Assignment 2 – Memory allocation and linked lists

Introduction

The purpose of this assignment is to practice the use of linked lists and 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. We provided you with a skeleton file, containing all the functions you need to implement. This file includes a main function containing the loop performed by the program. **DO NOT CHANGE THE MAIN FUNCTION! DOING SO WILL MOST LIKELY RESULT IN A ZERO GRADE!**

Examine the skeleton file a2.c carefully before beginning to write your own code. You will submit this file only. Do not change the file name, or function definitions! This will result in a 0 grade.

Structures

This assignment deals with registering students for courses. There are two structures with correspond to the information regarding students and courses:

```
typedef struct student {
char *name;
int  id;
struct clist *courses;
} student;

typedef struct course {
char *title;
int  number;
struct slist *students;
} course;
```

Each student has a name and id (that doesn't begin with 0), and has a list of courses he/she is registered to. Each course has a number (that doesn't begin with 0) and a title, as well as a list of students which are registered for it.

```
typedef struct slist {
student      *info;
struct slist *next;
} slist;

typedef struct clist {
course       *info;
struct clist *next;
} clist;
```

In the student/course lists, each node has a pointer to a student/course struct, and a pointer to the next node. IMPORTANT – note that these lists hold pointers. The purpose of this is to have **only one copy** of a student's or course's information. For example, a student may be registered to a few courses, so he will appear on all their student lists. However – these lists will hold only a pointer to his information, of which there is a single copy!

Basic Operations

You will implement these basic operations:

```
slist* add_student(slist *students, char *name, int id);
clist* add_course(clist *courses, char *title, int number);
```

Both these functions receive a list of students or courses, and add a new student/course to this list. Memory for the new student/course must be allocated dynamically, then filled with the relevant data, and added to the supplied list, which may be empty. The new list is returned by both the functions. You must copy strings, but there is no need to check for student ID / course number repetitions.

Registration

Students should be able to register to courses. Implement the following functions:

```
void reg_student(slist *students, clist *courses, int id, int number);
void unreg_student(slist *students, int id, int number);
```

The first function receives the list of students, the list of courses, a student ID, and a course number (where each is guaranteed to exist in the respective list), and registers the student to the course. This means that the course is added to the student's course list (just a pointer – not another copy!), and the student is added to the course's student list (again, just a pointer – not another copy!).

The second function will unregister a student from the course. Notice that this function does not receive as an argument the course list. This is not a mistake – you still must remove the student from the course's student list.

You can assume that no attempt to unregister an unregistered student is made and that no attempt to register a student to the same course twice will be made.

Printing out a report

You must implement the following functions:

```
void print_students(slist *students);
void print_courses(clist *courses);
```

The first function prints a list of all the students – for each student it prints ID, name, and a list of courses the student is registered for. The order in which the students are printed is by their ID from low to high. The order in which the courses that each student is registered for are printed is by their number, from low to high.

The second function prints a list of all the courses – for each course it prints number, title, and a list of all students that are registered to that course. The order in which the courses are printed is by their number, from low to high. The order in which the student lists are printed is by their ID from low to high.

Releasing all allocated memory

The release of memory will be done using the following function:

```
void free_all(slist *sl, clist *cl);
```

This function receives the student and course lists and releases all allocated memory. Make sure you release all the memory you allocate! Your program will be checked for memory leaks! A good way to check this is to print every time you allocate memory, and print before you free that memory.

The main function

The main function is provided to you in the skeleton file and is fully implemented. Do not change it!

```
int main() {     slist*
students = 0;     clist*
courses = 0;     char
c;     char  buf[100];
int   id, num;

        ....
        ....
        ....
}
```

The two lists defined in the beginning of the main function are the main lists of students and courses. These are the lists given to all the functions (including the free_all function). The rest of the variables are used in order to read the user's input.

Some important issues:

1. The assignment should be done by singles only.
2. Make sure you print exactly what is needed.
3. Make sure that you don't print anything other than what you are required to print (student and course lists as described).
4. You are not allowed to change the function prototypes you are given. Doing this will results in a zero grade.
5. Compilation warnings are not allowed besides for scanf, single line comments and infinite loops.
6. Make sure to add necessary comments.
7. For checking memory leaks you may use *valgrind*, figure out how to install it and use it 9. Files you have consider:
    a. a2.c – This is the skeleton. In this file you need to complete all the function as described above.
    b. a2.exe - This is the executable, you can run it on WINDOWS OS only so create your test files in WIN and compare them on UNIX.
    c. This pdf file.

Some likely questions:

1. Can I add a parameter to `unreg_student()`? -> NO.
2. There must be an error in the prototypes. -> THERE IS NOT.
3. What should I do in case `malloc()` fails? -> EXIST THE PROGRAM.
4. Is there a limit on the length of student names and course titles? -> NO.
    a. But there is a limit of 100 in the input -> That's right, but I can (and will) change it before I check your assignment.

Only questions that demonstrate that you read the entire assignment description and the skeleton file will be answered. **Before sending questions, make sure that they don't already appear in the FAQ.**

GOOD LUCK!