ATILIM UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

**Name: Name: Id:**

**Section:**

**Signature:**

**CMPE 226 Data Structures**

**Year** : 2018-2019 Fall

**Instructors** : E. Gökçay

**Mid-1** Examination

**Date**: 01.11.2018 **Time**: 16:30-18:00

**Duration**: 90 minutes

**WARNINGS**

* It is forbidden to bring electronic data storage equipments (mobile phones, MP3 players, flash disks and so on.) to exams.
* Students who either cheat, attempt to cheat or provide a help to other(s) in cheating, get 0 (zero) grade from this examination. Also, based on the regulations, a disciplinary action will be taken.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Q1 (10) | Q2 (15) | Q3 (25) | Q4 (10) | Q5 (15) | Total 100 |
|  |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Q1 (20) | Q2 (14) | Q3 (22) | Q4 (22) | Q5 (22) | Total 100 |
|  |  |  |  |  |  |

**Q1) (20pts)** Write a method called getLength() for the given LinkedList class to **calculate the number of nodes** in the list. Consider **extreme cases** like “empty list”. The prototype is given below. Write the method outside the class definition.

|  |  |
| --- | --- |
| **template <class T>**  **struct node {**  **T data;**  **node<T> \*next;**  **}** | **template <class T>**  **class LinkedList {**  **protected:**  **node<T> \*head;**  **public:**  **int getLength() ; // Prototype**  } |

**template <class T>**

**int LinkedList<T>::getLength(){**

**if ( head==NULL) return 0;**

**int cnt=0;**

**node<T> \*p=head;**

**while (p!=NULL) {**

**cnt++;**

**p=p->next;**

**}**

**return cnt;**

**}**

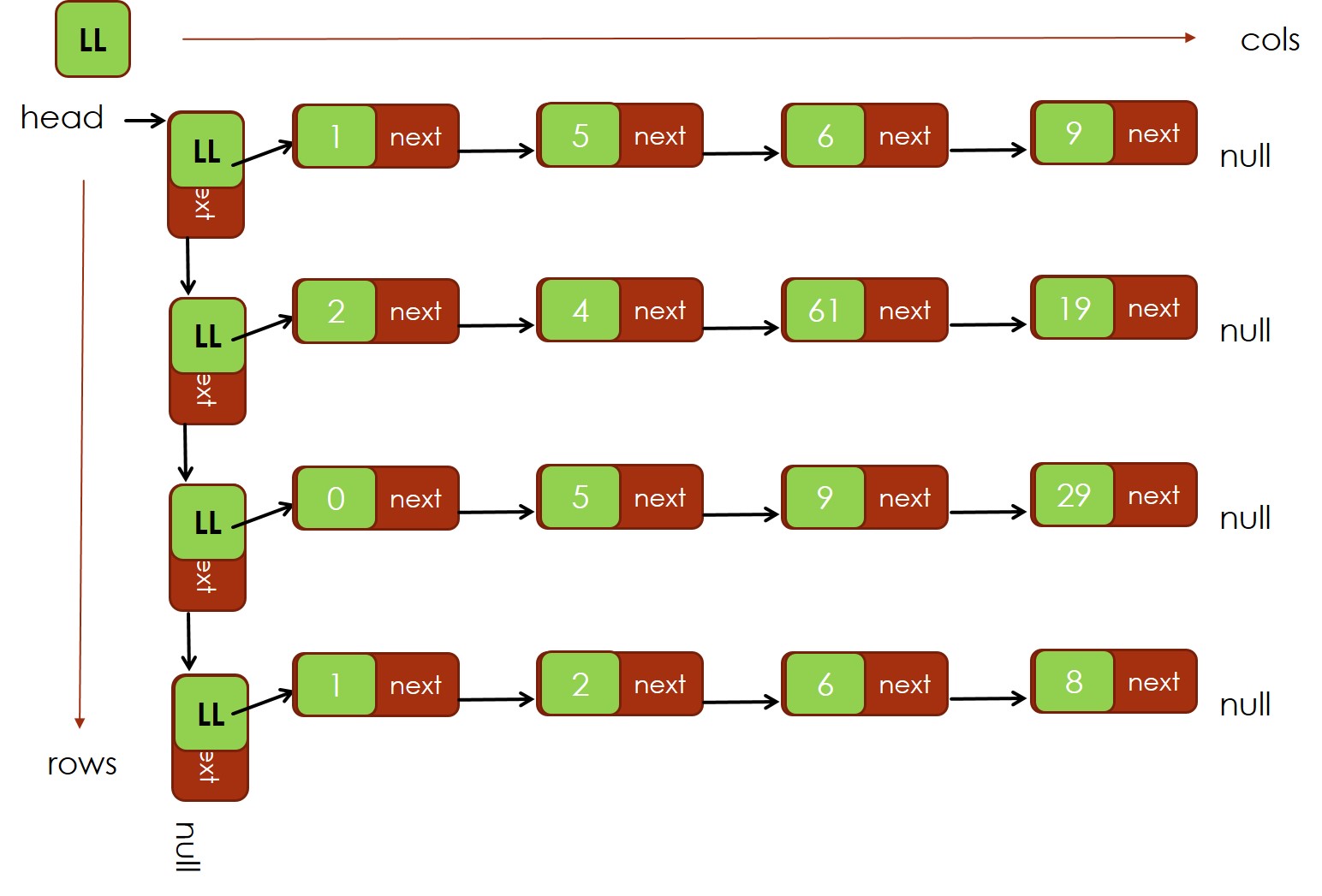
**Q2) (14pts)** You have the following data structure to represent a 2-dimensional array. What is the complexity **(9pts)** of reaching to a given index Arr2Dim(row,col) of this array? **Explain** your result. **Just giving the complexity alone will not be accepted**. Also write **the type definition** of this 2-dim array in the main program using the above LinkedList class **(5pts).**

**int main() {**

**…………………………………………. Arr2Dim;**

N

**}**

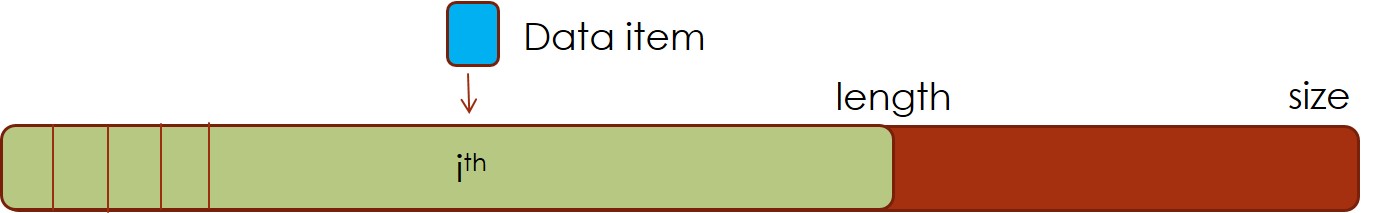
****

N

**We need to traverse the first LinkedList to find the given row number. This operation needs “row” # of iterations and proportional to N. After finding the given row, we have to traverse the LinkedList stored in that node to find col. That also requires “col” # of iterations and proportional to N. Total is 2N. Ignore constant. Complexity is O(N).**

**// Definition is**

**LinkedList< LinkedList<int> > Arr2Dim;**

**Q3) (22pts)** Assume that you have the following Array class. Add a method “insertData” to the class to **insert** a data element to a given index. **Update** all necessary class variables. Check **error cases** and return true if there is an error otherwise return false. Write the method outside the class definition. 

Error Cases:

* array has no room for insertion (2pts)
* array is not initialized (memory not allocated) yet. (2pts)
* index location is larger than available data items (2pts)

|  |
| --- |
| **template <class T>**  **class myArray {**  **protected:**  **T \*arr=NULL; // pointer to the array**  **int size=0; // The size of the array after initialization**  **int length=0; // Number of data items in the array**  **public:**  **myArray(int sz) {**  **arr = new T[sz];**  **size=sz;**  **length=0;**  **}**  **myArray( ) { }**  **bool insertData(T data, int index); // prototype of the method**  **}** |

**template <class T>**

**bool myArray<T>::insertData(T data, int index){**

**if (size==length ) return true;**

**if (size==0) return true;**

**if (index > length ) return true;**

**// Open space for the new item**

**// indexes start from 0**

**for (int n=length; n>index; n--){**

**arr[n]=arr[n-1];**

**}**

**arr[index]=data;**

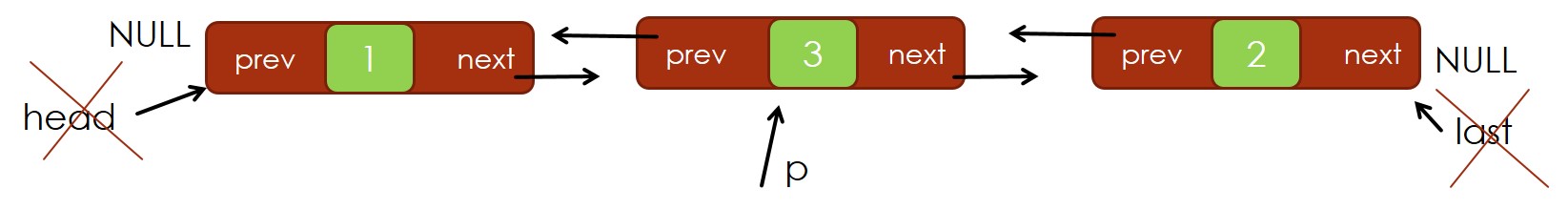
**length++;**

**return false;**

**}**

**Q4) (22pts)** The Double LinkedList class is given below. Assume that the information stored in head and last pointers is lost. But you have a pointer to an element in the list. **Write** a method to **recover** **and update head and last**. Consider **extreme cases** like p is pointing to the first node or to the last node. Visit each node once.

|  |  |
| --- | --- |
| **template <class T>**  **struct node {**  **T data;**  **node<T> \*next, \*prev;**  **};** | **template <class T>**  **class LinkedList {**  **protected:**  **node<T> \*head, \*last;**  **public:**  **void recoverList( node<T> \*p); //prototype**  **}** |

****

**template <class T>**

**void LinkedList<T>::recoverList( node<T> \*p){**

**// save p**

**node<T> \*q = p;**

**//find head**

**while (q->prev != NULL ){**

**q = q->prev;**

**}**

**head = q;**

**//find last**

**q = p;**

**while (q->next != NULL ){**

**q = q->next;**

**}**

**last=q;**

**}**

**Q5) (22pts)** Count the number of zeros in an integer array. **Write** a **recursive** method called getZeroCount( …… ). **Fill** the return type and parameter list.

**Hint**: If you need to pass a reduced size array to a method, use the original array and modified start/end indexes (Don’t create a smaller size copy of the array).

**int getZeroCount( int \*arr, int start, int end ){**

**if ( start==end) {**

**return arr[start]==0; // if equal return 1**

**} else {**

**return (arr[start]==0) + getZeroCount(arr,start+1,end);**

**}**

**}**