King Saud University
College of Computer and Information Sciences
Computer Science Department

CSC 212 **First Semester 1438-1439**

Tutorial #4

Problem 1

Method reverse ( )
Requires: none. **Input**: none **Output**: none.
Results: the elements of the list will be stored in reverse order.
Where the 1st, 2nd, 3rd, ... , i-1th, ith elements will be ith, i-1th, i-2th, ... , 2nd, 1st
Example: We have a List<Integer> in our main class.
With its elements looking like this:(14; 43; 28; 66; 33; 21)
Once we execute the reverse method they should look like this:(21; 33; 66; 28; 43; 14)

- Write the reverse method as an implementer of the LinkedList ADT

```
public void reverse() {
     // no need to reverse if the list have one or no elements
     if (head != null && head.next != null) {
          Node<T> pred = null, cur = head, succ = head.next;
          while (cur != null) {
               cur.next = pred;
               pred = cur;
               cur = succ;
               if (succ != null)
                    succ = succ.next;
          }
          head = pred;
     }
}
```

- Write the reverse method as a user of the List ADT

```
private static <T> void reverse(LinkedList<T> list) {
     if (!list.empty()) {
          list.findFirst();
          // first we will count the number of elements
          int count = 1; // for the last element
          while (!list.last()) {
               count++;
               list.findNext();
          }
          // no need to reverse if the list have one or no elements
          if (count > 1) {
               T left = null, right = null;
               list.findFirst();
               for (int i = 0; i < (count / 2); i++) {
                    left = list.retrieve();
                    for (int j = i; j < count - i - 1; j++)
                         list.findNext();
                    right = list.retrieve();
                    list.update(left);
                    list.findFirst();
                    for (int j = 0; j < i; j++)
                         list.findNext();
                    list.update(right);
                    list.findNext();
               }
          }
     }
}
```

Problem 2

A circular left shift (CLS) of a list consists in moving the first element to the last position while leaving the order of the remaining elements unchanged. Write a static method CLS (user of ADT) that takes as input a non-empty list l and an integer n (n >= 0) and applies n circular left shifts to the list l.

Example: assuming list: 1, 2, 3, 4. After calling CLS(list, 2) then list will be: 3, 4, 1, 2. **Method:** *public static<T> void CLS(List<T> list, int n)*

```
public static<T> void CLS(List<T> list, int n) {
     for (int i = 0; i < n; i++) {
          list.findFirst();
          T elem = list.retrieve();
          list.remove();
          if (!list.empty()) {
               while (!list.last())
                    list.findNext();
          }
          list.insert(elem);
     }
}
```

Problem 3
Write a static method *switch* that takes as input two lists and switches all the elements of the two lists except for the first element in both lists.

Example: assuming list1: 1, 2, 3 and list2: 4, 5. Calling *switch(list1, list2)* will result in list1: 1, 5 and list2: 4, 2, 3.

Method: *public static<T> void switch(List<T> list1, List<T> list2)*

```
public static <T> void switch (List <T> list1, List <T> list2){
     if (list1.empty() && list2.empty())
          return;
     if (!list2.empty())
          list2.findFirst();
     if (!list1.empty()){
          list1.findFirst();
          if (!list1.last()) {
               list1.findNext();
               while (!list1.last()) {
                    T elem = list1.retrieve();
                    list1.remove();
                    list2.insert(elem);
               }
               T elem = list1.retrieve();
               list1.remove();
               list2.insert(elem);
          }
     }
     if (!list2.last()) {
          list2.findNext();
          while (!list2.last()) {
               T elem = list2.retrieve();
               list2.remove();
               list1.insert(elem);
          }
          T elem = list2.retrieve();
          list2.remove();
          list1.insert(elem);
     }
}
```

Problem 4
Write the method *isPalindrome* part of the Double linkedlist ADT. It should return true if the list is a palindrome. False otherwise. A palindrome is a word, phrase or anything that reads the same forward or reversed.

Examples:

- l(13, 54, 76, 54, 13) → true
- l("A", "Bus", "Bus", "A") → true
- l(300, 400, 500) → false

Method: ***public boolean isPalindrome()***

```
public boolean isPalindrome() {
     if (head != null) {
          Node<T> start = head;
          Node<T> end = head;
          while (end.next != null)
               end = end.next;
          while (start != end && end.next != start) {
               if (!start.data.equals(end.data))
                    return false;
               start = start.next;
               end = end.previous;
          }
     }
     return true;
}
```