**Single Linked List**

**Problems about SingleLinkedList class problems**

1. Write a function removeDuplicate to be considered within the SingleLinkedList class and accepts one parameter value of type E. The function removes all the duplicates of the value from the list (It keeps the first one). The function heading is

public void removeDuplicate(E value)

public void removeDuplicate(E value)

{ Node<E> ptr=head;

boolean found =false;

while (ptr !=null && !found)

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

else ptr=ptr.next;

if(ptr != null) /\* if found \*/

{

Node<E> prev;

prev = ptr;

ptr = ptr.next;

while(ptr !=null)

{

if(ptr.data == value)

{

/\* value = removeAfter(prev);

ptr =prev.next; \*/

prev.next= ptr.next;

ptr=ptr.next;

size--;

}

else {prev = ptr;

ptr=ptr.next;

}

}

}

}

1. Write a function removeAll to be considered within the SingleLinkedList class and accepts one parameter value of type E. The function removes all data items equal to value in the. The function heading is

public void removeAll(E value)

public void removeAll(E value)

{ if(head == null) return;

while (head != null && head.data == value )

{

head = head.next;

size--;

}

if (head == null) return;

Node <E> prev, ptr;

Prev = head;

ptr = prev.next;

E value;

while(ptr !=null)

{

if(ptr.data == value)

{

/\* value = removeAfter(prev);

ptr =prev.next; \*/

prev.next= ptr.next;

ptr=ptr.next; /\*ptr= prev.next; \*/

size--;

}

else {prev = ptr;

ptr=ptr.next;

}

}

}

}

1. Write a function printAndReplaceLarger to be included in the SingleLinkedList class and accepts two parameters value and replaceItem of type E. The function prints all the data in the nodes that are greater than value and replaces them with replaceItem. The function prototype is

public void printAndReplaceLarger(E value, E replaceItem)

{

Node<E> ptr=head;

while (ptr !=null)

{

if (((Comparable)ptr.data).compareTo (( Comparable) value)>0)

{ system.out.println("----" + ptr.data + " -----");

Ptr.data = replaceItem;

}

Ptr=ptr.next;

}

}

1. Write a member InsertMark to be included in the singleLinkedList and accepts three parameters value1, value2 and value 3 of type E. The function checks if value1 and value2 are data of consecutive nodes, the function inserts a node with value3 as its data after the node having v2 as its data. The process will be done for the whole list. Assume always value3 does not equal value1. The function returns true in all cases unless the list is not empty. The function heading is

public boolean InsertMark(E value1, E value2, E value3)

{

Node<E> ptr=head;

if ( ptr == null) return false;

while (ptr !=null)

{

if ( ptr.data == value1)

if(ptr.next != null)

if (ptr.next.data == value2)

{ addAfter(ptr.next.next,value3;(

ptr=ptr.next.next.next; }

/\* { ptr.next.next = Node<E> (value3,ptr.next.next);

ptr=ptr.next.next.next;} \*/

else ptr=ptr.next;

}

else return true;

else ptr = ptr.next;

}

return true;

}

1. Write a function compareReverse to be outside the SingleLinkedList class that accepts two parameters l1 and l2 of type singleLinkedList. The function returns true, if l1 is the reverse of l2, and returns false in all other cases. The function heading is

public static <E> boolean compareReverse (SingleLinkedList<E> l1, SingleLinkedList<E> l2)

{

int j=l1.size()-1;

int i=0;

while (i < l1.size())

{

If( l1.get(i) != l2.get(j))

return false;

j--;

i++;

}

return true;

}

1. Write a function selfReverse to be outside the SingleLinkedList class that accepts one parameter l1 of SingleLinkedList class. The function checks if half of l1 is the reverse of the second half, it returns true, otherwise, it returns false. You have to be sure that the number of nodes is even to start your comparison or otherwise the function returns false. The function heading is

public static <E> boolean selfReverse (SingleLinkedList<E> l1)

{

If (l1. size() % 2 != 0) return false;

int half = l1. size()/2;

int i=0;

int j= l1. size()-1;

for (int i=0; i < half; i++)

{

if (l1.get(i) != l1.get(j)) return false

j--;

}

return true;

}

1. Write function to be considered within SingleLinkedList class insertInBetween that has three parameters v1,v2 and v3 of type E. The function inserts a node with data as v3 between the two nodes having data v1 and v2. We consider only the first occurrence of v1 and v2. If the insertion is happened the method returns true, otherwise the method returns false.

public boolean insertInBetween(E v1, E v2, E v3)

{

if (head == null) return false;

Node<E> ptr,prev;

boolean found=false;

prev = head;

while (prev != null && !found)

{

if(prev.data == v1) found=true;

else

prev=prev.next;

}

if (!found) return false;

ptr= prev.next;

if (ptr == null) return false;

if(ptr.data == v2)

{ prev.next = new Node<E>(v3,ptr);

return true;

/\* addAfter(prev,v3);

return true; \*/ }

else return false;

}

**Another solution**

public boolean insertInBetween(E v1, E v2, E v3)

{

If (head == null) return false;

Node<E> ptr, newNode;

ptr = head;

while(ptr != null && ptr.next !=null)

{

if((ptr.data == v1) && (ptr.next.data == v2))

{

ptr.next=new Node<E> (v3,ptr.next);

return true;

}

else ptr = ptr.next;

}

return false;

}