CMSC 216 Quiz 4 Worksheet

The next quiz for the course will be on Thu, Jul 7. 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](http://www.cs.umd.edu/~nelson/classes/utilities/examRules.html)
- **Regarding Piazza** - Feel free to post questions in Piazza regarding the worksheet and possible solutions to problems, but for coding questions please do not post code. You can post suggestions on how to solve coding problems, but your classmates will benefit more if they themselves actually solve the problems. Pretend you are a TA while addressing or providing help in Piazza 😊

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. What situation has occurred after the last assignment of the following code?
   ```c
   int *p = malloc(sizeof(int));
   *p = 400;
   p = NULL;
   ```

2. How are the malloc and calloc functions different?

3. Name one advantage of initializing pointer variables to NULL?

4. What is the problem in the following code?
   ```c
   double scores[100];
   double *p = scores;
   scores[0] = 13.2;
   free(p);
   ```

5. Implement the `append` function that has the prototype below. The function returns a string that represents the concatenation of all the strings present in an array of strings. For this problem, you can assume the end of the parameter array is marked by NULL. You need to allocate memory for the resulting string. You may not modify the array parameter.
   ```c
   char* append(char *data[]);
   ```

6. Write a function named `longest` that has the prototype below. The function returns a copy of the longest string in an array of strings. The end of the array is marked by an entry with a value of NULL. You can assume the `data` array will have at least one string. The function must free all the memory associated with the `data` array.
   ```c
   char* longest(char **data);
   ```
7. The following structures will be used for the questions that follow:

```c
#define MAX_LEN 80

typedef struct {
    char *title;
    int duration;
} Song;

typedef struct {
    Song* all_songs; /* array of songs */
    int number_of_songs;
    char album_name[MAX_LEN + 1];
} Album;
```

a. Given two Song structures s1 and s2, are there any problems with assigning one structure to another? Is deep copying taking place during the assignment of these two structures?

b. Define a function that initializes a Song using a duration and a string provided. The function will make a copy of the string parameter.

c. Define a function that will initialize an Album structure based on an array of Songs provided as parameter. The function will create a dynamically-allocated array of Songs and will initialize each entry with a copy of the corresponding song in the parameter array.

8. The following C statement creates an array of 14 characters.

```c
char *desc = (char *) malloc(sizeof(char) * 14);
```

Indicate whether the code can be simplified or not. If the code cannot be simplified indicate NO. If it can be simplified rewrite the code below.

9. The Course structure below represents a college course where name is a dynamically-allocated string and credits represents the course credits. For this problem you will implement the functions init and destroy.

```c
typedef struct course {
    char *name;
    int credits;
} Course;
```

a. The init function returns a dynamically-allocated Course structure where the name field points to dynamically-allocated memory that has been initialized with the string parameter (name). The only functions from the string library that you can use are strcpy and strlen (you may NOT use strdup). Remember to initialize the credits field with the parameter value.

```c
Course *init(const char *name, int credits)
```

b. The destroy function gives back to the memory allocator any dynamic memory associated with a Course structure that was created via the init function. In addition, the pointer to the Course structure is set to NULL after the memory has been given back.

```c
void destroy(Course **course)
```

Sample Driver and Output (Notice we are using a print_course function you do not need implement).

```
<table>
<thead>
<tr>
<th>Driver</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>int main() {</td>
<td>Name cmsc216</td>
</tr>
<tr>
<td>Course *course1 = init(&quot;cmsc216&quot;, 4);</td>
<td>Credits 4</td>
</tr>
<tr>
<td>print_course(course1);</td>
<td>Yes</td>
</tr>
<tr>
<td>destroy(&amp;course1);</td>
<td></td>
</tr>
<tr>
<td>printf(&quot;%s\n&quot;, course1 == NULL ? &quot;Yes&quot; : &quot;No&quot;);</td>
<td></td>
</tr>
<tr>
<td>return 0;</td>
<td></td>
</tr>
</tbody>
</table>
```

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:

```
   p
      NULL

   w

   m

   s
      NULL

   a
      82  7  30  95  0

   b

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

   s
      ____________
      
      ____