CIS 308 Exam 1 (50 points)
October 1/2, 2012

Name:_____Solution___________

1. (10 pts) Suppose that we have the following variable declarations:

```
int nums1[10];
int *nums2;
```

Further suppose that we have dynamically given `nums2` space for 5 elements, that `nums1` has the values {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, and that `nums2` has the values {4, 6, 8, 10, 12}. Show what is printed by each of the following statements, assuming that they are executed in order. (If the statement has an error, describe the error.)

a) `printf("%d\n", *nums1);`

1

b) `printf("%d\n", *(nums2+2));`

8

c) `printf("%d\n", (nums1+1)[1]);`

3

d) `printf("%d\n", *(nums1++));`

not allowed (nums1 is a constant array)

e) `printf("%d\n", *(++nums2));`

6

2. (6 pts) Write the **call-by reference** C function `triple` that takes the address of an integer and updates its contents to be three times the value of the original number. **Give an example** of calling `triple`.

```
void triple(int* num) {
     *num = *num*3;
}

Example:

int x = 4;
triple(&x);
```

3. (11 pts) In this question, you will perform operations on a dynamic two-dimensional array. Consider the following variable declarations:

```
int** triangle;
```

a) (3 pts) Allocate memory so that triangle can hold 10 rows. The first row must hold 1 element, the second row 2 elements, etc. Use a loop to allocate memory for each row.

```
int i;
triangle = malloc(10*sizeof(int*));
for (i = 0; i < 10; i++) {
     triangle[i] = malloc((i+1)*sizeof(int));
}
```

b) (3 pts) Use nested loops to initialize each element in `triangle` so that the array has the following pattern:

1
2 1
3 2 1
... (continue the pattern)

```
int j;
for (i = 0; i < 10; i++) {
        for (j = 0; j < i+1; j++) {
                triangle[i][j] = i-j+1;
        }
}
```

c) (3 pts) Write a code fragment that asks the user for a number from 0-9. (You may assume that the user enters a number in that range.) Write a loop that uses pointer notation to print every value from `triangle` on that row. (Note: it was announced during the exam that you didn't need to use pointer notation.)

```
int row;
printf("Enter row: ");
scanf("%d", &row);
for (i = 0; i < row+1; i++) {
   printf("%d\n", triangle[row][i]);
}
```

d) (2 pt) Free all of the memory allocated in this problem.

```
for (i = 0; i < 10; i++) {
     free(triangle[i]);
}

free(triangle);
```

4. (6 pts) Show what is printed by each of the three print statements.

```
char str1[20] = “first”;
char *str2 = “second”;

printf(“%d, %d\n”, strlen(str1), sizeof(str2));
strcat(str1, str2);
printf(“%s, %s\n”, str1, str2);
strcpy(str1, str2);
str1[2] = ‘f’;
printf(“%s %s\n”, str1, str2);
```

5, 4 (a pointer is a memory address [int], which usually uses 4 bytes)
firstsecond, second
second
sefond

5. (6 pts) Define a `struct` that represents an athlete. You should include fields for the **name** and the **sport** (can be swimming, running, or cycling – use an `enum`). Each athlete can be EITHER an **elite athlete** or a **recreational athlete** (use a `union`). We also want to keep track of whether we're storing an elite or recreational athlete (use an `enum`).

   For elite athletes (use a `struct`), we want to store the **sponsor** (Asics, Speedo, or Specialized – use an `enum`) and their **world ranking** (integer).

   For recreational athletes, we want to store their **age group** (use a character array) and the **number of years** they've been participating (integer).

   Use `typedef` to rename the overall type to `ATHLETE`.

```
typedef struct {
     char name[30];
     enum {swim, run, cycle} sport;
     enum {elite, rec} athType;
     union {
          struct {
               enum {Asics, Speedo, Specialized} sponsor;
               int rank;
          } eliteType;
          struct {
               char ageGroup[30];
               int years;
          } recType;
     } athUnion;
} ATHLETE;
```

6. (3 pts) Declare a pointer to an `ATHLETE` called `athPtr`. **Allocate memory** for the object, and **set its values** to “Joe”, swimming, elite, Speedo, and rank 2. Finally, **free** the allocated memory.

```
ATHLETE* athPTr = malloc(sizeof(ATHLETE);
strcpy(athPtr->name, "Joe");
athPtr->sport = swim;
athPtr->athType = elite;
athPtr->athUnion.eliteType.sponsor = Speedo;
athPtr->athUnion.eliteType.rank = 2;

free(athPtr);
```

7. (8 pts) Consider the `NODE` structure below, which represents a node in a singly-linked list.

```
typedef struct node {
     int data;
     struct node *next;
} NODE;
```

Write the C function:

```
NODE* addToEnd(NODE *head, int val)
```

In `addToEnd`, `head` is a pointer to the first node in a linked list. This function should add a new node with `val` as the data to the end of the linked list. It should also return the new head of the linked list. This function must be **recursive**.

```
NODE* addToEnd(NODE* head, int val) {
     NODE* newnode = malloc(sizeof(NODE));
     newnode->data = val;
     newnode->next = NULL;
     if (head == NULL) return newnode;
     else {
          head->next = addToEnd(head->next, val);
          return head;
     }
}
```