Not Another Committee!

If there's one thing that universities love to do, it's to create committees. Need to design a new degree? Have a committee figure it out! Dealing with a problem? A committee will sort it out! However, given the large number of committees, the university has realized it needs software to manage the composition of these committees.

More specifically, a committee is composed of several professors, each of which can belong to one or more fields (e.g., a professor could do research in both Computer Science and Mathematics). Furthermore, professors can either be tenured or not. In this problem, we will determine if a **bad** committee has been formed. A committee is *bad* if both of these conditions are true:

- There are only one or two unique fields represented in the committee.
- There are not enough tenured professors in the committee. If the committee has N professors, we need at least $(N//2) + 1$ of them to be tenured. The operator (//) means floor division (i.e., dividing and rounding down to the nearest integer.)

Here are some sample professors:

Name	Tenured (T/F)	Areas
bob	T	systems theory
sally	T	theory
ally	T	ai
rob	F	systems
ben	F	systems
jenny	F	systems ai

Here are some sample committees:

Name	Members	Bad? (T/F)
c1	bob sally ally	F
c2	bob rob jenny stephanie	T
c7	sally ally	F

We have supplied a class for representing professors (see below for the implementation of the `Professor` class). Your job is to complete the `Committee` class.

You are required to implement:

- the `Committee` constructor

- `addMember` - takes a `Professor` as an argument and add the professor, if they are not already on the committee. Returns `true` is the professor was successfully added to the committee and `false` otherwise. You may assume the professors have unique names.
- `numMembers` - returns the number of committee members
- `numTenured` - returns the number of tenured professor on the committee
- `getUniqueFields` - returns a set of the fields represented by the professors on the committee
- `isBadCommittee` - returns `true` if the committee is bad as defined above and `false` otherwise.

You are are welcome to write additional helper methods and to include additional attributes.

Here is the code for the header file for the `Professor` class.

```
#ifndef _Professor_
#define _Professor_
#include <unordered_set>
#include <string>
#include <vector>


/**
Class for representing a professor.

 Public attributes:
  name: the professor's name represented as a string.
  tenured: a boolean that will be true, if the professor has tenure and false
    otherwise.
  fields: a set of the names of the fields the professor works in
 **/
class Professor {
public:
std::string name;
bool tenured;
std::unordered_set<std::string> fields;

/**
Constructor:

 @param name the Professor's name represented as a string
 @param tenures a boolean that will be true, if the process has tenure and
    false otherwise.
 @param fields an array of strings with the names of the fields the professor
    works in
**/
Professor(std::string _name, bool _tenured, std::vector<std::string> _fields);

};

#endif
```

And here is the code for the `Professor` class.

```
#include "Professor.h"
#include <unordered_set>
#include <string>
#include <vector>

/**
    Class for representing a professor.

    Public attributes:
        name: the professor's name represented as a string.
        tenured: a boolean that will be true, if the professor has tenure and
            false otherwise.
        fields: a set of the names of the fields the professor works in
 **/
Professor::Professor(std::string _name, bool _tenured, std::vector<std::string> _fields):
name(_name), tenured(_tenured) {
    for (std::string field : _fields) {
        fields.insert(field);
    }
}
```

Here is the code for the header file for the `Committee` class.

```
#ifndef _Committee_
#define _Committee_
#include <vector>
#include <string>
#include <unordered_set>
#include "Professor.h"

/**
Class for representing Committees

 Public attributes:
  name: the name of the committee
  members:a list of the Professors assigned to the committee

 Public methods:
  addMember: add a Professor as a member to the committee, if they are not
      already on the committee.
  isBadCommittee: determine whether the committee as currently constituted is "bad"
 **/
class Committee {
public:
std::string name;

/**
Constructor: the constructor for Committee

 @param name the name of the committee represented as a string
**/
Committee(std::string _name);

/**
add a Professor to the committee, if they are not already on the committee.

 @param possible a possible member of the committee represented as a Professor

 @return true, if the specified Professor is not already on the committee,
      false otherwise.
**/
bool addMember(Professor* possible);

/**
determine whether the committee as currently constituted is "bad".

 @return true, if the committee meets the definition of bads (fewer than two
      unique fields represented
      and an insufficient number of tenured professors), false otherwise.
**/
bool isBadCommittee();


private:
std::vector<Professor*> members;

/**
 *  numMembers: computes the number of tenured committee members
 **/
int numMembers();

/**
 *  numTenured: computes the number of tenured committee members
 **/
int numTenured();
std::unordered_set<std::string> getUniqueFields();
};

#endif
```

And here is the skeleton code for the `Committee` class.

```
#include "Committee.h"
#include <vector>
#include <string>
#include <unordered_set>
#include "Professor.h"

/**
Class for representing Committees

 Public attributes:
  name: the name of the committee
  members:a list of the Professors assigned to the committee

 Public methods:
  addMember: add a Professor as a member to the committee, if they are not
    already on the committee.
  isBadCommittee: determine whether the committee as currently constituted is "bad"
 **/
Committee::Committee(std::string _name) {
    // COMPLETE THIS METHOD
}

/**
add a Professor to the committee, if they are not already on the committee.

 @param possible a possible member of the committee represented as a Professor

 @return true, if the specified Professor is not already on the committee,
    false otherwise.
**/
bool Committee::addMember(Professor* possible) {
    // COMPLETE THIS METHOD
    // return included to allow the skeleton code to compile
    return false;
}

/**
 *  numMembers: computes the number of committee members
 **/
int Committee::numMembers() {
    // COMPLETE THIS METHOD
    // return included to allow the skeleton code to compile
    return 0;
}


/**
 *  numTenured: computes the number of tenured committee members
 **/
int Committee::numTenured() {
    // COMPLETE THIS METHOD
    // return included to allow the skeleton code to compile
    return 0;
}

std::unordered_set<std::string> Committee::getUniqueFields() {
    // COMPLETE THIS METHOD
    // return included to allow the skeleton code to compile
    return std::unordered_set<std::string>();
}

/**
determine whether the committee as currently constituted is "bad".

 @return true, if the committee meets the definition of bads (fewer than two
      unique fields represented and an insufficient number of
      tenured professors), false otherwise.
**/
bool Committee::isBadCommittee() {
    // COMPLETE THIS METHOD
    // return included to allow the skeleton code to compile
    return false;
}
```

We have provided files named `Professor.h`, `Professor.cpp`, `Committee.h`, and `Committee.cpp` with the relevant code for your convenience. For the actual exam, you would be expected to copy the different pieces of code into the appropriate files.