# JOBSHEET 12

# Double Linked Lists

## 1.1. Learning Objective

After learning this lab activity, students will be able to:

1. Understand Double Linked List algorithm
2. Create and declare double linked list algorithm
3. Implement double linked list algorithm in various case studies

## 1.2. Lab Activities 1

In this lab activity, we will create Node class and DoubleLinkedList class that has operations to insert data in multiple way. (from the beginning or the tail of the list)

### 1.2.1. Steps

1. Take this class diagram as your reference for creating the **DoubleLinkedList clas**

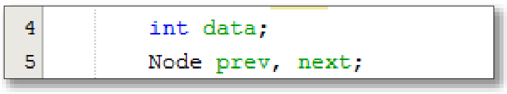
| Node |
|---|
| data: int<br>prev: Node<br>next: Node |
| Node(prev: Node, data:int, next:Node) |

| DoubleLinkedLists |
|---|
| head: Node<br>size : int |
| DoubleLinkedLists()<br>isEmpty(): boolean<br>addFirst (): void<br>addLast(): void<br>add(item: int, index:int): void<br>size(): int<br>clear(): void<br>print(): void |

**2.** Create a new package named **DoubleLinkedList**

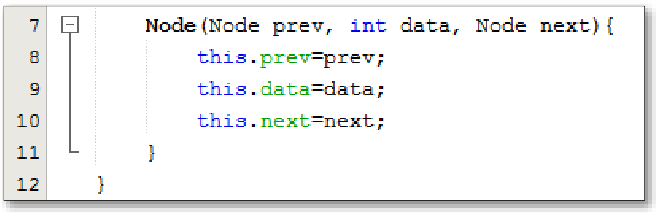
**3.** Create a new class in that package named **Node**

4. In that class, declare the attributes as described in the class diagram



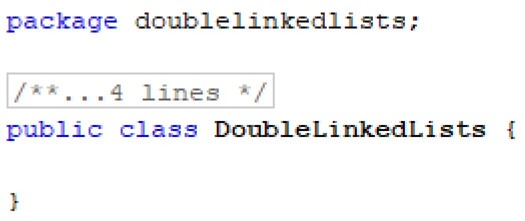
```
    int data;
    Node prev, next;
```

5. Next, add the default constructor in Node class



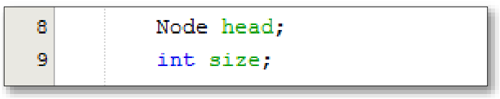
```
    Node(Node prev, int data, Node next){
        this.prev=prev;
        this.data=data;
        this.next=next;
    }
}
```

6. Create a new class named **DoubleLinkedList** in the same package with the node as following image:



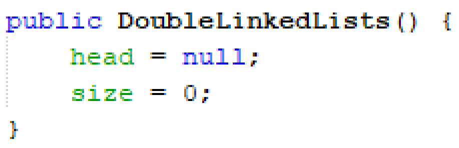
```
package doublelinkedlists;

/**...4 lines */
public class DoubleLinkedLists {

}
```

7. Next, we add the attributes



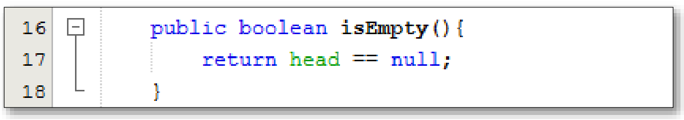
```
    Node head;
    int size;
```

**8.** Then, add the constructor in class **DoubleLinkedList**



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

9. Create method isEmpty(), this method will be used to check whether the linked list is empty or not



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

10. Then add method **addFirst().** This method will be executed when we want to add data in the beginning of the list

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

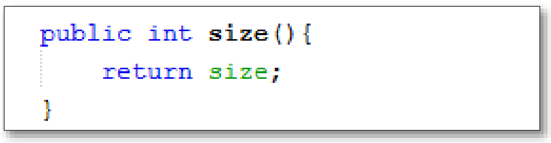
11. Let’s not forget about adding the data in the end of the list. We can do it after adding these lines of code in **addLast()** method

```
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++;
    }
}
```

12. If we want to add a data that specified by a certain index, we will need to provide additional method to do so. It can be done by creating the **add()** method

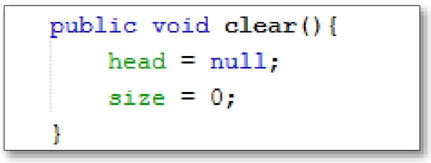
```
public void add(int item, int index) throws Exception{
    if(isEmpty()){
        addFirst(item);
    }else if(index < 0 || index > size){
        throw new Exception("Index out of bound");
    }else{
        Node current = head;
        int i = 0;
        while (i < index) {
            current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

13. We want to make our list has an easy access to retrieve the length of the list. That's why we create method **size()**



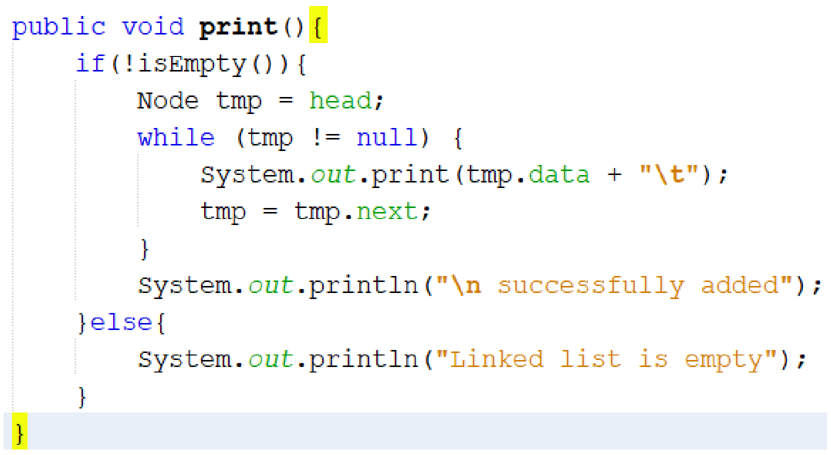
```
public int size(){
    return size;
}
```

14. We create a method **clear()** to remove all the data that are exist in linked lists



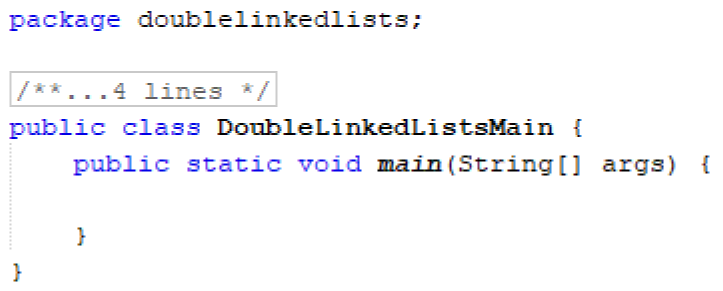
```
public void clear(){
    head = null;
    size = 0;
}
```

15. Next up, to print the whole data in the list, we need to create a method print().



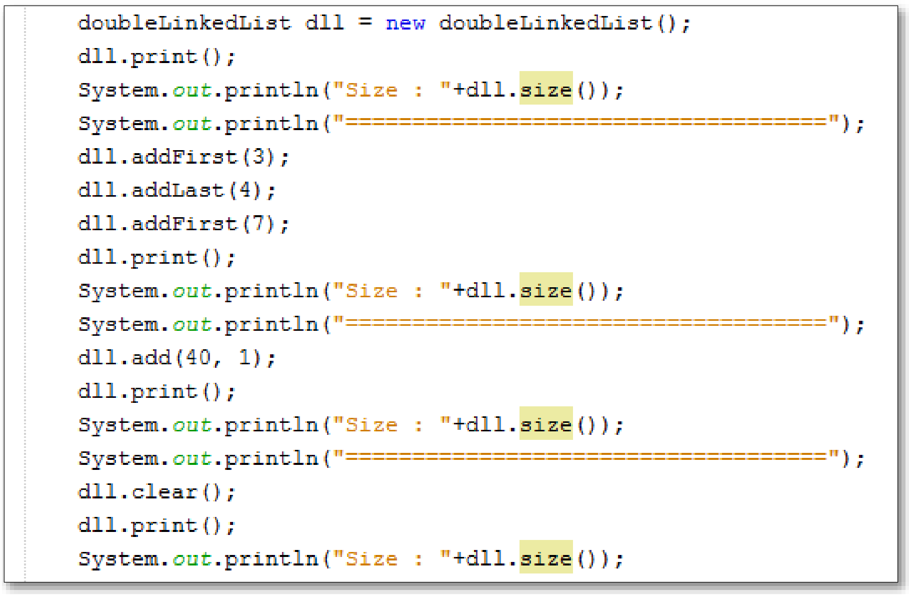
```
public void print(){
    if(!isEmpty()){
        Node tmp = head;
        while (tmp != null) {
            System.out.print(tmp.data + "\t");
            tmp = tmp.next;
        }
        System.out.println("\n successfully added");
    }else{
        System.out.println("Linked list is empty");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

16. After creating the blueprint classes, we will need one main class so that all of that can be included in the program. Create **DoubleLinkedListMain class** to do so



```
package doublelinkedlists;

/**...4 lines */
public class DoubleLinkedListsMain {
    public static void main(String[] args) {

    }
}
```

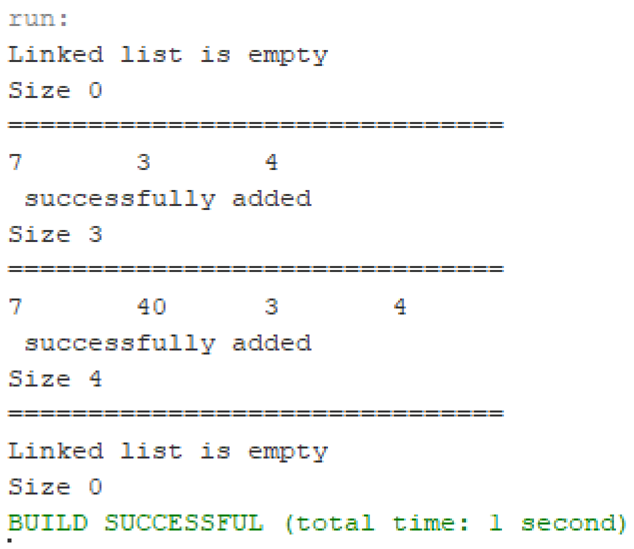
17. Instantiate an object from **DoubleLinkedList** class in the main method. Then apply these program code



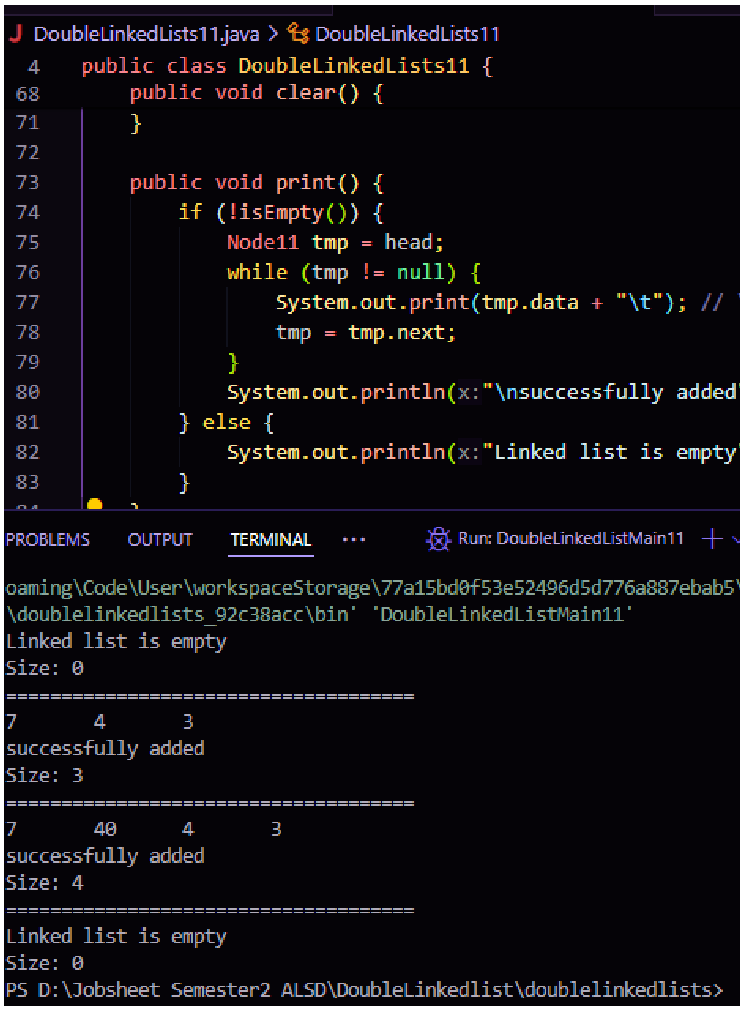
```java
doubleLinkedList dll = new doubleLinkedList();
dll.print();
System.out.println("Size : "+dll.size());
System.out.println("====================================");
dll.addFirst(3);
dll.addLast(4);
dll.addFirst(7);
dll.print();
System.out.println("Size : "+dll.size());
System.out.println("====================================");
dll.add(40, 1);
dll.print();
System.out.println("Size : "+dll.size());
System.out.println("====================================");
dll.clear();
dll.print();
System.out.println("Size : "+dll.size());
```

### 1.2.2. Result

Compile the program and see if the result matches with following image



```
run:
Linked list is empty
Size 0
===============================
7       3       4
 successfully added
Size 3
===============================
7       40      3       4
 successfully added
Size 4
===============================
Linked list is empty
Size 0
BUILD SUCCESSFUL (total time: 1 second)
```



DoubleLinkedLists11.java > DoubleLinkedLists11
PROBLEMS
OUTPUT
TERMINAL
Run: DoubleLinkedListMain11

```java
public class DoubleLinkedLists11 {
    public void clear() {
    }

    public void print() {
        if (!isEmpty()) {
            Node11 tmp = head;
            while (tmp != null) {
                System.out.print(tmp.data + "\t"); // 
                tmp = tmp.next;
            }
            System.out.println(x:"\nsuccessfully added
        } else {
            System.out.println(x:"Linked list is empty
        }
```

```
oaming\Code\User\workspaceStorage\77a15bd0f53e52496d5d776a887ebab5\
\doublelinkedlists_92c38acc\bin' 'DoubleLinkedListMain11'
Linked list is empty
Size: 0
====================================
7       4       3
successfully added
Size: 3
====================================
7       40      4       3
successfully added
Size: 4
====================================
Linked list is empty
Size: 0
PS D:\Jobsheet Semester2 ALSD\DoubleLinkedlist\doublelinkedlists>
```

### 1.2.3. Questions

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

   ➢ **Single Linked List:**

   - **Node Structure: One pointer (next).**
   - **Traversal: Only from front to back.**
   - **Insertion and Deletion: Changes to one pointer.**
   - **Extra Space: Requires a little extra space.**
   - **Complexity: Slightly lower for insert and delete operations.**

   ➢ **Double Linked List:**

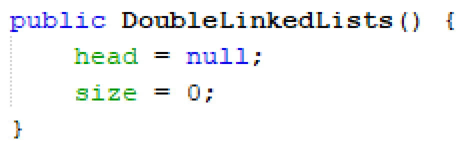
   - **Node Structure: Two pointers (prev and next).**
   - **Traversal: From front to back and vice versa.**
   - **Insertion and Deletion: Changes to two pointers.**
   - **Extra Space: Requires more extra space.**
   - **Complexity: Slightly higher for insert and delete operations.**

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

**In a doubly linked list, the `next` attribute points to the next node in the list, facilitating forward traversal, while the `prev` attribute points to the previous node, enabling backward traversal.**

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;
   }
   ```

   **The constructor on `DoubleLinkedLists` initializes the `head` and `size` attributes of the linked list. By setting `head` to `null` and `size` to `0`, the constructor specifies that the linked list is initially empty, ready to be added to.**

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);
```

**In the `addFirst()` method, creating a `Node` object with the `prev` parameter is considered `null` because the added element will be the first element (head) in the linked list, so it has no previous nodes.**

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);
   ```

   **In `addLast()` method, creation of `Node` object by passing parameter `prev` as `current` and `next` as `null` is done to add new element at the end of linked list. In this way, new node will be the last node in linked list, so `prev` points to previous last node (`current`) and `next` is set as `null` since there is no node after it.**

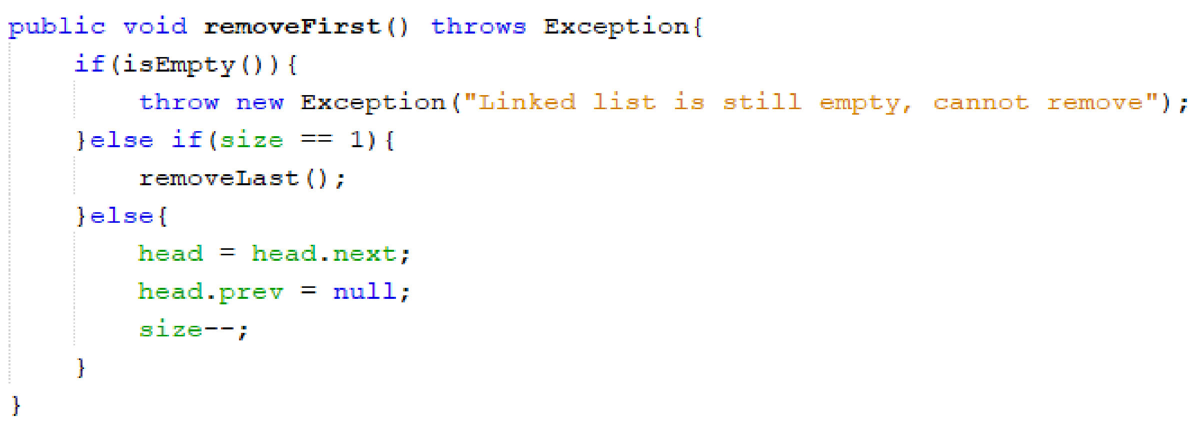
## 1.3. Lab Activities 2

In this lab activity, we have added some methods from our 1st lab activity. Now, we added some ways for the users to remove a data in the beginning of the list, the tail, or with specified index. For more details, pay attention to this class diagram:

| DoubleLinkedLists |
|---|
| head: Node<br>size : int |
| DoubleLinkedLists()<br>isEmpty(): boolean<br>addFirst (): void<br>addLast(): void<br>add(item: int, index:int): void<br>size(): int<br>clear(): void<br>print(): void<br>**removeFirst(): void**<br>**removeLast(): void**<br>**remove(index:int):void** |

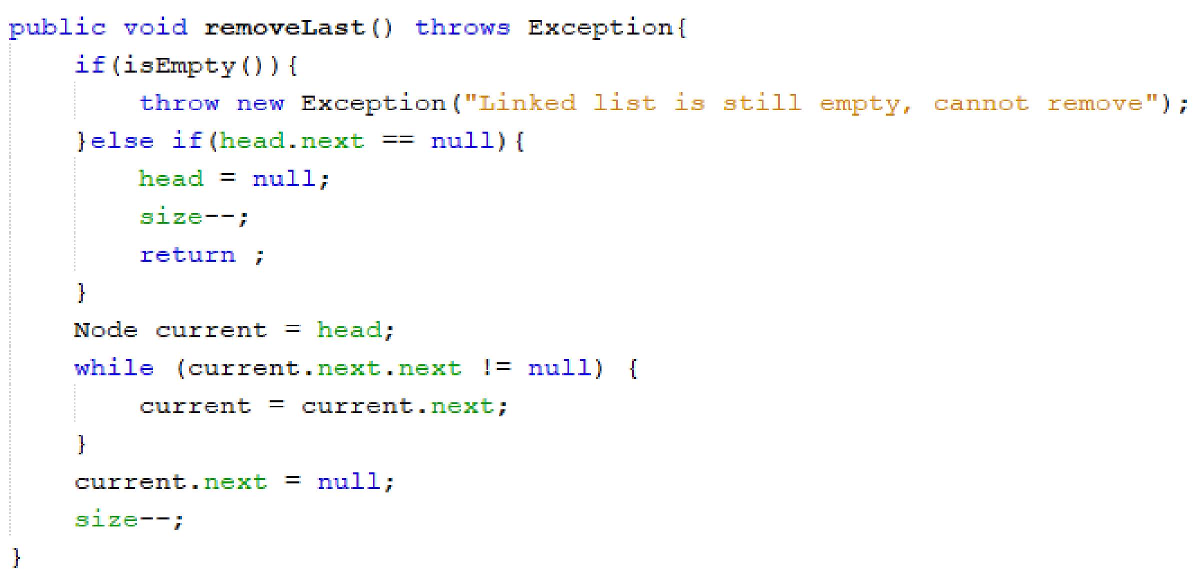
### 1.3.1. Steps

1. Create method **removeFirst()** in **class DoubleLinkedList**



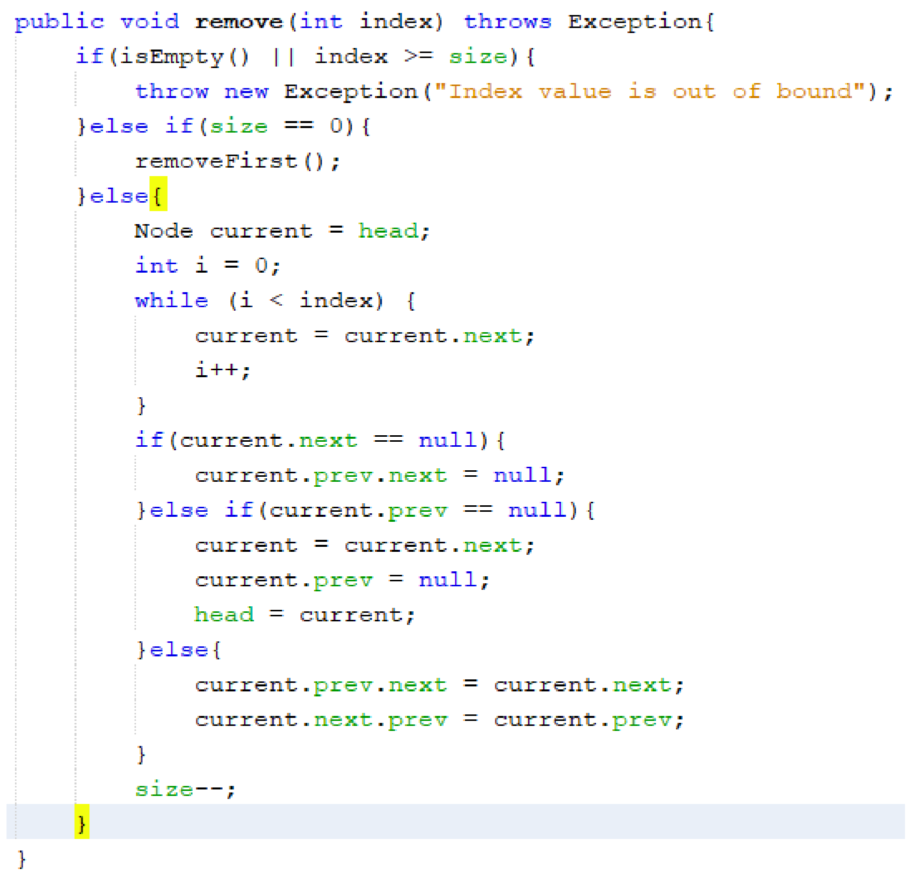
```
public void removeFirst() throws Exception{
    if(isEmpty()){
        throw new Exception("Linked list is still empty, cannot remove");
    }else if(size == 1){
        removeLast();
    }else{
        head = head.next;
        head.prev = null;
        size--;
    }
}
```

2. Create method **removeLast()** in **class DoubleLinkedList**



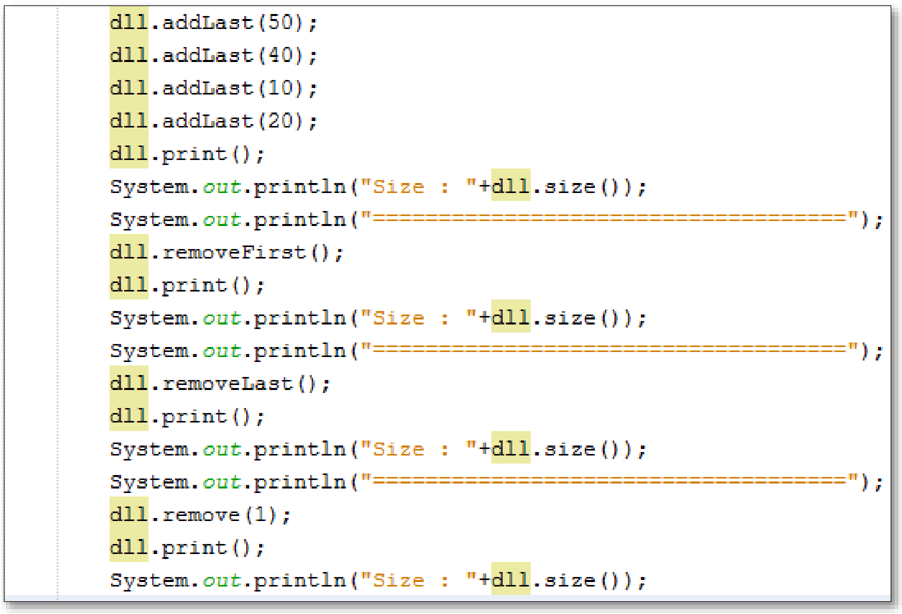
```java
public void removeLast() throws Exception{
    if(isEmpty()){
        throw new Exception("Linked list is still empty, cannot remove");
    }else if(head.next == null){
        head = null;
        size--;
        return ;
    }
    Node current = head;
    while (current.next.next != null) {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

3. Create method **remove()** in **class DoubleLinkedList,** alongside with its parameter



```java
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--;
    }
}
```

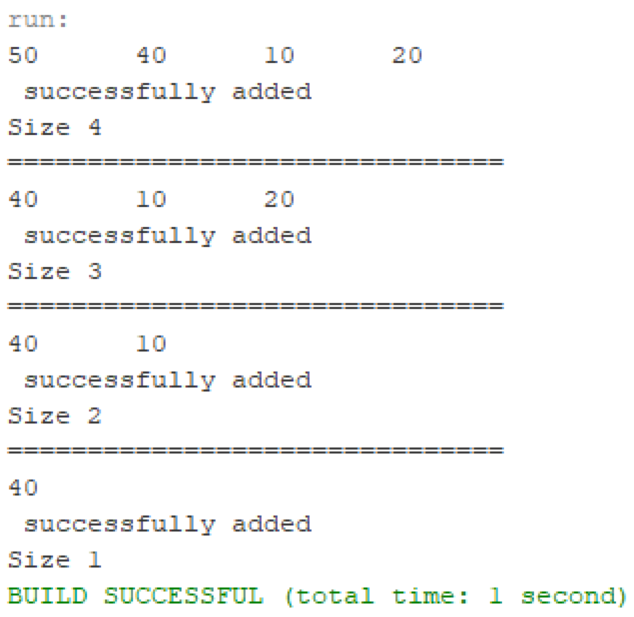
4. To execute additional codes we've just added, also make addition in the main class as well



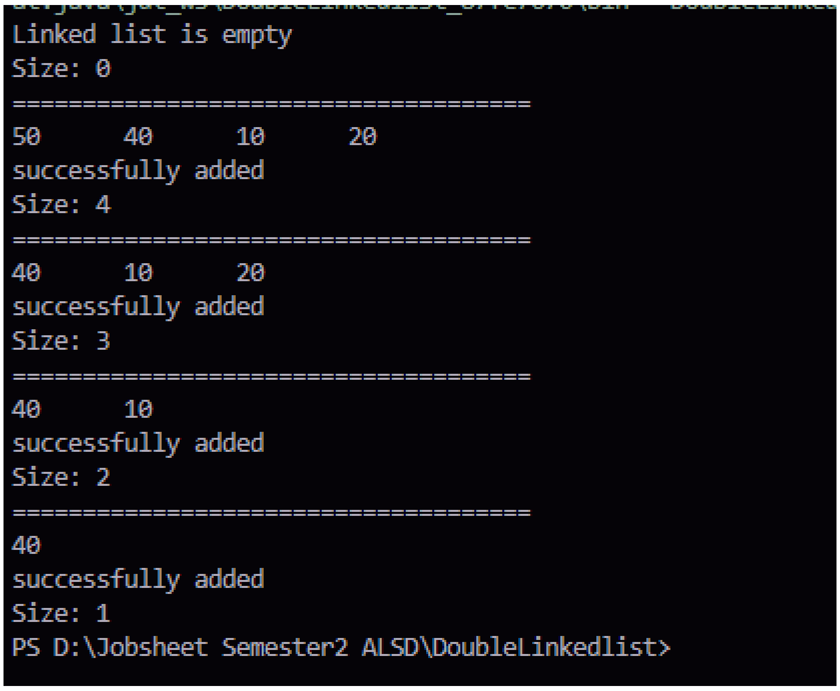
```
dll.addLast(50);
dll.addLast(40);
dll.addLast(10);
dll.addLast(20);
dll.print();
System.out.println("Size : "+dll.size());
System.out.println("====================================");
dll.removeFirst();
dll.print();
System.out.println("Size : "+dll.size());
System.out.println("====================================");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

### 1.3.2. Result

Compile the program and see if the result matches with following image



```
run:
50      40      10      20
 successfully added
Size 4
================================
40      10      20
 successfully added
Size 3
================================
40      10
 successfully added
Size 2
================================
40
 successfully added
Size 1
BUILD SUCCESSFUL (total time: 1 second)
```



```
Linked list is empty
Size: 0
=====================================
50      40      10      20
successfully added
Size: 4
=====================================
40      10      20
successfully added
Size: 3
=====================================
40      10
successfully added
Size: 2
=====================================
40
successfully added
Size: 1
PS D:\Jobsheet Semester2 ALSD\DoubleLinkedlist>
```

### 1.3.3. Questions

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

   - Cek apakah linked list kosong:

   ```
   if (isEmpty()) {
     throw new Exception("Linked list is still empty, cannot remove");
       }
   ```

   Jika kosong, lempar pengecualian.

   - Cek apakah hanya ada satu elemen:

   ```
   else if (size == 1) {
     removeLast();
       }
   ```

   Jika ada satu elemen, panggil `removeLast()`.

   - Hapus elemen pertama jika lebih dari satu elemen:

   ```
   else {
     head = head.next;
     head.prev = null;
     size--;
       }
   ```

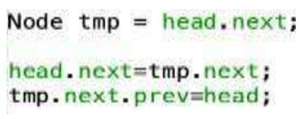
   Pindahkan `head` ke elemen berikutnya, putuskan hubungan dengan elemen pertama, dan kurangi ukuran.

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

   **removeLast()**?

   **can use an additional variable to store the previous position when iterating through a linked list. When it reaches the last element, the previous position will point to the last element, and use this information to delete the last element.**

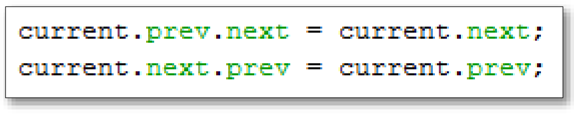
3. Explain why this program code is not suitable if we include it in **remove** command!



```
Node tmp = head.next;

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

   **The code does not handle the special case when the node to be deleted is the first node (head) of the linked list. When the first node is deleted, it is necessary to change the head to point to the next node.**

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



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

   **performs the deletion of a node from a linked list by setting the next reference of the previous node to point to the node after it, and setting the prev reference of the node after it to point back to the previous node, thereby removing the node from the linked list.**

   **With these two lines of code, the current node is removed from the linked list by changing the references of the previous node and the following node to "skip" the node to be removed.**

## 1.4. Lab Activities 3

In this 3rd lab activity, we will test if we can retrieve a data in linked list in various needs. The first is we can get a data in the beginning of the list, at the end of the list, or in specified index of the list. We will create 3 methods to realize the idea. For more details, feel free to check this class diagram

| DoubleLinkedLists |
|---|
| head: Node<br>size : int |
| DoubleLinkedLists()<br>isEmpty(): boolean<br>addFirst (): void<br>addLast(): void<br>add(item: int, index:int): void<br>size(): int<br>clear():  void<br>print(): void<br>removeFirst(): void<br>removeLast(): void<br>remove(index:int):void<br>**getFirst(): int**<br>**getLast() : int**<br>**get(index:int):  int** |

### 1.4.1. Steps

1. Create a method **getFirst()** in class **DoubleLinkedList** to retrieve the first  data in the list

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

2. Create a method **getLast()** in class **DoubleLinkedList** to retrieve the data in the list

```java
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;
}
```

3. Create a method **get(int index)** in class **DoubleLinkedList** to retrieve the data in specified index of the list

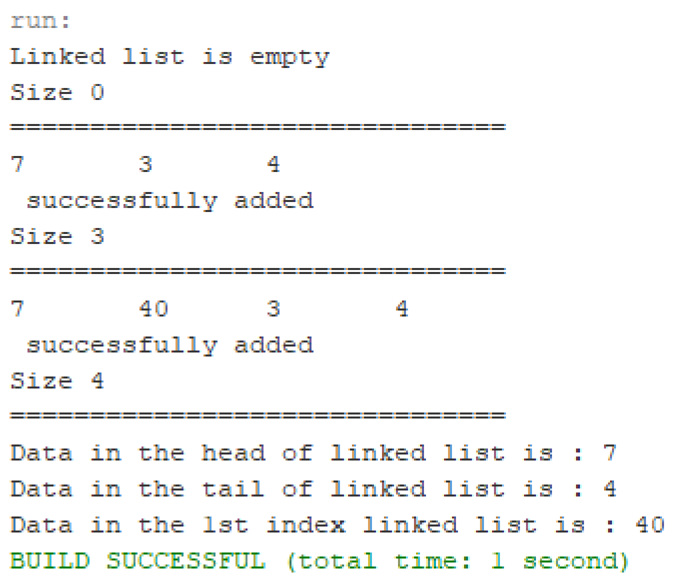
```java
public int get(int index) throws Exception{
    if(isEmpty()){
        throw new Exception("Linked list still empty");
    }
    Node tmp = head;
    for (int i = 0; i < index; i++) {
        tmp = tmp.next;
    }
    return tmp.data;
}
```

4. In the main class, add the program code as follows and see the result

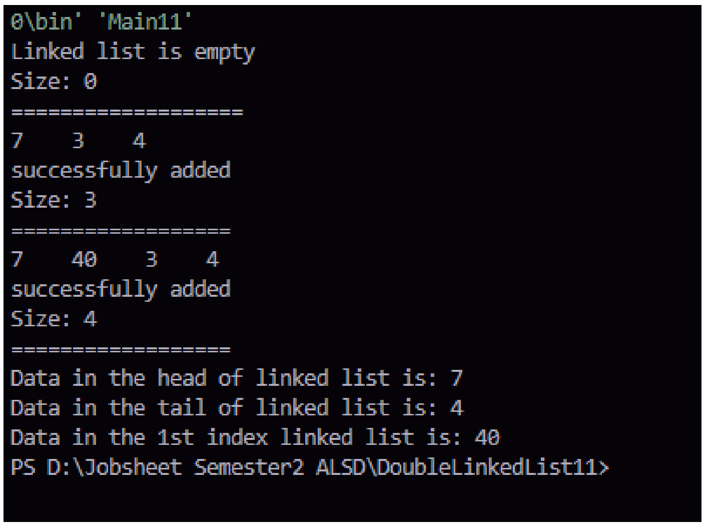
```java
public static void main(String[] args) throws Exception {
    DoubleLinkedList dll = new DoubleLinkedList();

    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));

}
```

### 1.4.2. Result

Compile the program and see if the result matches with following image



```
run:
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
BUILD SUCCESSFUL (total time: 1 second)
```



```
0\bin' 'Main11'
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
PS D:\Jobsheet Semester2 ALSD\DoubleLinkedList11>
```

### 1.4.3. Questions

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

   **The `size()` method in the `DoubleLinkedList` class functions to return the number of elements in the linked list.**

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

   **To start an index in a doubly linked list from 1 instead of 0, it is necessary to adjust how you manipulate the index in add, delete, or element access operations. This involves customizing methods such as `add()`, `get()`, and other operations that use indexes.**

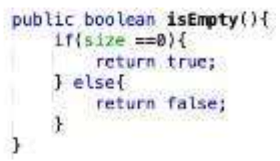
3. Please explain the difference between method **Add()** in double linked list and single linked list !

   ➢ **The difference between the `add()` method in a double linked list and a single linked list:**

   - **Double Linked List:**
     - **Allows adding elements anywhere in the list with O(1) performance, because each node has references to the nodes before and after it.**

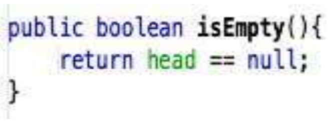
   - **Single Linked List:**
     - **Usually only supports adding elements at the beginning or end of the list with O(1) performance, but adding in the middle of the list requires searching the previous node, which results in O(n) performance.**

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



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

(a)



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

(b)

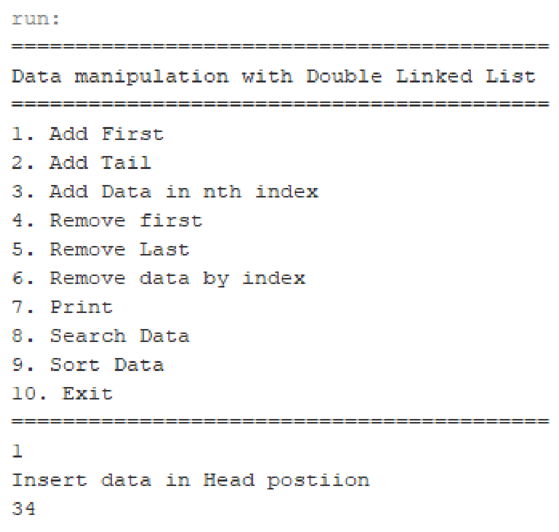
   ➢ **The differences between the two methods are:**
   - **First Method:**
     - **Checking for empty linked lists using the `size` variable.**
     - **Does not provide a return value if the linked list is empty.**
     - **Use of `if` syntax is incorrect and no return value is specified if the linked list is empty.**

   - **Second Method:**
     - **Checking for empty linked lists using the `head` pointer.**
     - **Returns `true` if the linked list is empty.**
     - **Simpler and clearer in determining empty linked lists.**

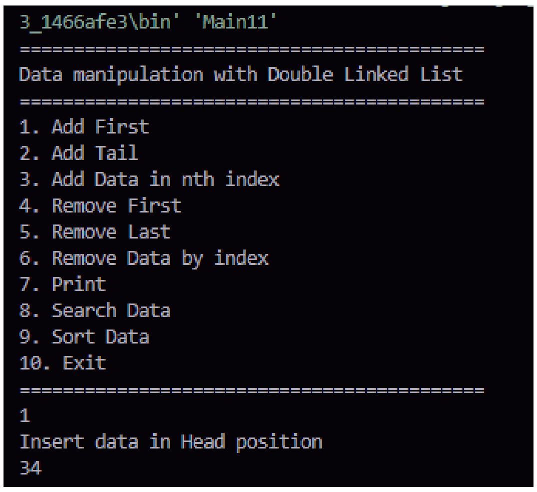
**1.5. Assignment**

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

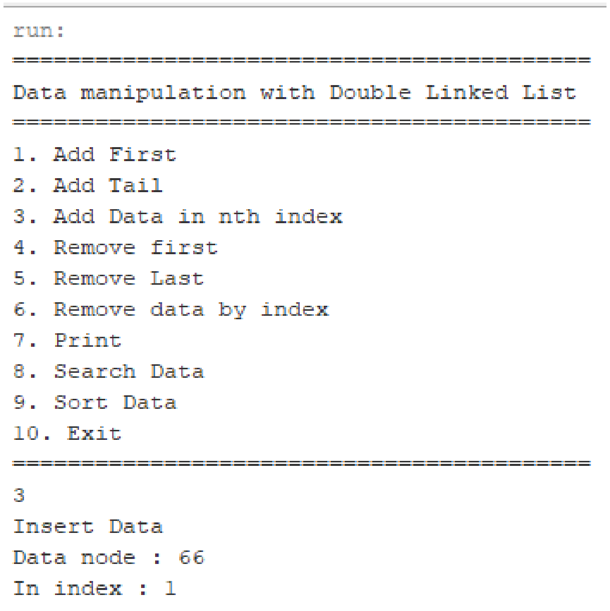


```
run:
============================================
Data manipulation with Double Linked List
============================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
============================================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

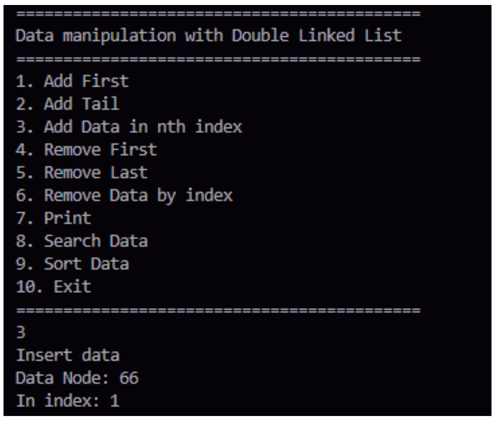


```
3_1466afe3\bin' 'Main11'
============================================
Data manipulation with Double Linked List
============================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove First
5. Remove Last
6. Remove Data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
============================================
1
Insert data in Head position
34
```

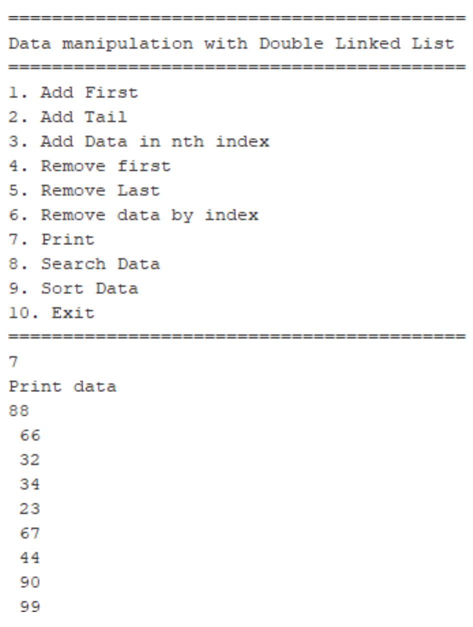
## Add data in specified index and display the result



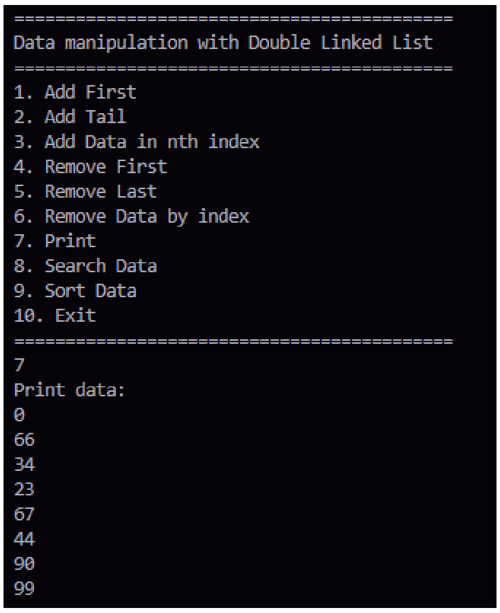
```
run:
============================================
Data manipulation with Double Linked List
============================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
============================================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

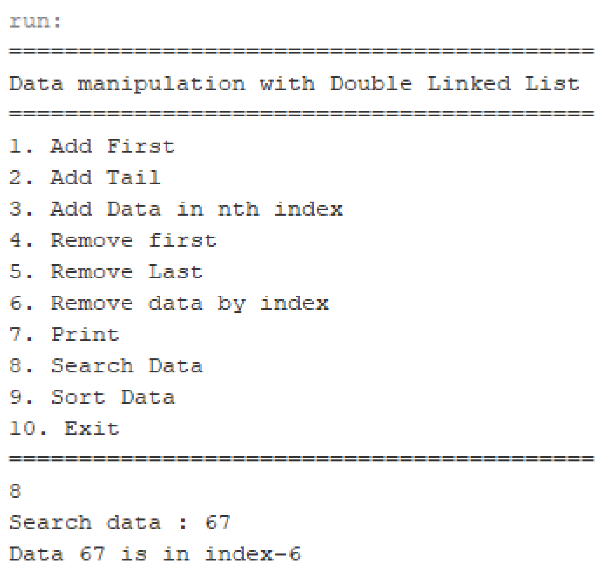


```
============================================
Data manipulation with Double Linked List
============================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
============================================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



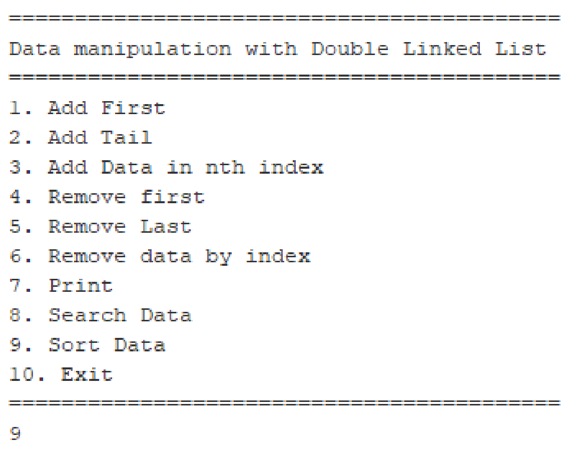
```
============================================
Data manipulation with Double Linked List
============================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove First
5. Remove Last
6. Remove Data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
============================================
7
Print data:
0
66
34
23
67
44
90
99
```

**Search Data**

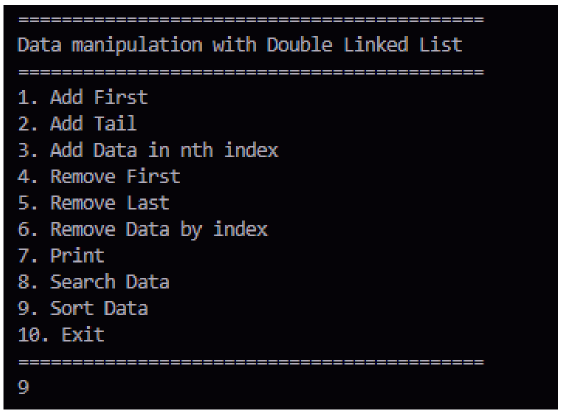


```
run:
============================================
Data manipulation with Double Linked List
============================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
============================================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

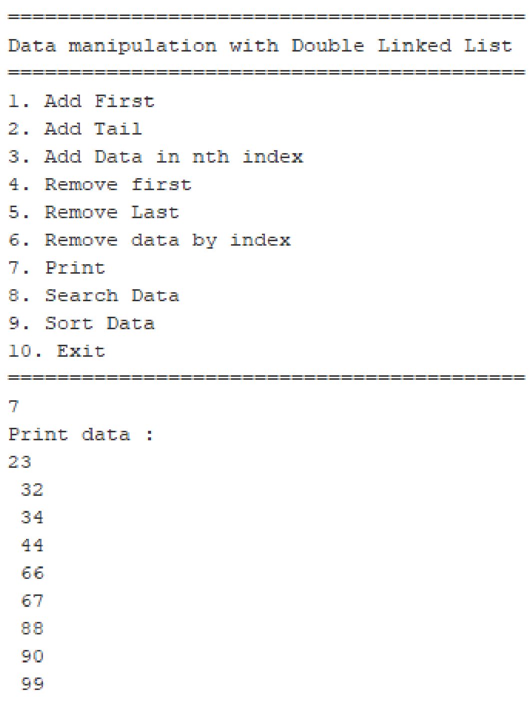
**Sorting Data**



```
============================================
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
============================================
9
```

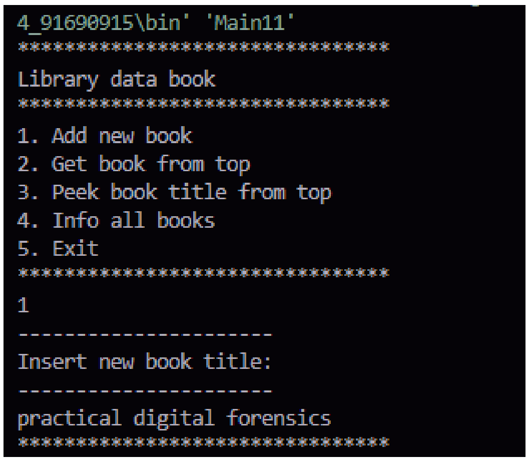


```
============================================
Data manipulation with Double Linked List
============================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove first
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
============================================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

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)**

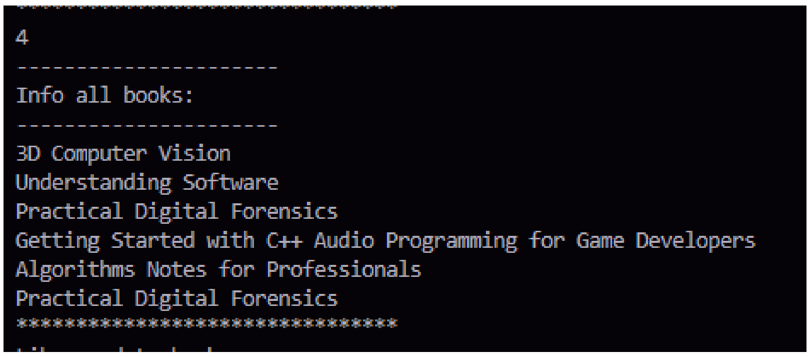
```
****************
Library data book
****************
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
****************
1
-------------------------
Insert new book title
-------------------------
Practical Digital Forensics
```



```
4_91690915\bin' 'Main11'
********************************
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**

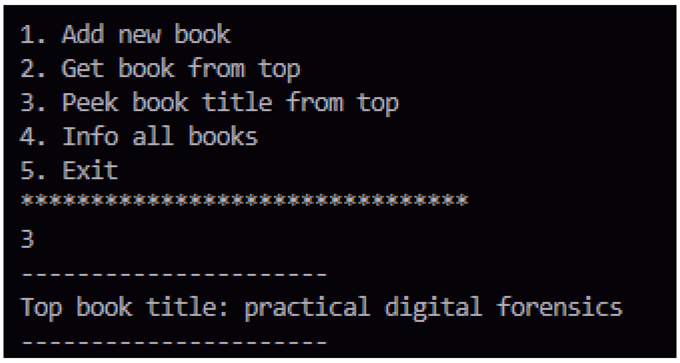
```
****************
Library data book
****************
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
****************
4
-------------------------
Info all books
-------------------------
3D Computer Vision
Understanding Software
Algorithms Notes for Professionals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



```
********************************
4
----------------------
Info all books:
----------------------
3D Computer Vision
Understanding Software
Practical Digital Forensics
Getting Started with C++ Audio Programming for Game Developers
Algorithms Notes for Professionals
Practical Digital Forensics
********************************
```

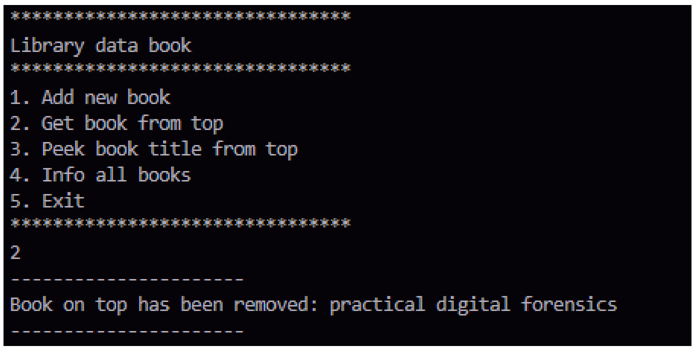
**See the data on top of the stack**

```
****************
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
------------------------
```



```
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: practical digital forensics
----------------------
```

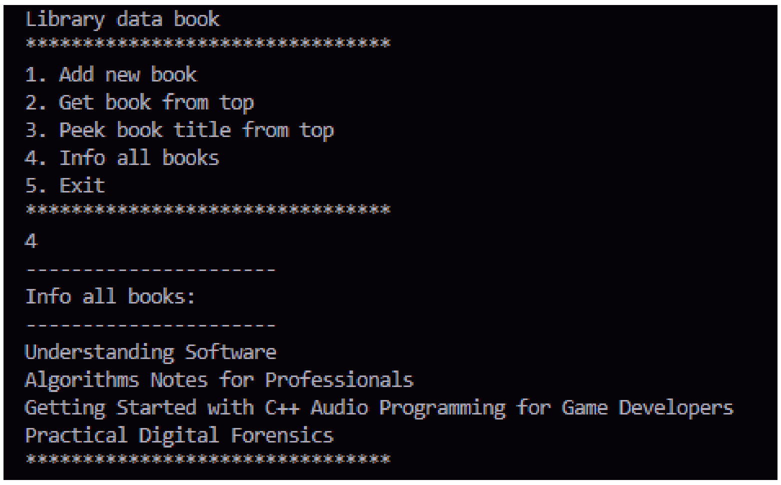
```
****************
Library data book
****************
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
****************
2
------------------------
Book om top has been removed
------------------------
```



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

**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
****************
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)
```

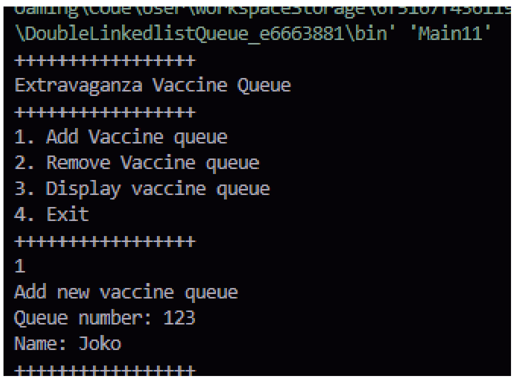


```
Library data book
********************************
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
********************************
4
---------------------
Info all books:
---------------------
Understanding Software
Algorithms Notes for Professionals
Getting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

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**)

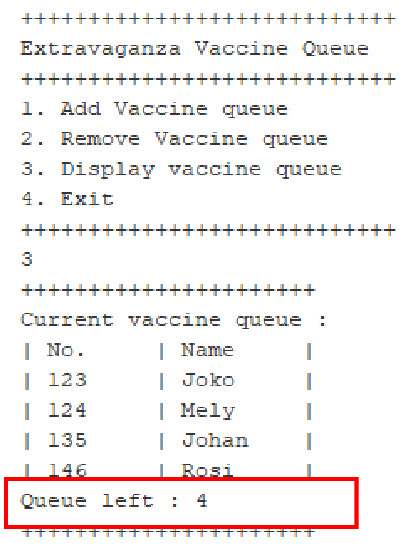
**Initial menu and adding a data**

```
++++++++++++++++++++++++++++
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
```

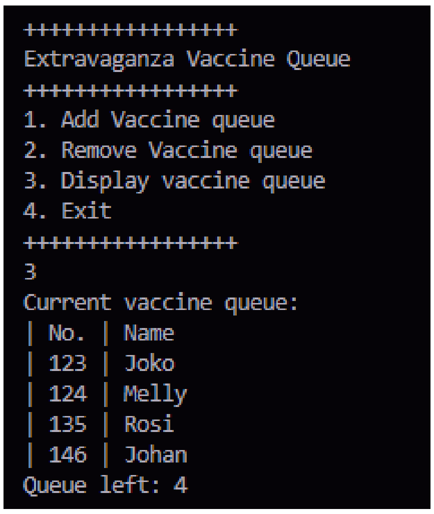


```
aming\Code\User\workspaceStorage\6f5167145611
\DoubleLinkedlistQueue_e6663881\bin' 'Main11'
++++++++++++++++
Extravaganza Vaccine Queue
++++++++++++++++
1. Add Vaccine queue
2. Remove Vaccine queue
3. Display vaccine queue
4. Exit
++++++++++++++++
1
Add new vaccine queue
Queue number: 123
Name: Joko
++++++++++++++++
```

**Print data (notice the highlighted red in the result)**

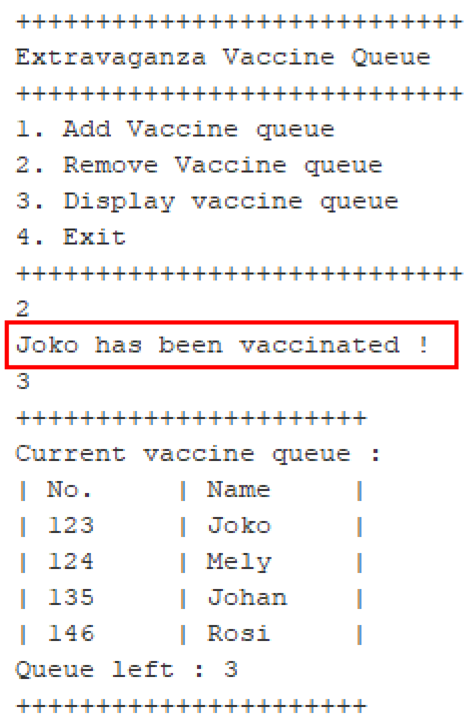


```
++++++++++++++++++++++++++++
Extravaganza Vaccine Queue
++++++++++++++++++++++++++++
1. Add Vaccine queue
2. Remove Vaccine queue
3. Display vaccine queue
4. Exit
++++++++++++++++++++++++++++
3
++++++++++++++++++++++
Current vaccine queue :
| No.      | Name      |
| 123      | Joko      |
| 124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

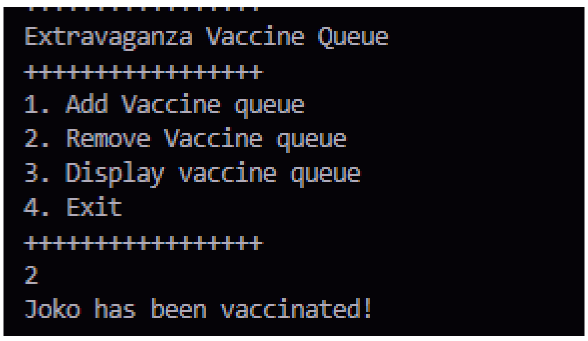


```
+++++++++++++++++
Extravaganza Vaccine Queue
+++++++++++++++++
1. Add Vaccine queue
2. Remove Vaccine queue
3. Display vaccine queue
4. Exit
+++++++++++++++++
3
Current vaccine queue:
| No. | Name
| 123 | Joko
| 124 | Melly
| 135 | Rosi
| 146 | Johan
Queue left: 4
```

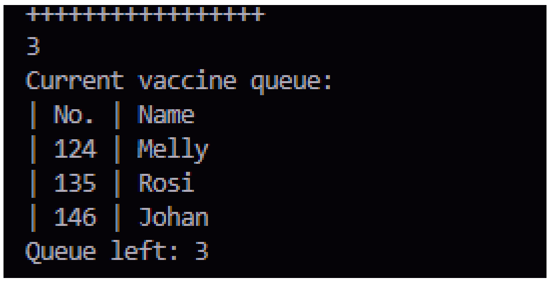
**Remove Data (the highlighted red must displayed in the console too)**



```
++++++++++++++++++++++++++++
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
++++++++++++++++++++++
```



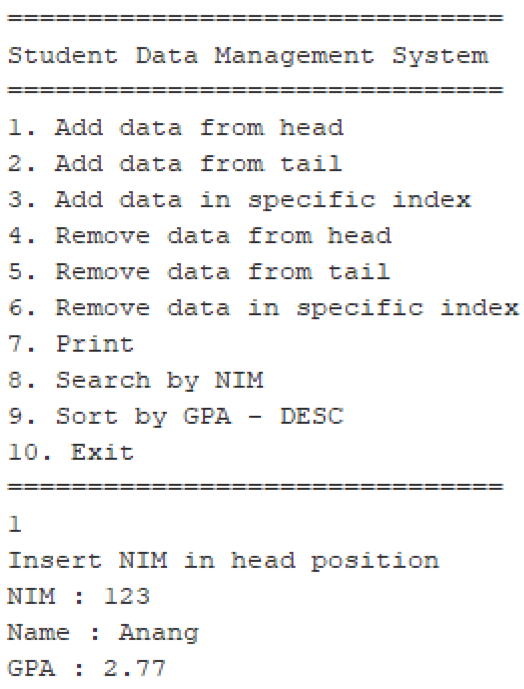
```
+++++++++++++++++
Extravaganza Vaccine Queue
+++++++++++++++++
1. Add Vaccine queue
2. Remove Vaccine queue
3. Display vaccine queue
4. Exit
+++++++++++++++++
2
Joko has been vaccinated!
```



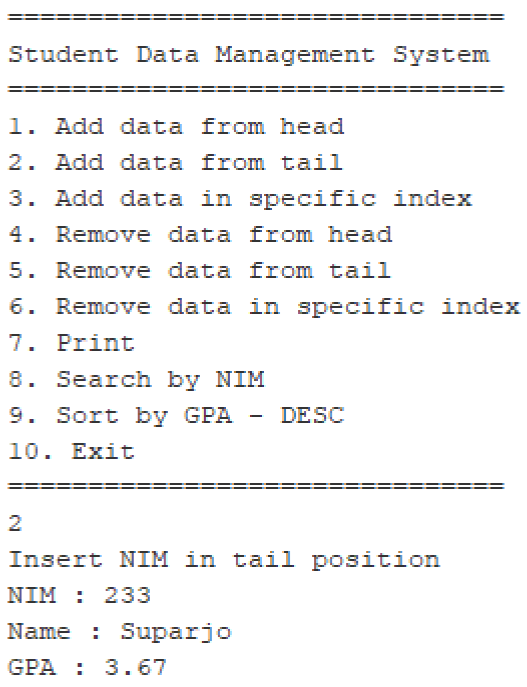
```
+++++++++++++++++
3
Current vaccine queue:
| No. | Name
| 124 | Melly
| 135 | Rosi
| 146 | Johan
Queue left: 3
```

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**

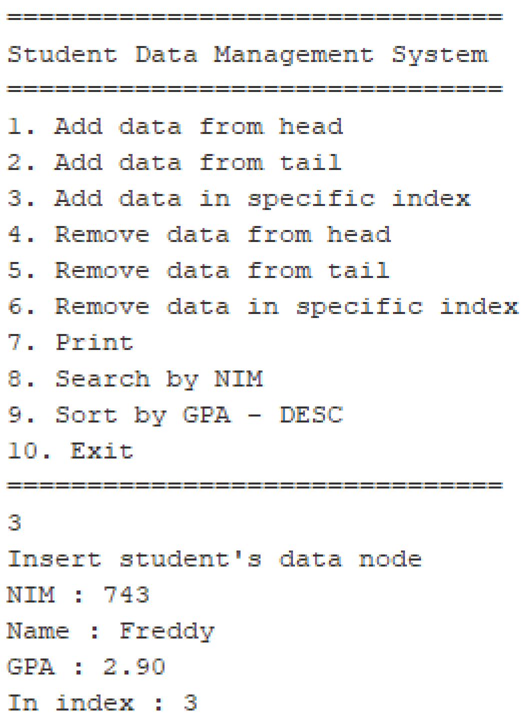
**Initial menu and adding data**



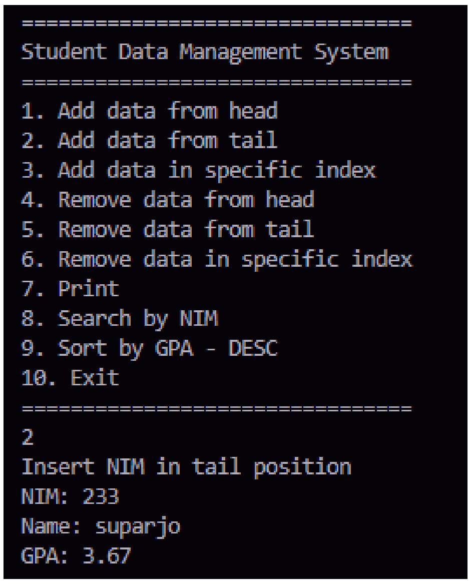
```
=================================
Student Data Management System
=================================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA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



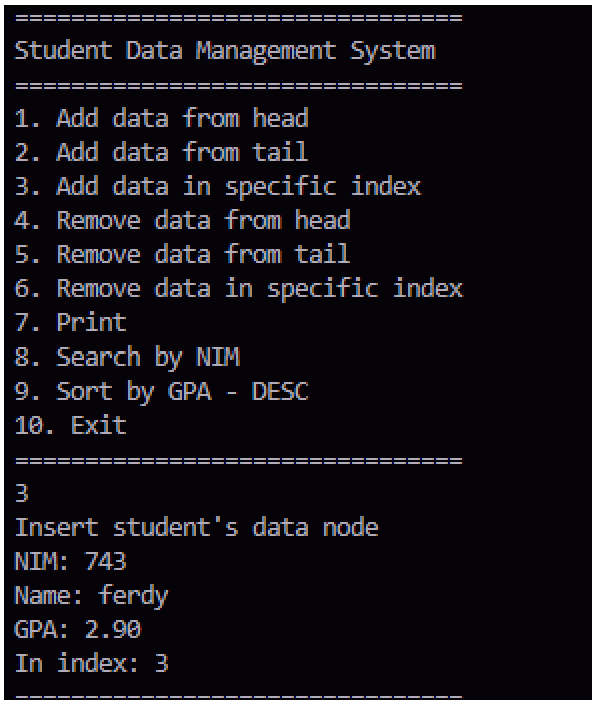
```
=================================
Student Data Management System
=================================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA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



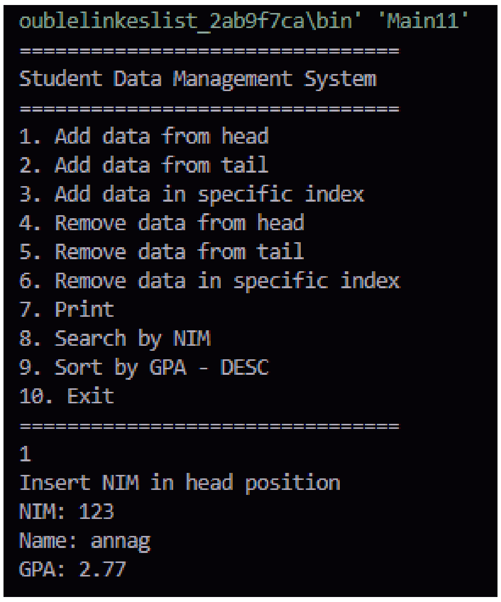
```
=================================
Student Data Management System
=================================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA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



```
================================
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
```



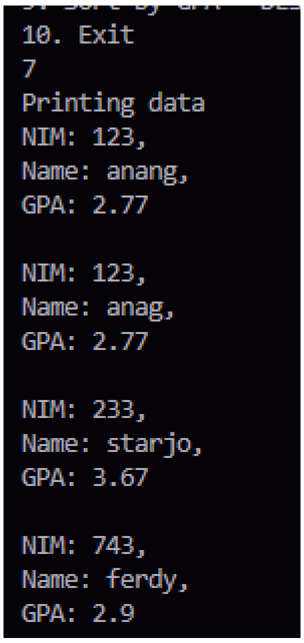
```
================================
Student Data Management System
================================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA -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



```
oublelinkeslist_2ab9f7ca\bin' 'Main11'
================================
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: annag
GPA: 2.77
```

**Printing data**

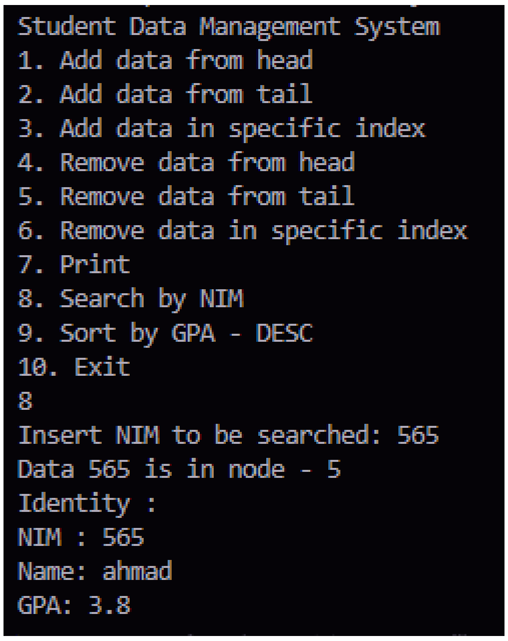
```
================================
Student Data Management System
================================
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA -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



```
10. Exit
7
Printing data
NIM: 123,
Name: anang,
GPA: 2.77

NIM: 123,
Name: anag,
GPA: 2.77

NIM: 233,
Name: starjo,
GPA: 3.67

NIM: 743,
Name: ferdy,
GPA: 2.9
```

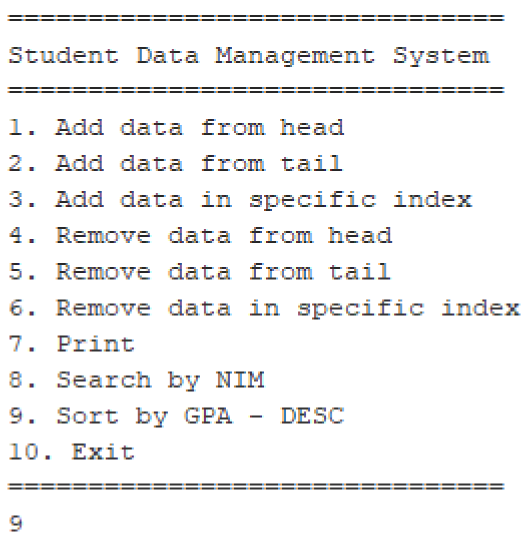
**Searching data**

```
================================
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
```

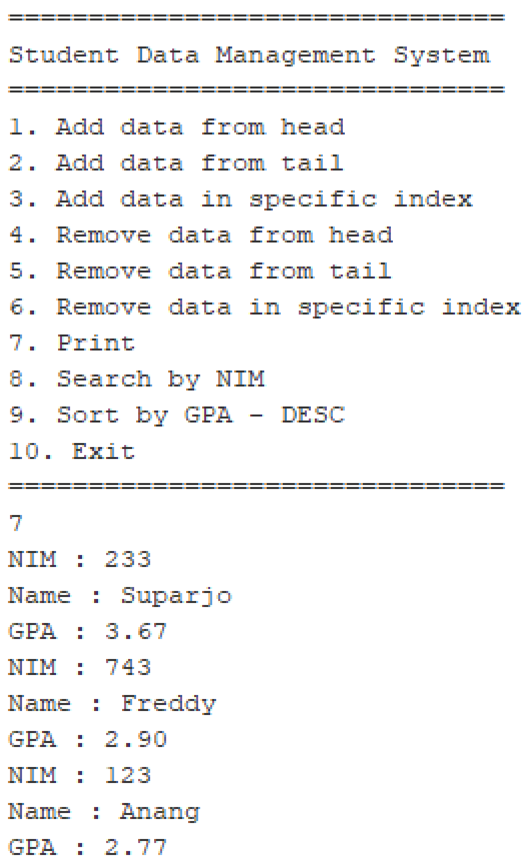


```
Student Data Management System
1. Add data from head
2. Add data from tail
3. Add data in specific index
4. Remove data from head
5. Remove data from tail
6. Remove data in specific index
7. Print
8. Search by NIM
9. Sort by GPA - DESC
10. Exit
8
Insert NIM to be searched: 565
Data 565 is in node - 5
Identity :
NIM : 565
Name: ahmad
GPA: 3.8
```

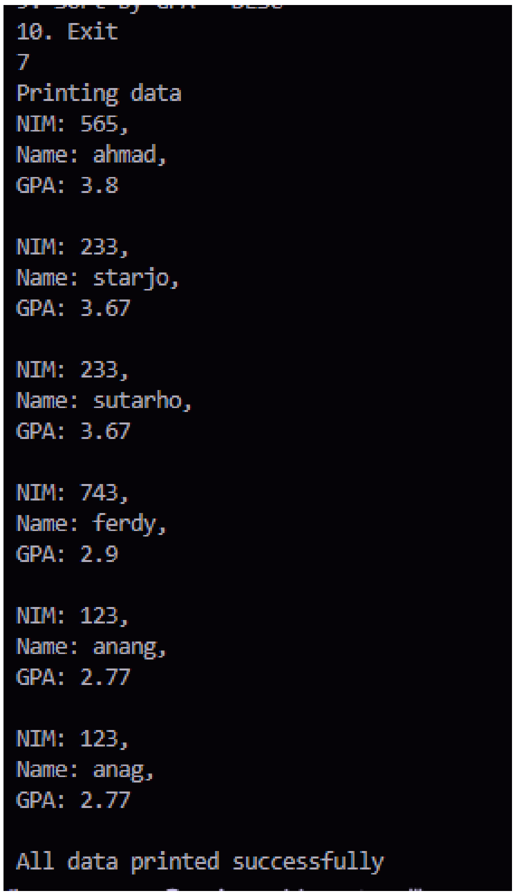
**Sorting data**



```
================================
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
```



```
10. Exit
7
Printing data
NIM: 565,
Name: ahmad,
GPA: 3.8

NIM: 233,
Name: starjo,
GPA: 3.67

NIM: 233,
Name: sutarho,
GPA: 3.67

NIM: 743,
Name: ferdy,
GPA: 2.9

NIM: 123,
Name: anang,
GPA: 2.77

NIM: 123,
Name: anag,
GPA: 2.77

All data printed successfully
```