05 Linked Lists Problems

Insert element at a position in the list

```
LinkedList insert(Person p, int pos, LinkedList l) {
    if (pos == 0) {
        Node* D = (Node *) malloc(sizeof(Node));
        D->data = p;
```

```
        D->next = l;
        return D;
    } else {
        l->next = insert(p, pos-1, l->next);
        return l;
    }
}
```

Concatenate 2 lists

```
LinkedList concat(LinkedList l1, LinkedList l2) {
    if (l1 == NULL) {
        return l2;
    } else {
        l1->next = concat(l1->next, l2);
        return l1;
    }
}
```

Reverse a list

```
LinkedList reverse(LinkedList l) {
    int s = size(l);
    LinkedList l2 = NULL;
    for (int i = 0; i < s; i++) {
        l2 = insert(*element_at(s-i-1, l), i, l2);
    }
    return l2;
```

```
}
```

Full code

```
#include "stdio.h"
#include "stdlib.h"
#define MAX_NAME_LEN 100

typedef struct Person {
    char name[MAX_NAME_LEN];
    int age;
} Person;

typedef struct Node {
    Person data;
    struct Node* next;
} Node;

typedef Node* LinkedList;

void print_list(LinkedList l) {
    printf("-----------------------------------\n");
    while (l != NULL) {
        printf("%s\t\t%d\n",l->data.name, l->data.age);
        l = l->next;
    }
    printf("-----------------------------------\n");
}

int size(LinkedList l) {
```

```
    int s = 0;
    while (l != NULL) {
        l = l->next;
        s ++;
    }
    // Single line recursive solution
    // return l == NULL? 0 ; 1 + size(l->next);
    return s;
}

Person* element_at(int pos, LinkedList l) {
    int s = 0;
    while (l != NULL) {
        if (s == pos) return &(l->data);
        l = l->next;
        s ++;
    }
    return NULL;
}

LinkedList append(Person p, LinkedList l) {
    if (l == NULL) {
        // Node D = {{"Raj", 18}, NULL};
        Node* D = (Node *) malloc(sizeof(Node));
        D->data = p;
        D->next = NULL;
        return D;
    } else {
        l->next = append(p, l->next);
    }
    return l;
}
```

```
LinkedList insert(Person p, int pos, LinkedList l) {
    if (pos == 0) {
        Node* D = (Node *) malloc(sizeof(Node));
        D->data = p;
        D->next = l;
        return D;
    } else {
        l->next = insert(p, pos-1, l->next);
        return l;
    }
}

LinkedList concat(LinkedList l1, LinkedList l2) {
    if (l1 == NULL) {
        return l2;
    } else {
        l1->next = concat(l1->next, l2);
        return l1;
    }
}


LinkedList reverse(LinkedList l) {
    int s = size(l);
    LinkedList l2 = NULL;
    for (int i = 0; i < s; i++) {
        l2 = insert(*element_at(s-i-1, l), i, l2);
    }
    return l2;
}
```

```c
int main() {
    Node third = {
        {"Alice", 22},
        NULL
    };
    Node second = {
        {"Bob", 26},
        &third
    };
    Node first = {
        {"Charlie", 20},
        &second
    };
    Person D = {"Raj", 18};
    Node l2 = { D, NULL};

    LinkedList l = &first;
    printf("Size of the list is %d\n", size(l));
    print_list(l);
    l = concat( &l2,l);
    print_list(l);
    print_list(reverse(l));
    // print_list(insert(D,2,l));

    // printf("Element at 1st position: %s\n", element_at(1,l)->name);
    // printf("Element at 2nd position: %s\n", element_at(2,l)->name);
    // append(D, l);
    // printf("List after appending\n");
    // print_list(l);
    return 0;
}
```

HW: Reverse a LinkedList in place

```
void reverse_inplace(LinkedList l) {
    // If l is a->b->c->d
    // after executing reverse_inplace(l)
    // l should become d->c->b->a
    // the function also should not use another linkedlist or array
}
```

HW: Shuffle 2 lists

```
LinkedList shuffle(LinkedList l1, LinkedList l2) {
    // If l1 is a->b->c->d and l2 is 1->2->3->4
    // shuffle(l1,l2) should return the list
    // a->1->b->2->c->3->d->4
}
```

Can you make the above function also `in place`? That is it should not use any additional array or linked list other than l1 and l2.

HW: Free memory in a LinkedList

```
void free(LinkedList l) {
    // free all memory used by a linked list l
}
```

HW: Sort a LinkedList

```
LinkedList sort(LinkedList l) {
    // sort the linked list l and return it.
}
```

HW: Write the social network program using LinkedList

Use a Linked list instead of array in the social network program to save memory. You can used a linked list instead of the members array in Social Net. Can we replace the friends array (in Person) also with a LinkedList?

```
typedef struct Person {
    char name[100];
    int age;
    RelStatus relstatus;
    int count_friends;
    Person* friends[5];
} Person;

typedef struct SocialNet {
    struct Person members[100];
    int size;
} SocialNet;
```

Implement the `check_mutual_friendship` function from the last days homework with the social network made using linked lists.

Solution: Shuffle in place

```
LinkedList shuffle_inplace(LinkedList l1, LinkedList l2) {
    Node* head = l1;
    Node* temp1;
    Node* temp2;
    while (l1 != NULL && l2 != NULL) {
        temp1 = l1->next;
        temp2 = l2->next;
        l1->next = l2;
        if (temp1 == NULL) {
            break;
        }
        l2->next = temp1;
        l1 = temp1;
        l2 = temp2;
    }
    return head;
}
```