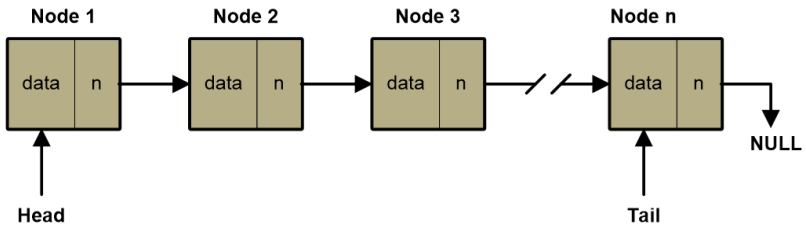
Exercise 2
List

Exercise Objectives:

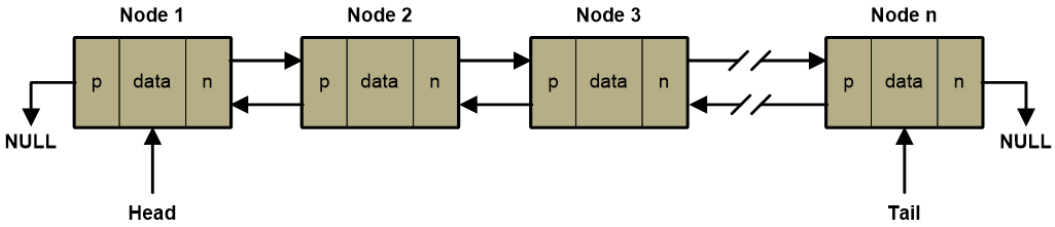
1. To write program to implement following operation in lists - creation, insertion, deletion and display.
2. Create program to implement LIST operations and ADTs in a common engineering design situations.

Singly Linked-List



Singly Linked list is a linked list in which each node contains only one link field pointing to the next field.

Doubly Linked-List



Doubly linked list is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains two fields, called links, which are references to the previous and to the next node in the sequence of nodes.

Hands-on Exercise:

[2_SinglyLL]

Create a singly linked list. The program can insert (front, rear, at) the elements into a singly linked list, delete (front, rear, at) the elements from that list and display all elements in that list.

- **insertFront(L, x)** – insert element at the front/start/first of the list
- **insertRear(L, x)** – insert element at the rear/end/last of the list
- **insertAt(L, x, p)** – insert element at a specified position in list
- **deleteFront(L)** – delete element at the front/start/first of the list
- **deleteRear(L)** – delete element at the rear/end/last of the list
- **deleteAt(L, p)** – delete element at a specified position in list
- **displayAll(L)** – display all elements in the list.

Data Structures

```
struct node{
    int data;
    struct node *next;
};
```

```
struct list{
    int count;
    struct node *head, *tail;
};
```

The main() executes the following sequence of operations:

```
1)  insertFront() data: 2
2)  displayAll()
3)  insertFront() data: 1
4)  displayAll()
5)  insertFront() data: 0
6)  displayAll()
7)  insertRear()  data: 3
8)  displayAll()
9)  insertRear()  data: 5
10) displayAll()
11) insertAt()    data: 4  position: 4
12) displayAll()
13) deleteFront()
14) displayAll()
15) deleteRear()
16) displayAll()
17) insertAt()    data: 88 position: 2
18) displayAll()
19) deleteAt()             position: 2
20) displayAll()
```

[3_DoublyLL]

Create a doubly linked list, insert (front, rear, at) the elements in to a doubly linked list, delete (front, rear, at) the elements from that list and display (from front, from rear) all elements in that list.

insertFront(L, x) – insert element at the front/start/first of the list

insertRear(L, x) – insert element at the rear/end/last of the list

insertAt(L, x, p) – insert element at a specified position in list

removeFront(L) – remove element at the front/start/first of the list

removeRear(L) – remove element at the rear/end/last of the list

removeAt(L, p) – remove element at a specified position in list

printList(L) – display all elements in the list from front to rear.

printMirror(L) – display all elements in the list from rear to front.

Data Structures

```
struct node{
    int data;
    struct node *prev, *next;
};
```

```
struct list{
    int count;
    struct node *head, *tail;
};
```

The main() executes the following sequence of operations:

```
1)  insertFront() data: 3
2)  printList()
3)  insertFront() data: 2
4)  printList()
5)  insertFront() data: 1
6)  printList()
7)  insertRear()  data: 5
8)  printList()
9)  insertRear()  data: 6
10) printList()
11) insertAt()    data: 4 position: 3
12) printList()
13) insertAt()    data: 7 position: 6
14) printList()
15)  insertAt()   data: 0 position: 0
16)  printList()
17)  removeFront()
18)  printList()
19)  removeRear()
20)  printList()
21)  removeRear()
22)  printList()
23)  insertAt()   data: 8 position: 2
24)  printList()
25)  removeAt()   position: 2
26)  printList()
27)  printMirror()
```

Practical Exercise: [4_ Statistics]

Problem Statement

Create a program to help a statistician customer to automate some of his work. Implement the following required operations of LIST and STATISTICS with the given data structure and function prototypes:

```
#ifndef STATISTICS_H
#define STATISTICS_H

#define TRUE 1
#define FALSE 0

typedef bool Boolean;
typedef struct node *nodePtr;

struct node {
     int item;
     nodePtr next;
};
typedef nodePtr Statistician, DataStore;

Statistician newStatistician();
void destroyStatistician(Statistician *s);
void add(Statistician s, int x);
void remove(Statistician s, int x);
void displayData(Statistician s);
Boolean isEmpty(Statistician s);

int minimum(Statistician s);
int maximum(Statistician s);
int range(Statistician s);
float mean(Statistician s);
float median(Statistician s);
DataStore mode(Statistician s);
float variance(Statistician s);
float standardDeviation(Statistician s);

#endif
```