COP 3502 Quiz #1 Version B (SLMP, Dynamic Memory Allocation) Solutions

1) (6 pts) Write three lines of code which do the following: (1) read in an integer from the user and store it in a variable called n, (this is two lines) (2) dynamically allocate an array of n doubles called nums, (3) set each of those values to 0.0. (#1 takes 2 lines and 2 and 3 can be done in one line.)

```
int n;
scanf("%d", &n);
double* nums = calloc(n, sizeof(double));
```

Grading: 1 pt declare n, 1 pt scanf(must have & to get the pt), 4 pts last line (1 pt LHS, 1 pt calloc, 1 pt n, 1 pt sizeof(double))

2) (12 pts) Edit the code below so that the code returns a dynamically allocated array of integers representing the list of unique items that belong to either list. To do this, you must initially allocate an array the size of the sum of the sizes of the two input arrays. Then, resize the array after you fill it to the proper size. Blank lines have been provided for you to add the necessary code. (Note: *news is set to the size of the returned list.)

```
// Pre-condition: list1 is length n, sorted, with unique values.
//                list2 is length m, sorted with unique values.
// Post-condition: All elements belonging to both lists are
//                 returned in a dynamically allocated array.
int* slist_union(int list1[], int n, int list2[], int m, int* news) {

    int* res = calloc(n+m, sizeof(int)) ;  // 2 pts many correct ans

    int i = 0, j = 0, k = 0;
    while (i < n || j < m) {
        if (j==m || (i<n && list1[i] < list2[j])) {

            res[k] = list1[i] ;   // 2 pts, 1 pt LHS, 1pt RHS
            i++; k++;
        }
        else if (i==n || list2[j] < list1[i]) {

            res[k] = list2[j] ;   // 2 pts, 1 pt LHS, 1pt RHS
            j++; k++;
        }
        else {
            res[k] = list1[i] ;// 2 pts, 1 pt LHS, 1pt RHS(list2[j]ok)
            i++; j++; k++;
        }
    }

    res = realloc(res, k*sizeof(int));      // 3 pts, 1 real, 1 res
                                            //   1 pt k*so(i)
    *news = k;                          // 1 pt
    return res;
}
```

3) (12 pts) One can create a custom addition table using one array of integers storing the first operand and another array of integers storing the second operand. For example, if the array [3, 2, 6] represented the possible first operands and the array [5, 7, 9, 12] represented the second operand, then the following table would be built to represent all possible multiplications that could be done:

	5	7	9	12
3	8	10	12	15
2	7	9	11	14
6	11	13	15	18

Note: The actual table has 3 rows and 4 columns and its entries are highlighted.

Write a complete function which takes in the first array, list1, the length of the first array, len1, the second array, list2 and the length of the second array, len2, and dynamically allocates a 2D integer array of size len1 by len2, fills it with the custom multiplication table defined by the two input lists, and returns a pointer to that 2D array.

```
int** makeCustomAddTable(int* list1, int len1, int* list2, int len2) {


    int** res = malloc(len1*sizeof(int*));        // 2 pts
    for (int i=0; i<len1; i++)                    // 1 pt
        res[i] = malloc(len2*sizeof(int));        // 2 pts

    for (int i=0; i<len1; i++)                    // 1 pt
        for (int j=0; j<len2; j++)                // 2 pts
            res[i][j] = list1[i]+list2[j];        // 3 pts

    return res;

}
```

Grading Note: If no nested loops max credit is 7 out of 12 (because inner lines only make sense if they are properly nested).

4) (5 pts) Assume that in the main function that called makeCustomAddTable, the pointer to the 2D array is called addtable and the variables len1 and len2 store the dimensions of the 2D array. (So, there are len1 pointers to 1D arrays and each of those 1D arrays are of length len2.) Write code to free all of this associated memory.

```
for (int i=0; i<len1; i++)        // 1 pt (must be to len1 to get credit)
    free(addtable[i]);            // 3 pts (1 pt for each item)
free(addtable);                   // 1 pt
```