CMSC 216 Quiz 3 Worksheet

The next quiz for the course will be on Wed, Feb 19. The following list provides additional information about the quiz:

- The quiz will be a written quiz (no computer).
- The quiz will be in lab session.
- Closed book, closed notes quiz.
- Answers must be neat and legible.
- Quiz instructions can be found at http://www.cs.umd.edu/~nelson/classes/utilities/examRules.html
- Make sure you know your section number and your TA's name.

The following exercises cover the material to be included in this quiz. Solutions to these exercises will not be provided, but you are welcome to discuss your solutions with the TA or instructor during office hours. If is recommended that you try this exercises on paper first (without using the computer).

Exercises

1. What makes a character array a string? Is every character array a string?

2. The function **filter** has the following prototype:

```
static int filter(const int src[], int dest[], int array_size,
                  int lower_bound, int upper_bound)
```

   The function initializes the dest array with elements that have values in the range defined by lower_bound(inclusive) and upper_bound (inclusive). The size of src and dest is array_size. The function will return the number of elements placed in dest. You can assume lower_bound is less than or equal to upper_bound.

3. Implement the function **maximum** that has the prototype below. The function computes the maximum in the array and returns that value via the max parameter. If the array has a size of 0, the pointer variable associated with the argument must be set to NULL. The following code fragment illustrates how the function will be used.

```
int b[] = {30, 5, 80, 4};
int max;
int * max_ptr = &max;

maximum(b, 4, &max_ptr);
if (max_ptr == NULL) {
   printf("Array size is 0\n");
} else {
   printf("%d\n", max);
}
```

   You can assume the array passed to the function will have positive elements (if the array size is different from 0).

```
static void maximum(const int a[], int a_size, int **max)
```

4. Implement the function **intersection** that has the following prototype:

```
static int intersection(const int *a, int size_of_a, const int *b, size_of_b,
                        int *common);
```

   The function will initialize the common array with elements that are present in arrays a and b. The function will return the number of elements in common. You can assume the common array is large enough to fit the result.

5. Write the output generated by the program below. In cases where you are asked to print an address, write **NULL** or **MEMORY_ADDRESS** (for any other value).

```
#include <stdio.h>

#define SIZE 5

void task(int *a, int **b) {
   printf("R5: %d\n", a[2]);
   a = NULL;
   *b = NULL;
}

int main() {
   int data[SIZE] = {3, 9, 7, 11};
   int *p = data;

   printf("R1: %d\n", *p);
   printf("R2: %d\n", *data);
   p[1] += 100;
   printf("R3: %d - %d\n", data[0], data[1]);
   printf("R4: %d\n", data[4]);

   task(data, &p);
   printf("R6: %p\n", (void *) data);
   printf("R7: %p\n", (void *) p);

   return 0;
}
```

6. Implement the function **get_nonnegative_values** that has the prototype below. The function assigns to the **dest** array all the non-negative values (including 0) that are present in the **src** array. The last parameter (int *dest_size) is an out parameter, which the function must set to the number of non-negative values (including 0) found. The function returns the number of **negative** values present in the **src** array. For example, the following code fragment will initialize **dest** with the values {10, 2, 0, 6}, **dest_size** with 4 and **negatives** with 3. You can assume SIZE is 7.

```
int data[SIZE] = {10, -1, 2, -8, 0, 6, -5};
int dest[SIZE], dest_size;
int negatives = get_nonnegative_values(data, 7, dest, &dest_size);
```

```
int get_nonnegative_values(const int src[], int src_size, int dest[], int
                                   *dest_size)
```