CS 6100 Program 3: Social Choice (40 points)

Purpose

The reasons I'm asking you to do this lab are:
• I want you to play with social welfare functions so that you get familiar with the concept.
• I want you to see that the various voting mechanisms choose very different candidates (so that
  you do not mistakenly believe that all of them will select the same "most preferred" candidate).
• I want you to try to develop a social choice mechanism yourself.

The Problem

This program should run on CSILM. Arrow's impossibility theorem says that there is no social choice
mechanism that takes individual preference patterns and generates a fair societal preference pattern.
Arrow defined fairness according to axioms, and showed that all the axioms could not be
simultaneously satisfied. Voting methods are attempts to take individual preference patterns and create
a "fair" societal preference pattern. We should be able to identify situations where the voting
mechanism breaks down. Since there is no way for voting to be fair, the task of somebody who is
designing a voting mechanism is to minimize the unfairness.

Consider a society of seven voters (A-G) who are trying to reach a consensus on which of the
alternatives they want (say, Coke, Sunkist, Root Beer, Milk, OJ) for the title “CS Department Favorite
Drink”. 1 represents the first choice and 5 the last choice. Each agent ranks them as 1 (meaning the
best) and 5 (meaning the worst).

One method will utilize a confidence in the vote. Each individual has a varying confidence in his/her
vote. We will use confidence as follows: Since voter B has confidence 4 (conf 4) in its ranking, we
will treat voter B’s first place vote for coke as if 4 different voters all ranked coke first. (This same
concept exists when different parties have a different number of votes, like in the electoral college.)

To facilitate experimentation, let the following values be the initial values. Also, allow the user to
request random preferences and confidences be used.

<table>
<thead>
<tr>
<th>Choice/Agent</th>
<th>A (conf 5)</th>
<th>B (conf 4)</th>
<th>C (conf 3)</th>
<th>D (conf 3)</th>
<th>E (conf 4)</th>
<th>F (conf 2)</th>
<th>G (conf 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coke</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Sunkist</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Root Beer</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Milk</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>OJ</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
Your job is to help these agents (as a group) choose one of the alternatives from the set of candidates.

The Experiment

I want you to determine what choice is made when the following social choice functions are applied.

- **Majority Rule** (Plurality Protocol – each person votes for first choice only). Identify the name of the winner and how many first choice votes it got. Utilize the confidence of each voter in making this decision.

- **Condorcet method.** The Condorcet winner is the candidate who would beat each of the other candidates in a run-off election. First, for each pair of candidates determine which candidate is preferred considering all voter preferences. For example, if we look at the comparison of Coke with Sunkist, the winner would be Coke as five of the seven voters prefer Coke to Sunkist.

If there is a candidate who 'wins' EVERY comparison with all other candidates, then this candidate is the winner. This comparison is done by considering how many times is candidate A preferred to candidate B when those are the only two contenders. If there is no such candidate, then there is no Condorcet winner. **Note:** we define a winning candidate as that candidate having a number of preferential votes which is greater than or equal to the number of preferential votes of all other candidates when the candidates are compared, pairwise.

We will computer the **Condorcet Score** as the number of times a candidate wins over an opponent minus the number of times a candidate loses to an opponent. (Thus, a Condorcet winner has a Condorcet score of n-1.)

For each candidate, identify its Condorcet score and whether there is a Condorcet winner.

- **Plurality with runoff:** First, restrict the set to two candidates having the most first choice votes. Call them candidate A and B. Then, for all voters, change the scores for candidate A to be 1 if the voter prefers A to B and 2 otherwise. Who gets the most first place votes now? Indicate the final two candidates, and indicate the name of the winner.

  **Note:** if two (or more) candidates have the same number of first place votes, look to their second-place (or if necessary, their third-place, etc.) votes to decide how to rank them.

- **Sequential Runoff** (*Hare system*): The sequential run-off scheme eliminates candidates one by one. First, we eliminate the candidate with the fewest first-place votes. Then we make up a preference schedule in which only the remaining candidates are listed; we renumber the ranking in every column so that the 4 remaining candidates have now ranks labeled by 1, 2, 3 and 4.

  We then repeat the procedure: we eliminate again the candidate with the fewest 1st place rankings, and renumber for the remaining 3 contenders, and so on. Indicate who was eliminated first, second, etc. Indicate the winner.

  **Note:** if at some step, two (or more) candidates have the same number of first place votes, look to their second-place (or if necessary, their third-place, etc.) votes to decide how to rank them.
o **Borda Protocol:** In the Borda protocol, each agent reveals his or her preferences and the social welfare function assigns 1 point to the top choice of the individual, 2 to the next choice, and so on, where N is the number of candidates. *(In class, the points were assigned in the opposite order, which is better if someone doesn’t rate all candidates. This method is easier, given our data.)* The points for each candidate are totaled, the candidates are sorted by point total, and the resulting ordering is the societal preference pattern. Try it two ways:

1. Single step (tally all the scores, and pick the winner) **Indicate the borda points for each candidate and the name of the winner.**
   
2. Iterated Borda (tally all the scores, eliminate the agent with the worst score, and repeat until a winner is chosen). **Indicate the name of the candidate which is eliminated each time, and the borda points of other candidate who are still in the running at the time the candidate is eliminated. Indicate the name of the winner.**

o **Approval Voting:** Each voter designates a subset of the outcomes of which he/she approves. The winning option is the one that appears in the highest number of approval lists. For this input, we will assume that the approval list is anything in the top half of the ordering. So, if there are five choices, anything in the top two slots is deemed "approved". **Indicate the count of number of approval lists containing each candidate and the winner.**

o Try a voting scheme you make up (or one that you find in the literature). **Explain your method (in the readme file).** Indicate the winner (and any other important results) using this method.

### The Output:

Allow the user to see the results for individual voting methods and easily compare the results. I suggest you arrange the results something like the following (with winner highlighted and other information put in the corresponding boxes)

<table>
<thead>
<tr>
<th>Majority</th>
<th>Condorcet</th>
<th>Plurality</th>
<th>Sequential</th>
<th>Borda</th>
<th>Iterated Borda</th>
<th>Approval Voting</th>
<th>My Own</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coke</td>
<td>Coke</td>
<td>Coke</td>
<td>Coke</td>
<td>Coke</td>
<td>Coke</td>
<td>Coke</td>
<td>Coke</td>
</tr>
<tr>
<td>Sunkist</td>
<td>Sunkist</td>
<td>Sunkist</td>
<td>Sunkist</td>
<td>Sunkist</td>
<td>Sunkist</td>
<td>Sunkist</td>
<td>Sunkist</td>
</tr>
<tr>
<td>Root Beer</td>
<td>Root Beer</td>
<td>Root Beer</td>
<td>Root Beer</td>
<td>Root Beer</td>
<td>Root Beer</td>
<td>Root Beer</td>
<td>Root Beer</td>
</tr>
<tr>
<td>Milk</td>
<td>Milk</td>
<td>Milk</td>
<td>Milk</td>
<td>Milk</td>
<td>Milk</td>
<td>Milk</td>
<td>Milk</td>
</tr>
<tr>
<td>OJ</td>
<td>OJ</td>
<td>OJ</td>
<td>OJ</td>
<td>OJ</td>
<td>OJ</td>
<td>OJ</td>
<td>OJ</td>
</tr>
</tbody>
</table>

The actual winners are not as important as how you analyze your results. Submit your code and a README report file. Use the scientific method (observe a phenomena, generate a hypothesis, test your hypothesis, and present supporting data). You should submit a report that summarizes the social choice functions you tested and an analysis of the results (what was good about each scheme, what was bad, why were the schemes good/bad). Feel free to experiment with different preference data.

**Hint:**
This is a trickier than it first appears. I would recommend thinking a bit about the design before you start coding. While you must do your own work, I would recommend comparing results with others.

Use all your good coding skills: meaningful variable names, print outs of current data, easy subscripting. For example, I had something called ctRank[i][j] which was the number of times product i was ranked j. Since j goes from 1 to 5, instead of 0-4, I just created the array to allow an extra 0th column that wasn’t used. That was simpler for me to think about than to always have to subtract 1 from the rank. I had something called stillIn[i] which told me whether or not item i was still in the running. That way I could ignore an item without having to rebuild the tables as much.

So my function to remove the worst from consideration and rebuild the ranks looked like:

```c
void remove(int worst, int rank[PROD][VOTER], bool stillIn[PROD])
{
    stillIn[worst]=false; //"remove " candidateName[worst]
    for (int i=0; i < prodCt; i++)
    {
        if (stillIn[i])
            for (int j=0; j < voterCt; j++)
                if(rank[i][j] > rank[worst][j])
                    rank[i][j]--; //
    }
    return rank;
}
```