<u>COP 3502 Quiz #1 Version A (SLMP, Dynamic Memory Allocation) Solutions</u>

1) (8 pts) Assume that the struct `artifact` had been created by the user and a typedef was set up to refer to the type as `artifact`. Write lines of code that do the following:

1) Read in an integer (from standard input) into an integer variable **`n`**, that is already declared.

2) Dynamically allocate memory for an array of size n, storing n **<u>pointers</u>** to `artifact`. Call the array **`arr`**.

3) Manually, using a for loop, set each of the pointers in the array to NULL.

```
scanf("%d", &n);

artifact** arr = calloc(n, sizeof(artifact*));

for (int i=0; i<n; i++)
    arr[i] = NULL;
```

Grading: 2 pts scanf, 3 pts either malloc or calloc, 3 pts loop (2 pts first line, 1 pt inside line)

2) (10 pts) Write a function that takes in an array, `array`, its length, `n`, and dynamically allocates **<u>a new array</u>** of size `2n`, and fills it with the contents of `array`, followed by those same contents reversed, so that the resulting array is a palindrome. So, if the input array stores [3,4,5,1,9], the newly returned pointer will point to a new array storing [3,4,5,1,9, 9,1,5,4,3]. No changes should be made to the original array.

```
// Pre-condition: array is of length n.
// Post-condition: No change is made to array and a new array of
//   size n is created storing array's contents followed by array's
//   contents reversed, and a pointer to this new array is returned.
int* makePalArray(int array[], int n) {

    // 3 pts, can use calloc or malloc.
    int* res = calloc(2*n, sizeof(int));

    // 2 pts for loop, assignment.
    for (int i=0; i<n; i++)
        res[i] = array[i];

    // 2 pts for loop, 2 pts for assignment.
    for (int i=n; i<2*n; i++)
        res[i] = array[2*n-1-i];

    // 1 pt for returning correct pointer.
    return res;
}
```

3) (10 pts) Write a function that takes in an array of strings, `words`, its length, `n`, representing the number of words in the array, and returns a pointer to a new integer array, dynamically allocated, where index i of the new array stores the length of the string stored in `words[i]`.

```
#include <string.h>
int* getStringLengths(char** words, int n) {

    // 4 pts, can use calloc or malloc.
    int* res = malloc(n*sizeof(int));

    // 2 pts for loop, 2 pts for assignment
    for (int i=0; i<n; i++)
        res[i] = strlen(words[i]);

    // 2 pts for returning correct pointer.
    return res;
}
```

4) (7 pts) When the `strcat` function is called, it's the programmer's responsibility to see if the first string passed to the function has enough memory allocated to it to have the second string concatenated to the end of it. Let `len1` be the amount of space allocated to the string `str1` and `len2` be the amount of space allocated to the string `str2`. In the code segment below, two strings are read into str1 and str2. In the space provided after that, please write the necessary code to reallocate memory for `str1` before the strcat function call. Make sure this realloc allocates precisely the correct amount of memory needed to store both strings.

```
#include <string.h>

int len1, len2;
scanf("%d%d", &len1, &len2);
char* str1 = malloc(len1*sizeof(char));
char* str2 = malloc(len2*sizeof(char));
scanf("%s%s", str1, str2);

// 4 pts for this calculation.
int need = strlen(str1) + strlen(str2) + 1;

// 3 pts for this realloc. LHS and assignment unnecessary.
str1 = realloc(str1, need);

strcat(str1, str2);
```