CS 162C++
1D Array exercises

Averaging with an Array

Create an integer array of 100 elements. Read numbers from the console and store them in the array until you read a zero. Keep a count of how many numbers you input (not including the final zero). You do not need to validate that the inputs are integers.

Once all the numbers are entered, add them up.

Display the sum and average, followed by listing the numbers five per line.

GradeBook

Create an integer array of 100 elements. Ask the user how many students there are. Then read in that number of scores (each score is between 1 and 100) and store them in the array. Validate your inputs.

After you have read in the numbers, go through it and then print out how many A’s (90-100), B’s (80-89), C’s (70-79), D’s (60-69) and F’s (0-59) there are.

Squares

For this exercise, you will create an array of 25 integers. For each location in the array you will store the square of its index.

Then you will print out these values in five rows of five numbers each.

Prime Numbers

For this exercise, you will use a sieve of Eratosthenes to find the prime numbers less than 100. Create an array of 100 integers and initialize them all to 1.

Then for each prime number up to 10 (2, 3, 5, 7) go through the array and mark any multiples of that number as not being a prime by changing its value to 0. (Start by setting locations 0 and 1 to zero).

Finally, go through the array and print out the indices of all remaining elements that have a value of 1. These are the prime numbers less than 100.

As a more advanced version, you could do this for all prime numbers less than 1000. Once you have gone through the array for 2 and 3, the next value with a 1 is the next prime and so forth.

Roman Numerals

For this exercise, you will input an integer between 1 and 20. You then will display the Roman numeral equivalent, using an array of strings to do the conversion.

Create an array of 20 strings and use static initialization to set them to the first 20 Roman numerals (I, II, III, IV, V, VI, VII, VIII, IX, X, XI, XII, XIII, XIV, XV, XVI, XVII, XVIII, XIX, XX).

You should validate the input. Your program should have an input function that gets an integer and validates it and a convert function that returns the proper Roman numeral.

As a more advanced version, you could do this for any number up to 1000 (50 is L, 100 is C, 500 is D, 1000 is M). Find a way to do this without having to build a table of 1000 strings.

IsPalindrome

A palindrome is a word that reads the same forwards and backwards; For example: “noon” or “racecar”.

Write a function isPalindrome which takes an array of characters and the length of that array; returning true if the characters form a valid palindrome and false otherwise.

Your main program will use this function to test the following possible palindromes:

“racecar”, “step on no pets”, “devil lived”, “elephant”, “nothing”

and for each one output the string and then say whether it is a palindrome or not.

Chips and Salsa

Write a program that lets a maker of chips and salsa keep track of their sales for five different types of salsa they produce: mild, medium, sweet, hot, and zesty.

It should use two **parallel** five element arrays: an array of strings that holds the five salsa names and an array of integers that holds the number of jars sold during the past month for each salsa type.

The salsa names should be stored using an initialization list at the time the name array is created. The program should prompt the user to enter the number of jars sold for each type.

Once this sales data has been entered, the program should produce a report that displays sales for each salsa type, total sales, and the names of the highest selling and lowest selling type.