PROBLEM 1:  (Baseball is alive again: (25 pts))

Consider the following definition, *Baseball* iterator class, and constructor. The *Baseball* iterator class contains information on all the baseball teams in a league. In particular, it iterates over all the players in the league and for each player gives information about the player (name and number of homeruns hit) and the name of the team the player is on.

```cpp
struct Player {
    string name; // name of baseball player
    int numHRs; // number of homeruns hit this season

    void Print(); // print name of baseball player and number of HRs hit
};

class BaseBall {
public:
    BaseBall(); // constructor
    BaseBall(int teams, int playercount); // constructor

    // iterator
    void First(); // iterator set to first player
    void Next(); // iterator set to next player
    Player Current(); // returns name and number of HRs for current player
    string CurrentTeam(); // returns name of team for the current player
    bool IsDone(); // returns true if iterator doesn't have a current player, returns false otherwise

    void Print() const; // prints teams and players

private:
    void Read(); // reads in team names and player info
    tvector <string> myTeamNames; // names of teams
    tmatrix <Player> myTeamPlayers; // names and number of HRs for players, each row represents one team
    int myNumTeams; // number of teams
    int myNumPlayersPerTeam; // number of players per team
    int myCurrentRow; // current row in iterator
    int myCurrentCol; // current column in iterator
};
```
BaseBall::BaseBall(int teams, int playercount)  // constructor
    : myTeamNames(teams), myTeamPlayers(teams, playercount),
      myNumTeams(teams), myNumPlayersPerTeam(playercount),
      myCurrentRow(0), myCurrentCol(0)
{
    Read();  // code not shown, reads in team names and player info
}

Here is a sample code segment that uses the \textit{Baseball} class to list out all the teams and for each team to list the players on that team and the number of homeruns they have hit this season. Then it prints a blank line followed by the name and number of homeruns of the first player. Note for the purposes of testing, we are using a small data file with just 4 teams and 5 players.

\begin{verbatim}
BaseBall year98(4,5);
year98.Print();  // list out all teams and players
    cout << endl;
year98.First();
year98.Current().Print();  // list out first player and number of homeruns
\end{verbatim}

Here is the corresponding output. On each row is the team name followed by players and the number of homeruns for each player, then the first player is printed again.

\begin{verbatim}
Mets Lotze 21 Sharma 4 Hanna 8 Allen 33 Fath 12
Reds Krauss 43 Ptak 22 Oka 17 Wong 2 Bostrom 70
Cards Caton 18 McGuire 70 Byrd 3 Fan 28 Sun 11
Cubs Zeng 4 Liu 9 Meng 16 Sosa 66 Finley 3
Lotze 21
\end{verbatim}

The function \texttt{Read} reads in information about the baseball teams. The name of a baseball team is stored in the vector \texttt{myTeamNames}, and information about the corresponding players on that team are stored in the matrix \texttt{myTeamPlayers} in the same row number. The team names are stored in no particular order, and the players on a team are stored in no particular order. You may assume \texttt{Read} already exists, you \textbf{DO NOT} need to write it.

Complete the following functions and code segment below. Note that the iterator moves over all the players from all the teams and at any point must be able to return a current player (function \texttt{Current}) and the name of the team the player is on (function \texttt{CurrentTeam}). Do NOT add any additional private variables to the \textit{BaseBall} class.

- Complete the function \texttt{First} below (3 pts).

\begin{verbatim}
void BaseBall::First()
    // postcondition: iterator set to first player
{

}
\end{verbatim}
• Complete the function `IsDone` below (3 pts).

```cpp
bool BaseBall::IsDone()
// postcondition: returns true if iterator has no current player,
// otherwise returns false
{
}
```

• Complete the function `Next` below (5 pts).

```cpp
void BaseBall::Next()
// postcondition: iterator moved to next player
{
}
```

• Complete the function `Current` below (3 pts).

```cpp
Player BaseBall::Current()
// postcondition: returns current player in iterator
{
}
```

• Complete the function `CurrentTeam` below (3 pts).

```cpp
string BaseBall::CurrentTeam()
// postcondition: returns current team in iterator
{
}
```
For this part, complete code that is part of a client program using the `BaseBall` class. Compute the maximum number of homeruns hit by a player and the list of names hitting that many homeruns.

For example, if the code segment was applied to the data on page 5, the output would be:

```
70 homeruns
McGuire Bostrom
```

Complete the code segment below (6 pts).

```java
Baseball database(M,N);  // read in info for M teams, there are
                         // N players per team
```
Consider the following definition to keep track of the type and number of each type of animals in a zoo.

```cpp
struct Critter
{
    string type;    // type of animal
    int number;     // number of this type of animal

    Critter(string, int); // constructor
};

Critter::Critter(string ty, int num) :
    type(ty), number(num)
{
}
```

**PART A (5 pts):**
Write the function `Read` whose header is shown below. `Read` reads data from an input stream consisting of pairs of a word and number (type of animal and number of this type) until there is no more data. Information for each animal is read in and stored in the vector in the order it is read in.

For example, if the stream `input` contains the information:

- elephant 3
- zebra 5
- hippo 2
- elephant 3
- elephant 2
- coyote 3
- hippo 10

Then after the call `Read(NYzoo, numTypes, input), numTypes` will be set to 7, and `NYzoo` will look like:

```
NYzoo

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

- elephant 3
- zebra 5
- hippo 2
- elephant 2
- coyote 3
- hippo 10
```
Complete function \textit{Read} below the following header. You can assume there is enough capacity in \textit{zoo} and that you do not need to resize it.

\begin{verbatim}
void Read( tvector <Critter *> & zoo, int & size, istream & input)
// precondition: zoo is empty, but has plenty of capacity for the
// data in the stream input, input is open and bound to a file
// postcondition: zoo contains data from the stream input and
// size indicates the number of items stored in zoo
{

PART B \textit{(2 pts)}: If N is the number of pairs of words and numbers read in, what is the worst case running time (big-Oh) of the \textit{Read} function?
\end{verbatim}
PART C (7 pts):
Write the function *FindDuplicate* whose header is shown below. *FindDuplicate* is given a vector of pointers to *Critters* and returns true if there is a duplicate type in the vector, otherwise it returns false. In addition, if there is a duplicate, *FindDuplicate* sets *pos1* and *pos2* to the index positions of the duplicate. If there are multiple duplicates, *pos1* and *pos2* can be set to any pair of duplicates. Let *NYzoo* represent the vector shown in Part A. The call *FindDuplicate(NYzoo, numTypes, first, dup)* returns true and sets *first* equal to 0 and *dup* equal to 3 (or *first* and *dup* could be set to any of the duplicate pairs).

Complete function *FindDuplicate* below the following header.

```cpp
bool FindDuplicate(tvector <Critter *> zoo, int numTypes, int & pos1, int & pos2)
// precondition: zoo has numTypes items
// postcondition: returns true if there is a duplicate and sets pos1 and
// pos2 to the index positions of the duplicate,
// otherwise returns false
```

PART D (2 pts): If N is the number of pairs of words and numbers read in, what is the worst case running time (big-Oh) of the *FindDuplicate* function?
PART E (9 pts):
Write the function `RemoveDuplicates` whose header is shown below. `RemoveDuplicates` is given a vector of pointers to `Critters` and combines duplicate types by combining their counts and removing the duplicate entry.

Let `NYzoo` represent the vector shown in Part A. After the call `RemoveDuplicates(NYzoo, numTypes)`, `numTypes` is set to 4 and `NYzoo` looks like:

In writing `RemoveDuplicates`, you may use the function `FindDuplicate` from part C. Assume the function `FindDuplicate` works correctly regardless of what you wrote for part C.

Complete function `RemoveDuplicates` below the following header.

```cpp
void RemoveDuplicates(tvector <Critter *> & zoo, int & numTypes)
// postcondition: all duplicates have combined their counts and the duplicates
// removed from zoo and numTypes has been adjusted to the current
// number of items in zoo
{
    // implementation...
}
```
PROBLEM 3:  (It’s a Mystery To Me: (6 pts))

Consider the following definitions and function Mystery.

```c
struct MysNode
{
    int number;
    MysNode * next;
};

MysNode * Mystery(MysNode * list)
{
    if (list != NULL && list->next != NULL)
    {
        return (Mystery(list->next->next));
    }
    else
        return list;
}
```

MysNode * temp;

1. Assume `temp` points to the following list.

```
      5 −→ 8
```

Show the list `temp` points to after the call: `temp = Mystery(temp)`

2. Assume `temp` points to the following list.

```
      2 −→ 9 −→ 7 −→ 4
```

Show the list `temp` points to after the call: `temp->next = Mystery(temp->next)`

3. Write a recurrence relation for the function `Mystery`. DO NOT SOLVE the recurrence relation. Assume the linked list has N nodes.
Consider the following definition for a node in a doubly linked list.

```c
struct Node{
    string fish; // type of fish
    Node * next; // forward pointer
    Node * prev; // backward pointer
};
```

PART A (6 pts):
Write the function `Find` whose header is shown below. `Find` returns a pointer to the first occurrence of a specified type of fish. `Find` returns NULL if the type of fish does not appear in the given list.

In the example below, the result of the call `Find(fishlist,"betta")` is illustrated by showing a pointer to a node in `fishlist` with the value "betta".

![Diagram of fish list]

Complete function `Find` below the following header.

```c
Node * Find(Node * list, const string & type)
// postcondition: returns a pointer to the first occuring node with "type" value
// returns NULL if the type does not appear in the list.
{
```
PART B (8 pts):
Write the function *RemoveDuplicates* whose header is shown below. *RemoveDuplicates* is given a linked list of fish and a type of fish, and removes all but one occurrence of this type of fish.

Let fishlist represent the list shown in Part A. The figure below shows one possible result of the call *RemoveDuplicates(fishlist,"discus")*. All but one node containing "discus" have been removed from the list.

```
void RemoveDuplicates(Node * & list, const string & type)
// postcondition: remove all but one occurrence of "type" from the list
{
```

Consider the following struct declarations used to represent a linked list of clubs, each club containing a linked list of members.

```c
struct Member
{
    string name;
    Member * next;

    Member(string newname, Member * newnext = 0) // constructor
    {
        name = newname;
        next = newnext;
    }
};

struct Club
{
    string clubName;
    Member * list;
    Club * next;

    Club(string newname, Member * newlist = 0, Club * newnext = 0) // constructor
    {
        clubName = newname;
        list = newlist;
        next = newnext;
    }
};
```

```
    tennis  →  cps  →  debate  →  juggling
     |          |          |          |
   dhatt     sharp     yoho     slepack
     |          |          |          |
   roh       tye       magee     sharp
     |          |          |          |
bloom     howell
```

PROBLEM 5:  (The club of clubs (16 points))

12
Part A: 4 points
Write the function IsMember whose header is given below. IsMember returns true if a person with the given name is in the list, e.g., IsMember(duke->list,"roh") returns true because "roh" is in the tennis club (the club pointed to be duke->list) and IsMember(duke->next->list,"magee") returns false because "magee" is NOT in the cps club (the club pointed to by duke->next->list).

```c++
bool IsMember(Member * list, string name)
// postcondition: returns true if name is in list
// otherwise returns false
{
}
```

Part B: 4 points
Complete the function FindClubList whose header is given below. FindClubList returns a pointer to the first student member in a specified club. For example on the previous page, FindClubList(duke, "juggling") returns a pointer to the node with "slepack", the first member in the "juggling" club list.

```c++
Member * FindClubList(Club * clubList, string clubname)
// postcondition: returns a pointer to the list of members in the
// club clubname (returns 0 if no club named clubname)
{
}
```
Part C: 8 points

Complete the function \texttt{CloneClubs} whose header is given below. \texttt{CloneClubs} creates a new club. The new club is formed by combining the members of two other clubs. For example using the earlier diagram, \texttt{CloneClubs(duke, "cps","juggling","music")} creates a new club called \textit{music} with the members from the \textit{cps} and \textit{juggling} clubs. The members of the music club are: sharp, tye, howell, and slepack (note that sharp is a member of both clubs, but should just appear once in the new club). The cps and juggling clubs should still remain, do not destroy them.

In writing \texttt{CloneClubs} you may call the function \texttt{IsMember} and the function \texttt{FindMember} you wrote in Parts A and B; assume these functions works as specified regardless of what you wrote in Parts A and B.

\begin{verbatim}
void CloneClubs(Club * & clubList, string club1, string club2, string newName)
// precondition: no club named newName is in clubList
// postcondition: a club named newName is added to clubList, the
// members of the new club are all members from
// clubs club1 and club2, with no member listed twice
\end{verbatim}