PROBLEM 1:  \((\textit{Sneetches Play: (17 pts)})\)

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 \texttt{PrintWithStars} whose header is shown below. \texttt{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.

The output for the call \texttt{PrintWithStars(list)} would be:

```
stacie eric junfei
```

Complete function \texttt{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 \textit{InsertFront} whose header is shown below. \textit{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 \textit{list} is the linked list from Part A, then the list below is the result of calling \textit{InsertFront}(list, "carlton", 'S').

\begin{figure}[h]
  \centering
  \includegraphics[width=0.8\textwidth]{sneetch_list.png}
  \caption{Sneetch list}
\end{figure}

Complete function \textit{InsertFront} below the following header.

\begin{verbatim}
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
{

}
\end{verbatim}
PART C (8 pts):
Write the function \textit{RemoveFirstStarSneetch} whose header is shown below. \textit{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 \textit{list} is the linked list from Part A, then the list below is the result of calling \textit{RemoveFirstStarSneetch(list)}, that is stacie is removed.

\begin{center}
\begin{tikzpicture}
\node[rectangle] (kate) at (0,0) {kate W} ;
\node[rectangle] (enrico) at (1,0) {enrico W} ;
\node[rectangle] (faith) at (2,0) {faith W} ;
\node[rectangle] (eric) at (3,0) {eric S} ;
\node[rectangle] (junfei) at (4,0) {junfei S} ;
\draw (kate) -- (enrico) ;
\draw (enrico) -- (faith) ;
\draw (faith) -- (eric) ;
\draw (eric) -- (junfei) ;
\end{tikzpicture}
\end{center}

Complete function \textit{RemoveFirstStarSneetch} below the following header.

\begin{verbatim}
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
{
\end{verbatim}
Consider the following definition to keep track of the load weight capacity and current load weight for trucks for a trucking company.

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

```
int FindSmallest(const tvector<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).

```
0 1 2 3 4 5 6 7 8 9 10
Blue Rider 250.0 500.0
Green Rider 450.0 600.0
Road Runner 350.0 500.0
Orange Rider 200.0 250.0
Brown Rider 400.0 500.0
Black Rider 350.0 500.0
```

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.

```cpp
void FillUpTruck(tvector <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
{
}
```
Consider the following definitions to represent a list of CD’s.

```cpp
struct CDinfo {
    string artist;  // name of artist
    string title;   // title of CD
    double price;   // price of CD
};

tvector <CDinfo *> ByArtist;  // CD’s sorted by artist
btvector <CDinfo *> ByTitle;  // CD’s sorted by title
```

Assume there are N CDs. ByArtist contains pointers to the CDs, sorted by artist, and ByTitle contains pointers to the same CDs, sorted by title.

Considering the most efficient algorithm, what is the best representation for the worst case running time (big-Oh) for each of the following?

1. Assume the titles are unique. Given a title, find its CD. __________

2. Print all the titles that contain a phrase. (For example, if the phrase is "for" then the title "Automatic for the People" would be printed in addition to all other titles that contained the phrase "for"). __________

3. Print the CD whose title is first in alphabetical order. __________

4. Suppose that an artist has M CDs in the list of CDs. Given an artist, return the number of CDs by that artist (that is, calculate and return the value of M). __________

5. Print the name of the artist who has the most CD’s in the list. __________

Consider the following program (includes not shown).

```c
int Mystery(int n)
{
    if (n <= 0)
        return 0;
    else
        return 2 + Mystery(n-1);
}

void Something(int n)
{
    if (n > 0)
    {
        Something(n-1);
        cout << n << " ";
        Something(n-2);
    }
}

int main()
{
    cout << Mystery(4) << endl;
    Something(3);
    cout << endl;
}
```

1. What is the output of the program?

2. Give a recurrence relation for Mystery. (Do not solve it).

3. Give a recurrence relation for Something. (Do not solve it).
PROBLEM 5:  (Stack it many ways: (11 pts))

Consider the tstack class given in lecture.

PART A (5 pts):
Write the Stack member function \textit{tstack::max} whose header is shown below. The function \textit{tstack::max} returns the maximum element in the stack.

For example, consider the stack \textit{S} shown below.

\begin{verbatim}
stack S
    myTop → 49
    52
    67
    43
    98
    37
\end{verbatim}

The call \textit{S.max()} would return 98.

Complete function \textit{tstack::max} below.

\begin{verbatim}
template <class Type>
Type
tstack<Type>::max() const
// precondition: stack is not empty
// postcondition: returns the maximum element in the stack
{
\end{verbatim}
PART B (6 pts):

Write the free (not a member function) function `FindMiddle` whose header is shown below. The function `FindMiddle` returns the middle element in the stack.

For example, consider the stack `S` shown below.

```
stack S
myTop ← 49
  52
  67
  43
  98
  37
```

The call `FindMiddle(S)` should return 43. If there are two middle elements (as in this case), the one lower down on the stack is returned.

Complete function `FindMiddle` below. Note that `stk` is passed by const reference.

```cpp
int FindMiddle(const tstack<int> & stk)
// postcondition: returns the middle element in the stack,
// returns 0 if the stack is empty
{
```
Consider the *NumberList* class shown below.

```cpp
struct Node {
    int number;
    Node * next;

    Node(int newNumber, Node * newNext); // constructor
};

Node::Node(int newNumber, Node * newNext) : number(newNumber), next(newNext) {
}

class NumberList {
public:
    NumberList();
    int NumOccurences(int value);
    void InsertAfterLarger(int num);
    void RemoveNegatives();

    // other public members not shown

private:
    Node * myList; // list of numbers
};

NumberList N;

PART A (4 pts):
Write the *NumberList* member function *NumOccurences* whose header is shown below. The function *NumOccurences* returns the number of occurrences of *value* in the list of numbers.
For example, assume the NumberList N is represented by the list shown below

```
```

The call *N.NumOccurences(8)* would return 3, and the call *N.NumOccurences(-1)* would return 2.
Complete function *NumberList::NumOccurences* below.
int NumberList::NumOccurences(int value)  
// postcondition: returns the number of occurrences of value in list.  
{

PART B (6 pts):

Write the NumberList member function RemoveNegatives whose header is shown below. The function RemoveNegatives removes all the negative numbers from the list.

For example, assume the NumberList N is represented by the list shown in Part A. After the call N.RemoveNegatives(), the list is shown below.

```
myList
```

Complete function NumberList::RemoveNegatives below.

```
void NumberList::RemoveNegatives()  
// postcondition: all nodes with negative values have been removed from list  
{
```
PART C (6 pts):
Write the NumberList member function InsertAfterLarger whose header is shown below. The function InsertAfterLarger inserts the value \textit{num} after the first node whose value is larger than \textit{num}, or at the end of the list if there is no value larger.

For example, assume the NumberList \textit{N} is represented by the list shown in Part B. After the call \textit{N.InsertAfterLarger(6)}, the list is shown below.

Complete function \textit{NumberList::InsertAfterLarger} below.

```cpp
void NumberList::InsertAfterLarger(int num)
// postcondition: inserts num after the first node that
// contains a larger value. If no node contains a larger value,
// num is inserted at the end of the list.
{
```