06 More Linked List Problems

Reverse a LinkedList in place

```
LinkedList node_append(Node* n, LinkedList l) {
    if (l == NULL) {
        return n;
    } else {
        l->next = node_append(n, l->next);
        return l;
    }
}

// Recursive Solution
LinkedList reverse_in_place(LinkedList l) {
    if (l == NULL) {
        return l;
    } else {
```

```
        LinkedList rev = reverse_in_place(l->next);
        l->next = NULL;
        rev = node_append(l, rev);
        return rev;
    }
}
```

Free memory in a LinkedList

```
void free_linked_list(LinkedList l) {
    if (l == NULL) {
        return;
    } else {
        LinkedList tail = l->next;
        free(l);
        free_linked_list(tail);
    }
}
```

Sort a LinkedList

```
// assuming a < b
// swaping is done by copy the data field in Node
LinkedList swap(LinkedList l, int a, int b) {
LinkedList head = l;
   Person temp;
   Node* a_ptr;
   Node* b_ptr;
```

```
    // Find above
    while(a >= 1) {
     l = l->next;
     a--;
     b--;
    }
    a_ptr = l;
    while(b >= 1) {
     l = l->next;
     b--;
    }
    b_ptr = l;
    temp = a_ptr->data;
    a_ptr->data = b_ptr->data;
    b_ptr->data = temp;
    return head;
}

LinkedList sort(LinkedList l) {
    // sort the linked list l and return it.
    // use swap to implement sorting
}
```

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?

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

```c
#include <stdbool.h>

typedef enum RelStatus {
    NotMentioned,
    Single,
    Engaged,
    Married
} RelStatus;

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

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

typedef Node* LinkedList;

typedef struct SocialNet {
    LinkedList members;
    int size;
} SocialNet;

void print_person(struct Person p) {
    char status_string[][20] = {
        "Not Mentioned",
        "Single",
```

```
        "Engaged",
        "Married"
    };
    printf("%s\t\t%d\t%s\t\t\t", p.name, p.age, status_string[p.relstatus]);
    for (int i = 0; i < p.count_friends; i++) {
        printf("%s, ",p.friends[i]->name);
    }
    printf("\n");
}

void print_network(SocialNet social_net) {
    LinkedList l = social_net.members;
    printf(
        "-------------------------------------------------------------------\n"
        "Name\t\tAge \tRel Status\t\tFriends\n"
        "-------------------------------------------------------------------\n");
    while (l!=NULL) {
        print_person(l->data);
        l = l->next;
    }
    printf(
        "-------------------------------------------------------------------\n");

}

Person* find_person(char* name, SocialNet *sn) {
    // TODO
    LinkedList l = sn->members;
    while(l!= NULL) {
        if (strcmp(l->data.name, name) == 0) {
            return &(l->data);
        }
        l = l->next;
```

```
    }
    return NULL;
}

bool check_mutual_friendship(char* name1, char* name2, SocialNet* sn) {
    Person* p = find_person(name1, sn);
    Person* q = find_person(name2, sn);
    bool q_in_fl_of_p = false;
    bool p_in_fl_of_q = false;
    for(int i =0; i< p->count_friends; i++) {
        if (q == p->friends[i]) {
            q_in_fl_of_p = true;
        }
    }
    for(int i =0; i< q->count_friends; i++) {
        if (p == q->friends[i]) {
            p_in_fl_of_q = true;
        }
    }

    if (p_in_fl_of_q && q_in_fl_of_p) {
        return true;
    } else {
        return false;
    }
}

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;
}

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;
}

int main() {
    SocialNet social_net = { .members=NULL, .size = 0};

    Person A = { "Alice", 24, Single, 2};
    Person B = { "Bob", 20, Engaged, 0};
    Person C = { "Charlie", 26, Married, 1};

    social_net.members = append( A,social_net.members);
    social_net.members = append( B,social_net.members);
    social_net.members = append( C,social_net.members);

    //social_net.members[0].friends[0] =  &(social_net.members[1]);
    // social_net.members[0].friends[1] =  &(social_net.members[2]);
    social_net.members->data.friends[0] = element_at(1, social_net.members);
    social_net.members->data.friends[1] = element_at(2, social_net.members);
```

```
    // social_net.members[2].friends[0] =  &(social_net.members[1]);
    element_at(2, social_net.members)->friends[0]
        = element_at(0, social_net.members);

    print_network(social_net);

    print_person(*find_person("Alice", &social_net));
    print_person(*find_person("Bob", &social_net));
    print_person(*find_person("Charlie", &social_net));

    printf("Is Alice and Charlie mutual friends: %d\n",
           check_mutual_friendship("Alice", "Charlie", &social_net));
    printf("Is Alice and Bob mutual friends: %d\n",
           check_mutual_friendship("Alice", "Bob", &social_net));

    return 0;
}
```

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

Shuffle in place

```
// 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
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;
}
```