Exercise Linked List

For the following questions consider the following struct definition:

```
Struct node{
    int data;
    Int node *next;
}
```

1. Write a function that takes a pointer to the head of a linked list and changes the list by adding an integer n (passed in as a parameter) to each node of the list.

```
void addN(struct node* list, int n){
    if (list=NULL) printf("NO Data");
    else{
        while(list !=NULL){
            list -> data+=n;
            list = list->next;
        }
}
```

2. Write a function that deletes the first node in a linked list and returns a pointer to the new head of the list. If there are no items in the original list, NULL should be returned.

```
struct node* deleteFirst(struct node* list) {
    if (list == NULL) printf("NO Data");
    else{
struct node* temp = malloc(sizeof(struct node));
    temp= list;
    list = list->next;
    free(temp); }
}
```

3. Write a function that makes a copy of an input list and returns a pointer to it. Note: This function should call malloc once for each node in the original list.

```
struct node* copy(struct node* list) {
    struct node* current = list;
    struct node* new = NULL;
    struct node* tail = NULL;
    while(current != NULL) {
        if (new == NULL) {
        new = malloc (sizeof(struct node));
        new->data = current->data;
        new->next = NULL;
        }
        else {
        tail->next = malloc (sizeof(struct node));
        tail = tail->next;
        tail->data = current->data;
        tail->next = NULL;
        }
       current = current->next;
    }
  return new;
}
```

4. p contains the elements 66, 9, 14, 52, 87, 14 and 17, in that order. Consider running the following line of code:

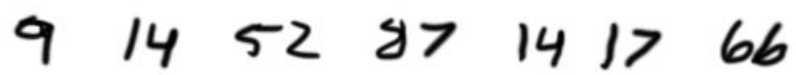
p = question4(p);

where question4 is the function defined below. Show the contents of p ***after*** the function call.

```
struct node* question4(struct node *list) {
        struct node* a = list;
        struct node* b = list;
        struct node* c;
        if (a == NULL) return NULL;
        while ( a->next != NULL)
                a = a ->next;
        a->next = b;
        c = b->next;
```

```
        b->next = NULL;
        return c;
}
```

Answer:



5. Write a function that takes in a pointer to the front of a linked list and returns 1 if all the nodes in the linked list are in sorted order (from smallest to largest, with repeats allowed), and 0 otherwise. The prototype is given below:

```
int isSorted(struct node* list) {
    if(list != NULL){
     int min = list->data;
     int bool = 1;
     while(list->next != NULL){
          list = list->next;
          if( list->data < min) bool = 0;
     }
     return bool;
    else return -99999;
    }
```

6. Write a function that takes in a pointer to the head of a linked list, a value to insert into the list, *val*, and a location in the list in which to insert it, *place*, which is guaranteed to be greater than 1, and does the insertion. If *place* number of items aren't in the list, just insert the item in the back of the list. You are guaranteed that the linked list into which the inserted item is being added is not empty. The prototype is given below:

```
void insertToPlace(struct node* list, int val, int place) {
    if(list == Null || place <= 0) return;
    struct node* temp = malloc(sizeof(struct node));
    temp->data = val;
    int count = 2;
    while(list->next != Null && count < place - 1){
        list = list->next;
        count++;
    }
    temp->next = list->next;
    list->next = temp;
}
```

7. Write a function that operates on a linked list of integers. Your function should insert a new node containing the value 2 after every node that contains the value 4. Make use of the list node struct and function header below.

```
void list_42(struct node* list) {
    struct node* current = list;
    struct node* temp;
    while(current != NULL){
        if(current->data == 4){
            temp = malloc(sizeof(struct node));
            temp->data = 2;
            temp->next = current->next;
            current->next = temp;
        }
        current = current->next;
    }
}
```

8. Write a function which returns a pointer to a linked list which is the result of moving the first

node in the list pointed to by alpha to the end of the list. (For example, if alpha points to a list that contains 3, 7, 2, 1, and 8, then when the function is called, a pointer to a list that contains 7, 2, 1, 8, and 3 should be returned.) If alpha has zero or one nodes, just return a pointer to the original list without making any changes. Utilize the function prototype provided below:

```
struct node* moveFrontToBack(struct node* alpha) {
  if (alpha == NULL || alpha->next == NULL)
    return alpha;

  struct *main = alpha->next
  Struct *reverse = alpha;
  reverse->next = NULL;
while (main != Null) {
  node *temp = main;
  main = main->next;
  temp->next = reverse;
  reverse = temp;
}

return reverse;

}
```

9. (Try this later after learning stacks. No need to submit it)You want to implement stack using linked list. Push function will insert an item to the head in the linked list and pop() function will remove and return the head item from the linked list. Implement the push and pop function using linked list and test it!

10. Implement Queue using linked list. [All the necessary steps are available in the slide]. We will try it at the lab].You don’t need to submit it with the exercise submission.