4. Consider the problem of assigning passengers to seats on airline flights. Three types of information are needed —passenger information, seat information, and flight information. Three classes will be used to represent this information, respectively: `Passenger`, `Seat`, and `Flight`.

   You will write three member functions for the `Flight` class:

   (a) `EmptySeatCount` that returns the number of empty seats of a specified type,

   (b) `FindBlock` that returns information about the location of an empty block of seats, and

   (c) `AssignGroup` that attempts to assign a group of passengers to adjacent seats.

   Passenger information is abstracted by a class and includes a name and other information. A default passenger, used to indicate "no passenger" in a seat, has the empty string `""` as its name. The declaration for class `Passenger` is as follows.

```
class Passenger
{
  public:
    Passenger();    // default passenger with name ""

    apstring GetName() const;
    // postcondition: returns passenger's name

    // ... other public and private members not shown
};
```

   Seat information includes the passenger assigned to the seat and the type of the seat ("window", "aisle", "middle"). The `Seat` function `GetPassenger` returns the passenger assigned to the seat; if the seat is empty, `GetPassenger` returns a default passenger. The declaration for the class `Seat` is as follows.

```
class Seat
{
  public:
    Passenger GetPassenger() const;
    // postcondition: returns passenger in this seat

    apstring GetType() const;
    // postcondition: returns the type of this seat

    void SetPassenger(const Passenger & p);
    // postcondition: assigns p to this seat (i.e., GetPassenger() == p)

    // ... constructors and other public and private members not shown
};
```

Seat assignments are processed by the public member functions of the class `Flight`. The seating arrangement is represented internally by a matrix of seats in the class `Flight`. The declaration for the class `Flight` is as follows.

```
class Flight
{
  public:
    int EmptySeatCount(const apstring & seatType) const;
    // postcondition: returns the number of empty seats
    //                whose type is seatType;
    //                if seatType is "any", returns the
    //                total number of empty seats

    int FindBlock(int row, int seatsNeeded) const;
    // postcondition: returns column index of the first (lowest index)
    //                seat in a block of seatsNeeded adjacent
    //                empty seats in the specified row;
    //                if no such block exists, returns -1

    bool AssignGroup(const apvector<Passenger> & group);
    // postcondition: if possible, assigns the group.length() passengers
    //                from group to adjacent empty seats in a single row
    //                and returns true;
    //                otherwise, makes no changes and returns false

    // ... constructors and other public member functions not shown

  private:
    apmatrix<Seat> mySeats;

    // ... other private data members not shown
};
```

(a) You will write the `Flight` member function `EmptySeatCount`, which is described as follows. `EmptySeatCount` returns the number of empty seats of the specified type `seatType`. Recall that an empty seat holds a default passenger whose name is `""`. If `seatType` is `"any"`, then every empty seat should be counted in determining the number of empty seats. Otherwise, only seats whose type is the same as `seatType` are counted in determining the number of empty seats.

For example, consider the diagram of passengers assigned to seats as stored in `mySeats` for `Flight ap2002` as shown below.

| | [0] | [1] | [2] | [3] | [4] | [5] |
|---|---|---|---|---|---|---|
| [0] | window<br>"Kelly" | middle<br>"Robin" | aisle<br>"" | aisle<br>"Sandy" | middle<br>"" | window<br>"Fran" |
| [1] | window<br>"Chris" | middle<br>"Alex" | aisle<br>"" | aisle<br>"" | middle<br>"Pat" | window<br>"Sam" |

The following table shows several examples of calling `EmptySeatCount` for this flight.

| Function Call | Value Returned |
|---|---|
| `ap2002.EmptySeatCount("aisle")` | `3` |
| `ap2002.EmptySeatCount("window")` | `0` |
| `ap2002.EmptySeatCount("middle")` | `1` |
| `ap2002.EmptySeatCount("any")` | `4` |

Complete function `EmptySeatCount` below.

```
int Flight::EmptySeatCount(const apstring & seatType) const
// postcondition: returns the number of empty seats
//                whose type is seatType;
//                if seatType is "any", returns the
//                total number of empty seats
```

(b) You will write the `Flight` member function `FindBlock`, which is described as follows. `FindBlock` searches for a block of `seatsNeeded` adjacent empty seats in the specified row. If such a block of seats is found, `FindBlock` returns the column index of the first (i.e., the lowest index) seat in the block; otherwise, it returns -1.

The seating diagram for passengers of `Flight ap2002` is repeated here for your convenience.

| | [0] | [1] | [2] | [3] | [4] | [5] |
|---|---|---|---|---|---|---|
| | window | middle | aisle | aisle | middle | window |
| [0] | "Kelly" | "Robin" | "" | "Sandy" | "" | "Fran" |
| | window | middle | aisle | aisle | middle | window |
| [1] | "Chris" | "Alex" | "" | "" | "Pat" | "Sam" |

The following table shows several examples of calling `FindBlock` for `Flight ap2002 as` shown.

| Function Call | Value Returned |
|---|---|
| `ap2002.FindBlock(0, 1)` | `2` or `4` |
| `ap2002.FindBlock(0, 2)` | `-1` |
| `ap2002.FindBlock(1, 2)` | `2` |

Complete function `FindBlock` below.

```
int Flight::FindBlock(int row, int seatsNeeded) const
// postcondition: returns column index of the first (lowest index)
//                seat in a block of seatsNeeded adjacent
//                empty seats in the specified row;
//                if no such block exists, returns -1
```

(c) You will write the `Flight` member function `AssignGroup`, which is described as follows. The parameter to the `Flight` member function `AssignGroup` is an array of passengers, `group`. These passengers require a block of adjacent seats in a single row. `AssignGroup` searches for `group.length()` adjacent empty seats in some row. If such a block of seats is found, the passengers in `group` will be assigned to those seats, and `AssignGroup` returns `true`. Otherwise, no passengers are assigned to seats, and `AssignGroup` returns `false`.

For example, the seats in `Flight ap314` are as shown in the first diagram below. If the array `adults` contains three passengers, the call `ap314.AssignGroup(adults)` makes no changes to `ap314` and returns `false`, because there is no block of three adjacent empty seats in a single row. On the other hand, suppose the array `kids` contains passengers `"Sam"` and `"Alex"`. The call `ap314.AssignGroup(kids)` will assign `"Sam"` and `"Alex"` to the seats shown in the second diagram below and return `true`.

Contents of `mySeats` for `ap314` before any call to `AssignGroup`

| | [0] | [1] | [2] | [3] | [4] |
|---|---|---|---|---|---|
| [0] | window<br>"Kelly" | aisle<br>"" | aisle<br>"Sandy" | middle<br>"" | window<br>"Fran" |
| [1] | window<br>"Chris" | aisle<br>"" | aisle<br>"" | middle<br>"Pat" | window<br>"" |

Contents of `mySeats` for `ap314` after call to `ap314.AssignGroup(kids)`

| | [0] | [1] | [2] | [3] | [4] |
|---|---|---|---|---|---|
| [0] | window<br>"Kelly" | aisle<br>"" | aisle<br>"Sandy" | middle<br>"" | window<br>"Fran" |
| [1] | window<br>"Chris" | aisle<br>"Sam" | aisle<br>"Alex" | middle<br>"Pat" | window<br>"" |

In writing `AssignGroup`, you may call `FindBlock` specified in part (b). Assume that `FindBlock` works as specified, regardless of what you wrote in part (b).

Complete function `AssignGroup` below.

```
bool Flight::AssignGroup(const apvector<Passenger> & group)
// postcondition: if possible, assigns the group.length() passengers
//                from group to adjacent empty seats in a single row
//                and returns true;
//                otherwise, makes no changes and returns false
```

**END OF EXAMINATION**