CMSC 216 Quiz 4 Worksheet

The next quiz for the course will be on Wed, Mar 29. 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
- **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 ☺

Exercises

1. What is wrong with the following code?

```
#include <stdio.h>
#include <stdlib.h>

int *get_val(int y) {
   int x = 200;

   x += y;

   return &x;
}

int main() {
   int *p, y = 10;

   p = get_val(y);
   printf("%d\n", *p);

   exit(0);
}
```

2. What situation has occurred after the last assignment of the following code?

```
int *p = malloc(sizeof(int));
*p = 400;
p = NULL;
```

3. How are the malloc and calloc functions different?

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

5. What is the problem with the following code fragment?

```
double scores[100];
double *p = scores;
scores[0] = 13.2;
free(p);
```

6. 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.

```
char* append(char *data[]);
```

7. 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.

```
char* longest(char **data);
```

8. The following structures will be used for the questions that follow:

```
#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. Feel free to add to the function any parameters you understand are needed.

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

```
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.

10. The following two structures will be used for the questions below. A **Customer** structure represents a bank customer and the **Account** structure a bank account.

```
typedef struct {
   int number;
   double balance;
} Account;

typedef struct {
   char *name;
   Account *account;
} Customer;
```

   a. Implement the **init** function that has the prototype below. The function will:

      i. Dynamically allocate memory for a **Customer** structure and assign the address of that structure to the Customer pointer variable associated with the **customer** out parameter.
      ii. Dynamically allocate enough memory for the **name** field of the Customer structure so we can copy the **name** parameter.
      iii. Dynamically allocate memory for an **Account** structure.
      iv. Initialize the **number** and **balance** fields based on the parameter values.
      v. If the **name** or **customer** parameters are NULL the function will not do any processing and will return false; otherwise the function will complete the processing described above and return true.
      vi. You can assume memory allocations are successful (you do not need to check values returned by malloc nor calloc).

```
int init(const char *name, int number, double balance, Customer **customer)
```

b. Implement the **destroy_customer** function that has the prototype below. The function gives back to the memory allocator any dynamic memory created via the **init** function.

```
void destroy_customer(Customer *customer)
```

Below we have provided a sample driver and the expected output. Feel free to ignore it if you know what to implement. Notice you do not need to implement the print_customer function.

Driver	**Output**

`int main() {`
`  Customer *customer;`
 	
`  if (init("Peter", 124, 500.00, &customer)) {`
`  print_customer(customer);`
`  destroy_customer(customer);`
`  }`
 	
`  return 0;`
`}`
	`Name: Peter`
	`Account Number: 124`
	`Account Number: 500.00`