Job Sheet 12 Double Linked List

From:

AL AZHAR RIZQI RIFA'I FIRDAUS

Class:

1 I

Absence:

01

Student Number Identity:

2241720263

Department:

Information Technology

Study Program:

Informatics Engineering

Practicum 1

Code :

- Main

```java
package practicum1;

public class DoubleLinkedListMain {
    public static void main(String[] args) throws Exception {
        DoubleLinkedList dll = new DoubleLinkedList();
        dll.print();
        System.out.println();
        System.out.println("Size : " + dll.size());
        System.out.println("========================");
        dll.addFirst(3);
        dll.addLast(4);
        dll.addFirst(7);
        dll.print();
        System.out.println("Size : " + dll.size());
        System.out.println("========================");
        dll.add(40, 1);
        dll.print();
        System.out.println("Size : " + dll.size());
        System.out.println("========================");
        dll.clear();
        dll.print();
        System.out.println("Size : " + dll.size());
    }
}

```

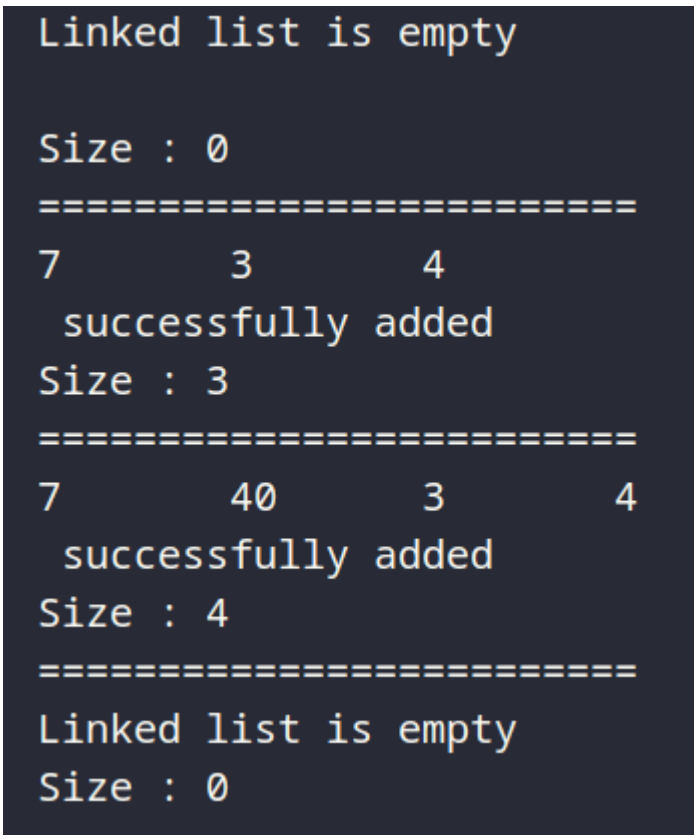
- Double Linked List

```java
package practicum1;

public class DoubleLinkedList {
    public Node head;
    public int size;

    public DoubleLinkedList() {
        head = null;
        size = 0;
    }

    public boolean isEmpty() {
        return head == null;
    }

    public void addFirst(int item) {
        if (isEmpty()) {
            head = new Node(null, item, head);
        } else {
            Node newNode = new Node(null, item, head);
            head.prev = newNode;
            head = newNode;
        }
        size++;
    }

    public void addLast(int item) {
         if (isEmpty()) {
            addFirst(item);
         } else {
            Node current = head;
            while (current.next != null) {
                current = current.next;
            }
            Node newNode = new Node(current, item, null);
            current.next = newNode;
            size++;
         }
    }

    public void add(int item, int index) throws Exception {
        if (isEmpty()) {
            addFirst(item);
        } else if (index < 0 || index > size ) {
            throw new Exception("Index out of bound");
        } else {
            Node current = head;
            int i = 0;
            while (i < index) {
                current = current.next;
                i++;
            }
            if (current.next == null) {
                Node newNode = new Node(null, item, current);
                current.prev = newNode;
                head = newNode;
            } else {
                Node newNode = new Node(current.prev, item, current);
                newNode.prev = current.prev;
                newNode.next = current;
                current.prev.next = newNode;
                current.prev = newNode;
            }
        }
        size++;
    }

    public int size() {
        return size;
    }

    public void clear() {
        head = null;
        size = 0;
    }

    public void print() {
        if (!isEmpty()) {
            Node tmp = head;
            while (tmp != null) {
                System.out.print(tmp.data + "\t");
                tmp = tmp.next;
            }
            System.out.println("\n successfully added");
        } else {
            System.out.println("Linked list is empty");
        }
    }
}

```

- Node

```java
package practicum1;

public class Node {
    public int data;
    Node prev, next;

    public Node(Node prev, int data, Node next) {
        this.prev = prev;
        this.next = next;
        this.data = data;
    }
}

```

- Result



Questions

1. What's the difference between single linked list and double linked list?

The main difference between a single linked list and a double linked list lies in the way nodes are connected.

In a single linked list:

- **Each node contains a reference (usually called next) to the next node in the list.**
- **Traversal can only be done in one direction, starting from the head and moving towards the tail.**
- **Removing or modifying a node requires traversing the list until the desired node is reached.**

In a double linked list:

- **Each node contains two references: one to the previous node (usually called prev) and one to the next node (usually called next).**
- **Traversal can be done in both directions: forward (from head to tail) and backward (from tail to head).**
- **Removing or modifying a node can be done more efficiently because the prev reference allows direct access to the previous node.**

2. In Node class, what is the usage of attribute next and prev ?

The next and prev attributes in the Node class are used to establish the connections between nodes in a double linked list.

3. In constructor of DoubleLinkedList class. What's the purpose of head and size attribute in this following code?

```
public DoubleLinkedLists() {
    head = null;
    size = 0;
}
```

In the constructor of the DoubleLinkedList class, the head and size attributes are initialized.

- **head: The head attribute represents the first node or the starting point of the double linked list. It keeps track of the first node in the list. In the constructor, head is set to null initially because the list is empty when it is created.**
- **size: The size attribute keeps track of the number of nodes in the double linked list. It represents the current size or length of the list. In the constructor, size is set to 0 because the list is empty when it is created.**

4. In method addFirst(), why do we initialize the value of Node object to be null at first?

Node newNode = new Node(null, item, head);

The reason for initializing the prev reference of the newNode object to null is because the new node will become the first node in the list. Since there is no node before it, the prev reference is set to null to indicate that there is no previous node.

5.In method addLast(), what's the purpose of creating a node object by passing the prev parameter with current and next with null ?

Node newNode = new Node(current, item, null);

- **prev parameter: By passing current as the prev parameter, it establishes the current last node in the list (current) as the previous node of the new node. This connects the new node to the existing last node, ensuring the continuity of the double linked list.**
- **next parameter: By setting null as the next parameter, it indicates that the new node does not have any node after it. Since the new node is being added at the end of the list, there won't be any node after it. Therefore, the next reference of the new node is set to null.**

Practicum 2

Code :

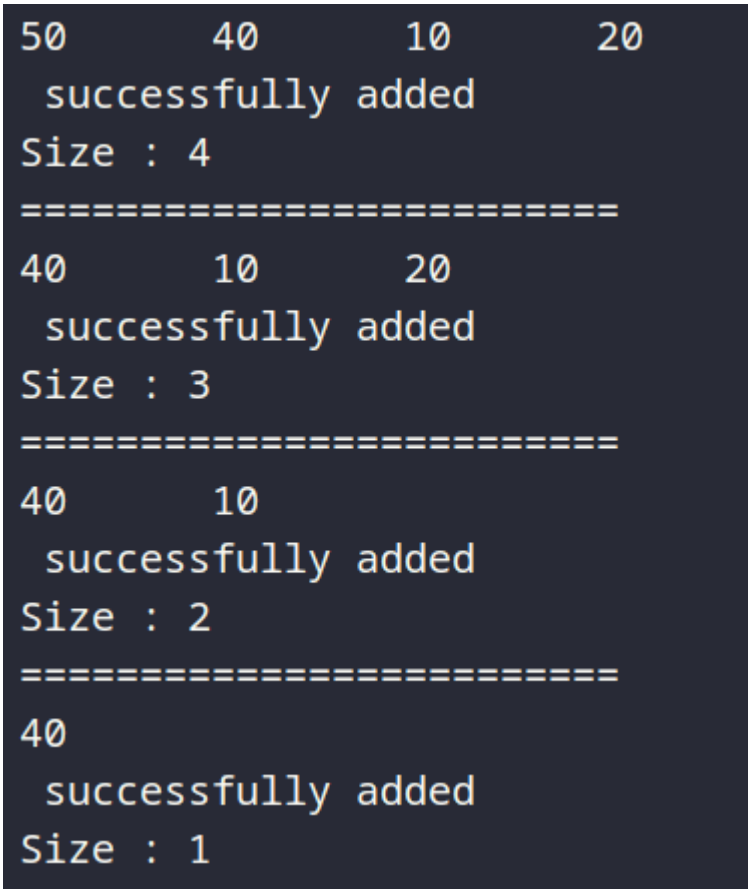
- Double Linked List

```java
// add practicum 2
public void removeFirst() throws Exception {
    if (isEmpty()) {
        throw new Exception("Linked list is still empty, cannot remove");
    } else if (size == 1) {
        removeLast();
    } else {
        head = head.next;
        head.prev = null;
        size--;
    }
}

public void removeLast() throws Exception {
    if (isEmpty()) {
        throw new Exception("Linked list is still empty, cannot remove");
    } else if (head.next == null) {
        head = null;
        size--;
        return;
    }
    Node current = head;
    while (current.next.next != null) {
        current = current.next;
    }
    current.next = null;
    size--;
}

public void remove(int index) throws Exception {
    if (isEmpty() || index >= size) {
        throw new Exception("Index value is out of bound");
    } else if (size == 0) {
        removeFirst();
    } else {
        Node current = head;
        int i = 0;
        while (i < index) {
            current = current.next;
            i++;
        }
        if (current.next == null) {
            current.prev.next = null;
        } else if (current.prev == null) {
            current = current.next;
            current.prev = null;
            head = current;
        } else {
            current.prev.next = current.next;
            current.next.prev = current.prev;
        }
        size--;
    }
}
```

- Main

```java
// add practicum 2
dll.addLast(50);
dll.addLast(40);
dll.addLast(10);
dll.addLast(20);
dll.print();
System.out.println("Size : " + dll.size());
System.out.println("=========================");
dll.removeFirst();
dll.print();
System.out.println("Size : " + dll.size());
System.out.println("=========================");
dll.removeLast();
dll.print();
System.out.println("Size : " + dll.size());
System.out.println("=========================");
dll.remove(1);
dll.print();
System.out.println("Size : " + dll.size());
```

- Result



Questions

1. What's the meaning of these statements in removeFirst() method?

- **if (isEmpty()): This condition checks if the linked list is empty. If it is empty, it means there are no nodes in the list. In such a case, an exception is thrown with the message "Linked list is still empty, cannot remove."**
- **else if (size == 1): This condition checks if there is only one node in the list. If there is only one node, it means that node is both the first and the last node. In this case, the removeLast() method is called to remove the single node from the list.**
- **else: If the above conditions are not met, it means there are multiple nodes in the list. In this case, the following steps are executed:**
  - **head = head.next;: The head reference is updated to point to the second node in the list, effectively removing the first node.**
  - **head.prev = null;: The prev reference of the new first node (previously the second node) is set to null, indicating that there is no previous node before it.**
  - **size--;: The size of the list is decremented by 1 to reflect the removal of the first node.**

2. How do we detect the position of the data that are in the last index in method removeLast()?

In the removeLast() method, to detect the position of the data that is in the last index, the following steps are performed:

- **If the linked list is empty, an exception is thrown because there are no elements to remove.**
- **If there is only one node in the linked list (i.e., head.next == null), the head is set to null, indicating an empty list, and the size is decremented by 1. This means that the data at the last index (which is also the first and only index) is removed.**
- **If there are multiple nodes in the linked list, the following steps are executed:**
    - **A reference variable current is assigned the value of the head node.**
    - **A while loop is used to traverse the linked list until the second-to-last node (i.e., current.next.next) is reached. This is done by checking if current.next.next is not null.**
    - **Inside the loop, the current reference is updated to point to the next node in each iteration until the second-to-last node is reached.**
    - **After the loop, the last node (current.next) is set to null, effectively removing it from the list.**
    - **The size of the linked list is decremented by 1 to reflect the removal of the last node.**

By traversing the linked list until the second-to-last node, the position of the data that is in the last index can be detected and subsequently removed from the list.

3. Explain why this program code is not suitable if we include it in remove command!

```
Node tmp = head.next;

head.next=tmp.next;
tmp.next.prev=head;
```

- **The provided code snippet is not suitable if we include it in the remove command because it skips a crucial step in unlinking the node to be removed from the linked list. The code assumes that the node to be removed is the second node (head.next) in the linked list. It attempts to remove this node by updating the references of head.next and tmp.next.prev. However, this code snippet overlooks the proper handling of the previous node's reference.**

4. Explain what's the function of this program code in method remove!

```
current.prev.next = current.next;
current.next.prev = current.prev;
```

- **current.prev.next = current.next;: This code sets the next reference of the previous node (current.prev) to point directly to the next node (current.next). By doing this, the node to be removed (current) is bypassed in the linked list, and the previous node is connected directly to the next node, skipping the node to be removed.**
- **current.next.prev = current.prev;: This code sets the prev reference of the next node (current.next) to point directly to the previous node (current.prev). By doing this, the next node is connected directly to the previous node, bypassing the node to be removed.**

Practicum 3

Code :

- Double Linked List

```java
// add practicum 3
public int getFirst() throws Exception {
    if (isEmpty()) {
        throw new Exception("Linked list still empty");
    }
    return head.data;
}

public int getLast(int index) throws Exception {
    if (isEmpty()) {
        throw new Exception("Linked list still empty");
    }
    Node tmp = head;
    while (tmp.next != null) {
        tmp = tmp.next;
    }
    return tmp.data;
}

public int get(int index) throws Exception {
    if (isEmpty()) {
        throw new Exception("Linked list still empty");
    }
    Node tmp = head;
    for (int i = 0; i < index; i++) {
        tmp = tmp.next;
    }
    return tmp.data;
}
```

- Main

```java
// add practicum 3
dll.print();
System.out.println("Size : " + dll.size());
System.out.println("=========================");
dll.addFirst(3);
dll.addLast(4);
dll.addFirst(7);
dll.print();
System.out.println("Size : " + dll.size());
System.out.println("=========================");

dll.add(40, 1);
dll.print();

System.out.println("Size : " + dll.size());
System.out.println("=========================");
System.out.println("Data in the head of linked list is : " + dll.getFirst());
System.out.println("Data in the tail of linked list is : " + dll.getLast(0));
System.out.println("Data in the 1st index  linked list is : " + dll.get(1));
```

- Result

```
Linked list is empty
Size : 0
============================
7       3       4
 successfully added
Size : 3
============================
7       40      3       4
 successfully added
Size : 4
============================
Data in the head of linked list is : 7
Data in the tail of linked list is : 4
Data in the 1st index  linked list is : 40
```

Questions

1. What is the function of method size() in DoubleLinkedList class ?

- **The size() method in the DoubleLinkedList class serves the purpose of returning the current size or number of elements in the double-linked list.**

2. How do we set the index in double linked list so that it starts from 1st index instead of 0th index?

Modify this code with change 0 value to 1

```java
public void add(int item, int index) throws Exception {
    if (isEmpty()) {
        addFirst(item);
    } else if (index < 1 || index > size ) {
        throw new Exception("Index out of bound");
    } else if (index == 1) {
        addFirst(item);
    } else {
        Node current = head;
        int i = 1;
        while (i < index) {
            current = current.next;
            i++;
        }
        if (current.next == null) {
            Node newNode = new Node(null, item, current);
            current.prev = newNode;
            head = newNode;
        } else {
            Node newNode = new Node(current.prev, item, current);
            newNode.prev = current.prev;
            newNode.next = current;
            current.prev.next = newNode;
            current.prev = newNode;
        }
    }
    size++;
}
```

```java
public void remove(int index) throws Exception {
    if (isEmpty() || index >= size) {
        throw new Exception("Index value is out of bound");
    } else if (index == 1) {
        removeFirst();
    } else {
        Node current = head;
        int i = 1;
        while (i < index) {
            current = current.next;
            i++;
        }
        if (current.next == null) {
            current.prev.next = null;
        } else if (current.prev == null) {
            current = current.next;
            current.prev = null;
            head = current;
        } else {
            current.prev.next = current.next;
            current.next.prev = current.prev;
        }
        size--;
    }
}
```

```java
public int get(int index) throws Exception {
    if (isEmpty()) {
        throw new Exception("Linked list still empty");
    } else if (index < 1 || index > size) {
        throw new Exception("Linked list still empty");
    }
    Node tmp = head;
    for (int i = 1; i < index; i++) {
        tmp = tmp.next;
    }
    return tmp.data;
}
```

3. Please explain the difference between method Add() in double linked list and single

linked list !

The add() method in a double linked list and a single linked list differs in how they handle inserting a new node into the list.

- **In a single linked list, the add() method typically adds a new node at the specified position by adjusting the pointers of the previous node and the new node. This involves updating the next pointer of the previous node to point to the new node, and updating the next pointer of the new node to point to the node that was previously at that position.**
- **On the other hand, in a double linked list, the add() method needs to handle the pointers of both the previous node and the next node. It creates a new node with appropriate connections to the previous and next nodes, and updates the next pointer of the previous node to point to the new node, as well as the prev pointer of the next node to point to the new node. This ensures that the new node is inserted correctly between the previous and next nodes.**

4. What's the logic difference of these 2 following codes?

- a

```
public boolean isEmpty(){
    if(size ==0){
        return true;
    } else{
        return false;
    }
}
```

- b

```
public boolean isEmpty(){
    return head == null;
}
```

Both code snippets serve the same purpose of checking whether a linked list is empty or not. However, there is a logic difference in how they determine the emptiness of the linked list.

- **Code snippet "a" checks the emptiness based on the size of the linked list. If the size is equal to 0, it means the list is empty, so it returns true. Otherwise, if the size is greater than 0, it means the list is not empty, so it returns false.**
- **On the other hand, code snippet "b" checks the emptiness based on the head node of the linked list. If the head node is null, it means the list is empty, so it returns true. Otherwise, if the head node is not null, it means the list is not empty, so it returns false.**

Assignment

1. Create a program with double linked list implementation that allows user to choose a menu as following image! The searching uses sequential search approach and the program should be able to sort the data in descending order. You may any choose sorting approach you prefer (bubble sort, selection sort, insertion sort, or merge sort)

Adding a data

```
run:
==========================================
Data manipulation with Double Linked List
==========================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
==========================================
1
Insert data in Head postiion
34
```

Add data in specified index and display the result

```
run:
==========================================
Data manipulation with Double Linked List
==========================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
==========================================
3
Insert Data
Data node : 66
In index : 1
```

```
==========================================
Data manipulation with Double Linked List
==========================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
==========================================
7
Print data
88
 66
 32
 34
 23
 67
 44
 90
 99
```

Search Data

```
run:
==========================================
Data manipulation with Double Linked List
==========================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
==========================================
8
Search data : 67
Data 67 is in index-6
```

Sorting Data

```
==========================================
Data manipulation with Double Linked List
==========================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
==========================================
9
```

```
==========================================
Data manipulation with Double Linked List
==========================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
==========================================
7
Print data :
23
 32
 34
 44
 66
 67
 88
 90
 99
```

Code :

Double Linked List

```java
package assnum1;

public class DoubleLinkedList {
    public Node head;
    public int size;

    public DoubleLinkedList() {
        head = null;
        size = 0;
    }

    public boolean isEmpty() {
        return head == null;
    }

    // choose 1
    public void addFirst(int item) {
        if (isEmpty()) {
            head = new Node(null, item, head);
        } else {
            Node newNode = new Node(null, item, head);
            head.prev = newNode;
            head = newNode;
        }
        size++;
    }

    // choose 2
    public void addLast(int item) {
         if (isEmpty()) {
            addFirst(item);
         } else {
            Node current = head;
            while (current.next != null) {
                current = current.next;
            }
            Node newNode = new Node(current, item, null);
            current.next = newNode;
            size++;
         }
    }

    // choose 3
    public void add(int item, int index) throws Exception {
        if (isEmpty()) {
            addFirst(item);
        } else if (index < 0 || index > size ) {
            throw new Exception("Index out of bound");
        } else {
            Node current = head;
            int i = 0;
            while (i < index) {
                current = current.next;
                i++;
            }
            if (current.next == null) {
                Node newNode = new Node(null, item, current);
                current.prev = newNode;
                head = newNode;
            } else {
                Node newNode = new Node(current.prev, item, current);
                newNode.prev = current.prev;
                newNode.next = current;
                current.prev.next = newNode;
                current.prev = newNode;
            }
        }
        size++;
    }

    // choose 7
    public void print() {
        if (!isEmpty()) {
            Node tmp = head;
            while (tmp != null) {
                System.out.println(tmp.data);
                tmp = tmp.next;
            }
            // System.out.println("\n successfully added");
        } else {
            System.out.println("Linked list is empty");
        }
    }

    // add practicum 2
    // choose 4
    public void removeFirst() throws Exception {
        if (isEmpty()) {
            throw new Exception("Linked list is still empty, cannot remove");
        } else if (size == 1) {
            removeLast();
        } else {
            head = head.next;
            head.prev = null;
            size--;
        }
    }
```

```java
    // choose 5
    public void removeLast() throws Exception {
        if (isEmpty()) {
            throw new Exception("Linked list is still empty, cannot remove");
        } else if (head.next == null) {
            head = null;
            size--;
            return;
        }
        Node current = head;
        while (current.next.next != null) {
            current = current.next;
        }
        current.next = null;
        size--;
    }

    // choose 6
    public void remove(int index) throws Exception {
        if (isEmpty() || index >= size) {
            throw new Exception("Index value is out of bound");
        } else if (size == 0) {
            removeFirst();
        } else {
            Node current = head;
            int i = 0;
            while (i < index) {
                current = current.next;
                i++;
            }
            if (current.next == null) {
                current.prev.next = null;
            } else if (current.prev == null) {
                current = current.next;
                current.prev = null;
                head = current;
            } else {
                current.prev.next = current.next;
                current.next.prev = current.prev;
            }
            size--;
        }
    }

    // add practicum 3
    public int getFirst() throws Exception {
        if (isEmpty()) {
            throw new Exception("Linked list still empty");
        }
        return head.data;
    }

    public int getLast(int index) throws Exception {
        if (isEmpty()) {
            throw new Exception("Linked list still empty");
        }
        Node tmp = head;
        while (tmp.next != null) {
            tmp = tmp.next;
        }
        return tmp.data;
    }

    public int get(int index) throws Exception {
        if (isEmpty()) {
            throw new Exception("Linked list still empty");
        }
        Node tmp = head;
        for (int i = 0; i < index; i++) {
            tmp = tmp.next;
        }
        return tmp.data;
    }

    // choose 8
    public int sequentialSearch(int item) {
        Node current = head;
        for (int index = 0; current != null; index++) {
            if (current.data == item) {
                return index;
            }
            current = current.next;
        }

        return -1;
    }

    public void showPosition(int x, int pos) {
        if (pos != -1) {
            System.out.println("Data : " + x + " is found in index-" + pos);
        } else {
            System.out.println("Data : " + x + " is not found!");
        }
    }

    // choose 9
    public void bubbleSort() {
        if (head == null || head.next == null) {
            return; // No need to sort an empty list or a list with only one element
        }

        boolean swapped;
        Node current;

        for (int i = 0; i < size - 1; i++) {
            swapped = false;
            current = head;

            for (int j = 0; j < size - 1 - i; j++) {
                if (current.data < current.next.data) {
                    int temp = current.data;
                    current.data = current.next.data;
                    current.next.data = temp;
                    swapped = true;
                }
                current = current.next;
            }

            if (!swapped) {
                break;
            }
        }
    }
}
```

Node :

```java
package assnum1;

public class Node {
    public int data;
    Node prev, next;

    public Node(Node prev, int data, Node next) {
        this.prev = prev;
        this.next = next;
        this.data = data;
    }
}

```

Main :

```java
package assnum1;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) throws Exception{
        Scanner sc = new Scanner(System.in);
        DoubleLinkedList dll = new DoubleLinkedList();

        int choose;
        do {
            menu();
            choose = sc.nextInt();
            switch (choose) {
                case 1:
                    System.out.print("Insert data in head position : ");
                    int data1 = sc.nextInt();
                    dll.addFirst(data1);
                    break;
                case 2:
                    System.out.print("Insert data in tail position : ");
                    int data2 = sc.nextInt();
                    dll.addLast(data2);
                    break;
                case 3:
                    System.out.print("Insert data : ");
                    int data3 = sc.nextInt();
                    System.out.print("Insert index : ");
                    int index3 = sc.nextInt();
                    dll.add(data3, index3);
                    break;
                case 4:
                    dll.removeFirst();
                    break;
                case 5:
                    dll.removeLast();
                    break;
                case 6:
                    System.out.print("Insert index data : ");
                    int index6 = sc.nextInt();
                    dll.remove(index6);
                    break;
                case 7:
                    dll.print();
                    break;
                case 8:
                    System.out.print("Search data : ");
                    int data8 = sc.nextInt();
                    int position = dll.sequentialSearch(data8);
                    dll.showPosition(data8, position);
                    break;
                case 9:
                    dll.bubbleSort();
                    dll.print();
                    break;
                case 10:
                    System.exit(0);
                    break;

                default:
                    break;
            }

        } while (choose >= 1 && choose <= 10);

        sc.close();
    }

    public static void menu() {
        System.out.println("========================================");
        System.out.println("Data manipulation with doule linked list");
        System.out.println("========================================");
        System.out.println("Choose Menu : ");
        System.out.println("1. Add first");
        System.out.println("2. Add tail");
        System.out.println("3. Add data with index");
        System.out.println("4. remove first");
        System.out.println("5. Remove last");
        System.out.println("6. Remove data by index");
        System.out.println("7. Print");
        System.out.println("8. Search data");
        System.out.println("9. Sort data");
        System.out.println("10. Exit");
        System.out.println("========================================");
    }
}

```

Result :

```
==========================================
Data manipulation with doule linked list
==========================================
Choose Menu :
1. Add first
2. Add tail
3. Add data with index
4. remove first
5. Remove last
6. Remove data by index
7. Print
8. Search data
9. Sort data
10. Exit
==========================================
1
Insert data in head position : 56
==========================================
Data manipulation with doule linked list
==========================================
Choose Menu :
1. Add first
2. Add tail
3. Add data with index
4. remove first
5. Remove last
6. Remove data by index
7. Print
8. Search data
9. Sort data
10. Exit
==========================================
1
Insert data in head position : 34
```

```
Data manipulation with doule linked list
=========================================
Choose Menu :
1. Add first
2. Add tail
3. Add data with index
4. remove first
5. Remove last
6. Remove data by index
7. Print
8. Search data
9. Sort data
10. Exit
=========================================
2
Insert data in tail position : 345
=========================================
Data manipulation with doule linked list
=========================================
Choose Menu :
1. Add first
2. Add tail
3. Add data with index
4. remove first
5. Remove last
6. Remove data by index
7. Print
8. Search data
9. Sort data
10. Exit
=========================================
3
Insert data : 99
Insert index : 1
```

```
==========================================
Data manipulation with doule linked list
==========================================
Choose Menu :
1. Add first
2. Add tail
3. Add data with index
4. remove first
5. Remove last
6. Remove data by index
7. Print
8. Search data
9. Sort data
10. Exit
==========================================
7
34
99
56
345
```

```
==========================================
Data manipulation with doule linked list
==========================================
Choose Menu :
1. Add first
2. Add tail
3. Add data with index
4. remove first
5. Remove last
6. Remove data by index
7. Print
8. Search data
9. Sort data
10. Exit
==========================================
8
Search data : 99
Data : 99 is found in index-1
```

```
==========================================
Data manipulation with doule linked list
==========================================
Choose Menu :
1. Add first
2. Add tail
3. Add data with index
4. remove first
5. Remove last
6. Remove data by index
7. Print
8. Search data
9. Sort data
10. Exit
==========================================
9
345
99
56
34
```

2. We are required to create a program which Implement Stack using double linked list. The features are described in following illustrations:

Initial menu and add Data (push)

```
******************
Library data book
******************
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
******************
1
-------------------------
Insert new book title
-------------------------
Practical Digital Forensics
```

Print All Data

```
*****************
Library data book
*****************
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
*****************
4
------------------------
Info all books
------------------------
3D Computer Vision
Understanding Software
Algorithms Notes for Professionals
Getting Started with C++ Audio Programming for Game Developers
Practical Digital Forensics
```

See the data on top of the stack

```
******************
Library data book
******************
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
******************
3
------------------------
Peek book title from top
------------------------
```

Pop the data from the top of the stack

```
****************
Library data book
****************
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
****************
2
------------------------
Book om top has been removed
------------------------

****************
Library data book
****************
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
****************
4
------------------------
Info all books
------------------------
Understanding Software
Algorithms Notes for Professionals
Getting Started with C++ Audio Programming for Game Developers
Practical Digital Forensics
BUILD SUCCESSFUL (total time: 1 second)
```

Code :

Main

```java
package assnum2;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        Stack stack = new Stack();

        int choose;
        do {
            menu();
            choose = sc.nextInt();
            switch (choose) {
                case 1:
                    System.out.print("Insert new book title : ");
                    String data = sc.next();
                    sc.nextLine();
                    stack.push(data);
                    break;
                case 2:
                    try {
                        String topElement = stack.pop();
                        System.out.println("Removed book title: " + topElement);
                    } catch (Exception e) {
                        System.out.println(e.getMessage());
                    }
                    break;
                case 3:
                    try {
                        String topElement = stack.peek();
                        System.out.println("Top book title: " + topElement);
                    } catch (Exception e) {
                        System.out.println(e.getMessage());
                    }
                    break;
                case 4:
                    stack.print();
                    break;
                case 5:
                    System.exit(0);
                    break;

                default:
                    break;
            }
        } while (choose >= 0 && choose <=5);

        sc.close();
    }

    public static void menu() {
        System.out.println("========================================");
        System.out.println("Library data book");
        System.out.println("========================================");
        System.out.println("Choose Menu : ");
        System.out.println("1. Add new book");
        System.out.println("2. Get book from top");
        System.out.println("3. Peek book title from top");
        System.out.println("4. Info all books");
        System.out.println("5. Exit");
        System.out.println("========================================");
    }
}

```

Stack

```java
package assnum2;

public class Stack {
    private Node top;
    private int size;

    public Stack() {
        top = null;
        size = 0;
    }

    public boolean isEmpty() {
        return top == null;
    }

    public void push(String item) {
        Node newNode = new Node(null, item, top);
        if (top != null) {
            top.prev = newNode;
        }
        top = newNode;
        size++;
    }

    public String pop() throws Exception {
        if (isEmpty()) {
            throw new Exception("Stack is empty");
        }
        String topData = top.data;
        top = top.next;
        if (top != null) {
            top.prev = null;
        }
        size--;
        return topData;
    }

    public String peek() throws Exception {
        if (isEmpty()) {
            throw new Exception("Stack is empty");
        }
        return top.data;
    }

    public void print() {
        if (!isEmpty()) {
            Node tmp = top;
            System.out.println("Info all books");
            System.out.println("========================================");
            while (tmp != null) {
                System.out.println(tmp.data);
                tmp = tmp.next;
            }
        } else {
            System.out.println("Stack is empty");
        }
    }
}

```

```java
package assnum2;

public class Node {
    public String data;
    Node prev, next;

    public Node(Node prev, String data, Node next) {
        this.prev = prev;
        this.next = next;
        this.data = data;
    }
}

```

Result :

```
==========================================
Library data book
==========================================
Choose Menu :
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
==========================================
1
Insert new book title : How-to-become-hacker
```

```
==========================================
Library data book
==========================================
Choose Menu :
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
==========================================
4
Info all books
==========================================
Ethical-hacking
Practical-digital-forensics
How-to-become-hacker
```

```
==========================================
Library data book
==========================================
Choose Menu :
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
==========================================
3
Top book title: Ethical-hacking
==========================================
Library data book
==========================================
Choose Menu :
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
==========================================
2
Removed book title: Ethical-hacking
```

```
=========================================
Library data book
=========================================
Choose Menu :
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
=========================================
4
Info all books
=========================================
Practical-digital-forensics
How-to-become-hacker
```

3. Create a program that helps vaccination process by having a queue algorithm alongside with double linked list as follows (the amount left of queue length in menu print(3) and recent vaccinated person in menu Remove data (2) should be displayed)

Initial menu and adding a data

```
++++++++++++++++++++++++++++++
Extravaganza Vaccine Queue
++++++++++++++++++++++++++++++
1. Add Vaccine queue
2. Remove Vaccine queue
3. Display vaccine queue
4. Exit
++++++++++++++++++++++++++++++
1
Add new vaccine queue
Queue number : 123
Name : Joko
```

Print data (notice the highlighted red in the result)

```
++++++++++++++++++++++++++++
Extravaganza Vaccine Queue
++++++++++++++++++++++++++++
1. Add Vaccine queue
2. Remove Vaccine queue
3. Display vaccine queue
4. Exit
++++++++++++++++++++++++++++
3
++++++++++++++++++++++
Current vaccine queue :
| No.      | Name      |
| 123      | Joko      |
| 124      | Mely      |
| 135      | Johan     |
| 146      | Rosi      |
Queue left : 4
++++++++++++++++++++++
```

Remove Data (the highlighted red must displayed in the console too)

```
++++++++++++++++++++++++++++
Extravaganza Vaccine Queue
++++++++++++++++++++++++++++
1. Add Vaccine queue
2. Remove Vaccine queue
3. Display vaccine queue
4. Exit
++++++++++++++++++++++++++++
2
Joko has been vaccinated !
3
++++++++++++++++++++++
Current vaccine queue :
| No.      | Name      |
| 123      | Joko      |
| 124      | Mely      |
| 135      | Johan     |
| 146      | Rosi      |
Queue left : 3
++++++++++++++++++++++
```

Code :

Double Linked List

```java
package assnum3;

public class DoubleLinkedList {
    public Node head;
    public int size;

    public DoubleLinkedList() {
        head = null;
        size = 0;
    }

    public boolean isEmpty() {
        return head == null;
    }

    public void addFirst(int number, String name) {
        if (isEmpty()) {
            head = new Node(null, number, name, head);
        } else {
            Node newNode = new Node(null, number, name, head);
            head.prev = newNode;
            head = newNode;
        }
        size++;
    }

    public void addLast(int number, String name) {
        if (isEmpty()) {
            addFirst(number, name);
        } else {
            Node current = head;
            while (current.next != null) {
                current = current.next;
            }
            Node newNode = new Node(null, number, name, head);
            current.next = newNode;
            newNode.prev = current;
            size++;
        }
    }

    public void add(int number, String name, int index) throws Exception {
        if (isEmpty()) {
            addFirst(number, name);
        } else if (index < 0 || index > size) {
            throw new Exception("Index out of bound");
        } else {
            Node current = head;
            int i = 0;
            while (i < index) {
                current = current.next;
                i++;
            }
            if (current.next == null) {
                Node newNode = new Node(null, number, name, head);
                current.prev = newNode;
                head = newNode;
            } else {
                Node newNode = new Node(null, number, name, head);
                newNode.prev = current.prev;
                newNode.next = current;
                current.prev.next = newNode;
                current.prev = newNode;
            }
            size++;
        }
    }

    public int size() {
        return size;
    }

    public void clear() {
        head = null;
        size = 0;
    }

    public void print() {
        if (!isEmpty()) {
            Node tmp = head;
            int count = 0;
            while (tmp != null) {
                System.out.println("No = " + tmp.number);
                System.out.println("Name = " + tmp.name);
                System.out.println();
                tmp = tmp.next;
                count++;
                if (count == size) {
                    break;
                }
            }
        } else {
            System.out.println("Linked list is empty");
        }
    }
```

```java
    public void removeFirst() throws Exception {
        if (isEmpty()) {
            throw new Exception("Linked list is still empty, cannot remove");
        } else if (size == 1) {
            removeLast();
        } else {
            head = head.next;
            head.prev = null;
            size--;
        }
    }

    public void removeLast() throws Exception {
        if (isEmpty()) {
            throw new Exception("Linked list is still empty, cannot remove");
        } else if (head.next == null) {
            head = null;
            size--;
            return;
        }
        Node current = head;
        while (current.next.next != null) {
            current = current.next;
        }
        current.next = null;
        size--;
    }

    public void remove(int index) throws Exception {
        if (isEmpty() || index >= size) {
            throw new Exception("Index value is out of bound");
        } else if (size == 0) {
            removeFirst();
        } else {
            Node current = head;
            int i = 0;
            while (i < index) {
                current = current.next;
                i++;
            }
            if (current.next == null) {
                current.prev.next = null;
            } else if (current.prev == null) {
                current = current.next;
                current.prev = null;
                head = current;
            } else {
                current.prev.next = current.next;
                current.next.prev = current.prev;
            }
            size--;
        }
    }

    public int getFirst() throws Exception {
        if (isEmpty()) {
            throw new Exception("Linked list is still empty");
        }
        return head.number;
    }

    public int getLast() throws Exception {
        if (isEmpty()) {
            throw new Exception("Linked list is still empty");
        }
        Node tmp = head;
        while (tmp.next != null) {
            tmp = tmp.next;
        }
        return tmp.number;
    }

    public int get(int index) throws Exception {
        if (isEmpty()) {
            throw new Exception("Linked list is still empty");
        }
        Node tmp = head;
        for (int i = 0; i < index; i++) {
            tmp = tmp.next;
        }
        return tmp.number;
    }
}

```

Main

```java
package assnum3;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Queue queue = new Queue();

        int choose;
        do {
            menu();
            choose = sc.nextInt();
            switch (choose) {
                case 1:
                    System.out.println("Add new queue");
                    System.out.print("Queue number : ");
                    int number = sc.nextInt();
                    System.out.print("Name : ");
                    String name = sc.next();
                    queue.enqueue(number, name);
                    break;
                case 2:
                    try {
                        System.out.println("Removing vaccine queue");
                        queue.dequeue();
                    } catch (Exception e) {
                        System.out.println(e.getMessage());
                    }
                    break;
                case 3:
                    queue.print();
                    System.out.println("Current queue length: " + queue.size());
                    break;
                case 4:
                    System.exit(0);
                    break;

                default:
                    break;
            }
        } while (choose >= 0 && choose <= 4);

        sc.close();
    }

    public static void menu() {
        System.out.println("========================================");
        System.out.println("Extravaganza Vaccine Queue");
        System.out.println("========================================");
        System.out.println("Choose Menu : ");
        System.out.println("1. Add Vaccine queue");
        System.out.println("2. Remove vaccine queue");
        System.out.println("3. Display vaccine queue");
        System.out.println("4. Exit");
        System.out.println("========================================");
    }
}

```

Node

```java
package assnum3;

public class Node {
    public String name;
    public int number;
    Node prev, next;

    public Node(Node prev, int number, String name, Node next) {
        this.prev = prev;
        this.next = next;
        this.name = name;
        this.number = number;
    }
}
```

Queue

```java
package assnum3;

public class Queue {
    private DoubleLinkedList list;

    public Queue() {
        list = new DoubleLinkedList();
    }

    public boolean isEmpty() {
        return list.isEmpty();
    }

    public void enqueue(int number, String data) {
        list.addLast(number, data);
    }

    public void dequeue() throws Exception {
        if (isEmpty()) {
            throw new Exception("Queue is empty. Cannot dequeue.");
        } else {
            list.removeFirst();
        }
    }

    public int size() {
        return list.size();
    }

    public void print() {
        list.print();
    }
}

```

Result :

```
==========================================
Extravaganza Vaccine Queue
==========================================
Choose Menu :
1. Add Vaccine queue
2. Remove vaccine queue
3. Display vaccine queue
4. Exit
==========================================
1
Add new queue
Queue number : 1
Name : Azhar
==========================================
Extravaganza Vaccine Queue
==========================================
Choose Menu :
1. Add Vaccine queue
2. Remove vaccine queue
3. Display vaccine queue
4. Exit
==========================================
1
Add new queue
Queue number : 2
Name : Joko
```

```
==========================================
Extravaganza Vaccine Queue
==========================================
Choose Menu :
1. Add Vaccine queue
2. Remove vaccine queue
3. Display vaccine queue
4. Exit
==========================================
3
No = 1
Name = Azhar

No = 2
Name = Joko

Current queue length: 2
==========================================
Extravaganza Vaccine Queue
==========================================
Choose Menu :
1. Add Vaccine queue
2. Remove vaccine queue
3. Display vaccine queue
4. Exit
==========================================
2
Removing vaccine queue
```

```
==========================================
Extravaganza Vaccine Queue
==========================================
Choose Menu :
1. Add Vaccine queue
2. Remove vaccine queue
3. Display vaccine queue
4. Exit
==========================================
3
No = 2
Name = Joko

Current queue length: 1
```

4. Create a program implementation that list students score. Each student's data consist of their nim, name, and gpa. The program should implement double linked list and should be able to search based on NIM and sort the GPA in descending order. Students class must be implemented in this program

Initial menu and adding data

```
================================
Student Data Management System
================================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
================================
1
Insert NIM in head position
NIM : 123
Name : Anang
GPA : 2.77
```

```
================================
Student Data Management System
================================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
================================
2
Insert NIM in tail position
NIM : 233
Name : Suparjo
GPA : 3.67
```

```
================================
Student Data Management System
================================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
================================
3
Insert student's data node
NIM : 743
Name : Freddy
GPA : 2.90
In index : 3
```

Printing data

```
================================
Student Data Management System
================================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
================================
7
NIM : 123
Name : Anang
GPA : 2.77
Insert NIM in tail position
NIM : 233
Name : Suparjo
GPA : 3.67
Insert student's data node
NIM : 743
Name : Freddy
GPA : 2.90
In index : 3
All data printed successfully
```

Searching data

```
================================
Student Data Management System
================================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
================================
8
Insert NIM to be searched : 565
Data 565 is in node - 5
Identity :
NIM : 565
Name : Ahmad
GPA : 3.80
```

Sorting data

```
==============================
Student Data Management System
==============================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
==============================
9

==============================
Student Data Management System
==============================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
==============================
7
NIM : 233
Name : Suparjo
GPA : 3.67
NIM : 743
Name : Freddy
GPA : 2.90
NIM : 123
Name : Anang
GPA : 2.77
```

Code :

Double Linked List

```java
package assnum4;

public class DoubleLinkedList {
    public Node head;
    public int size;

    public DoubleLinkedList() {
        head = null;
        size = 0;
    }

    public boolean isEmpty() {
        return head == null;
    }

    // choose 1
    public void addFirst(String nim, String name, double gpa) {
        if (isEmpty()) {
            head = new Node(null, nim, name, gpa, head);
        } else {
            Node newNode = new Node(null, nim, name, gpa, head);
            head.prev = newNode;
            head = newNode;
        }
        size++;
    }

    // choose 2
    public void addLast(String nim, String name, double gpa) {
         if (isEmpty()) {
            addFirst(nim, name, gpa);
         } else {
            Node current = head;
            while (current.next != null) {
                current = current.next;
            }
            Node newNode = new Node(current, nim, name, gpa, null);
            current.next = newNode;
            size++;
         }
    }

    // choose 3
    public void add(String nim, String name, double gpa, int index) throws Exception {
        if (isEmpty()) {
            addFirst(nim, name, gpa);
        } else if (index < 0 || index > size ) {
            throw new Exception("Index out of bound");
        } else {
            Node current = head;
            int i = 0;
            while (i < index) {
                current = current.next;
                i++;
            }
            if (current.next == null) {
                Node newNode = new Node(null, nim, name, gpa, current);
                current.prev = newNode;
                head = newNode;
            } else {
                Node newNode = new Node(current.prev, nim, name, gpa, current);
                newNode.prev = current.prev;
                newNode.next = current;
                current.prev.next = newNode;
                current.prev = newNode;
            }
        }
        size++;
    }

    // choose 7
    public void print() {
        if (!isEmpty()) {
            Node tmp = head;
            while (tmp != null) {
                System.out.println("NIM = " + tmp.nim);
                System.out.println("Name = " + tmp.name);
                System.out.println("GPA = " + tmp.gpa);
                tmp = tmp.next;
            }
            // System.out.println("\n successfully added");
        } else {
            System.out.println("Linked list is empty");
        }
    }

    // add practicum 2
    // choose 4
    public void removeFirst() throws Exception {
        if (isEmpty()) {
            throw new Exception("Linked list is still empty, cannot remove");
        } else if (size == 1) {
            removeLast();
        } else {
            head = head.next;
            head.prev = null;
            size--;
        }
    }
```

```java
    // choose 5
    public void removeLast() throws Exception {
        if (isEmpty()) {
            throw new Exception("Linked list is still empty, cannot remove");
        } else if (head.next == null) {
            head = null;
            size--;
            return;
        }
        Node current = head;
        while (current.next.next != null) {
            current = current.next;
        }
        current.next = null;
        size--;
    }

    // choose 6
    public void remove(int index) throws Exception {
        if (isEmpty() || index >= size) {
            throw new Exception("Index value is out of bound");
        } else if (size == 0) {
            removeFirst();
        } else {
            Node current = head;
            int i = 0;
            while (i < index) {
                current = current.next;
                i++;
            }
            if (current.next == null) {
                current.prev.next = null;
            } else if (current.prev == null) {
                current = current.next;
                current.prev = null;
                head = current;
            } else {
                current.prev.next = current.next;
                current.next.prev = current.prev;
            }
            size--;
        }
    }

    // choose 8
    public int sequentialSearch(String nim) {
        Node current = head;
        for (int index = 0; current != null; index++) {
            if (current.nim.equals(nim)) {
                return index;
            }
            current = current.next;
        }

        return -1;
    }

    public void showPosition(String x, int pos) {
        if (pos != -1) {
            System.out.println("Data : " + x + " is found in index-" + pos);
            Node current = head;
            for (int i = 0; i < pos; i++) {
                current = current.next;
            }
            System.out.println("NIM \t = " + current.nim);
            System.out.println("Name \t = " + current.name);
            System.out.println("GPA \t = " + current.gpa);
        } else {
            System.out.println("Data : " + x + " is not found!");
        }
    }

    // choose 9
    public void bubbleSort() {
        if (head == null || head.next == null) {
            return; // No need to sort an empty list or a list with only one element
        }

        boolean swapped;
        Node current;

        for (int i = 0; i < size - 1; i++) {
            swapped = false;
            current = head;

            for (int j = 0; j < size - 1 - i; j++) {
                if (current.getGpa() < current.next.getGpa()) {
                    // Swap GPA
                    double tempGpa = current.getGpa();
                    current.setGpa(current.next.getGpa());
                    current.next.setGpa(tempGpa);

                    // Swap NIM
                    String tempNim = current.getNim();
                    current.setNim(current.next.getNim());
                    current.next.setNim(tempNim);

                    // Swap Name
                    String tempName = current.getName();
                    current.setName(current.next.getName());
                    current.next.setName(tempName);

                    swapped = true;
                }
                current = current.next;
            }

            if (!swapped) {
                break;
            }
        }
    }
}

```

```java
package assnum4;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        DoubleLinkedList dll = new DoubleLinkedList();

        int choose;
        do {
            menu();
            choose = sc.nextInt();
            switch (choose) {
                case 1:
                    System.out.println("Insert student in head position");
                    System.out.print("NIM : ");
                    String nim1 = sc.next();
                    System.out.print("Name : ");
                    String name1 = sc.next();
                    System.out.print("GPA : ");
                    double gpa1 = sc.nextDouble();
                    dll.addFirst(nim1, name1, gpa1);
                    break;
                case 2:
                    System.out.println("Insert student in head position");
                    System.out.print("NIM : ");
                    String nim2 = sc.next();
                    System.out.print("Name : ");
                    String name2 = sc.next();
                    System.out.print("GPA : ");
                    double gpa2 = sc.nextDouble();
                    dll.addLast(nim2, name2, gpa2);
                    break;
                case 3:
                    System.out.println("Insert student by index");
                    System.out.print("NIM : ");
                    String nim3 = sc.next();
                    System.out.print("Name : ");
                    String name3 = sc.next();
                    System.out.print("GPA : ");
                    double gpa3 = sc.nextDouble();
                    System.out.print("Index : ");
                    int index3 = sc.nextInt();
                    dll.add(nim3, name3, gpa3, index3);
                    break;
                case 4:
                    dll.removeFirst();
                    break;
                case 5:
                    dll.removeLast();
                    break;
                case 6:
                    System.out.print("Remove data in specific index : ");
                    int index6 = sc.nextInt();
                    dll.remove(index6);
                    break;
                case 7:
                    dll.print();
                    break;
                case 8:
                    System.out.print("Search student by NIM : ");
                    String nim8 = sc.next();
                    int position = dll.sequentialSearch(nim8);
                    dll.showPosition(nim8, position);
                    break;
                case 9:
                    dll.bubbleSort();
                    dll.print();
                    break;
                case 10:
                    System.exit(0);
                    break;

                default:
                    break;
            }
        } while (choose >= 0 && choose <= 10);

        sc.close();
    }

    public static void menu() {
        System.out.println("========================================");
        System.out.println("Student Data Management System");
        System.out.println("========================================");
        System.out.println("Choose Menu : ");
        System.out.println("1. Add data from head");
        System.out.println("2. Add data from tail");
        System.out.println("3. Add data with index");
        System.out.println("4. remove data from head");
        System.out.println("5. Remove data from tail");
        System.out.println("6. Remove data by index");
        System.out.println("7. Print");
        System.out.println("8. Search by NIM");
        System.out.println("9. Sort by GPA");
        System.out.println("10. Exit");
        System.out.println("========================================");
    }
}
```

```java
package assnum4;

public class Node {
    public String nim, name;
    public double gpa;
    Node prev, next;

    public Node(Node prev, String nim, String name, double gpa, Node next) {
        this.prev = prev;
        this.next = next;
        this.nim = nim;
        this.name = name;
        this.gpa = gpa;
    }

    public String getNim() {
        return nim;
    }

    public void setNim(String nim) {
        this.nim = nim;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getGpa() {
        return gpa;
    }

    public void setGpa(double gpa) {
        this.gpa = gpa;
    }
}

```

Result :

```
==========================================
Student Data Management System
==========================================
Choose Menu :
1. Add data from head
2. Add data from tail
3. Add data with index
4. remove data from head
5. Remove data from tail
6. Remove data by index
7. Print
8. Search by NIM
9. Sort by GPA
10. Exit
==========================================
1
Insert student in head position
NIM : 123
Name : azhar
GPA : 4
```

```
==========================================
Student Data Management System
==========================================
Choose Menu :
1. Add data from head
2. Add data from tail
3. Add data with index
4. remove data from head
5. Remove data from tail
6. Remove data by index
7. Print
8. Search by NIM
9. Sort by GPA
10. Exit
==========================================
9
NIM = 123
Name = azhar
GPA = 4.0
NIM = 345
Name = rifai
GPA = 3.9
NIM = 234
Name = rizqi
GPA = 3.5
```

```
==========================================
Student Data Management System
==========================================
Choose Menu :
1. Add data from head
2. Add data from tail
3. Add data with index
4. remove data from head
5. Remove data from tail
6. Remove data by index
7. Print
8. Search by NIM
9. Sort by GPA
10. Exit
==========================================
8
Search student by NIM : 123
Data : 123 is found in index-0
NIM      = 123
Name     = azhar
GPA      = 4.0
```