COP 3223 Assignment #9

Program A: List Reverser (listreverse.c):

One of the graders has gotten lazy, and instead of grading assignments, he has decided to just assign grades randomly (don't worry, this isn't a true story). He has a list of student names sorted by the grade they were given on the last assignment. In order to make his laziness less noticeable, he has decided to reverse the list to assign grades this time. Please help him.

The first input to the console will be a positive integer, *N*, the number of students on the list. Following this will be *N* lines each containing a student's full name and ID. Each student's name will consist of, space separated, a last and first name (each less than 50 characters), each with no spaces within. IDs will be alphanumeric less than 20 characters with no spaces.

Once the list of names is input, output the list in reverse, in the same format as it was input.

You must have and use the following struct in your program:

```
struct Student {
      char first[50];
      char last[50];
      char id[20];
};
```

Sample Run (user input in bold and italics):

3
Dent Arthur 12345ABC
Prefect Ford 54321CBA
McMillan Tricia AB9876

```
McMillan Tricia AB9876
Prefect Ford 54321CBA
Dent Arthur 12345ABC
```

Program B: Bingo (bingo.c):

You and some of your friends have started playing bingo. Each of your friends has one bingo card. For simplicity, we'll assume that each bingo card just consists of a single row, so a player wins by having every number on their card (the entire row) called (not necessarily in order). But you don't believe your friends are playing correctly. So you have decided to implement a program that will automate the game for everyone playing.

The first line of input from standard input will be an integer, *N*, the number of players this round. The following *N* lines will each contain a first name (a string less than 50 characters with no spaces) and 5 distinct positive integers, the numbers on that player's card.

Following this, a series of numbers will be input. These numbers are the numbers being "called". When a number is called, each player marks it on their card. As soon as a player has marked all numbers on their card, they should call "bingo!" (your program should output the name(s) of the people who have bingo at that point) and the program should end (since the round now ends). It is guaranteed that there will always be enough queries entered to reach this end state in some way. Note: the program should finish when at least one player has won, regardless of any other attempted input.

You must have and use the following struct in your program:

```
struct Player {
      char name[50];
      int board[5];
      int board_size;
};
```

You must have and use the following function in your program:

```
// Removes the given number from the player's board if it exists in
// that board, or else does nothing. Adjusts the board's contents and
// size to reflect this removal.
void remove_number(Player* player, int number);
```

Sample Run (User input in bold and italics):

3
John 1 2 3 4 5
Tyler 1 2 3 4 6
Evan 1 2 4 5 6
6 4 2 3 5 1
John wins!
Tyler wins!
Evan wins!

Sample Run (User input in bold and italics):

3
John 1 2 3 4 5
Tyler 1 2 3 4 6
Evan 1 2 4 5 6
4 3 2 5 1
John wins!

Sample Run (User input in bold and italics):

3
John 1 2 3 4 5
Tyler 1 2 3 4 6
Evan 1 2 4 5 6
1 2 3 4 5 6
John wins!

Program C: Password Recovery (passrecover.c):

Oh no! You've forgotten a very important password! You really need to recover it or you're going to be in big trouble. You do remember a list of all possible passwords you may have used, but you can't just guess as to what it is, you need to narrow it down more accurately.

Luckily, you shared this password with all of your friends. Unluckily, your friends also have bad memories (and really, why would you want them remembering your password, anyway?). Each of your friends was able to give you a hint as to what your password was. But some of them may be wrong. Hints will be strings (less than 50 characters each) consisting of alphanumeric characters and *s. Any * means that the person did not remember which letter was in that position. Any alphanumeric character means the part of the password that that friend remembers had that character in that location (case-sensitive). In order for a hint to be considered matching a password possibility, they must both be the same length and each non-* character must match in the same location in the hint and password. No password possibilities given will contain any *s.

You will be given to standard input a positive integer, *N*, followed by *N* unique possible passwords (each alphanumeric without spaces less than 50 characters). Following this will be a positive integer, *M*, followed by *M* password hints. You should output the password possibility with the highest number of hints matching it. If there are multiple possible passwords with the same number of hint matches, output the one that comes lexicographically first (remember strcmp).

Hint: to check a hint against a possibility, first check that their lengths are equal, then loop character by character to compare each respective position (if the hint is non-*), making sure all of those match.

You must have and use the following struct in your program:

```
struct Option {
      char password[50];
      int matches;
};
```

You must have and use the following function in your program:

```
// Checks to see if the given password matches the given pattern.
// Returns false (0) if it does not, or true (1) if it does.
int is_match(char password[], char pattern[])
```

Sample Run (User input in bold and italics):

3
password
secret11
qwertyui
4
*******1***
s****
****e******
*******q***
secret11

Sample Run (User input in bold and italics):

4
abcdefg
abcdeff
abcdefa
abcdefb
3
a**d*
*******a***
*******b***
abcdefa

Deliverables:

Please submit three separate .c files for your solutions to these problems via WebCourses by the designated due date:

Program A: **listreverse.c**
Program B: **bingo.c**
Program C: **passrecover.c**