

Linked Lists

Programming in C (Spring 2009/2010) Assignment 5

Deadline: **Thursday, June 3, 2010**

Introduction

The purpose of this assignment is to practice the use of dynamically allocated data structures in programming. You must implement several functions as requested in the assignment, but **feel free** to define additional functions as you see fit. You can assume correct function inputs.

The required function signatures must **exactly match** those in the questions. It is recommended that you test the functions you write in `main`, but we won't check `main`—the functions will be checked directly after we link our program with yours.

In this assignment we have provided several structure definitions. You must define these structures in your program as-is.

In addition, we have published an executable file. The purpose of the executable is to demonstrate how the functions should work. It is recommended that you implement the same functionality in your `main` function in order to be able to compare the output that your program produces.

Structures

In this assignment, we deal with fans who buy tickets to events. We keep lists of fans and lists of events, where each fan knows to which event they hold tickets, and each event organizer also knows which fans have bought the relevant tickets.

The structures `fan` and `event` are informational:

```
typedef struct fan {
    char *name;
    char *character;
    long id;
    struct event_list *events;
} fan;

typedef struct event {
    char *title;
    char *code;
    struct fan_list *fans;
} event;
```

The structures `fan_list` and `event_list` keep the data structure:

```
typedef struct fan_list {
    fan *info;
    struct fan_list *next;
} fan_list;

typedef struct event_list {
    event *info;
    struct event_list *next;
} event_list;
```

Note that the nested data structures are kept via pointers as well—in order to be able to keep data pertaining to the same fan or event in several lists, without duplication.

Basic Operations (20%)

Implement functions with the following prototypes:

```
fan_list* add_fan(fan_list *fans,
                 char *name, char *character,
                 long id);
event_list* add_event(event_list *events,
                     char *title, char *code);
```

Each function should dynamically allocate the relevant data structure, fill it with the supplied data, add it to the supplied list (which may be empty), and return the new list. The order of elements in the list should be the same as produced by our executable (new elements are added at the front).

You must copy strings (you cannot assume that they are immutable), but there is no need to check for fan ID / event code repetitions.

Tickets (40% + 10%)

Fans should be able to buy event tickets, and to attend events. Implement the functions

```
void buy_ticket(fan_list *fans,
               event_list *events,
               long id, char *code);
void use_ticket(event_list *events,
               long id, char *code);
```

that receive a list of fans, a fan ID and an event code (where each is guaranteed to exist in the respective list), and either add or remove the fan to/from the event. You can assume that no attempts to use a non-existing ticket are made (but someone may buy a ticket twice—this is OK, and is dealt with in the next section).

In addition, implement the query function

```
int has_ticket(fan_list *fans,
              long id, char *code);
```

that checks whether a fan has a ticket to some event.

Consistence (30%)

Implement the function

```
int is_consistent(fan_list *fans,
                  event_list *events);
```

that checks whether two lists of fans and of events are consistent:

- no two fans have the same ID;
- no two events have the same (case-sensitive) code;
- no fan has bought two tickets to some event.

Final Remarks

Remember:

1. If you want to be sure that your functions work as intended, the output of your program must match exactly the output of the executable file that is published together with this assignment.

2. **The program must compile without errors or warnings in Visual Studio 2005, when using the project settings in clang102.vcproj.**

3. **The program must not cause a runtime error at any point of its execution.**

4. The source code should be correctly indented and well-commented.

5. **Function prototypes must exactly match the prototypes in this assignment. If you don't understand what it means, it is very likely that your grade for the assignment will be 0.**

6. Any block of memory that cannot be of use anymore (i.e., cannot be accessed by subsequent function calls), must be freed.

All questions regarding the assignment should be sent to Michael Orlov (orlovm@cs.bgu.ac.il). Remember that you must consult the FAQ regularly for updates. Questions that are redundant (answered in this text or in the FAQ) will not be answered.

Good luck!