

COMPUTER PROGRAMMING : SECTION

Instructor: Girish Varma • Course Code: CS0.101 • IIT Hyderabad

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <stdbool.h>
5
6  typedef enum RelStatus {
7      NotMentioned, Single, Engaged, Married
8  } RelStatus;
9
10 typedef struct Node Node;
11
12 typedef Node* LinkedList;
13
14 typedef struct Person {
15     char name[100]; int age;
16     RelStatus relstatus;
17     LinkedList friends;
18 } Person;
19
20 struct Node {
21     struct Person* data; struct Node* next;
22 };
23
24 typedef struct SocialNet {
25     LinkedList members;
26 } SocialNet;
27
28 LinkedList append(Person* p, LinkedList l) {
29     if (l == NULL) {
30         Node* D = (Node *) malloc(sizeof(Node));
31         D->data = p;
32         D->next = NULL;
33         return D;
34     } else {
35         l->next = append(p, l->next);
36     }
37     return l;
38 }
39
40 void print_person(Person* p) {
41     char status_string[15] = {
42         "Not Mentioned", "Single", "Married", "Engaged"
43     };
44     printf("%s\t\t%d\t\t%s\t\t\t",
45         p->name, p->age, status_string[p->relstatus]);
46     LinkedList f = p->friends;
47     while (f != NULL) {
48         printf("%s, ", f->data->name);
49         f = f->next;
50     }
51     printf("\n");
52 }
53
54 void print_network(LinkedList m) {
55     printf(
56         "-----\n"
57         "Name\t\tAge\t\tStatus\t\tFriends\n"
58         "-----\n");
59     while (m != NULL) {
60         print_person(m->data);
61         m = m->next;
62     }
63     printf(
64         "-----\n");
65 }
66
67 Person* find_person(char* name, LinkedList l) {
68     // Either find the person with a particular name
69     // if not found return NULL
70     while(l != NULL) {
71         if (strcmp(l->data->name, name) == 0) {
72             return l->data;
73         }
74     }
75 }
76
77 }
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
```

```

73         }
74         l = l->next;
75     }
76     return NULL;
77 }
78
79 int popularity(char* name, LinkedList l) {
80     // Q1: Return the number of people who has the person
81     // named 'name' among their friends. (3 marks)
82 }
83
84 LinkedList filterby_age(LinkedList l, int lower, int upper) {
85     // Q2: Return the link list of people in l with age
86     // between lower and upper (3 marks)
87 }
88
89 bool transitive_friendship(LinkedList members) {
90     // Q3: check if the friendship relation is transitive
91     // ie for any X,Y, Z, if Y is a friend of X and
92     // Z is a friend of Y then Z is a friend of X
93     // Also print all the links that violates transitivity
94     // (4 marks)
95 }
96
97 int main()
98 {
99     SocialNet s = { NULL };
100     Person A = {"Alice", 23, Single, NULL};
101     Person B = {"Bob", 26, Engaged, NULL};
102     Person C = {"Charlie", 21, NotMentioned, NULL};
103     Person D = {"Don", 28, Married, NULL};
104
105     s.members = append(&A, s.members);
106     s.members = append(&B, s.members);
107     s.members = append(&C, s.members);
108     s.members = append(&D, s.members);
109
110     A.friends = append(&B, A.friends);
111     A.friends = append(&C, A.friends);
112     B.friends = append(&D, B.friends);
113     C.friends = append(&D, C.friends);
114     D.friends = append(&A, D.friends);
115
116     // prints
117     // -----
118     // Name           Age      Status           Friends
119     // -----
120     // Bob            26       Married        Don,
121     // Don            28       Engaged        Alice,
122     // -----
123     print_network(filterby_age(s.members, 24, 28));
124
125     // For the above social network,
126     // `transitive_friendship(s.members)`
127     // returns false and prints
128     // -----
129     // Links that are not Transitive
130     // -----
131     // Alice->Bob->Don, but there is no Alice->Don
132     // Alice->Charlie->Don, but there is no Alice->Don
133     // Bob->Don->Alice, but there is no Bob->Alice
134     // Charlie->Don->Alice, but there is no Charlie->Alice
135     // Don->Alice->Bob, but there is no Don->Bob
136     // Don->Alice->Charlie, but there is no Don->Charlie
137     // -----
138     transitive_friendship(s.members);
139
140     return 0;
141 }

```