UML DIAGRAM

```
Node
-----------------------------
data : int
next : Node*
-----------------------------
+getData() : int
~~+getNext() : Node~~
+setData(int val) : void
+SetNext(Node* V) : void
~~+Node()~~
```

```
LinkedList
-----------------------------
header : Node*
-----------------------------
+addFront(int newItem) : void
+addEnd(int newItem) : void
+addAtPosition(int position, int newItem) : void
+search(int ~~new~~Item) : int
+deleteFront() : void
+deleteEnd() : void
+deletePosition(int position) : void
+getItem(int position) : int
+printItems() : void
+LinkedList()
+LinkedList(int array[], int size)
```

Description :-

Node

* getData() → returns the value of the data member data which is an integer type of class ~~node~~ Node.

Camlin Page
Date / /

* getNext() → returns next which is of type ~~[illegible]~~ Node* of class Node.
* setData(int val) → The function sets the value of val into the data member data (i.e data = value).
* SetNext(Node* V) → The function sets Node* V into ~~the~~ ~~va~~ Node* next data member (i.e next = v).
* Node() → A default constructor of class Node that initializes the data variables of class Node.

LinkedList
* addFront(int newItem) → The function inserts a new node, containing the newItem, at the beginning of the list
* addEnd(int newItem) → The function inserts a new node, containing the newItem, at the end of the list.
* addAtPosition(int position, int newItem) :→ The function inserts a new node, containing the newItem, such that it is the position-th member of the list. i.e we assume the 1st element of the list is in position 1. If position is larger than the size of the list, the new item is added to the end of the list. If position < 1, the new item is added at the beginning of the list.
* search(int item) → The function searched the list for the item, and found it both prints the position of the item (followed by a space) and returns the position of the item in the list. If not found both prints 0 and returns 0.
* deleteFront() → The function deletes the 1st element of the list.
* deleteEnd() → The function deletes the last element of the list.
* deletePosition(int position) → The function deletes the element at the given position of the list. If the position < 1 or larger than list size print "outside range".

✱ getItem(int position) -> The function both prints the value of the item (followed by a space) and returns the value of the item at the given position of the list. If beyond size of the array, both prints std::numeric_limits<int>::max() and returns std::numeric_~~lim~~limits<int>::max() and should add/include <limits>.

✱ printItems() -> The function prints the value of the items of the list from head to tail. If empty list nothing is printed.

✱ LinkedList() -> A constructor with no parameters, which makes an empty list.

✱ LinkedList(int array[ ], int size) -> A constructor that takes an array of integers and makes a linked list, containing all the elements of the array in some order. It takes size of array as 2nd parameter.

Test Cases

	Input	Output
1)	5 2 7 10 AP 3 9	5 2 9 7 10
2)	3 4 2 1 DP 3 0	3 4 1
3)	45 20 2 10 GT 3 0	2 45 20 2 10