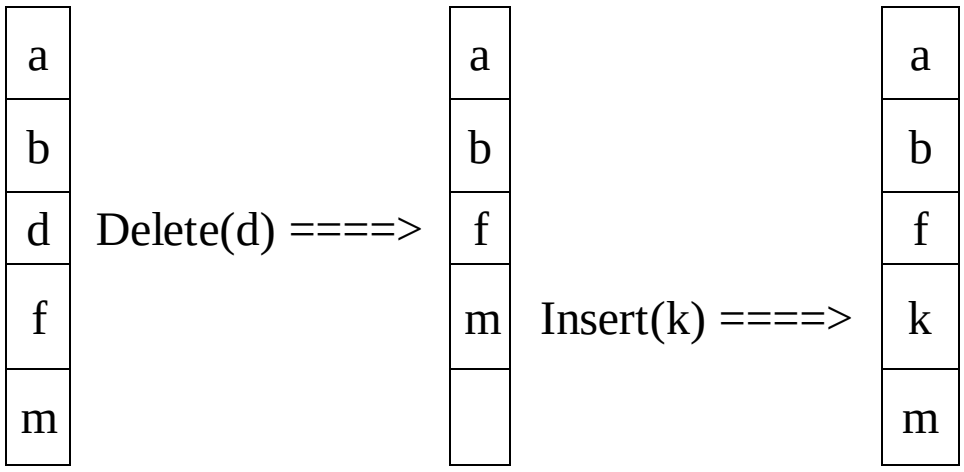
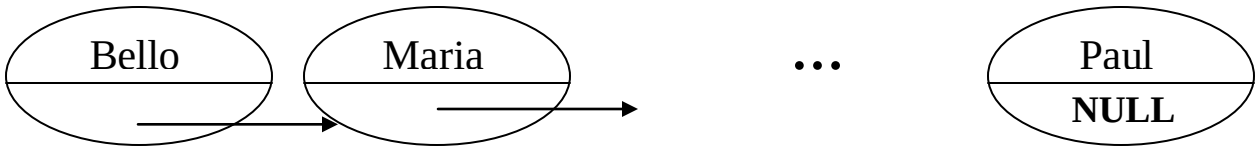
Linked Lists Structures

- Motivation:
  - Insertion & Deletion in an ordered array require moving elements up or down (left or right).
  - Manipulating ordered lists of varying sizes
- Example:



- Solution:
  - Use linked lists
  - creation of variable's storage space in memory during the execution (rather than compilation) of the program
  - Each data (a set of data) is associated with an address: form a node.

- Linked Lists:
  - A linked list consists of a number of nodes, each of which has a reference to the next link.



  - Adding and removing a node in the middle of a linked list is efficient.
  - Visiting the nodes of a linked list in sequential order is efficient
  - Random access is not efficient
  - Operations:
    - **Insert:**
      - insertFirst(Object obj)
      - insertLast(Object obj)
    - **Delete:**
      - removeFirst()
      - removeLast()
    - **Find:**
      - findNode()

- Implementation:
    - Use memory heap
    - Allocation of variable's storage space: needs a variable that holds the address of that space.
        - Java: An instance of a given class.
        - Pascal: pointer variable
        - Ada: Access variable
        - C: For any type in C, there is a corresponding type pointer-to-T: The value of the pointer is an address of memory.

- Requirements:
    - Mechanism to define the structure of a node (all fields)
    - A way to create a node
    - A way to free no longer needed nodes in some languages such as C.

- Declaration of a node: Java

```
private int data;
private Node next;
public Node(){
        data = 0;
        next = null;
}
```

```
        public Node(int d, Node l){
            data = d;
            next = l;
        }
        public int getData(){
            return(data);
        }
        public Node getNext(){
            return(next);
        }
}
```

- **Node p = new Node(1, null);**

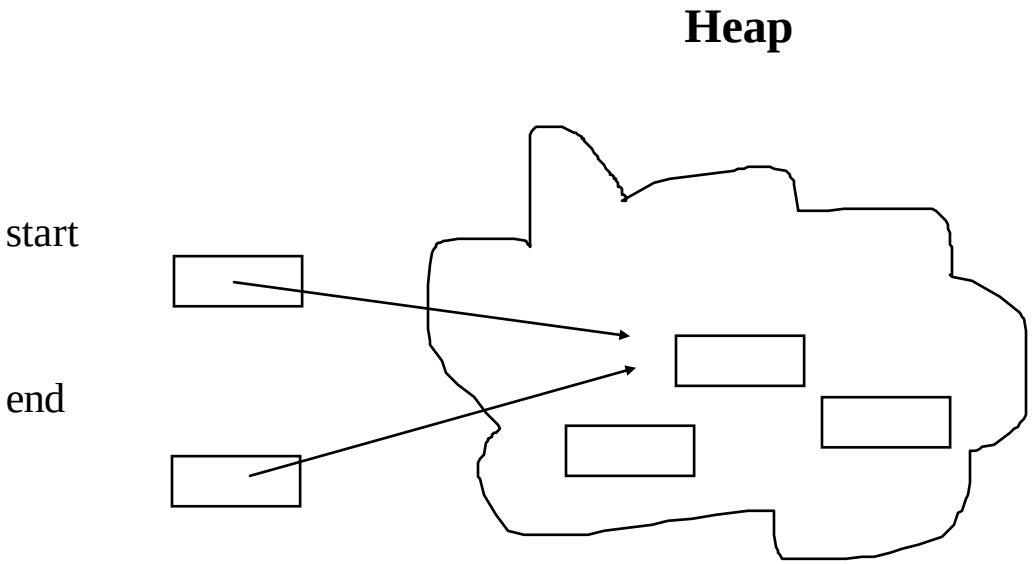
- **Constant**:

  - the pointer does not point to any node
  - Node first = nil or null;

- **Operations on pointers:**

  - comparison: start == end, etc.
  - Assignments: start = end, etc.
  - Arithmetic: depending on the language

- **Creation of a node:**
  - Primitive procedure: malloc, new, etc.

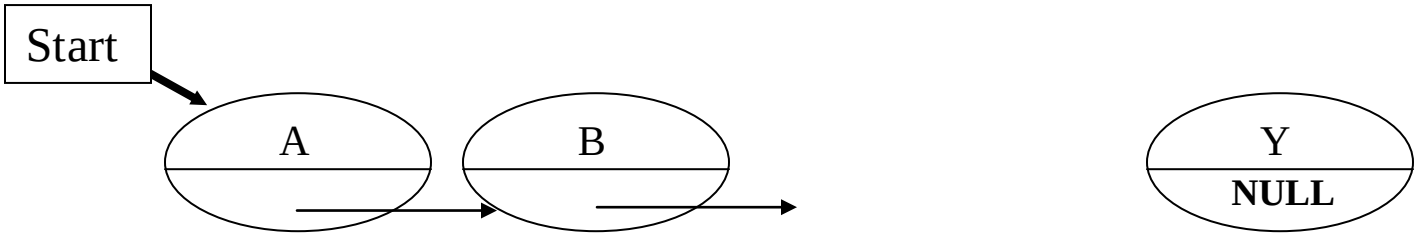


- **Disposition of a node:**
  - Primitive procedure: free(p), dispose(p), etc.
  - Java: done my JVM

- **Singly Linked Lists**
  - One-way linked lists
  - One-way linked lists with head and tail
  - Circular one-way linked lists

- **One-way linked lists**

  - **An example:**



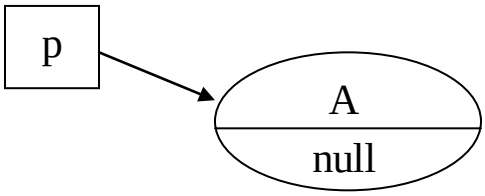
  - **Operations**: Sorted or Unsorted

    - Insertion
    - Deletion
    - Find

  - **Insertion in an unsorted list:**

    - Creation of a node:

      Node p = new Node(A,null);



    - Case 1: When the list is not empty; start != null.

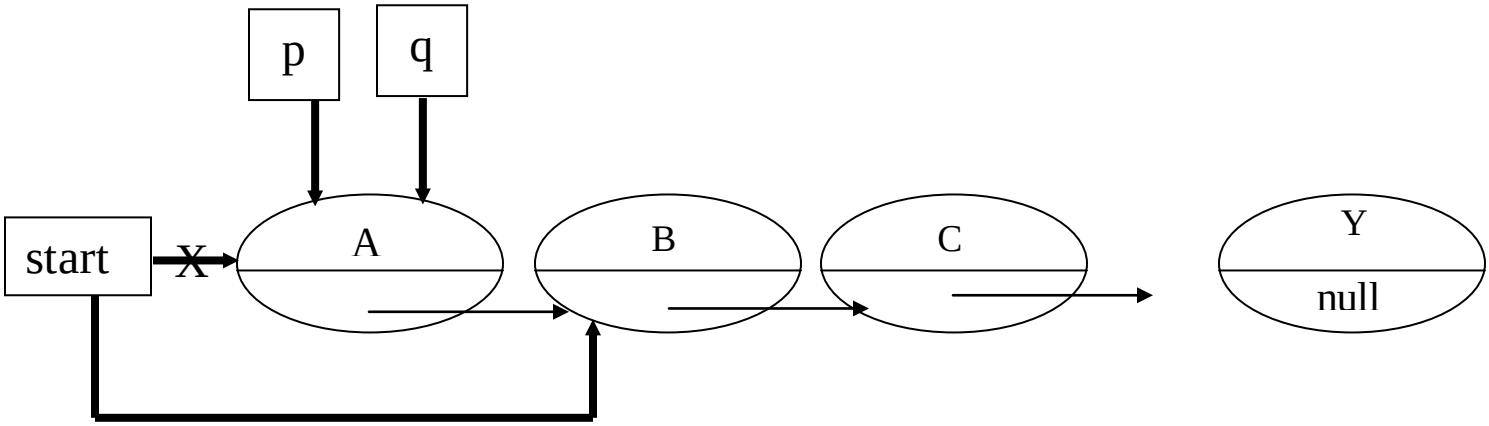
      p.next = start;
      start = p;

    - Case 2:When the list is empty; start=null.

      start = p;

- **Deletion** :

  - Case 1: The first element of the list

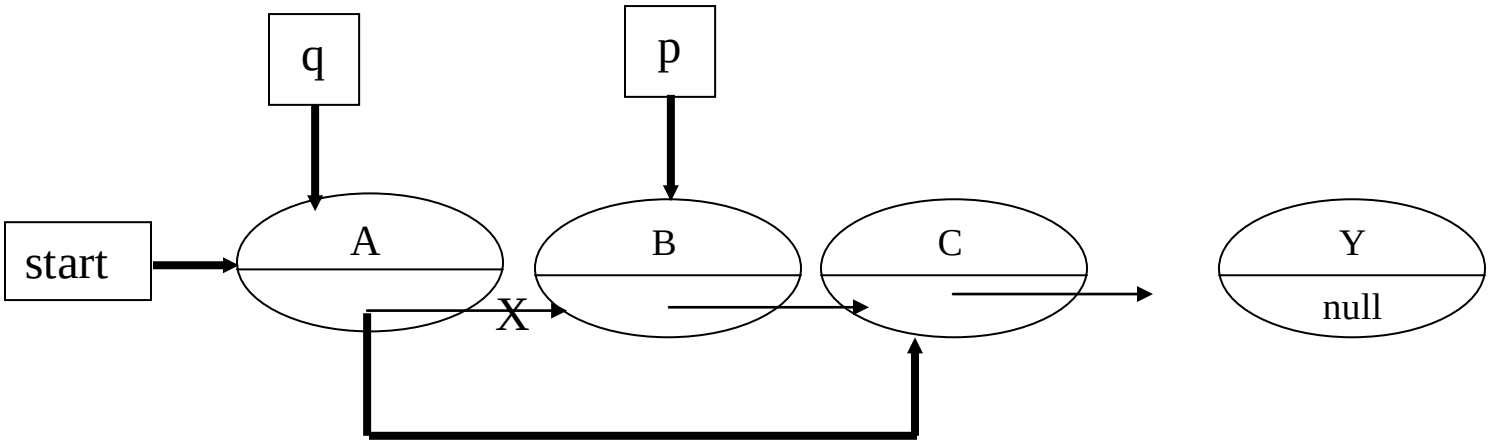


Delete(A):

p = q = start

start = start.next;

  - Case 2: An element different than the first one in the list
    Delete the node pointed by p

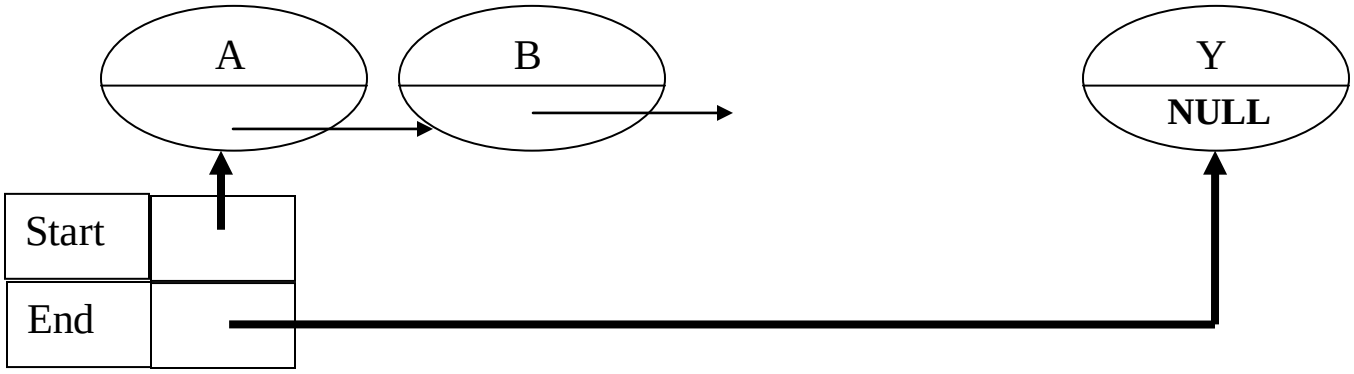


q.next = p.next;

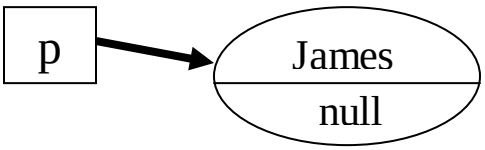
// You would need to free the pointer p if the language does not automatically free memory like in C: free(p);

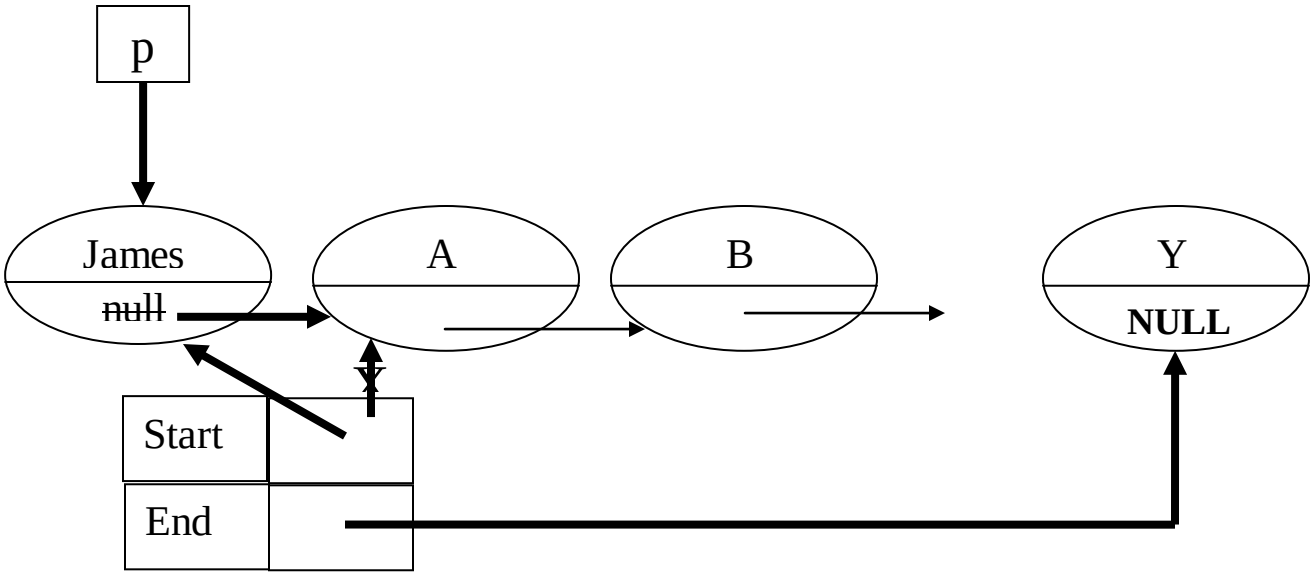
//In Java, JVM takes care of garbage collection.

- **One-way linked lists with Head and Tail**
    - **An example:**



- **Operations**: Sorted or Unsorted
    - Insertion
    - Deletion
    - Find
- **Insertion in an unsorted list:**
    - Creation of a node:



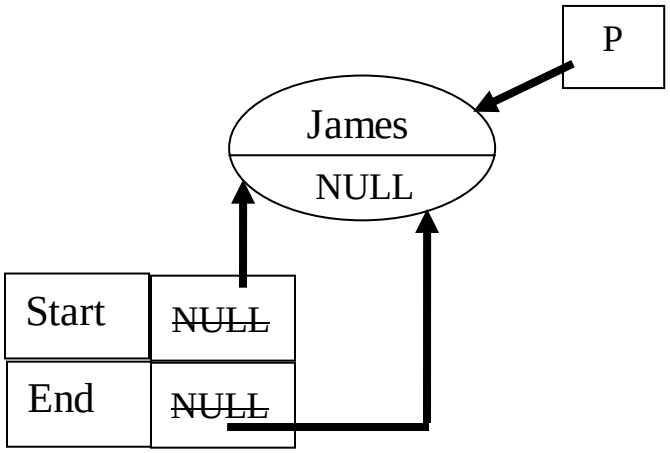


- Case 1: When the list is not empty; start != null and End != null;

  p.next = start;
  start = p;

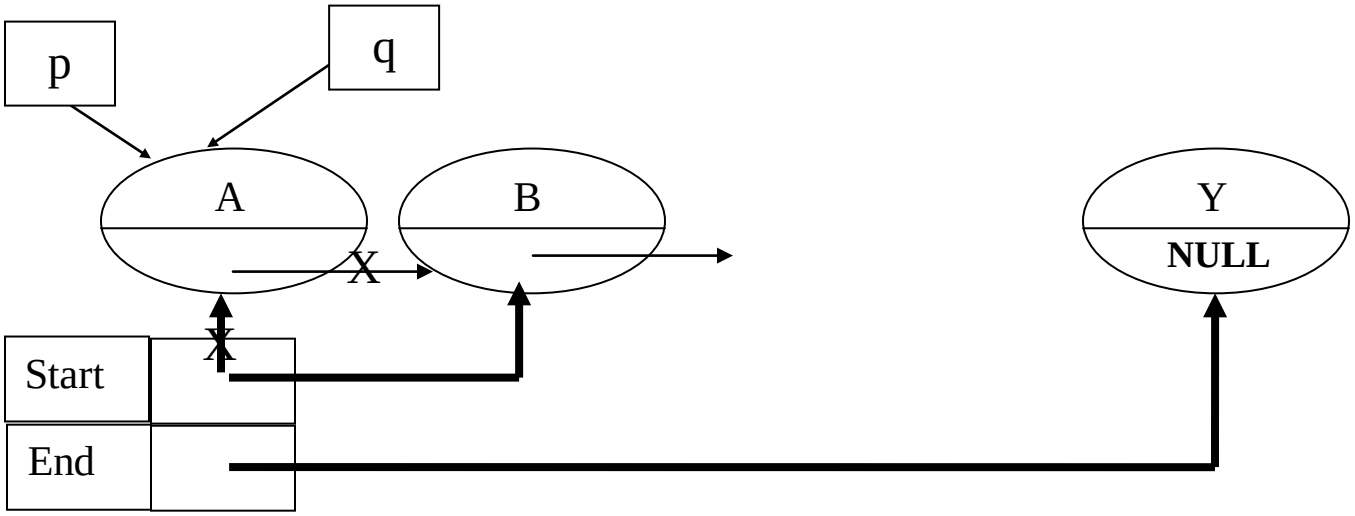
- Case 2:When the list is empty; start=null.

  start = p;
  End = p;



- **Deletion** :

  - Case 1: The first element of the list



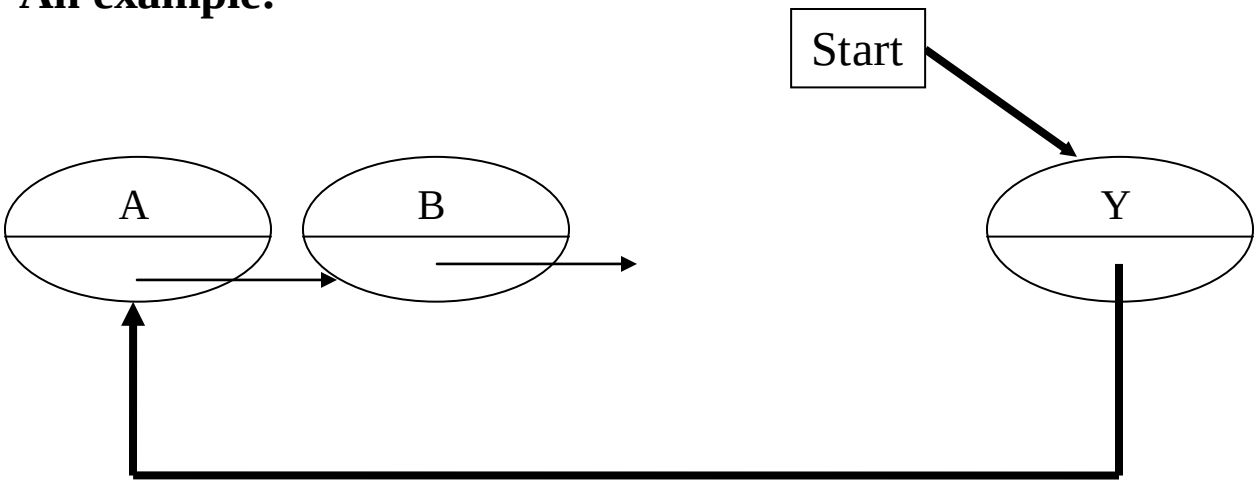
Delete(A):

```
p = q = Start
Start = Start.next;
// You would need to free the pointer p if the language does
not automatically free memory like in C: free(p);
//In Java, JVM takes care of garbage collection.
```

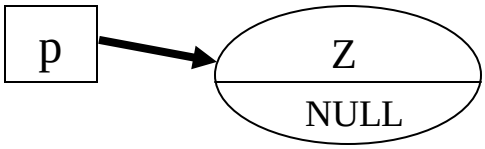
  - Case 2: An element different than the first one in the list

Same as in the case of one-way linked list.

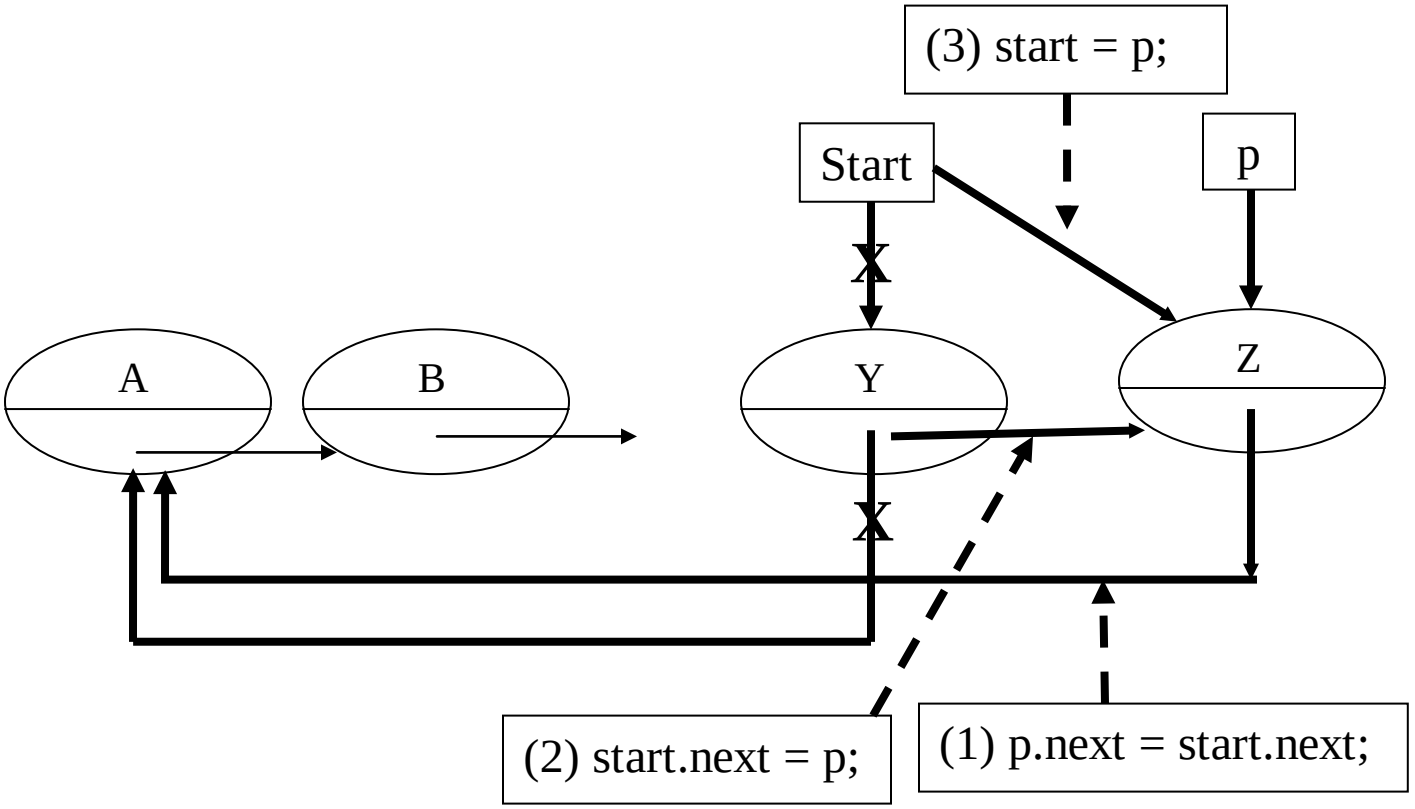
- **Singly Circular Linked Lists**
  - **An example:**



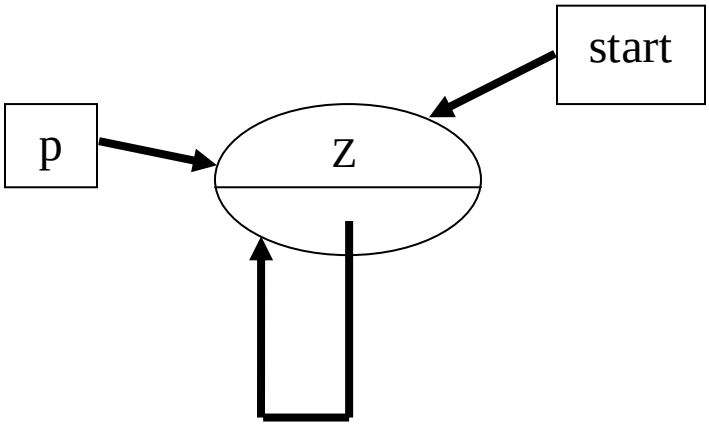
  - **Operations**: Sorted or Unsorted
    - Insertion
    - Deletion
    - Find
  - **Insertion in an unsorted list:**
    - Creation of a node:



- Case 1: When the list is not empty; start != null;



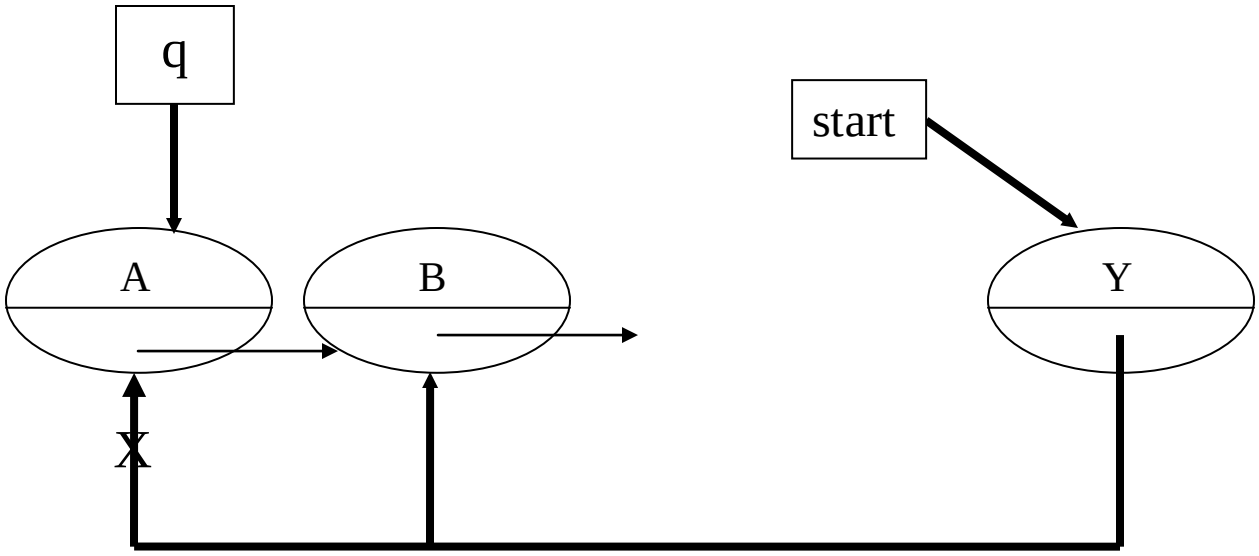
- Case 2:When the list is empty; start=null.



start = p;
p.next = p;

- **Deletion** :

  - Case 1: The first element of the list



Delete(A):

```
q = start.next;
start.next = q.next;
// You would need to free the pointer p if the language does
not automatically free memory like in C: free(p);
//In Java, JVM takes care of garbage collection.
```

  - Case 2: The last element of the list

    Try this case on your own.

  - Case 3: An element different from the first and last element of the list.

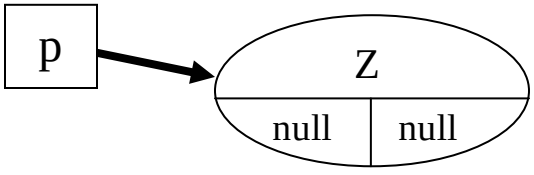
    Same as in the case of one-way linked list.

- **Doubly Linked Lists**

  - Type of nodes: Declaration of a node:

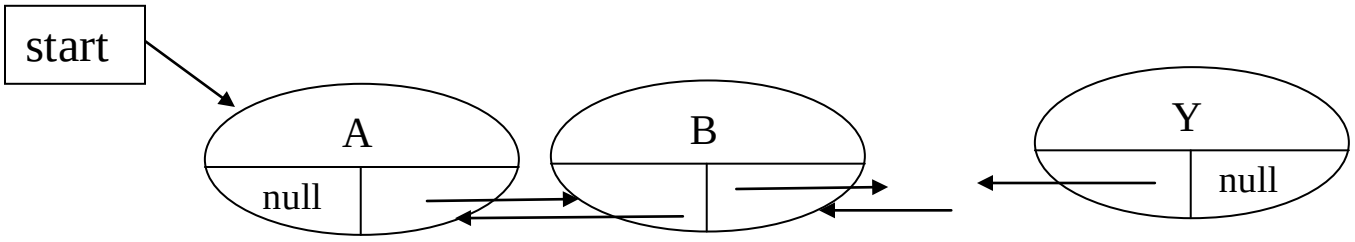
```
        private int data;
        private Node Rnext;
        private Node Lnext;

        public Node(){
              data = 0;
              Lnext = null;
              Rnext = null;
        }
        public Node(int d, Node l, Node r){
              data = d;
              Rnext = l;
              Lnext = r;
        }
        public int getData(){
              return(data);
        }
        public Node getRnext(){
              return(Rnext);
        }
        public Node getLnext(){
              return(Lnext);
        }
}
```

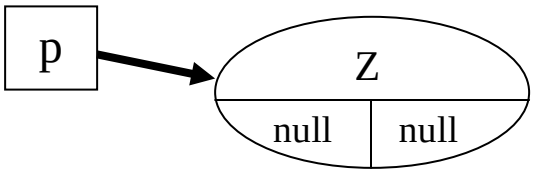


- Types:
  - Simple doubly linked lists
  - Circular doubly linked lists

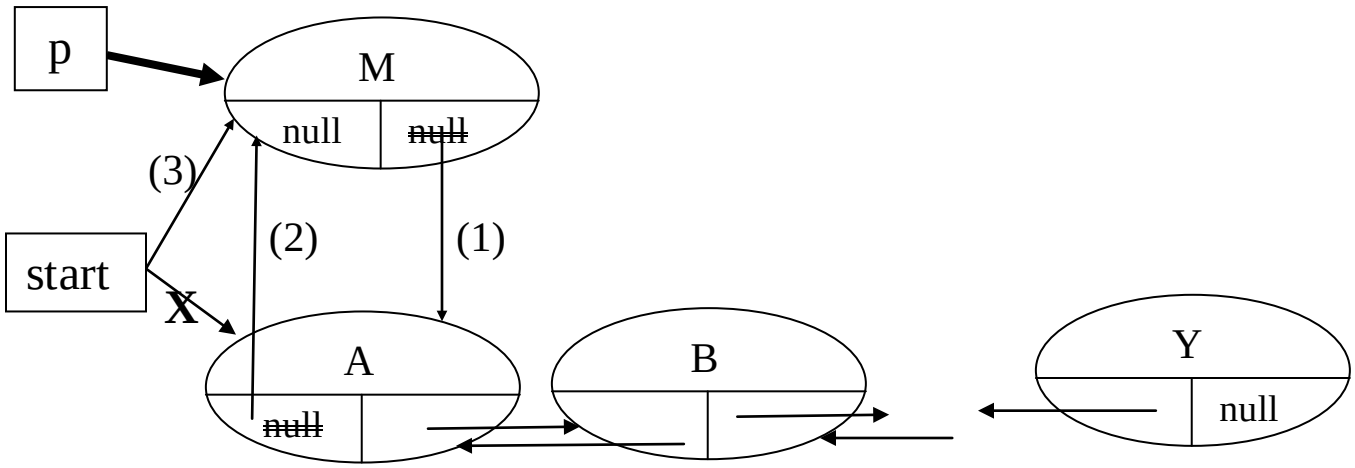
- **Simple doubly linked lists**
  - **An example:**



  - **Operations**: Sorted or Unsorted
    - Insertion
    - Deletion
    - Find
  - **Insertion in an unsorted list:**
    - Creation of a node:



- Case 1: When the list is not empty; start != null;



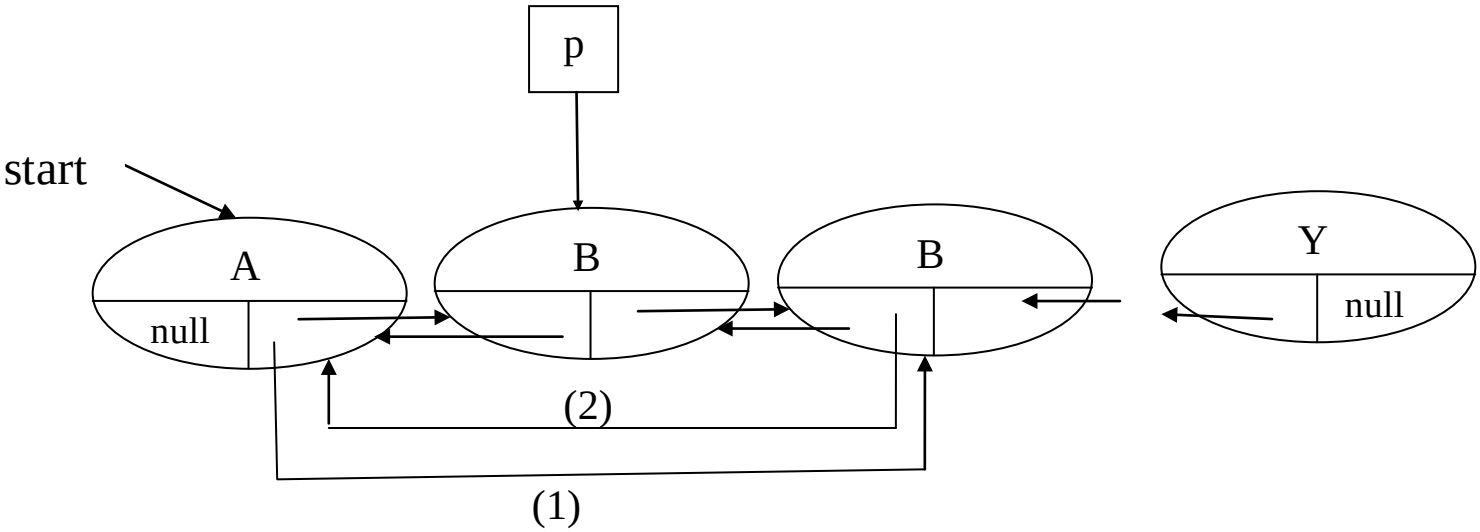
(1) p.Rnext = start;
(2) start.Lnext = p;
(3) start = p;

- Case 2: When the list is empty; start=null.

  start = p;

o **Deletion** :

- Case 1: An element different from the first and last element of the list.



Delete(B):

(1) p.Lnext.Rnext = p.Rnext;
(2) p.Rlink.Lnext = p.Lnext;
// You would need to free the pointer p if the language does not automatically free memory like in C: free(p);
//In Java, JVM takes care of garbage collection.

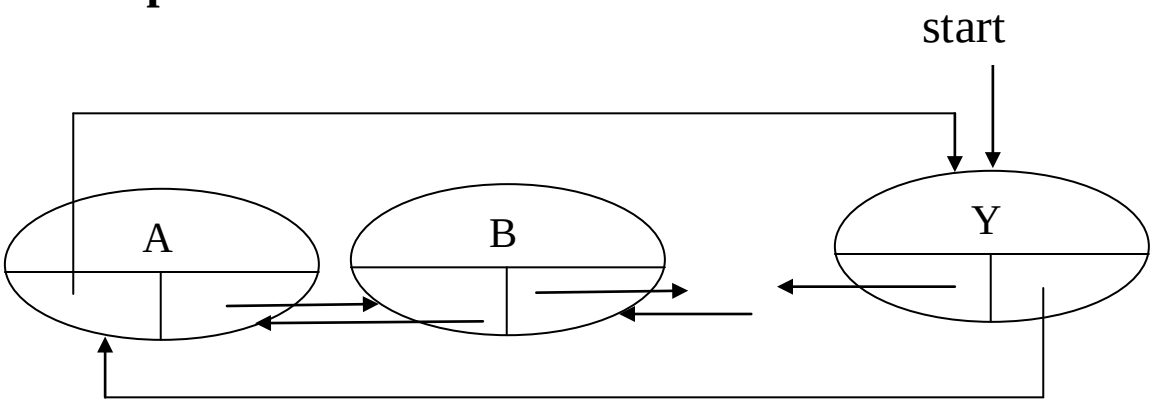
- Case 2: The first element of the list.

  Try this case on your own.

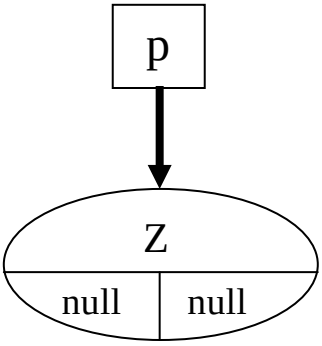
- Case 3: The last element of the list.

  Try this case on your own.

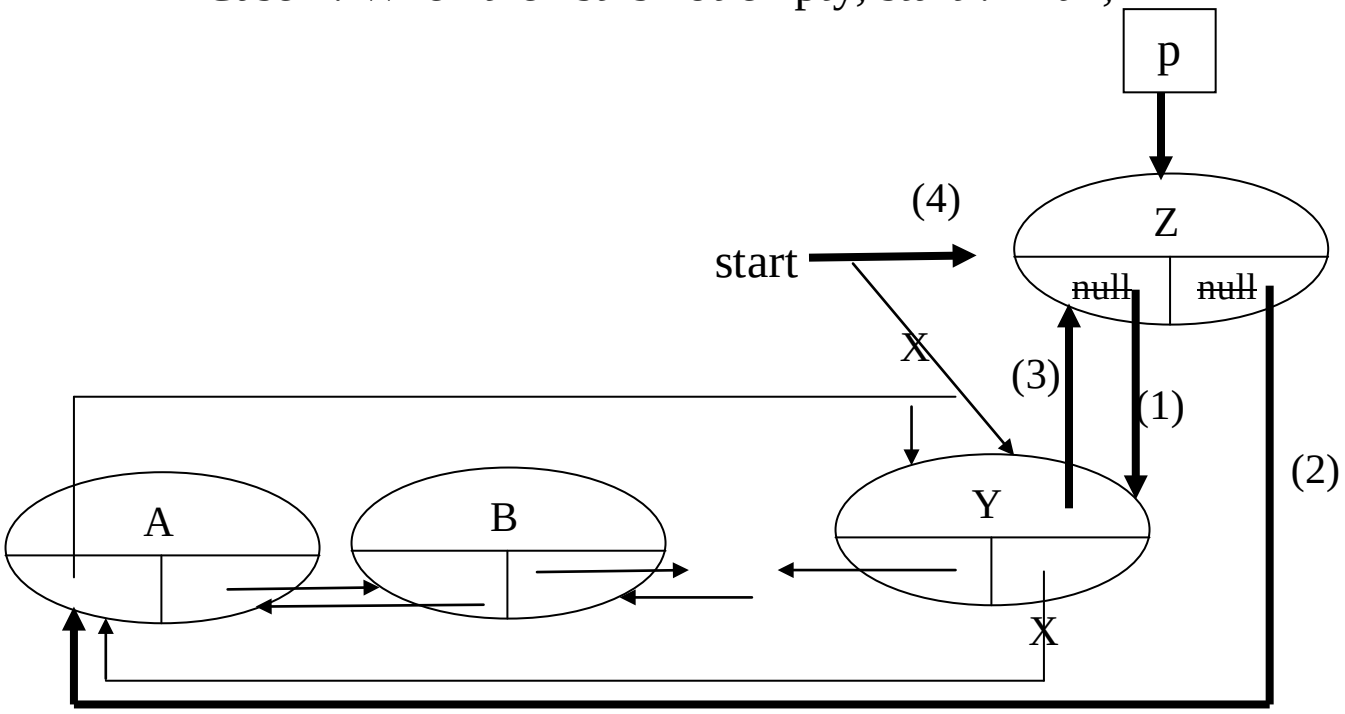
- **Circular doubly linked lists**
  - **An example:**



  - **Operations**: Sorted or Unsorted
    - Insertion
    - Deletion
    - Find
  - **Insertion in an unsorted list:**
    - Creation of a node:

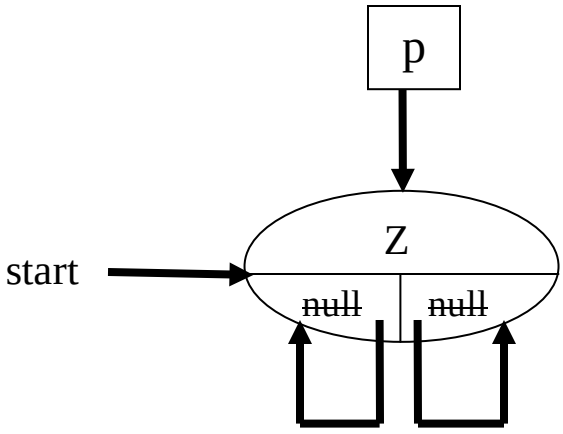


- Case 1: When the list is not empty; start != null;



(1) p.Lnext = start;
(2) p.Rnext = start.Rnext
(3) start.Rnext = p;
(4) start = p;

- Case 2: When the list is empty; start=null.



- **Deletion** :

  - Case 1: An element different from the first and last element of the list.

  - Case 2: The first element of the list.

  - Case 3: The last element of the list.

- Lab Assignment:
  - Given the following Java package for a singly linked list, complete the implementation of the missing methods.

```
package List;

public class Node {
	private int data;
	private Node next;
	public Node(){
		data = 0;
		next = null;
	}
	public Node(int d, Node l){
		data = d;
		next = l;
	}
	public int getData(){
		return(data);
	}
	public Node getNext(){
		return(next);
	}
}
```

```
package List;

import java.util.NoSuchElementException;

public class OneWayLinkedList {
	private Node Start  = null;

	public OneWayLinkedList(){
		Start = null;
	}
	// Returns true if the list is empty
	public boolean isEmpty(){
		return Start == null;
	}
	// Inserts a new node at the beginning of this list.
	public void addFirst(int element){
		Start = new Node(element, Start);
	}
	//  Returns the first element in the list.
	public int getFirst(){
		if(Start == null) throw new NoSuchElementException();
			return Start.getData();
	}
	//-----------------------------------------------------------------
	// Removes the first element in the list.
	public boolean deleteFirst(){
		// Implement this function
	}
	// Find whether and element is in the list.
	public boolean findData(int d){
		//Implement this function
	}
	//-----------------------------------------------------------------
```

```
	// Print list.
	public void printList(){
		if (Start == null)
			System.out.println("Your list is empty)");
		else {
			Node move = Start;
			System.out.println("------- Print Your List ------- ");
			while (move != null) {
				System.out.print("-->" + move.getData());
				move = move.getNext();
			}
		}
	}


}
```