Can We Make a Reservation?

Businesses often use software to manage certain aspects of their operations. For example, restaurants might use a reservation booking service like OpenTable or Resy. In this problem, we will supply a `Reservation` class and you will complete the methods for a `Restaurant` class for use in a booking service.

To request a reservation, a customer needs to provide: the name of the restaurant, the name of their party, and the number of guests in their party. (See below for the implementation of the `Reservation` class.)

A restaurant has a name and a number of tables. Each table will have four seats and can be used only once a night. The tables are numbered 0 through $N - 1$, where N is the number of tables.

A given reservation can take up more than one table, but there will be at most one party per table. For example, a reservation for a party of six will have exclusive use of two tables. Parties are seated at available tables in numeric order (i.e., starting from table 0 through N - 1). If the restaurant does not have enough tables to accommodate the reservation, then no tables are assigned.

For this example, assume that Nella has 5 tables and Medici has 8 tables and that the reservations are processed in order.

Restaurant Name	Party Name	Number of Guests	Tables Assigned
Nella	Hammond	5	0, 1
Medici	Bolton	5	0, 1
Medici	Lumbergh	3	2
Nella	Malcolm	3	2
Medici	Smykowski	8	3, 4
Nella	Grant	4	3
Medici	Joanna	9	5, 6, 7
Medici	Waddams	2	–

Your task is to implement the constructor and two additional methods— `makeReservation` and `checkReservation`— for the `Restaurant` class.

The method `makeReservation` takes a `Reservation` and, if possible, updates it to include the table(s) assigned to the party making the reservation. The method should return `true` when there were enough tables to fulfill the reservation and `false` otherwise.

The method `checkReservation` takes the name of a party and an array of table numbers. It should return `true` if the specified party was assigned the specified tables and `false` otherwise.

You are are welcome to write additional helper methods and to include additional attributes., The constructor, `makeReservation`, and `checkReservation` are

required.

Your implementation must take advantage of the attributes and methods provided by the Reservation class.

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

```
#ifndef _Reservation_
#define _Reservation_
#include <vector>
#include <string>

/**
 Class for representing requests for restaurant reservations.

 Public attributes:
    partyName: the name of the party making the reservation represented as a string
    partySize: the number of people in the party represented as an integer

 Public methods:
    assignTables: update the tables assigned to a reservation.

    confirmTables: confirm the tables assigned to a given reservation.
 */
class Reservation {
public:
std::string restaurantName;
std::string partyName;
int partySize;

/**
Constructor for Reservation

 @param partyName the name of the party making the reservation represented as a string
 @param partySize the number of people in the party represented as an integer
**/

Reservation(std::string& restaurantName, std::string& partyName, int partySize);


/**
assign specific tables to the reservation.

 @param tables an array of integer table numbers to be assigned to the reservation.
**/
void assignTables(std::vector<int> tables);
```

```
/**
confirm that the specified tables have been assigned to the reservation
(in the order given).

 @param tables an array of integer table numbers to be assigned to the reservation.

 @return True, if the tables assigned to the reservation match the specified tables.
    False, otherwise.
**/
bool confirmTables(std::vector<int> tables);


private:
std::vector<int> assignedTables;
};

#endif
```

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

```
#include "Reservation.h"
#include <vector>
#include <string>


/**
    Class for representing requests for restaurant reservations.

    Public attributes:
    partyName: the name of the party making the reservation represented as a string
    partySize: the number of people in the party represented as an integer

    Public methods:
    assignTables: update the tables assigned to a reservation.

    confirmTables: confirm the tables assigned to a given reservation.
*/
Reservation::Reservation(std::string& _restaurantName,
                         std::string& _partyName,
                         int _partySize): restaurantName(_restaurantName),
                                          partyName(_partyName),
                                          partySize(_partySize),
                                          assignedTables(std::vector<int>(0, 0)) {}

/**
assign specific tables to the reservation.
```

```
 @param tables an array of integer table numbers to be assigned to the reservation.
**/
void Reservation::assignTables(std::vector<int> tables) {
    for (int i = 0; i < tables.size(); ++i) {
        assignedTables.push_back(tables[i]);
    }
}

/**
confirm that the specified tables have been assigned to the reservation
(in the order given).

 @param tables an array of integer table numbers to be assigned to the reservation.

 @return True, if the tables assigned to the reservation match the specified tables.
    False, otherwise.
**/
bool Reservation::confirmTables(std::vector<int> tables) {
    if ((assignedTables.size() == 0) || assignedTables.size() != tables.size()) {
        return false;
    }

    for (int i = 0; i < tables.size(); ++i) {
        if (assignedTables[i] != tables[i]) {
            return false;
        }
    }
    return true;
}
```

Here is the header file for the `Restaurant` class.

```
#ifndef _Restaurant_
#define _Restaurant_
#include <vector>
#include <string>
#include "Reservation.h"

/**
Class for representing restaurants.

Public attributes:
name: the name of the restaurant represented as a string

Public methods:
makeReservation: assign tables if the reservation is feasible
based on the number of people in the party and the number of
```

```
tables available.

checkReservation: verify that the table assigned to a given party.
**/
class Restaurant {
public:
std::string name;
int numTablesAvailable;

/**
Constructor for Restaurant

 @param name the name of the restaurant represented as a string
 @param numTables the number of tables in the restaurant
     */
Restaurant(std::string name, int numTables);

/**
assign tables if the reservation is feasible based on the number of people
in the party and the number of tables available.

  @param res a reservation request represented as a Reservation

  @return True if the reservation tables are assigned, false otherwise.
**/
bool makeReservation(Reservation res);

/**
verify the tables assigned to a given party.

 @param partyName the name of the party associated with a reservation
 @param tables an array of table numbers

 @return True if the tables listed in the reservation for the specified
 party matches the specified tables. False, otherwise.
**/
bool checkReservation(std::string partyName, std::vector<int> tables);

private:
int nextTable;
std::vector<Reservation> reservations;
};

#endif
```

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

```
#include "Restaurant.h"
#include <vector>
#include <string>

/**
    Class for representing restaurants.

    Public attributes:
    name: the name of the restaurant represented as a string

    Public methods:
    makeReservation: assign tables if the reservation is feasible
    based on the number of people in the party and the number of
    tables available.

    checkReservation: verify that the table assigned to a given party.
**/
Restaurant::Restaurant(std::string _name, int _numTables) {
    // COMPLETE THIS METHOD
}

/**
assign tables if the reservation is feasible based on the number of people
in the party and the number of tables available.

  @param res a reservation request represented as a Reservation

  @return True if the reservation tables are assigned, false otherwise.
**/
bool Restaurant::makeReservation(Reservation res) {
    // COMPLETE THIS METHOD
    // Return included to allow the sketelon code to complile
    return false;
}

/**
verify the tables assigned to a given party.

 @param partyName the name of the party associated with a reservation
 @param tables an array of table numbers

 @return True if the tables listed in the reservation for the specified
 party matches the specified tables. False, otherwise.
**/
bool Restaurant::checkReservation(std::string partyName, std::vector<int> tables) {
    // COMPLETE THIS METHOD
```

```
    // Return included to allow the sketelon code to complile
    return false;
}
```

We have provided files named `Reservation.h`, `Reservation.cpp`, `Restaurant.h`, and `Restaurant.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.