Project 2

Fair Division

Introduction

Dividing a pie among any number of people is a simple problem. If there are n people, then cut the pie into n equal slices and give a slice to each person. But what if, instead of a pie, we are dividing a painting among n people? We cannot cut the painting into n pieces and hand them out. Clearly, only one person could receive the painting. What about the others? If they should receive a cash settlement, then how much cash? In this project, we will program an elegant solution to the problem of dividing m indivisible assets among n people.

Dividing the Assets

The items to be divided will be called the *assets* and the people among whom they are divided will be called the *players*. We will assume that we have m assets and n players. Any positive values are possible. That is, we could have any of the possibilities $m < n$, $m = n$, or $m > n$.

The procedure begins with each player placing a private *bid* on each asset. Their bids represent the values that they place on those assets. Once all the players have placed their bids, the bids are opened and compared. For each asset, the player who placed the highest bid on that asset receives that asset.

If two or more highest bids are equal, then the asset is given to one of them at random.

An Example

For example, suppose that there are two assets, a house and a yacht, and that there are three players, Joe, Jim, and Jack. They place the following bids:

Player	House	Yacht
Joe	\$100,000	\$200,000
Jim	\$150,000	\$180,000
Jack	\$225,000	\$195,000

Then Jack receives the house and Joe receives the yacht.

The Cash Settlement

Of course, so far, this is not fair because Jim has received nothing. To make it fair, we follow this with a cash settlement to even things up. The amount of cash paid or received is based on each player's perceived *fair share*. A player's fair share is an equal portion of the total value that he places on the assets. For example, Joe placed a total value of $300,000 on the assets and there are 3 players, so, in Joe's view, his fair share is $100,000. Similary, Jim's fair share is $110,000 and Jack's fair share is $140,000.

The players who received assets add up the values of those assets, based on their bids. If the value of the assets is more than their fair share, then they pay the difference into the *kitty*. If it is less than their fair share, then they receive the difference from the kitty.

At this point, all the players have received exactly their perceived fair share. However, unless all the bids for each asset were equal, there is money left over in the kitty. This remaining amount is divided evenly among all the players.

The Example Continued

We see that Joe received the yacht, which he valued at $200,000, and his fair share is $100,000, so he must pay $100,000 into the kitty. Jack received the house, which he valued at $225,000, and his fair share is $140,000, so he must pay $85,000 into the kitty. On the other hand, Jim received no asset, and his fair share is $110,000, so he receives $110,000 from the kitty.

That leaves $75,000 in the kitty. Divided 3 ways, each player receives an additional $25,000. Thus, the final settlement is

- Joe receives the yacht and pays $75,000.
- Jim receives $135,000.
- Jack receives the house and pays $60,000.

Note that each player ended up with $25,000 more than his perceived fair share. So everyone is happy, very happy.

The `Player` Class

To facilitate the design of this program, we will create a `Player` class. This class is described in detail in the document `The Player Class`. A `Player` object contains the player's name, the number of assets (which is also the number of bids), and an array of bids.

The Program

The application program should define and use the following functions.

- `string* readAssets(int& numAssets)` – Reads the names of the assets from a file into a dynamically allocated array of strings. The function returns a pointer to the array and the parameter `numAssets` is assigned the size of the array.

- `Player* readPlayers(int& numPlayers)` – Reads the players as `Player` objects from a file into a dynamically allocated array of `Player`s. The function returns a pointer to the array and the parameter `numPlayers` is assigned the size of the array.

- `double* getFairShares(Player* player, int numPlayers)` – Returns a pointer to a dynamically allocated array containing the fair shares of the players.

- `int* assetSettlement(Player* player, int numPlayers, int numAssets)` – Returns a pointer to a dynamically allocated array of indexes (`int`s), one index for each asset. Each index refers to the player who was awarded the asset.

- `double* cashSettlement(Player* player, int numPlayers, int* recipient, int numAssets, double* fairShare)` – Returns a pointer to a dynamically allocated array of `double`s that represent the cash settlement for the players, one value for each player. A positive quantity indicates that the player *paid* that amount into the kitty. A negative quantity indicates that the player *received* that amount from the kitty. The parameter `recipient` is a pointer to the array of indexes of players who won the assets, the same pointer that was returned by `assetSettlement()`.

- `void reportSettlement(Player* player, int numPlayers, double* fairShare, string* asset, int numAssets, int* recipient, double* cash)` – Outputs a report showing the players' fair shares, the settlement of assets (who received what), and the cash settlement (who paid and who received how much).

The names of the assets should be stored in a dynamically allocated array of strings. The `Player` objects should be stored in a dynamically allocated array of `Player` objects. The fair shares, asset settlements, and cash settlements should also be stored in dynamically allocated arrays of the appropriate types.

Place the files `FairDivision.cpp`, `player.h`, and `player.cpp` in a folder named `Project_2` and place it in the drop box. Your work is due by Friday, Feb 15.