**Examples on double linked list using ListIterators. You can do the same problems without ListIterator as exercise.**

Write a method addWithCondition to be included in the class KWLinkedList and accepts two parameters item and item1 of type E. The methods finds the first node having its data as item and inserts before it a node having its data as item1 and returns true, otherwise it returns false.

/\* using iterator \*/

public boolean addWithCondition(E item, E item1)

{

if (head==null) return false;

ListIterator<E> iter = listIterator(0);

E a;

while(iter.hasNext())

{

a=iter.next();

if(a.equals(item))

{

int b=iter.nextIndex();

/\* if the node to be dded before the first node \*/

if ( b-2 < 0 )

{

addFirst(item1);

return true;

}

/\* to add the node some where in the middle or before the last node \*/

iter=listIterator(b-2);

iter.next();

iter.add(item1);

return true;

}

} /\* for while \*/

return false; /\* item is not existing \*/

}

**Another solution using Iterators**

public boolean addWithCondition(E item, E item1)

{

if(head == null) return false;

ListIterator<E> iter=listIterator();

while (iter.hasNext())

{

if(item.equals(iter.hasNext())

{

if(item.euals(iter.next()))

iter.previous();

iter.add(item1);

return true;

}

}

return false;

}

**/\* without iterator \*/**

public boolean addWithCondition(E item, E item1)

{

if (head==null) return false;

Node<E> ptr = head;

Node<E> newNode = new Node<E>(item1);

while (ptr !=null && !found)

{

if (ptr.data == item) found=true;

else ptr=ptr.next;

}

if (found)

{ if (ptr == head)

{ /\* you can call addFirst(item1) if permitted or do the below \*/

newNode.next = head;

head.prev = newNode;

head = newNode;

}

else

{ /\* in any other location other than the beginning \*/

ptr.prev.next = newNode;

newNode.next = ptr;

newNode.prev = ptr.prev;

ptr.prev=newNode;

}

size++;

return true;

}

else return false;

}

Write a function checkForEquality to be included in class KWLinkedList that does not have any parameter. Assume the list is having even number of nodes. The function returns true if the first half of the list is equal to the second half of the list in the reverse order, otherwise it returns false. If the list is empty, the function returns false;

**/\* with iterator \*/**

public boolean checkForEquality()

{

/\* forward iterator \*/

ListIterator<E> nextiter = listIterator(0);

/\* backward iterator \*/

ListIterator<E> previter = listIterator(size);

int count=0;

E a,b;

/\* No need for hasNext() \*/

while ( count < size/2)

{ a= nextiter.next();

b=previter.previous();

count++;

if (! a.equals(b)) return false;

}

return true;

}

**/\* without iterator \*/**

public boolean checkForEquality()

{

Node<E> fptr = head;

Node<E> bptr =tail;

int count=0;

while (count < size/2)

{

if (fptr.data != bptr.data) return false;

else

{

fptr= fptr.next;

bptr=bptr.prev;

count ++;

}

}

return true;

}

Write a function **TheCorrespondingIndex** to be included in KWLinkedList class that accepts one parameter of type KWLinkedList, the list has character elements, The functions searches for the first node that has "x" as its data and deletes the data of the corresponding node in "this" list and returns true. Assume both lists are of the same size. If "x" is not found, the function returns false.

**/\* with ListIterator \*/**

public boolean removeTheCorrespondingIndex (KWLinkedList<Character> list)

{

/\* This iterator for list linked list which has characters as data \*/

if (list.head == null) return false;

ListIterator<Character> iter= list.listIterator ();

boolean found=false;

char a;

int b;

while(iter.hasNext() && !found)

{

a=iter.next();

if (a=='x') found =true;

}

if (!found) return false;

b = iter.nextIndex()-1;

/\* This iterator for "this" list making the reference of nextItem

reference points to a node with b index then apply next method

and at the end we apply remove method \*/

ListIterator<E> iter1=listIterator(b);

iter1.next();

iter1.remove();

return true;

}

**/\*without ListIterator \*/**

public boolean removeTheCorrespondingIndex (KWLinkedList<Character> list)

{

Node<E> ptr = list.head;

if (list.head == null) return false;

boolean found =false;

int count = 0;

while (ptr!= null && !found)

{

if (ptr.data != 'x')

{ count++;

ptr = ptr.next;

}

}

if (found)

{

int c =0

Node<E> p =head;

while (c < count)

{ p = p.next;

c++;

}

if (c == 0) /\* in the first node \*/

{

head = head.next;

if (head == null) tail = null;

else

{ head=head.next; head.prev = null;}

size--;

return true;

}

if (c == size-1) /\* at the end \*/

{ tail=p.prev;

tail.next=null;

else /\* some where in the middle \*/

{

p.prev.next=p.next;

p.next.prev=p.prev;

}

size--;

return true;

}

}

else return false; /\* for found \*/

}

Write a function called insertNode which to be considered within the KWLinkedList. The function accepts one parameter item of type integer and returns false if “this” list (the list) is empty or has less than two nodes. The function inserts a new node before the last node containing item as its data if the summation of the first and last node data are less than item and return true, otherwise, it returns false.

public boolean insertNode(int item)

{

if (size < 2)

return false;

if ((Integer(header.data) + Integer (tail.data)) < item)

{

Node<E> newNode = new Node<E> (item);

newNode.next = tail;

newNode.prev = tail.prev;

tail.prev.next = newNode;

tail.prev = newNode;

size++;

return true;

}

else return false;

}