows the categories to be created with percentage weights by the instructor. This information

is stored to make the calculations. You must make your program modularized and modifiable to support ease of implementing different weights and categories. The grades can be 0 to 105.

Grading Scale:

A	90 or greater
B	Greater than or equal to 80 and less than 90
C	Greater than or equal to 70 and less than 80
D	Greater than or equal to 60 and less than 70
F	less than 60

Example 1: In your class the following is used to calculate your grade.

Class Activity (10%)	Homework (30%)	Project (30%)	Midterm (15%)	Final 15%

Output

```
This program will calculate the grades for these categories 1. Learning Activity 2. Homework 3. Project 4. Midterm 5. Final
The category weights are
Category 1 weight is 0.10
Category 2 weight is 0.30
Category 3 weight is 0.30
Category 4 weight is 0.15
Category 5 weight is 0.15

The correct order to enter grades for each student is: 1. Learning Activity 2. Homework 3. Project 4. Midterm 5. Final

Enter the grade for each category for student 1, category 1: 100
Enter the grade for each category for student 1, category 2: 85
Enter the grade for each category for student 1, category 3: 90
Enter the grade for each category for student 1, category 4: 75
Enter the grade for each category for student 1, category 5: 88

Enter the grade for each category for student 2, category 1: 78
Enter the grade for each category for student 2, category 2: 80
Enter the grade for each category for student 2, category 3: 78
```

```
Enter the grade for each category for student 2, category 4: 67
Enter the grade for each category for student 2, category 5: 71

Enter the grade for each category for student 3, category 1: 100
Enter the grade for each category for student 3, category 2: 95
Enter the grade for each category for student 3, category 3: 80
Enter the grade for each category for student 3, category 4: 96
Enter the grade for each category for student 3, category 5: 90

Enter the grade for each category for student 4, category 1: 77
Enter the grade for each category for student 4, category 2: 88
Enter the grade for each category for student 4, category 3: 99
Enter the grade for each category for student 4, category 4: 88
Enter the grade for each category for student 4, category 5: 77

Grades entered for each student

Student 1 :    100    85     90     75     88
Student 2 :    78     80     78     67     71
Student 3 :    100    95     80     96     90
Student 4 :    77     88     99     88     77

Final grades for students, respectively:
Student 1:     87.0   B
Student 2:     75.9   C
Student 3:     90.4   A
Student 4:     88.6   B

Class average is 85.5
```

Example 2: In another class this is how the categories are weighted.

Class Activity (10%)	Homework (40%)	Quizzes (30%)	Final 20%

Example 3:

In another class this is how the categories are weighted.

Class Activity (10%)	Homework (40%)	Quizzes (10%)	Midterm (15%)	Final 15%

Specification

1. You will define the following constants above main that can be modified before running the program to accommodate different course set ups. These constants will be implemented in your code to make the program easily modifiable. Think about how these constants could be read from a database or configuration file before running the program.

   #define STUDENTS 4
   #define GRADE_CATEGORIES 5
   #define CATEGORIES "1. Learning Activity 2. Homework 3. Project 4. Midterm 5. Final "
   const **double** GRADE_CATEGORY_WEIGHT[] = { 0.1, 0.3, 0.3, 0.15, .15 };

2. Modularize your code into functions where main makes user defined function calls. You may have a puts/printf to display a header but logic should be in functions.
3. Store the student grades for each category in a two-dimensional array and store the final grades in a 1 dimensional array
4. Implement the least privilege principle when passing arrays.
5. Validate inputted grades to be integers values 0 to 105 (inclusive).
6. Do not implement concepts not covered yet in class. Can include information from lectures 1 to 9 and chapters 1 to 6.

```
DEFINE STUDENTS as 4
DEFINE GRADE_CATEGORIES as 5
DEFINE MIN_GRADE as 0.0
DEFINE MAX_GRADE as 105.0
DEFINE CATEGORIES as "1. Learning Activity 2. Homework 3. Project 4. Midterm
5. Final "
DECLARE constant GRADE_CATEGORY_WEIGHTS[]  as { 0.1, 0.3, 0.3, 0.15, .15 };
DEFINE A_MIN_GRADE as 90.0
DEFINE B_MIN_GRADE as 80.0
DEFINE C_MIN_GRADE as 70.0
DEFINE D_MIN_GRADE as 60.0

FUNCTION main():
      PRINT the grade categories using CATEGORIES
```

```
      ITERATE through GRADE_CATEGORY_WEIGHTS using GRADE_CATEGORIES and a variable
                                 gradeCategoryNum:
            PRINT the weight for each category

      PRINT the order the user should enter grades in using CATEGORIES

      DECLARE a 2d double array of size STUDENTS by GRADE_CATEGORIES called
                           studentGrades

      CALL enterGrades() with studentGrades, STUDENTS and GRADE_CATEGORIES

      PRINT grades entered header
      ITERATE up to STUDENTS with a variable studentNum:
            PRINT “Student #:  “

            ITERATE up to GRADE_CATEGORIES with a variable grade Category No:
                  Print “<Grade>  “

            PRINT the new line character

      DECLARE a 1D double array called finalGrades of size STUDENTS

      PRINT final grades header
      ITERATE up to STUDENTS with a variable studentNum:
            DECLARE finalGrade as the CALL of calcFinalGrade of studentGrades,
                                 studentNum and GRADE_CATEGORIES

            SET the studentNum’th element of finalGrades to finalGrade

            PRINT “Student #: <finalGrade> <CALL of gradeLetter() of finalGrade>“

      PRINT “Class Average: <CALL of calcAverage of finalGrades>“


FUNCTION calcAverage(arr[], arrSize):
      DECLARE sum as 0.0

      ITERATE through arr using arrSize:
            ADD the current element to sum

      RETURN sum divided by arrSize

FUNCTION calcFinalGrade(studentGrades[][GRADE_CATEGORIES], studentNum,
                           numGradeCategories):
      DECLARE sum as 0.0
```

```
	ITERATE through the studentNum row of studentGrades:
		ADD the current grade times GRADE_CATEGORY_WEIGHTS of the current grade
				to sum

	RETURN sum

FUNCTION enterGrades(studentGrades[][GRADE_CATEGORIES], numStudents,
                                                    numGradeCategories):
	ITERATE up to numStudents with a variable studentNum:
		ITERATE up to numGradeCategories with a variable gradeCategoryNum:
			SET studentGrades[studentNum][gradeCategoryNum] to the CALL of
				getGrade(studentNum, gradeCategoryNum)

		PRINT an empty line

FUNCTION gradeLetter(grade):
	DECLARE gradeLetter

	IF grade is at least A_MIN_GRADE:
		SET gradeLetter to 'A'
	ELSE IF grade is at least B_MIN_GRADE:
		SET gradeLetter to 'B'
	ELSE IF grade is at least C_MIN_GRADE:
		SET gradeLetter to 'C'
	ELSE IF grade is at least D_MIN_GRADE:
		SET gradeLetter to 'D'
	ELSE:
		SET gradeLetter to 'F'

	RETURN gradeLetter

FUNCTION getGrade(studentNum, gradeCategoryNum):
	DECLARE retVal
	DECLARE grade

	DECLARE isInputValid

	DO:
		PRINT the user prompt for studentNum's gradeCategoryNum grade

		SET retVal to the CALL of scanf("%lf", &grade)

		CLEAR the buffer

		SET isInputValid to if retVal >= 1 and the CALL of isGradeValid(grade)
```

```
        WHILE not isInputValid

        RETURN grade

FUNCTION isGradeValid(grade):
        RETURN grade ≥ MIN_GRADE and grade ≤ MAX_GRADE
```

Reflection and Learning

1. Explain how to implement a function to work with an array so that the function could be reused with different sized arrays. Use an example from your code in your explanation.
To implement a function which works with arrays of different sizes, be sure to input its dimensions as size_t arguments too. Take for example my calcAverage() function: notice how it takes in both an array and a size_t length indicator.

```
//! Calculates the average value of an array.
/*!
  \param arr the array whose values will be used
  \param size the size of the array
  \return The average of the array
 */
double calcAverage(const double arr[], size_t size)
{
        // Calculate the sum of the array
        double sum = 0.0;

        for (int i = 0; i < size; i++) {
        sum += arr[i];
        }

        // Return the sum divided by the total elements
        return sum / size;
}
```

2. How did you implement the least privilege principle in this assignment when passing arrays to a function? Use an example from your code in your explanation.

To implement the principle of least privilege, I made sure to implement array arguments as const arguments wherever they did not need to be modified inside the method. This is true for all functions in my code except enterGrades(). A good example of a const array argument is my calcFinalGrade() function: notice how it takes a const array input and only reads from said array.

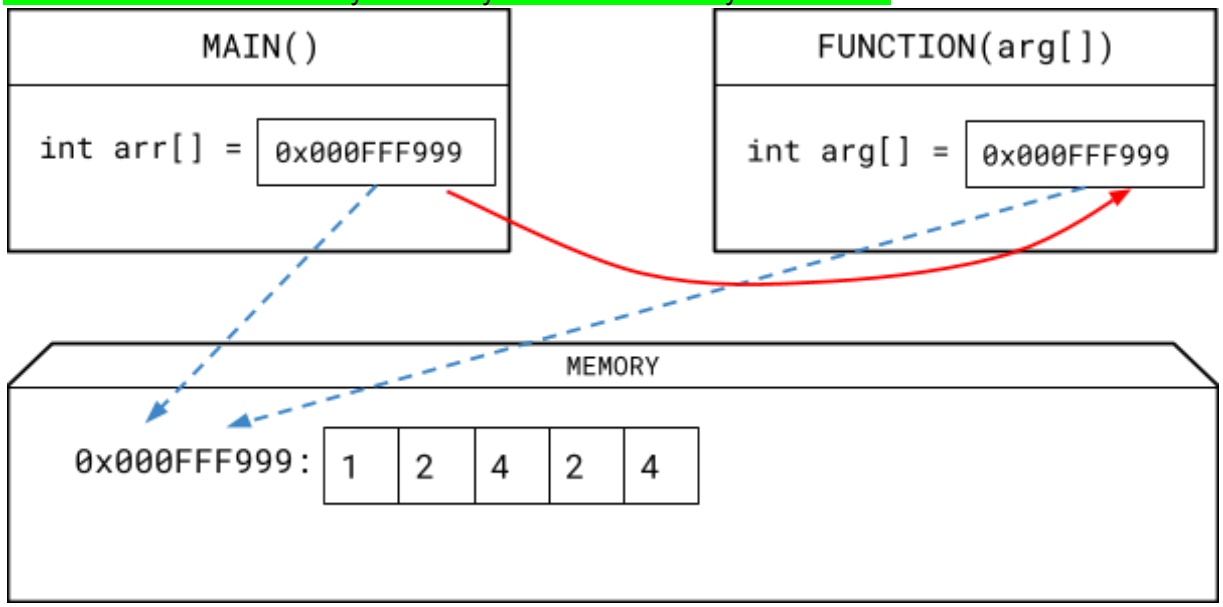
```
//! Calculates the final grade of a student for the class.
/*!
  \param studentGrades the input 2D array of class grades
  \param studentNum the students number / row in the 2D array
  \param numGradeCategories the number of grade categories stored in the array
  \return The final grade of a student
 */
double calcFinalGrade(const double studentGrades[][GRADE_CATEGORIES],
                      unsigned int studentNum, unsigned int numGradeCategories)
{
      double sum = 0.0;

      // Add the product of each weight and grade category to the sum
      for (int gradeCategoryNum = 0; gradeCategoryNum < numGradeCategories;
                                  gradeCategoryNum++) {
      sum += studentGrades[studentNum][gradeCategoryNum] *
            GRADE_CATEGORY_WEIGHTS[gradeCategoryNum];
      }

      return sum;
}
```

3. Explain how arrays are passed by reference and what is stored in the variable of an array identifier. Include in your explanation a drawing showing function stacks and what is happening in memory.

First, it is important to note that array variables store the address/pointer to the first element of an array in memory: it does not store the array itself.

Below, there is an illustration showing an array variable in main() called arr[] which contains a

memory address. The information at that memory address is shown in memory. If, per se, main were to call function() with arr[], the address, not the contents of the array, will be passed by value. This process, also known as pass by reference, is shown with the solid red arrow. Notice how there is still only one array of data in memory due to this.



4. What was the most difficult part of this assignment for you?

The most difficult part of this assignment was figuring out how to implement my functions in such a way that I could test each of them as I implemented it. For example, the calcFinalGrade() method depends on enterGrades() being validly implemented, enterGrades() depends on getGrade() being validly implemented, AND getGrade() depends on isGradeValid() being validly implemented.

5. What are you proud of from this assignment?

I'm proud of how relatively simple I managed to keep each of the functions in the program by modularizing it with new methods like letterGrade().