```
#include <iostream>

using namespace std;


/**
 * Prints the leaderboard in the appropriate (sorted by candy count) order
 *
 * Parameters:
 *   players: Array of player names
 *   candy: Array of candy counts (candy count at location i corresponds to
 *          the player name at location i in the players array)
 *   numplayers: Number of players and candy counts in the respective arrays
 */
void printLeaderboard(string players[], int candy[], int numplayers);


/**
 * Update the candy count of the given player to reflect the number
 * of pieces of candy they found. Do nothing (just return)
 * if the given player does not exist in the arrays already
 *
 * Parameters:
 *   players: Array of player names
 *   candy: Array of candy counts (candy count at location i corresponds to
 *          the player name at location i in the players array)
 *   playerName: Name of the player to update
 *   candyFound: Number of pieces of candy found by playerName
 *   numplayers: Number of players and candy counts in the respective arrays
 *
 */
void struckGold(string players[], int candy[], string playerName, int candyFound, int
numplayers);

/**
 * Add 2 to all player candy counts that are still in the game.
 *
 * Parameters:
 *   players: Array of player names
 *   candy: Array of candy counts (candy count at location i corresponds to
 *          the player name at location i in the players array)
 *   numplayers: Number of players and candy counts in the respective arrays
 *
 */
void rainingCandy(string players[], int candy[], int numplayers);

/**
```

```
 * Cut in half all candy counts of players who are in even positions on the
 * leaderboard (0-indexed). Use integer division, since the big bully
 * is greedy and doesn't like "half" pieces of candy.
 *
 * Parameters:
 *   players: Array of player names
 *   candy: Array of candy counts (candy count at location i corresponds to
 *          the player name at location i in the players array)
 *   numplayers: Number of players and candy counts in the respective arrays
 *
 */
void theft(string players[], int candy[], int numplayers);

/**
 * Gives one piece of candy from the top player to each other player in
 * reverse order.  Starting from the last player, Big Bully takes one
 * piece of candy from the top player and gives it to the last place player,
 * then repeats for the second  to last player, continuing until the top
 * player is out of candy or we've given one piece of candy to every other
 * player (i.e. we've reached the 2nd place player).
 *
 * Parameters:
 *   players: Array of player names
 *   candy: Array of candy counts (candy count at location i corresponds to
 *          the player name at location i in the players array)
 *   numplayers: Number of players and candy counts in the respective arrays
 *
 */
void toughLuck(string players[], int candy[], int numplayers);


/**
 * Returns the index of the specified player or -1 if it doesn't exist
 *
 * Parameters:
 *   players: Array of player names
 *   playerName: Name of the player to search for
 *   numplayers: Number of players in the players arrays
 *
 * Returns: Index of given playerName or -1
 */
int findPlayer(string players[], string playerName, int numplayers);

/**
 * Prints the main menu and returns the integer selection the user
 * wants to perform.  If the user provides an invalid selection,
 * simply return -1 which the calling function (e.g. main() can
 * use to detect the error.
 *
 * Returns: integer selection of the user or -1 if the selection
 *  was invalid
 */
int printPromptAndGetInput();


/**
 * Sort the player and candy array from highest candy count to lowest.
 *
 * We recommend one of the simplest sorting algorithms:  Selection sort
 * Look at the class notes or online resources for how to implement this.
 *
 * Parameters
 *   players: Array of player names
 *   candy: Array of candy counts (candy count at location i corresponds to
 *          the player name at location i in the players array)
 *   numplayers: Total number of players
 */
void sortLeaderboard(string players[], int candy[], int numplayers);

/**
 * Delete the given player (and its corresponding candy count from the arrays
 * moving all later players/candy counts up one spot
 *
 * Parameters:
 *   players: Array of player names
 *   candy: Array of candy counts (candy counts at location i corresponds to
 *          the player name at location i in the players array)
 *   playerName: Name of the player to delete
 *   numplayers: Number of players and candy counts in the respective arrays
 *
 * Returns: false if the specified player does not exist, or true if
 *          the player was successfully deleted.
 */
bool deletePlayer(string players[], int candy[], string playerName, int numplayers);

/**
 * Delete all players from the game who do not have positive candy counts.
 *
 * Note that when a player is deleted, the index of other players may
 * shift, so special care must be taken.
 *
 * Parameters
 *   players: Array of player names
 *   candy: Array of candy counts (candy count at location i corresponds to
 *         the player name at location i in the players array)
 *   numplayers: Total number of players before deleting.
 *
 * Returns: the number of players remaining after deleting.
 */
int dropLosers(string players[], int candy[], int numplayers);

/**
 * !!!!!!!!!!!!!!!! EXTRA CREDIT !!!!!!!!!!!!!!!!!!!!!!!!!
 *
 * By implementing this correctly and integrating it into your
 * project so that player names are displayed with spaces, even
 * though the user enters them without spaces you can earn
 * some additional extra credit.
 *
 * Breaks up a string at capital letters and inserts spaces
 * So if the input is "KatnissEverdeen" return "Katniss Everdeen".
 * You may assume the first letter is a caps (and even if it
 * not you will still get a reasonable result; so "johnBrown"
 * should still return "john Brown").
 *
 * Parameters:
 *  in: string with no spaces
 *
 * Returns: a version of the string with spaces
 */
string breakStringAtCaps(string in);

/*****************************************************************
 *  Write your implementations for each function prototyped
 *  above in the space below
 ****************************************************************/

void printLeaderboard(string players[], int candy[], int numplayers)
{

}
void struckGold(string players[], int candy[], string playerName, int candyFound, int
numplayers)
{

}

void rainingCandy(string players[], int candy[], int numplayers)
{

}
```

```
void theft(string players[], int candy[], int numplayers)
{

}

void toughLuck(string players[], int candy[], int numplayers)
{

}


int findPlayer(string players[], string playerName, int numplayers)
{

}

int printPromptAndGetInput()
{

}

void sortLeaderboard(string players[], int candy[], int numplayers)
{

}

string breakStringAtCaps(string in)
{

  /* !!!!!!!! Implement this for extra credit if you desire !!!!! */
  /* Otherwise leave this as is                                    */
  return in;

}

bool deletePlayer(string players[], int candy[], string playerName, int numplayers)
{

  int loc = findPlayer(players, playerName, numplayers);
  if(loc == -1)
  {
    return false;
  }

  for(int i = loc+1; i < numplayers; i++)
  {
    players[i-1] = players[i];
    candy[i-1] = candy[i];
  }

  return true;

}

int dropLosers(string players[], int candy[], int numplayers)
{

  int numLosers = 0;
  for (int i=0; i < numplayers; i++)
  {
    if (candy[i] <= 0)
    {
      numLosers++;
    }
  }

  while(numLosers > 0)
  {
    for (int i=0; i < numplayers; i++)
    {
      if (candy[i] <= 0)
      {
        deletePlayer(players, candy, players[i], numplayers);
        numLosers--;
        numplayers--;
        break;
      }
    }
  }

  return numplayers;

}



/************************************************************
 *  Main must be completed by you.
 ************************************************************/
int main()
{
  const int SIZE = 20;
  string players[SIZE];
  int candy[SIZE];
  int numplayers = 0;

  cout << "How many players will play? (Enter a number between 1 and 20 inclusive)" <<
endl;
  cin >> numplayers;
  if (numplayers < 1 || numplayers > 20){
          return 0;
  }

  cout << "Enter each player, followed by a nonzero number of starting candy" << endl;
  for (int i=0; i < numplayers; i++){
    cin >> players[i];
    cin >> candy[i];
  }
  cout << endl;

  sortLeaderboard(players, candy, numplayers);

  cout << "INITIAL LEADERBOARD:" << endl;
  printLeaderboard(players, candy, numplayers);
  cout << endl;

  /* Continue your code here */



  /* End your code here */

  cout << endl;
  cout << "FINAL LEADERBOARD:" << endl;
  printLeaderboard(players, candy, numplayers);


  return 0;
}
```