Quiz 2

Quiz 2

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

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

typedef struct Node Node;

typedef Node* LinkedList;

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

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

typedef struct SocialNet {
    LinkedList members;
} SocialNet;

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

void print_person(Person* p) {
    char status_string[][15] = {
        "Not Mentioned","Single",
        "Married", "Engaged"
    };
    printf("%s\t\t%d\t%s\t\t\t", p->name, p->age, status_string[p->relstatus]);
    LinkedList f = p->friends;
    while (f != NULL) {
        printf("%s, ", f->data->name);
        f = f->next;
    }
    printf("\n");
}

void print_network(LinkedList m) {
    printf(
        "------------------------------------------------------\n"
        "Name\t\tAge\tStatus\t\t\tFriends\n"
        "------------------------------------------------------\n");
    while (m != NULL) {
        print_person(m->data);
        m = m->next;
    }
    printf("------------------------------------------------------\n");
}

Person* find_person(char* name, LinkedList l) {
    // Either find the person with a particular name
    // if not found return NULL
    while(l!= NULL) {
        if (strcmp(l->data->name, name) == 0) {
            return l->data;
        }
        l = l->next;
    }
    return NULL;
}


int size(LinkedList l) {
    return l == NULL? 0: 1+size(l->next);
}

char* person_with_most_friends(LinkedList l) {
    // Q A1: Return the name of the person with most friends
    // (3 marks)
    int d = 0;
    Node* n = NULL;
    while(l != NULL) {
        int e = size(l->data->friends);
        if (e > d) {
            d = e;
            n = l;
        }
        l = l->next;
    }
    return n==NULL? "" : n->data->name;
}

int popularity(char* name, LinkedList l) {
    // Q B1: Return the number of people who has the person
    // named `name` amoung their friends. (3 marks)
    int count = 0;
    while ( l!= NULL) {
        if (find_person(name, l->data->friends) != NULL) {
            count++;
        }
    }
    return count;
}

LinkedList delete_by_name(char* name, LinkedList l) {
    // Q A2: Delete the person named `name` from l (3 marks)
    if (l == NULL) {
        return NULL;
    } else if (strcmp(name, l->data->name) == 0) {
        Node* tail = l->next;
        free(l);
        return tail;
    } else {
        l->next = delete_by_name(name, l->next);
        return l;z
    }
}

LinkedList filterby_age(LinkedList l, int lower, int upper) {
    // Q B2: Return the link list of people in l with age
    // between lower and upper (3 marks)
    LinkedList l2 = NULL;
    while(l != NULL) {
        if (l->data->age >= lower && l->data->age <= upper) {
            l2 = append(l->data, l2);
        }
        l = l->next;
    }
    return l2;
}

bool friends_triangle(LinkedList members) {
    // Q A3: Check if there is a triangle of friends
    // ie there exists X, Y, Z such that
    // Y is a friend of X, Z is a friend of Y, X is a friend of Z
    // ALso print all such triplets (4 marks)
    LinkedList f = members;
    printf(
        "-----------------------------\n"
        "Friend Triangles\n"
        "-----------------------------\n");
    bool found = false;
    while(f != NULL) {
        LinkedList s = f->data->friends;
        while (s != NULL) {
            LinkedList t = s->data->friends;
            while (t != NULL) {
                LinkedList l = t->data->friends;
                while (l != NULL) {
                    if (strcmp(l->data->name, f->data->name)==0) {
                        printf("%s->%s->%s->%s\n", f->data->name, s->data->name,
                        found = true;
                    }
                    l = l->next;
                }
                t = t->next;
            }
            s = s->next;
        }
        f = f->next;
    }
    printf("-------------------------\n");
    return found;
}

bool transitive_friendship(LinkedList members) {
    // Q B3: check if the friendship relation is transitive
    // ie for any X,Y, Z, if Y is a friend of X and
    // Z is a friend of Y then Z is a friend of X
    // Also print all the links that violates transitivity
    // (4 marks)
    LinkedList f = members;
    printf(
        "-----------------------------\n"
        "Links that are not Transitive\n"
        "-----------------------------\n");
    bool found = false;
    while(f != NULL) {
        LinkedList s = f->data->friends;
        while (s != NULL) {
            LinkedList t = s->data->friends;
            while (t != NULL) {
                if (find_person(t->data->name, f->data->friends) == NULL) {
                    printf("%s->%s->%s, but there is no %s->%s\n", f->data->name
                    found = true;
                }
                t = t->next;
            }
            s = s->next;
        }
        f = f->next;
    }
    printf("-------------------------\n");
    return !found;
}

int main()
```

```
{
    SocialNet s = { NULL };

    Person A = {"Alice", 23, Single, NULL};
    Person B ={"Bob", 26, Engaged, NULL};
    Person C = {"Charlie", 21, NotMentioned, NULL};
    Person D ={"Don", 28, Married, NULL};

    s.members = append(&A, s.members);
    s.members = append(&B, s.members);
    s.members = append(&C, s.members);
    s.members = append(&D, s.members);

    A.friends = append(&B, A.friends);
    A.friends = append(&C, A.friends);
    B.friends = append(&D, B.friends);
    C.friends = append(&D, C.friends);
    D.friends = append(&A, D.friends);

    printf("List of people between ages 24 to 28:\n");
    print_network(filterby_age(s.members, 24, 28));

    printf("The person with most friends is %s.\n",person_with_most_friends(s.me

    // For above social network, `friends_triangle(s.members)`
    // returns `true` and prints
    // -----------------------------
    // Friend Triangles
    // -----------------------------
    // Alice->Bob->Don->Alice
    // Alice->Charlie->Don->Alice
    // Bob->Don->Alice->Bob
    // Charlie->Don->Alice->Charlie
    // Don->Alice->Bob->Don
    // Don->Alice->Charlie->Don
    // -------------------------
    friends_triangle(s.members);

    // For the above social network, `transitive_friendship(s.members)`
    // returns false and prints
    // -----------------------------
    // Links that are not Transitive
    // -----------------------------
    // Alice->Bob->Don, but there is no Alice->Don
    // Alice->Charlie->Don, but there is no Alice->Don
    // Bob->Don->Alice, but there is no Bob->Alice
    // Charlie->Don->Alice, but there is no Charlie->Alice
    // Don->Alice->Bob, but there is no Don->Bob
    // Don->Alice->Charlie, but there is no Don->Charlie
    // -------------------------
    transitive_friendship(s.members);

    return 0;
}
```