ITSE 1430- Fall 2016
Program Set #6

See C# Grading /Program Guide Sheet for directions and grading/submission information.

Note: Create a console application program for the following problem. Do not create a windows application (form).

1. One popular Olympic event is the High Jump. A bar is placed at some initial height, and competitors jump over it. Each competitor has three chances to clear the bar at the given height. If the jumper can't clear it in any of their three chances, they are eliminated. After all competitors finish at a certain height, the bar is raised, and competitors try to jump over the higher bar. Misses from previous heights don't carry over, thus all competitors will be able to take three attempts at all heights. Competitors are also allowed to skip an entire height, and thus will not have an entry at that height. Competitors that do this are still alive in the competition, but only receive credit for jumps that are successfully made.

When all (or all except for one) of the competitors are eliminated, the competitor who cleared the highest distance successfully is the winner. If two or more jumpers tie for first place, the tie-breakers are:

- The fewest misses at the height at which the tie occurred; and
- The fewest misses throughout the competition.

If there is still a tie, then the result is a tie. (In the Olympics, this is resolved by having a "jump-off" between the two competitors). Write a C# console application to process High Jump scores for an event and determine the winner (or winners). Input will be from a text file, where the first input will be a number, `n`, $n \geq 1$, denoting the number of events to score for. Each of the `n` events will consist of a number, `m`, containing the number of entrants ($1 \leq m \leq 10$). Then there will be a series of heights in the following form:

```
Height
k
Contestant number1 Attempts
Contestant number2 Attempts
…
Contestant numberk Attempts
```

"`Height`" is a decimal value with exactly one digit before the decimal point and two digits after the decimal point which corresponds to the height of the bar. `k` is the number of contestants who attempted the jump at this height. Then there will be `k` lines, each denoting a contestant by number (contestant numbers start at 1) and their attempts. The attempts will consist of three characters, each character will be either `'Y'`, signifying they made the jump on that attempt, `'N'`, signifying they missed the jump on that attempt. Attempts after a success are designated by `'S'`, signifying they skipped that attempt (since they made it successfully). The heights will be in increasing order, but there is no special ordering for the contestants within a height. The height entries will conclude with a height of `0.00`. There will be at most 10 distinct heights. For each input case, output one line stating:

`Event i: The winner is contestant n, with a height of h.`

where `i` is the event number (starting at 1) `n` is the contestant number of the winner, and `h` is their last successful jump. If there is a tie and no way to determine the winner, output the line:

`There is a tie between the following contestants: n1 n2 … nn`

There is a single space after the colon. Where the various n_i are the contestant numbers of the tied contestants, listed in increasing numerical order and separated by spaces. Let the user input the file name from the keyboard. Output should look similar to below.

Sample Input File:

```
3
3
1.00
3
1  N Y S
2  S S S
3  N N N
1.10
2
1  Y S S
2  N N N
0.00

2
1.00
2
1  N N Y
2  Y S S
0.00

2
1.00
2
1  N N Y
2  N N Y
0.00
```

Sample Run:

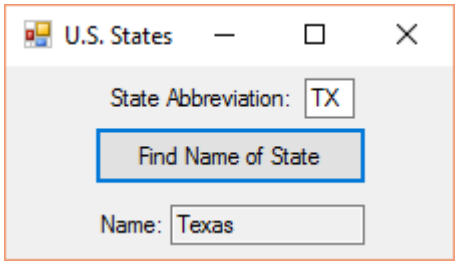
```
Enter the file name: jumps.txt

Event 1: The winner is contestant 1, with a height of 1.10
Event 2: The winner is contestant 2, with a height of 1.00
Event 3: There is a tie between the following contestants:
1 2
```

Name the program: `HighJumpXX.cs`, where XX are your initials.

Note: Create a windows application (form) program for the following problem.

2. Write a C# windows application to look up the name of the state whose abbreviation is given. The state abbreviation is entered into a textbox. The form title should be named `U.S. States`. The XML file will be provided by your professor. Output should look similar to below.



Name the application: `XMLStateAbbXX`, where XX are your initials.

Note: For Problem 3 create a C# Web application.

3. Write a C# Web application for the following: The TCC-SE computer club is selling T-shirts. Create a website that allows users to enter their first and last names, phone number, and e-mail address. Allow users to select sizes (S, M, L, XL, and XXL) and quantity. Add statements that process the order by calculating the total cost. All shirts except the XXL are $20; the XXL shirt is $25 dollars. Retrieve and display the name of the customer placing the order. Display the total cost of their selection including %7 sales tax. Output should be user friendly.

Name the application: `ASPWebShirtsXX`, where XX are your initials.

Note: Create a windows application (form) program for the following problem.

4 (**). In any sport, scheduling the officials to work games in season can be a huge task. There are many factors involved, some that can be controlled others that cannot. In this project your task will be to write a C# windows application that schedules the 17 National Football League (NFL) officiating crews to 16 regular season games based on certain criteria. The criteria:

- No crew can work the same team (home or away) more than two times in the regular season.
- There must be a minimum of separation of six-weeks (6 or greater) before the crew can see the same team (home or away) again.
    - Example: Week 1 then again Week 6 or later.
- If a crew sees the same team twice during the season:
    - One must be at home and the other away- OR
    - Both times away
- Each crew (except one) will have two off or bye weeks in a 17-week season. Byes can't happen in consecutive weeks. Thus, each crew is guaranteed 15 games.
- No crew can work the same two matchups (home or away) during the season.
    - Example: `DAL @ WAS` and `WAS @ DAL`- even if all the other criteria stated above is met

Remember there are 17 weeks- but only 16 games in the NFL season. Also, due to team bye weeks, not all 32 NFL teams play every single week. Keep this mind when building crew schedules.

Input

- Input will come from two text files named: `crews.txt` and `games.txt`. The user will enter these file names from an appropriate user control.

  `crews.txt`
  - There are 17 lines- each line containing the referee's last name (crew chief).

  `games.txt`
  - The first number consists of the number of weeks in the season.
  - For each week, one line containing:
    - The week number, and a list of 13 to 16 games separate by a semicolon. For each game the away team is listed first then the home team, each separated by a space.

Output

- Using the criteria stated above, the user should have the choice to:
  - View schedules by week-the crews that work/do not work each week (`OFF`)- for each of the 17 weeks

  OR

  - Two (2) possible regular season schedules (options) for each of the 17 crews. For this option, output a list of the unique (in order) 6 per line the NFL teams they will work in a season as well as which teams they will not see (in order) 6 per line.

Output either choice to a text box or to file in an easily readable format. Use appropriate controls (labels, buttons, etc.). in your application. The form title should be named `Crew Scheduler`. Make the application user friendly.

Name the application: `CrewSchedulingXX`, where `XX` are your initials.