CMSC 216 Quiz 4 Worksheet

The next quiz for the course will be on Fri, Nov 13. The following list provides additional information about the quiz:

- Do not post any solutions to this worksheet in Piazza. That represents an academic integrity violation.
- 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). **At the end we have provided an example of a memory map, so you know exactly what we are expecting while drawing maps. Take a look at the example before drawing any maps.**

Exercises

1. How do you run valgrind on an executable named my_prog?

2. Which of the following pointer variables uses the largest number of bytes?

```
int *p;
char *q;
```

3. What will happen when the following code is executed? Explain briefly.

```
int *p = NULL;
*p = 20;
```

4. What will happen when the following code is executed? Explain briefly.

```
int *p;
*p = 20;
```

5. What will happen when the following code is executed? Notice that **a** has not been initialized Explain briefly.

```
int a;
int *a_ptr = &a;

printf("%p\n", (void *)a_ptr);
```

6. What is the difference between a void pointer variable and a non-void pointer variable (e.g., an integer pointer variable)?

7. Write a memory map for the program below. In cases where you are asked to print an address, write **NULL** or **MEMORY_ADDRESS** (for any other value). To help you understand pointers better you may assume some memory addresses while drawing the memory map (as we did in lecture).

```
#include <stdio.h>

void process(float *passed_card);

void process(float *passed_card) {
   *passed_card += 200;
   printf("In process_one %.2f\n", *passed_card);
   passed_card = NULL;
}

int main() {
   float bank_account = 500.00;
   float *card_one, *card_two;

   card_one = card_two = &bank_account;
   printf("V1: %.2f\n", bank_account);
   printf("V2: %.2f\n", *card_one);
   printf("V3: %.2f\n", *card_two);
   printf("V4: %p\n", (void *)card_two);
   printf("V5: %p\n", (void *)&bank_account);

   *card_two += 20.0;
   printf("V6: %.2f\n", *card_two);
   card_two = NULL;
   printf("V7: %.2f\n", *card_one);

   process(card_one);
   printf("V8: %p\n", (void *)card_one);
   printf("V9: %.2f\n", bank_account);

   return 0;
}
```

8. 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)
```

9. 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;
}
```

10. 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)
```

Sample Memory Map

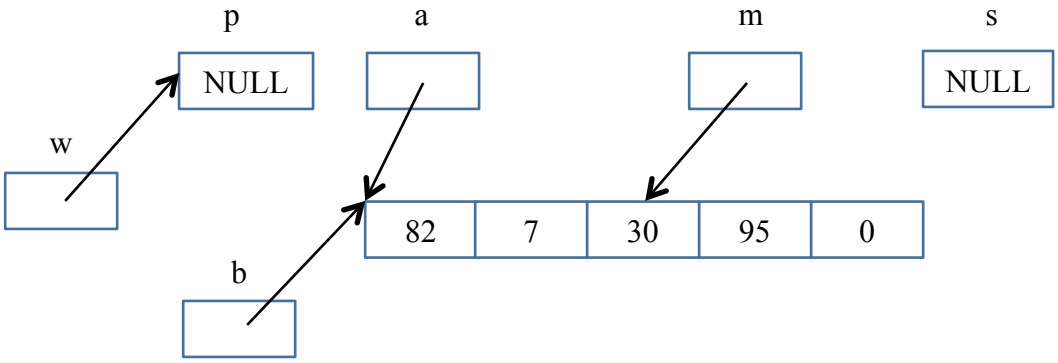
We are providing this example so you know what we are expecting for memory maps.

Example

Draw a memory map for the following program up to the point indicated by the comment **/*HERE */**.

```
#include <stdio.h>

#define MAX_LEN 5

void process(int *b, int *s, int **w) {
    b[0] = 82;
    s[1] = 95;
    s = NULL;
    *w = NULL;
    /* HERE */
}

int main() {
   int a[MAX_LEN] = {10, 7, 30, 40};
   int *p = a;
   int *m = a + 2;

   process(p, m, &p);

   return 0;
}
```

Answer:



Note: You can also replace NULL with the ground symbol as done in lecture. For example, s above could be represented as: