Vector class

// for a vector of Items use Vector<Item>, e.g., Vector<int> intvector;
//
// note: Item must have a default constructor
//
/// constructors:
/// Vector() -- default, vector of size 0 (no entries)
/// Vector(int size) -- vector with size entries
/// Vector(int size,
///       Item fillValue) -- vector w/ size entries all == fillValue
/// Vector(const Vector & vec) -- copy constructor
///
/// int Length() -- returns size of vector (capacity)
/// void SetSize(int newSize) -- resizes the vector to newSize elements
/// (can result in losing elements if
///   new size < old size)
/// void resize(int newSize) -- synonym for SetSize
///
/// void Fill(Item fillValue) -- set all entries == fillValue
///
/// operator = -- assignment operator works properly
/// operator [] -- indexes both const and non-const vectors
///
/// examples of use:
///
/// Vector<double> dlist(100); // a list of 100 doubles
/// Vector<double> dzlist(100,0.0); // initialized to 0.0
///
/// Vector<String> slist(300); // 300 strings
///
/// Vector<int> ilist; // has room for 0 ints

PROBLEM 1:  (Searching for that Perfect Book: (27 pts))

Consider the following definitions and constructors.

struct Book
{
    string title;       // title
    string author;      // author
    char rating;        // rating of book: R(recommend), A(acceptable), M(marginal)
    int lowAge;         // lowest age recommended for reading
The BookList iterator class contains information on books, including ratings and appropriate ages. In particular, BookList iterates over all the books for a specified age.

Here is a sample code segment that uses the BookList class to list out one book appropriate for a three year old, and one book appropriate for an eight year old.
BookList library;
library.First(3);
cout << "... a book for a 3 year old is ..." << endl;
if (!library.IsDone())
    library.Current().Print();
library.First(8);
cout << "... a book for an 8 year old is ..." << endl;
if (!library.IsDone())
    library.Current().Print();

Following might be the corresponding output.

... a book for a 3 year old is ...
Jamberry, by Degan.
Ages 2-4. Rating R.
... a book for an 8 year old is ...
The Cat in the Hat, by Dr. Seuss.
Ages 4-10. Rating R.

The function Read reads in information about books from a file, storing this information in the vector myBooks and the number of the books in myNumBooks. The books are stored in myBooks in no particular order. You may assume Read already exists, you DO NOT need to write it.

Complete the following functions below. Note that the iterator moves over all the books in a specified age range and returns a current book (function Current) for that specified age. Do NOT add any additional private variables to the BookList class.

- Complete the function First below (5 pts).

```cpp
void BookList::First(int age)
// postcondition: iterator set to first book in specified age range
{
}
```

- Complete the function IsDone below (3 pts).

```cpp
bool BookList::IsDone() const
// postcondition: returns true if iterator has no current book in
// specified age range, otherwise returns false
{
}
```

- Complete the function Next below (5 pts).

```cpp
...
void BookList::Next()
// postcondition: iterator moved to next book in specified age range
{

}

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

Book BookList::Current()
// precondition: iterator is pointing to a valid book in specified age range
// postcondition: returns current book in iterator
{

}

• Complete the function *CurrentRating* below (3 pts).

char BookList::CurrentRating()
// precondition: iterator is pointing to a valid book in specified age range
// postcondition: returns rating of current book in iterator
{

}

• For this part, complete the function PrintRating that is part of a client program using the *BookList* class. PrintRating is given a rating and an age range and prints information on all the books of the given rating in the specified age range.

For example, the call PrintRating('R', 3, 8, library) would print information on the two books *Jamberry* (which is for ages 2-4) and *The Cat in the Hat* (which is for ages 4-10) in the same format as it printed these two books in the earlier example, plus all the other books that are appropriate for 3-8 year-olds with the rating of R.

**NOTE:** Each appropriate book should only be printed once.

Complete the function PrintRating below (6 pts).

void PrintRating(char rating, int lowage, int highage, BookList & library)
// precondition: rating is a valid rating, 0 <= lowage <= highage
// postcondition: prints information on all books from library with the
// specified rating and age range.
{

• Explain why library is passed by reference in PrintRating (2 pts)?
Dr. Seuss wrote a story about Sneetches, a fictional animal. There are two types of Sneetches, those with a star on their belly and those without a star on their belly. 
Consider the following definition for a linked list node containing information about Sneetches.

```cpp
struct Node
{
    string name;  // name of sneetch
    char type;   // 'S' - with star, 'W' - without star
    Node * next;

    Node(const string &nm, char ty, Node * nx);
};

Node::Node(const string &nm, char ty, Node * nx)
    : name(nm), type(ty), next(nx)
{};
```

PART A (6 pts):
Write the function *PrintWithStars* whose header is shown below. *PrintWithStars* outputs the names of Sneetches in the given linked list that have a star on their belly (type is 'S'). Names are printed on one line separated by blanks.

For example, consider the linked list shown below.

```
  list  kate  enrico  stacie  faith  eric  junfei
       W    W     S      W   S    S
```

The output for the call *PrintWithStars(list)* would be:

stacie eric junfei

Complete function *PrintWithStars* below.

```cpp
void PrintWithStars(Node * list)
{ // postcondition: prints the names of Sneetches in the list with stars on
// their bellies on one line, separated by blanks
}
```

PART B (3 pts):
Write the function *InsertFront* whose header is shown below. *InsertFront* is given a linked list and info on a Sneetch (its name and type). The new Sneetch is inserted at the front of the linked list.
For example, if *list* is the linked list from Part A, then the list below is the result of calling `InsertFront(list, "carlton", 'S').`

```
+-------+       +-------+       +-------+       +-------+       +-------+       +-------+       +-------+
|  carlton |       |  kate  |       |  enrico |       |  stacie |       |  faith  |       |  eric   |       |  junfei |
|         W |       |         W |       |         S |       |         W |       |         S |       |         S |       |         S 
```

Complete function *InsertFront* below the following header.

```c
void InsertFront(Node * & list, const string & name, char type)  
// precondition: Sneetch with name does not appear in list  
// postcondition: inserted Sneetch at the front of the list  
{
}
```

**PART C (8 pts):**

Write the function `RemoveFirstStarSneetch` whose header is shown below. `RemoveFirstStarSneetch` is given a linked list and removes the first star-bellied Sneetch in the list. It assumes that there is at least one star-bellied Sneetch.

For example, if *list* is the linked list from Part A, then the list below is the result of calling `RemoveFirstStarSneetch(list), that is stacie is removed.

```
+-------+       +-------+       +-------+       +-------+       +-------+       +-------+       +-------+
|  kate  |       |  enrico |       |  stacie |       |  faith  |       |  eric   |       |  junfei |
|         W |       |         W |       |         S |       |         W |       |         S |       |         S |       |         S 
```

Complete function *RemoveFirstStarSneetch* below the following header.

```c
void RemoveFirstStarSneetch(Node * & list)  
// precondition: there is at least one star-bellied Sneetch in list.  
// postcondition: removes the first star-bellied Sneetch in the list  
{
}
```

**PROBLEM 3 : (Truckin Down the Highway: (16 pts))**

Consider the following definition to keep track of the load weight capacity and current load weight for trucks for a trucking company.

```c
struct Truck  
{
    string name;  // name of truck
```
double load;    // weight of current load on truck
double capacity;  // capacity weight for truck

Truck(const string &, double, double);    // constructor
};

Truck::Truck(const string & nm, double ld, double cap)
: name(nm), load(ld), capacity(cap)
{
}

PART A (6 pts):
Write the function FindSmallest whose header is shown below. FindSmallest returns the index position of the truck with the smallest weight load from company.

For example, if company contains the truck info shown below (where the first number is the load and the second number is the capacity), then the call FindSmallest(company, 8) returns 4 (the Purple Rider truck), and the call FindSmallest(company, 3) returns 0 (the Red Rider truck).

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Rider</td>
<td>120.0</td>
<td>300.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blue Rider</td>
<td>250.0</td>
<td>500.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Green Rider</td>
<td>450.0</td>
<td>660.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yellow Rider</td>
<td>130.0</td>
<td>300.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purple Rider</td>
<td>100.0</td>
<td>300.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown Rider</td>
<td>400.0</td>
<td>500.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Black Rider</td>
<td>150.0</td>
<td>500.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orange Rider</td>
<td>200.0</td>
<td>250.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Complete function FindSmallest on the next page

```
int FindSmallest(const Vector <Truck *> & company, int size)
// precondition: size > 0, company has size elements
// postcondition: returns the index of the truck with the smallest load,
// if there is more than one truck with the smallest load, return
// index of any such one.
{
}
```

PART B (10 pts):
Write the function FillUpTruck whose header is shown below. FillUpTruck inserts a new truck into company with the given name and capacity. It then loads the truck as full as possible by removing loads from those trucks with the smallest loads. A load is removed from an existing truck and put on the new truck only if the existing truck has the smallest load and its complete load will fit on the new truck. Trucks with an empty load are removed.
For example, let `company` represent the vector shown in Part A, and `size` be the current number of elements in `company`. The call `FillUpTruck(company, size, "Road Runner", 500.0)` adds the truck `Road Runner` with capacity 500.0 at the end of the vector. It then puts the load from `Purple Rider`, the smallest load, onto `Road Runner` and removes `Purple Rider` from the vector. It then puts the load from `Red Rider`, now the smallest load, onto `Road Runner` and removes `Red Rider` from the vector. It then puts the load from `Yellow Rider`, now the smallest load, onto `Road Runner` and removes `Yellow Rider` from the vector. The next smallest load would not fit on `Road Runner`. After the call, the vector `company` might look like the following (the order of trucks could be different).

In writing `FillUpTruck`, you may call the function `FindSmallest` that you wrote in Part A. Assume `FindSmallest` works correctly, regardless of what you wrote in Part A.

Complete function `FillUpTruck` on the next page.

```cpp
void FillUpTruck(Vector <Truck *> & company, int & size, const string & name, double capacity)
// precondition: size > 0, company has size elements
// postcondition: add name to company filling it up with smallest loads
// and removing trucks whose loads are now empty, update size
{
}
```