In the following exercises, assume that you have a Simple Linked List class `List` with the following definition:

```
template <class keyType, class dataType>
class List
{
    public:
        // Member Functions
        List();  // Create an empty List
        ~List(); // Class Destructor
        // Functions Prototype Definitions
        bool listIsEmpty() const;
        bool curIsEmpty() const;
        void toFirst();  bool atFirst() const;
        void advance();  void toEnd();
        bool atEnd() const;  int listSize() const;
        void updateData (const dataType & );
        void retrieve (keyType &, dataType &) const;
        void insertFirst (const keyType &, const dataType & );
        void insertAfter (const keyType &, const dataType & );
        void insertBefore (const keyType &, const dataType & );
        void insertEnd (const keyType &, const dataType & );
        void deleteNode();  void deleteFirst();
        void deleteEnd();  void makeListEmpty();
        bool search (const keyType & );
        void orderInsert(const keyType &, const dataType & );
        void traverse();
    private:
        // Node Class
        class node
        {
            public:
                keyType key;   // key
                dataType data;  // Data
                node *next;  // pointer to next node
        }; // end of class node declaration
        typedef node * NodePointer;
        NodePointer head, cursor, prev;  // Pointers
    }; // End of class List declaration
```

**Group 1**

*Assume that both key and data fields are integers*
Write **user application** functions to:

1. Receive a list and to return how many of the nodes contain zeros in the data field.
2. Receive a list and to return how many of the nodes contain even numbers in the key field.
3. Receive a list and to return the sum of data stored in the list.
4. Receive a list (L) and to split it at the approximate middle into two separate lists, returning two new lists (L1) and (L2). After the split, (L) will become empty.
5. Receive a list (L) and to split it into two separate lists, returning two new lists (L1) containing positive data numbers and (L2) containing negative data numbers. After the split, (L) will become empty.
6. Receive two lists and return true if they are identical, ordered in the same way. Assume that the two lists are not empty.
7. Receive a list and a number (m) and to return the list after deleting the first m nodes. If the list size is less than or equal to (m), delete the whole list.
8. Receive a non-negative integer \( n \) and return a list in which each decimal digit of the number is placed in the data field in a node by itself and the order of the digit is to be put in the key field of the node. The order of the nodes should preserve the order of the digits in the original integer from left to right.
9. Receive a list (L1) and a number (m) and to return another list (L2) that contains an exact copy of the first (m) elements from (L1). If the size of (L1) is less than or equal to (m), copy the whole list into (L2). In any case, L1 should remain unchanged.
10. Receive two lists (L1) and (L2) and append (L2) to the end of (L1), returning the result in a new list (L) without changing the original lists.
11. Receive a list and to return true if the keys in the node are in ascending order.
12. Receive a long integer \( \text{Num} \) in the form of a string of digits (e.g. “5467876567890876643246765434543457”) and to return a list containing its digits in the data field (one digit per node). The order of the digit is to be put in the key field of the node. The digits will be ordered such that the least significant digit will be in the first node and the most significant digit in the last node.

**Group 2**

Add **member** functions to the **List** class with the following specifications:

1. A public function `.Recursive_List_Size()` to call a recursive private function to return the size of the list.
2. A public function `.DisplayList()` to call a recursive private function to display the contents of the nodes in the list.
3. A public function `.SameAs(List &L2)` to return true if the list is the same as another list \( L2 \)