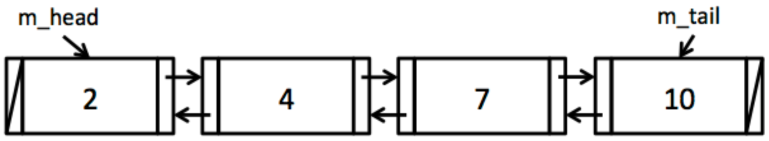
Midterm Practice

TA: Brian Choi

*** Make sure you try all exercises by hand! You won't have access to Visual C++ during the exam. ***

1. We will build a sorted doubly linked list without sentinel (dummy) nodes in this exercise. The following shows an example of a list with 4 elements, where the nodes are sorted in the increasing order of their values. The `m_prev` pointer of the head node and `m_next` pointer of the tail node are both `nullptr`. If the list is empty, head and tail pointers are both `nullptr`.



Assume the following declaration of `Node` structure and `SortedLinkedList` class.

```
struct Node                 class SortedLinkedList
{                           {
    ItemType m_value;           public:
    Node* m_prev;                   SortedLinkedList();
    Node *m_next;                   bool insert(const ItemType& value);
};                                  Node* search(const ItemType& value) const;
                                    void remove(Node* node);
                                    int size() const { return m_size; }
                                    void printIncreasingOrder() const;
                                private:
                                    Node* m_head;
                                    Node* m_tail;
                                    int m_size;
                            };
```

(a) Implement `SortedLinkedList()`.

```
SortedLinkedList::SortedLinkedList()
{




}
```

(b) Implement `insert()`. If a node with the same value is already in the list, do not insert a new node. Return true if a new node is successfully inserted, and return false otherwise. You may assume that `ItemType` has `<`, `>`, and `==` operators properly implemented.

```
bool SortedLinkedList::insert(const ItemType& value)
{

}
```

(c) Implement `search()`, which returns the pointer to the node with the specified value.

```
Node* SortedLinkedList::search(const ItemType& value) const
{

}
```

(d) Implement `remove()`. Assume `node` is either `nullptr` (in which case you would simply return) or a valid pointer to a Node in the list, as found in `search()`.

```
void SortedLinkedList::remove(Node* node)
{

}
```

(e) Implement `printIncreasingOrder()`, which prints the values stored in the list in the increasing order, one value in each line.

```
void SortedLinkedList::printIncreasingOrder() const
{

}
```

(f) The public interface of `SortedLinkedList` has a problem. More precisely, the user of this class can possibly break the integrity of the sorted linked list, only using the public interface of `SortedLinkedList`. Demonstrate this problem with an example. Also, suggest a fix, if you have an idea.