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

1) (6 pts) Describe the key differences between malloc and calloc in words.

malloc takes in 1 input parameter while calloc takes in 2 input parameters (3 pts).
malloc does NOT zero out the memory allocated but calloc does. (3 pts)

2) (12 pts) Complete the function below so it that takes in a sorted array, list, of size n, and two integers, low and high (with low < high), and creates a dynamically allocated integer array which stores the values from the original array list that are in between low and high, inclusive, in order, and returns a pointer to that array. (In addition, the function sets *news to the length of the returned array.) **You are guaranteed that at least one number in the array list is in between low and high.**

```
// Pre-condition: list is length n, sorted, with unique values.
//                list[0] <= max
// Post-condition: All elements in list in between low and high
//                 are stored in an array and a pointer to that
//                 array is returned. In addition, *newlen is set to
//                 the length of the returned array.
int* rangelist(int list[], int n, int low, int high, int* newlen) {

    int i=0;
    while ( i<n && list[i] < low) i++;      // 1 pt

    int j = i;

    while ( j<n  && list[j] <= high) j++ ;  // 1 pt, 2 pts


    int* res = malloc((j-i)*sizeof(int)) ;  // 4 pts - 1 pt malloc
                                            // 1 pt - so(i), 2 pts j-i

    for (int z=0; z<j-i; z++)

        res[z] = list[i+z];                 // 2 pts

    *newlen = j-i ;                         // 2 pts
    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

}
```

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
```