COP 3502 Quiz #1 Version A (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 chars called word, (3) set each of those chars to '\0'. (Note: The integer value of the null character is 0. Steps 2 and 3 are one line.)

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

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(char))

2) (12 pts) Edit the solution to the sorted list matching problem below so that instead of printing out the solution to the screen, the code returns a dynamically allocated array of integers representing the list of items common to both lists. To do this, you must initially allocate an array the size of either of the two input arrays, but after you have copied the result into that array, you must resize the array (using a function that we learned) before returning it. 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* slmp_linear(int list1[], int n, int list2[], int m, int* news) {

    int* res = malloc(n*sizeof(int)); // 3 pts many correct ans

    int i = 0, j = 0, k = 0;
    while (i < n && j < m) {
        if (list1[i] < list2[j]) i++; // 1 pt i=i+1; also ok
        else if (list2[j] < list1[i]) j++;
        else {

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

            k++ ;                 // 1 pt
            i++;
            j++;
        }
    }

    res = realloc(res, k*sizeof(int)) ;  // 4 pts - 1 realloc
                                         // 1 pt k, 1 pt *, 1 pt soi

    *news = k;                           // 1 pt
    return res;
}
```

3) (12 pts) One can create a custom multiplication 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	15	21	27	36
2	10	14	18	24
6	30	42	54	72

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** makeCustomMultTable(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;                                     // 1 pt

}
```

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 makeCustomMultTable, the pointer to the 2D array is called multtable 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(multtable[i]);           // 3 pts (1 pt for each item)
free(multtable);                  // 1 pt
```