CSE 130 Midterm 2 — Practice Version

1. Please sit at the machine to which you have been assigned. You can find the machine number on a label on the lid of each computer. **Be sure to sign the attendance sheet as well.**

2. Complete the problems that follow. The score for a perfect final exam is 150 points.

3. Partial credit may be assigned at the instructor's discretion. **Programs that do not compile will receive 0 points.**

4. The exam is open-book, open-notes, limited Internet. You will only have access to one Web site during the exam (the course Web page) which can be found at:

   http://www.cs.stonybrook.edu/~tashbook/spring2017/cse130/

5. **You may not use cell phones, tablets, e-readers, pagers, or any other electronic devices during the exam, except for USB flash drives, USB mice, and the computers that are installed in the exam room.** Please turn all prohibited devices **OFF** for the duration of the exam.

6. You **may not** communicate in any fashion (notes, texting, talking, etc.) with any other students during the exam. **Any suspected cheating will result in a grade of 0 for the exam and academic dishonesty charges.**

7. At the start of the exam, your instructor will tell you where to find the questions and the starting code for the exam problems. You will also be provided with instructions for submitting your completed work.

1. **Middle Names (50 points)**

   Complete the "middle.c" program, which reads in a full "name" (multiple words, separated by single spaces) from the user. This program prints out **all** of the middle name(s) from the input, on a single line; the first and last words (and their associated spaces) are omitted. You may assume that the user input always contains at least three words. **Hint:** use pointers to walk along the input string (character array) to locate the first and final spaces; for the last one, you may need to backtrack from the end of the string!

Sample Input	Expected Output
Mary Jane Watson	Jane
John Jacob Jingleheimer Schmidt	Jacob Jingleheimer
Juan Paco Pedro de la Mar	Paco Pedro de la

2. **Inverse Deltas of an Array (50 points)**

   Suppose that you have an array of integers. The ***delta*** (difference) value between two adjacent array elements is equal to the second element minus the first. For example, the array [4 6] has a single delta of 2 (which is 6 - 4). An array of *N* elements will thus have a delta array with *N-1* elements (one delta value per pair of adjacent elements). We can easily compute the delta values for an array of integers using a loop and a second integer array (with one fewer element).

   The ***inverse delta*** of two array values is just -1 times the delta value (in other words, the first value minus the second value). This is just as easy to compute as the original array of delta values; just switch the order of the operands in the subtraction.

   Finally, we can use our list of inverse delta values to construct a new array of integers from the original array's starting value. The second element of the new array is equal to the first element plus the first inverse delta. The third element of the new array is equal to the second element of the new array plus the second inverse delta, and so on.

   For example, consider the starting array [10 3 6 2 5]. The original delta values for this array would be [-7 3 -4 3]. The inverse delta values would thus be [7 -3 4 -3]. Applying these inverse delta values to the original starting value (10), we would get [10 17 14 18 15] (for example, 10 + 7 gives us 17, 17 + -3 gives us 14, 14 + 4 gives us 18, etc.).

   Complete the "inverse-deltas.c" program, which reads in a series of (positive and/or

negative) integers from the user. This program should compute the inverse deltas of this input array, and then print a new array that applies those inverse delta values to the first value entered by the user (we have provided two helper functions, `fillArray()` and `printArray()`, for this purpose).

Sample Input	Inverse Deltas	Final Array
3 10 2 3 5	-7 8 -1 -2	3 -4 4 3 1
2 -3 7 12 5 1 -7 5	5 -10 -5 7 4 8 -12	2 7 -3 -8 -1 3 11 -1
11 -5 13 4	16 -18 9	11 27 9 18

3. **Palindromes (50 points)**

A palindrome is a string that reads the same forward and backward. Complete the "palindrome.c" program, which reads in a string from the user and reports whether or not it is a palindrome (you may assume that the input is always all-lowercase). **Hint:** Use two pointers to move in from the ends of the string, comparing character values as they go, until they meet in the middle (or find a mismatch).