These are some review questions to test your understanding of the material. Some of these questions may appear on an exam.

1 Lists

Please refer to the definition of `ListNode` on page 3, `List` on page 3, and `ListItr` on page 4. Assume that all the member functions of the classes have been defined and are available for your use. You do not have to define them.

1.1 Suppose you have constructed a `List` named `lst` that may or may not have elements in it. Write valid `C++` code to construct an iterator for `lst` that is positioned at the last element. If `lst` is empty, the iterator is to be past end.

1.2 1. Write a member function for `List`

```
template <class Object>
void List<Object>::reversePrint(ostream & out)
```

that uses `ListItr`s to print the elements of the `List` to `out` in reverse order (from tail to head). The elements are to be comma-separated and enclosed in "`<>`" delimiters. Do not construct a list of the elements in reverse order, just use iterators. You may assume that `ListItr` has the following operators defined:

`bool operator==(const ListItr<Object> & rhs)` returns `true` if `rhs` iterator is set to same node as this iterator.

`const ListItr<Object> & operator=(const ListItr<Object> & rhs)` makes this iterator's current node the same as that of `rhs` and returns this iterator.

2. What is the "Big-Oh" asymptotic time performance of this method in worst case?

1.3 Write a definition for `splice`, a new member function of the `List` class:

```
template <class Object>
bool List<Object>::splice (const List<Object>& lst,
                           ListItr<Object> & pos)
```

This function "splices" the specified list `lst` into this `List` object at the specified position (`pos` is a `ListItr` over "this" `List`. `splice` returns false if `pos` is past end.

What is the worst-case Big-Oh asymptotic time performance of your method?

Example: Suppose `lst1 is (1,2,3,4,5)` and `lst2 is (10,20,30)`. Suppose `itr` is constructed as `lst1.first()`, and advanced twice so it is positioned at the "3" element. Then the expression `lst1.splice(lst2,itr)` returns `true` and causes `lst1` to become `(1,2,3,10,20,30,4,5)`

1.4 Given two lists of lengths $n1$ and $n2$:

$$L_1 = (x_1, x_2, \cdots, x_{n1})$$

$$L_2 = (y_1, y_2, \cdots, y_{n2})$$

the *shuffleMerge* of L_1 and L_2 is the new list

$$L_3 = (x_1, y_1, x_2, y_2, \cdots)$$

Examples:

```
L1 = <1,2,3>; L2 = <4,5,6>;  shuffleMerge ==> <1,4,2,5,3,6>

L1 = <1,2,3>; L2 = <4>;      shuffleMerge ==> <1,4,2,3>

L1 = <1,2,3>; L2 = <>;       shuffleMerge ==> <1,2,3>
```

Write a definition of:

```
template <class Object>
List<Object>
List<Object>::shuffleMerge(const List<Object> lst)
```

that returns a new List produced by the shuffle merge of this list with `lst`. An empty list is returned if the given lists are both empty.

1.5 Fill in the following table of Big-Oh worst-case asymptotic time performance for the given operations on `List` (page 3). Give your answers in terms of n, the number of elements in the list.

Operation	Worst Case Time
`insert`	
`find`	
`findPrevious`	
`remove`	
`makeEmpty`	
`isEmpty`	
`first`	

Definition of ListNode Class

This definition is directly from the text.

```
template <class Object>
class ListNode
{
  ListNode( const Object & theElement = Object( ),
            ListNode * n = NULL )
    : element( theElement ), next( n ) { }

  Object   element;
  ListNode *next;

  friend class List<Object>;
  friend class ListItr<Object>;
};
```

Definition of List Class

This definition is directly from the text.

```
template <class Object>
class List
{
  public:
    List( );
    List( const List & rhs );
    ~List( );
    bool isEmpty( ) const;
    void makeEmpty( );
    ListItr<Object> zeroth( ) const;
    ListItr<Object> first( ) const;
    void insert( const Object & x, const ListItr<Object> & p );
    ListItr<Object> find( const Object & x ) const;
    ListItr<Object> findPrevious( const Object & x ) const;
    void remove( const Object & x );
    const List & operator=( const List & rhs );
  private:
    ListNode<Object> *header;
};
```

Definition of `ListItr` Class

This definition is directly from the text.

```
template <class Object>
class ListItr
{
  public:
    ListItr( ) : current( NULL ) { }
    bool isPastEnd( ) const
      { return current == NULL; }
    void advance( )
      { if( !isPastEnd( ) ) current = current->next; }
    const Object & retrieve( ) const
      { if( isPastEnd( ) ) throw BadIterator( );
        return current->element; }
  private:
    ListNode<Object> *current;    // Current position
    ListItr( ListNode<Object> *theNode )
      : current( theNode ) { }

    friend class List<Object>; // Grant access to constructor
};
```