Week 12

Subject	Data Structure and Algorithm
Lecturer	Imam Fahrur Rozi ST. MT.
Type	Assignment
Semester	Semester 2
Time	@May 22, 2023
Files & Media	

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

Jobsheet 12

activities 1

```
package JB12.Prac;

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

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

```
package JB12.Prac;

public class DoubleLinkedLists
```

```
{
    Node head;
    int size;

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

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

    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++;
    }

    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++;
        }
    }

    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++;
    }

    int size()
    {
        return size;
    }

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

    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");
    }
}
```

```
package JB12.Prac;

public class DoubleLinkedListMain
{
    public static void main(String[] args) throws Exception {
        DoubleLinkedLists dll = new DoubleLinkedLists();
        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);
        dll.addLast(50);
        dll.addLast(40);
        dll.addLast(10);
        dll.addLast(20);
        dll.print();
        System.out.println("Size : " + dll.size);
        System.out.println("==================================");
    }
}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--;
        }
    }

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

    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--;
        }
    }
```

questions 1

1. single link list only uses one node on each list so if we want to access a data, we have to go through from the head, while double linked list uses two node on each data so if we want to access a certain data that is before the data that we are accessing, we can just go back using `prev`
2. `next` is used to go to the next data, while `prev` is used to access the previous data
3. it is used to construct the `DoubleLinkedLists` class so we know that since the list is still empty, the head must be `null` while the size is still at 0
4. because the previous data of the first data should be `null` or empty
5. the `current` is used to determine the last list and the `null` is used to identify that this method is used to add in the last lists so the next data should be `null` or none

activities 2

```
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--;
        }
    }

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

    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--;
        }
    }
```

```
        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);
```

questions 2

1. **yes**

2. first we check whether the list is only one or not, if it only one, then we remove the `head` then reduce the size by 1. if it more than one, we create a new temporary data called `current` set as the same as the `head` after that we loop the `current.next` `.next` to check whether if it's a `null` or not, if it isn't a null, then we change the `current` as a `current.next` if the loop is stopped, that means we already found the last 2 index, which become the `current` and the last index, so we remove the last index using `current.next = null;`

3. if we use the following code, it won't remove the data that we wanted to remove, because it will only skipping the data that we wanted to remove, but the data is still accessible

4. so the code is used to remove the `current` data by changing the `current` into `current.next` and `current.prev`

activities 3

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

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

    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;
```

```
        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));
```

questions 3

1. to know how big is the size of the linked list
2. we need to set the first index as 1 in the constructor. this will make the consequent index to start from 1
3. In double linked list, we need to also handle the previous node. while in single linked list we can just attach the new node and only care about the next reference, we also need to care about the prev node in double linked list
4. the first one checks if the list is empty by checking its size while the second one checks if the head is null or not

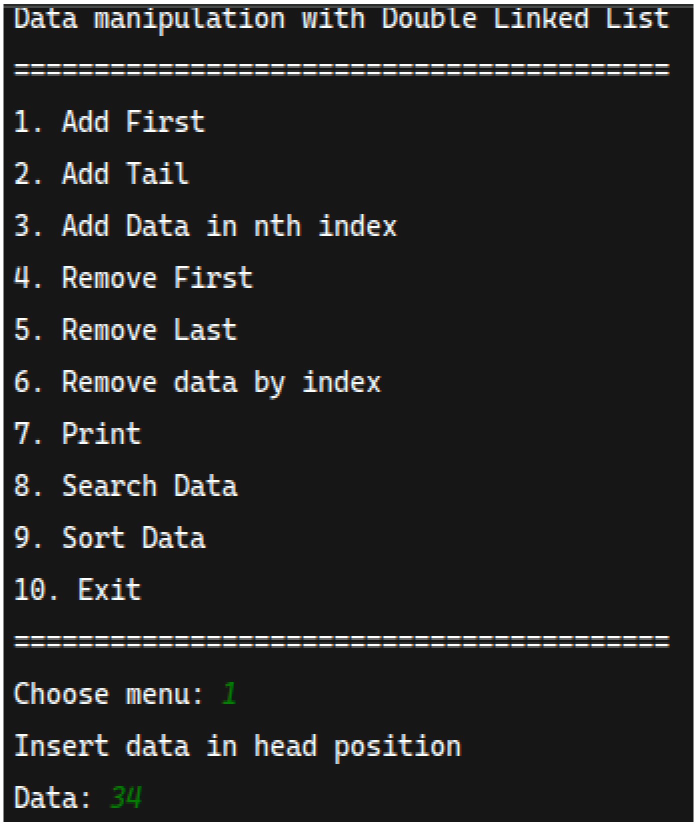
assignment

1. Codes

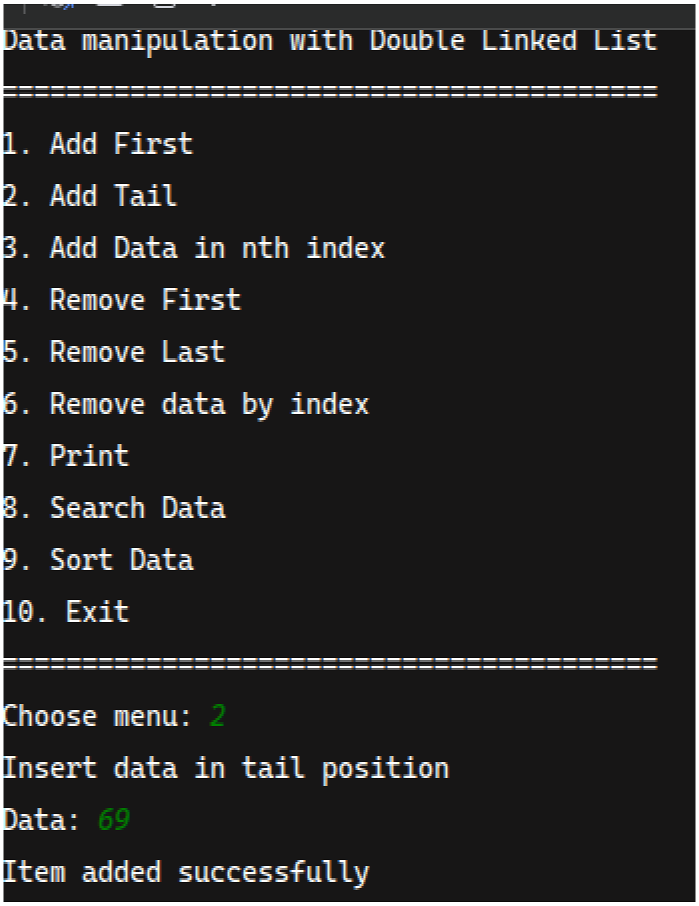
```
package JB12.Asg1;

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

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

```
package JB12.Asg1;

public class DoubleLinkedLists
{
    Node head;
    int size;

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

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

    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++;
    }

    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++;
        }
    }

    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++;
    }

    int size()
    {
        return size;
    }

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

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


    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--;
        }
    }

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

    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--;
        }
    }

    //act 3 starts from here

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

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

    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;
    }

    int sequentialSearch(int search) throws Exception
    {
        if (isEmpty()) throw new Exception("Linked list is still empty");
        Node tmp = head;
        for (int i = 0; i < size; i++)
        {
            if (tmp.data == search) return i;
            else tmp = tmp.next;
        }
        return -1;
    }

    void bubbleSort()
    {
        boolean swapped;
        Node current;
        Node tmp = head;

        do
        {
            swapped = false;
            current = head;

            while (current.next != null)
            {
                if (current.data > current.next.data)
                {
                    int temp = tmp.data;
                    tmp.data = tmp.next.data;
                    tmp.next.data = temp;
                    swapped = true;
                }
                current.data = current.next.data;
            }
        }
        while (swapped);
    }
}
```

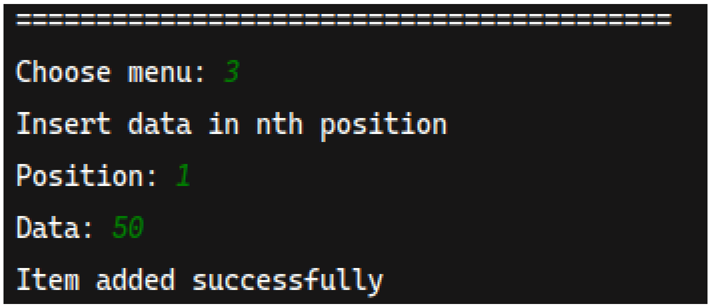
```
package JB12.Asg1;

import java.util.Scanner;

public class Main
{
    static Scanner sc = new Scanner(System.in);
    public static void main(String[] args) throws Exception {
        DoubleLinkedLists dll = new DoubleLinkedLists();
        int menu, data, index;
        do
        {
            System.out.println("=========================================");
            System.out.println("Data manipulation with Double Linked List");
            System.out.println("=========================================");
            System.out.println("1. Add First");
            System.out.println("2. Add Tail");
            System.out.println("3. Add Data in nth 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("=========================================");
            menu = sc.nextInt();
            switch (menu)
            {
                case 1:
                    System.out.println("Insert Data in Head position");
                    data = sc.nextInt();
                    dll.addFirst(data);
                    break;
                case 2:
                    System.out.println("Insert Data in Tail position");
                    data = sc.nextInt();
                    dll.addLast(data);
                    break;
                case 3:
                    System.out.println("Insert Data");
                    System.out.print("Data node : ");
                    data = sc.nextInt();
                    System.out.print("In index : ");
                    index = sc.nextInt();
                    dll.add(data, index);
                    break;
                case 4:
                    System.out.println("Removed First Data (" + dll.getFirst() + ")");
                    dll.removeFirst();
                    break;
                case 5:
                    System.out.println("Removed Last Data (" + dll.getLast(0));
                    dll.removeLast();
                    break;
                case 6:
                    System.out.println("Remove Data");
                    System.out.print("In Index : ");
                    index = sc.nextInt();
                    dll.remove(index);
                    break;
                case 7:
                    dll.print();
                    break;
                case 8:
                    System.out.print("Search Data : ");
                    int search = sc.nextInt();
                    int pos = dll.sequentialSearch(search);
                    if (pos == -1) System.out.println("Data: " + search + " isn't found");
                    else System.out.println("Data: " + search + " found at index-" + pos);
                    break;
                case 9:
                    System.out.println("Sort Data");
                    dll.bubbleSort();
                    System.out.println("Print Data :");
                    dll.print();
            }
        }
        while (menu != 10);
    }
}
```



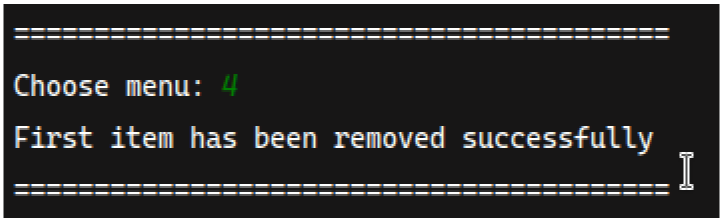
```
Data manipulation with Double Linked List
=========================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove First
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
=========================================
Choose menu: 1
Insert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



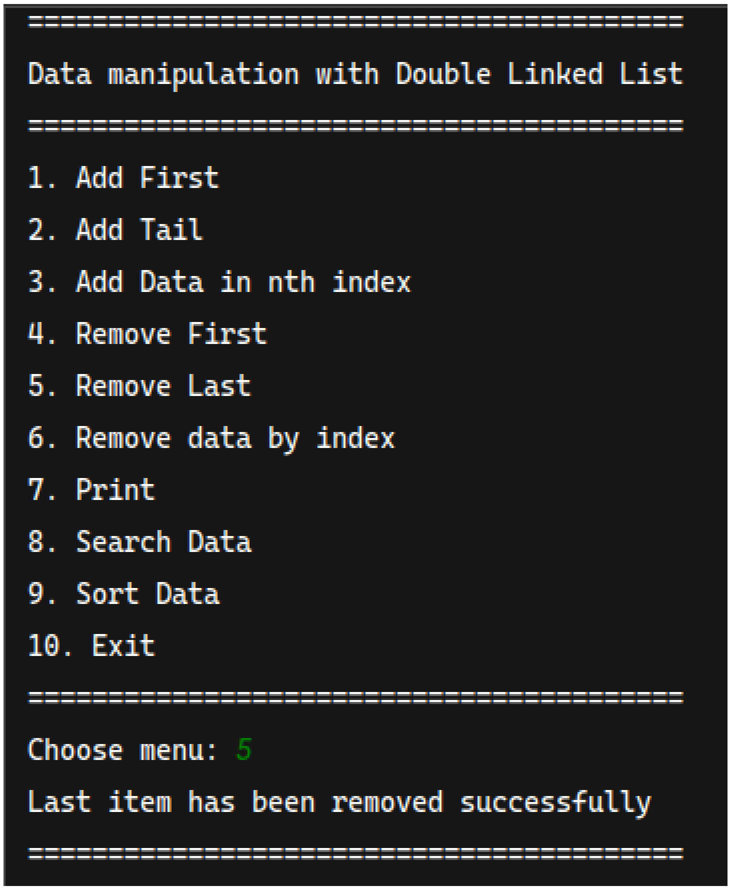
```
Data manipulation with Double Linked List
=========================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove First
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
=========================================
Choose menu: 2
Insert data in tail position
Data: 69
Item added successfully
```



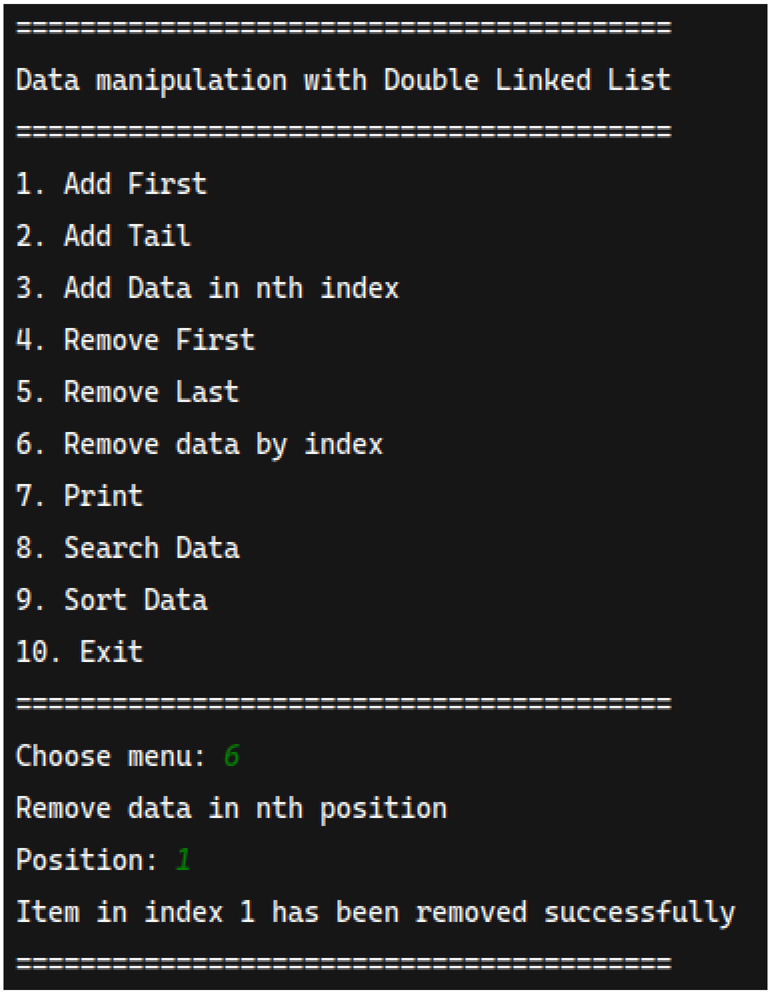
```
=========================================
Choose menu: 3
Insert data in nth position
Position: 1
Data: 50
Item added successfully
```



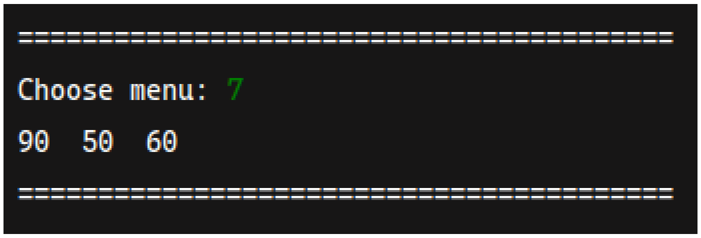
```
=========================================
Choose menu: 4
First item has been removed successfully
=========================================
```



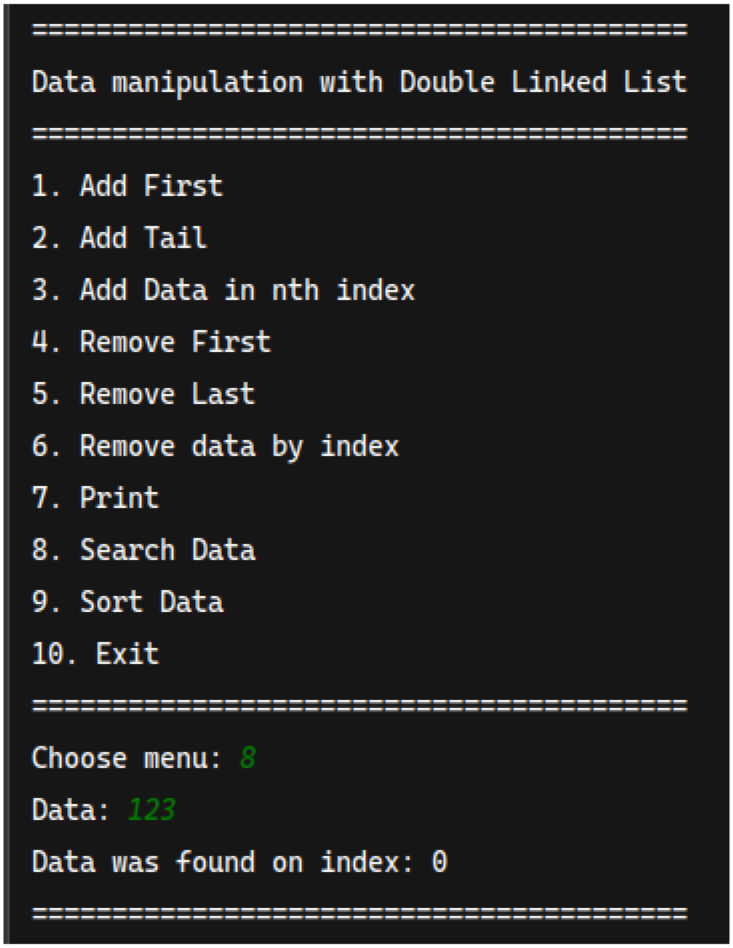
```
=========================================
Data manipulation with Double Linked List
=========================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove First
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
=========================================
Choose menu: 5
Last item has been removed successfully
=========================================
```



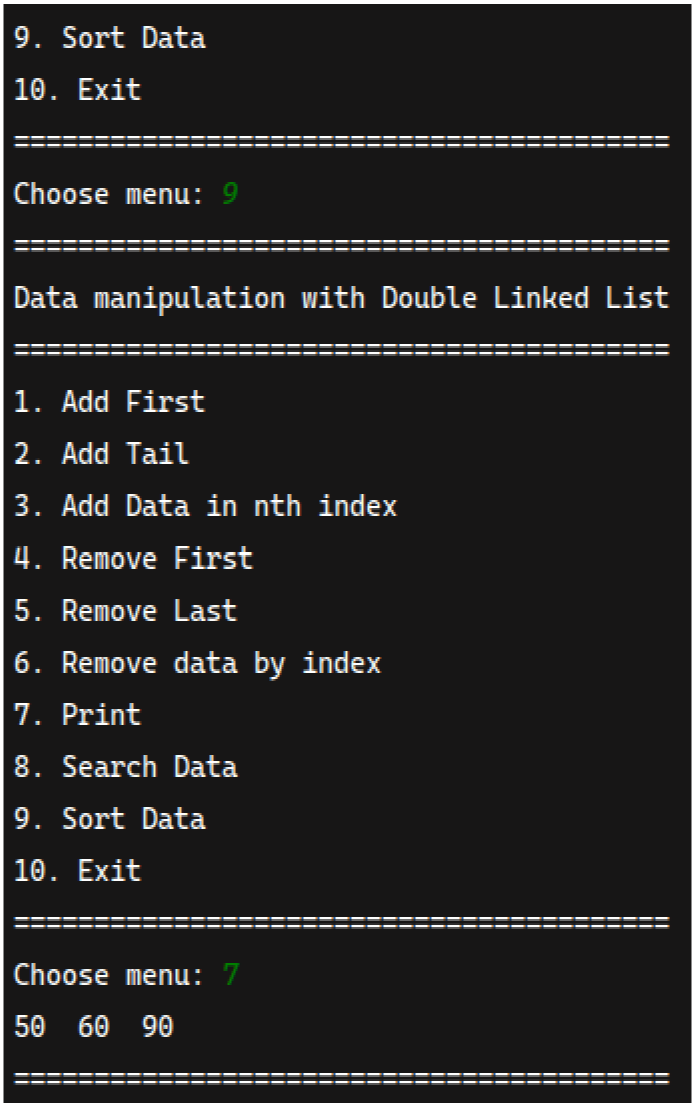
```
=========================================
Data manipulation with Double Linked List
=========================================
1. Add First
2. Add Tail
3. Add Data in nth index
4. Remove First
5. Remove Last
6. Remove data by index
7. Print
8. Search Data
9. Sort Data
10. Exit
=========================================
Choose menu: 6
Remove data in nth position
Position: 1
Item in index 1 has been removed successfully
=========================================
```



```
=========================================
Choose menu: 7
90  50  60
=========================================
```



```
=========================================
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
=========================================
Choose menu: 8
Data: 123
Data was found on index: 0
=========================================
```



```
9. Sort Data
10. Exit
=========================================
Choose menu: 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
=========================================
Choose menu: 7
50  60  90
=========================================
```

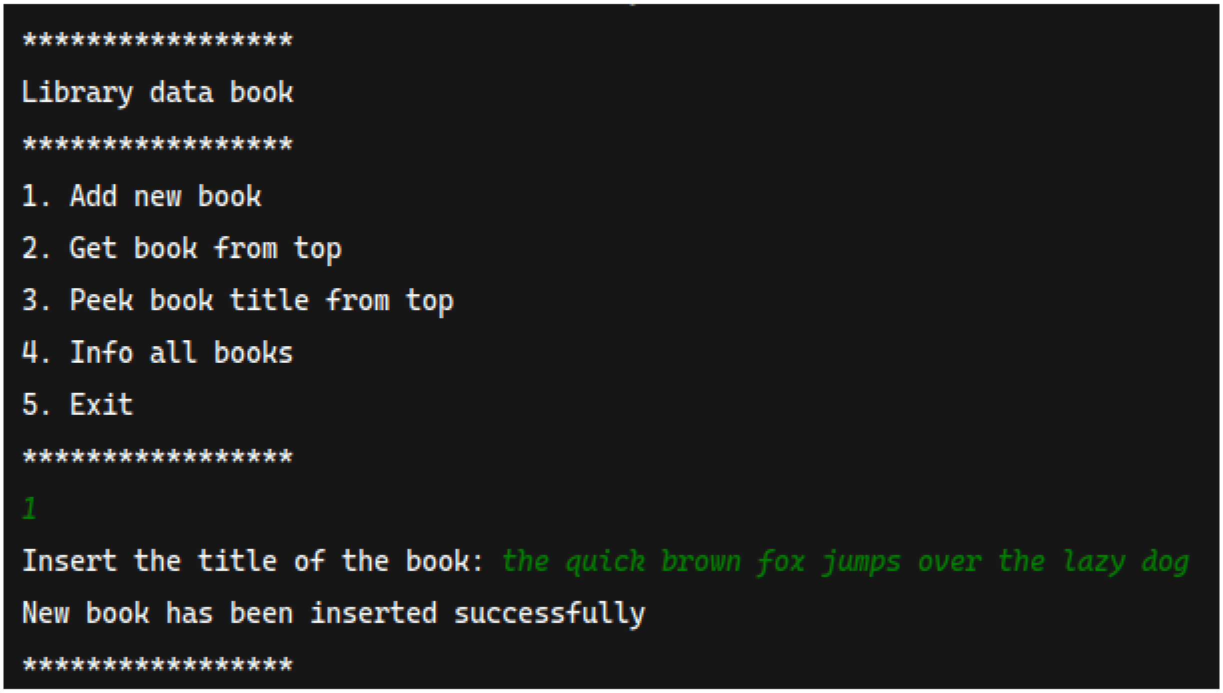
2. Code

```
public class StudentList
{
    private final DoubleLinkedList<Student> list;
    public StudentList()
    {
        list = new DoubleLinkedList<>();
    }
    // add data from head
    public void addFirst(Student data)
    {
        list.addFirst(data);
    }
    // add data from tail
    public void addLast(Student data)
    {
        list.addLast(data);
    }
    // add data in specific index from head
    public void addFrom(Student data, int index) throws Exception
    {
        list.addItem(data, index);
    }
    // remove data from head
    public void removeFirst() throws Exception
    {
        list.removeFirst();
    }
    // remove data from tail
    public void removeLast() throws Exception
    {
        list.removeLast();
    }
    // remove data in specific index
    public void remove(int index) throws Exception
    {
        list.remove(index);
    }
    // print
    public void print()
    {
        Node<Student> current = list.head;
        while (current != null)
        {
            System.out.println("| " + current.data.nim + " | " + current.data.name + "
            current = current.next;
        }
        System.out.println();
    }
    // search by nim
    public int search(String nim)
    {
        if (list.isEmpty()) return -1;
        Node<Student> current = list.head;
        int i = 0;
        while (current != null)
        {
            if (current.data.nim.equals(nim))
            return i;
            current = current.next;
            i++;
        }
        return -1;
    // sort by gpa - desc
    public void sortByGpa()
    {
        if (list.head == null || list.head.next == null)
        {
            return; // No need to sort if the list is empty or has only one element
        }
        boolean swapped;
        Node<Student> current;
        Node<Student> last = null;
        do
        {
            swapped = false;
            current = list.head;
            while (current.next != last)
            {
                if (current.data.gpa > current.next.data.gpa)
                {
                    swap(current, current.next);
                    swapped = true;
                }
                current = current.next;
            }
        }

        last = current;
        }
        while (swapped);
    }
    static void swap(Node<Student> left, Node<Student> right)
    {
        Student tmp = left.data;
        left.data = right.data;
        right.data = tmp;
    }
}
```

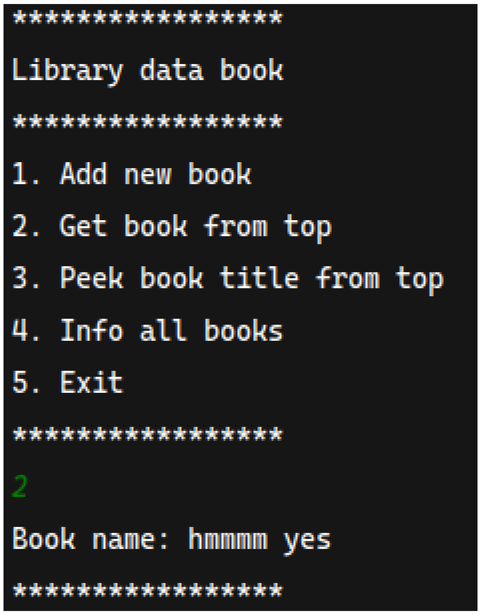
```
public class StudentMain {
public static void main(String[] args) throws Exception {
Scanner scanner = new Scanner(System.in);
StudentList studentList = new StudentList();
while (true) {
showMenu();
int chosenMenu = scanner.nextInt();
switch (chosenMenu) {
case 1: {
System.out.println("Add data from head");
System.out.print("NIM: ");
scanner.nextLine();
String nim = scanner.nextLine();
System.out.print("Name: ");
String name = scanner.nextLine();
System.out.print("GPA: ");
double gpa = scanner.nextDouble();
Student student = new Student(nim, name, gpa);
studentList.addFirst(student);
break;
```

```
}
case 2: {
System.out.println("Add data from tail");
System.out.print("NIM: ");
scanner.nextLine();
String nim = scanner.nextLine();
System.out.print("Name: ");
String name = scanner.nextLine();
System.out.print("GPA: ");
double gpa = scanner.nextDouble();
Student student = new Student(nim, name, gpa);
studentList.addLast(student);
break;
}
case 3: {
System.out.println("Add data to specific index");
System.out.print("Index: ");
int index = scanner.nextInt();
scanner.nextLine();
System.out.print("NIM: ");
String nim = scanner.nextLine();
System.out.print("Name: ");
String name = scanner.nextLine();
System.out.print("GPA: ");
double gpa = scanner.nextDouble();
Student student = new Student(nim, name, gpa);
studentList.addFrom(student, index);
break;
}
case 4: {
System.out.println("Remove data from head");
studentList.removeFirst();
break;
}
case 5: {
System.out.println("Remove data from tail");
studentList.removeLast();
break;
}
case 6: {
System.out.println("Remove data in specific index");
System.out.print("Index: ");
int index = scanner.nextInt();
studentList.remove(index);
break;
}
case 7: {
System.out.println("Print");
studentList.print();
break;
}
case 8: {
System.out.println("Search by NIM");
System.out.print("NIM: ");
```

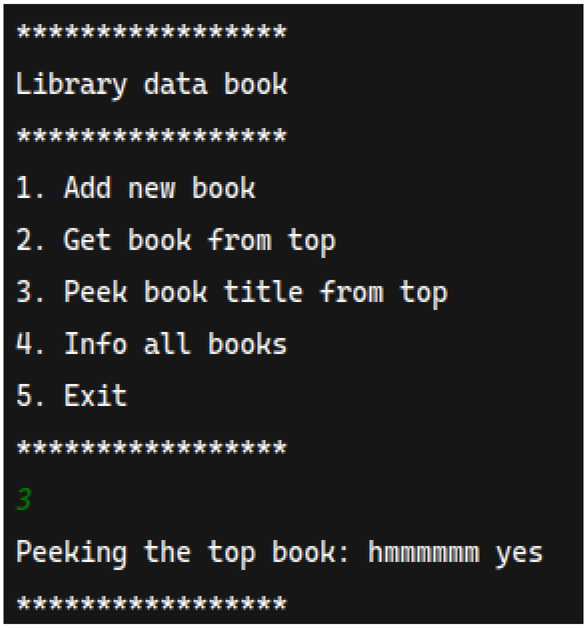
```
scanner.nextLine();
String nim = scanner.nextLine();
System.out.println("Index: " + studentList.search(nim));
break;
}
case 9: {
System.out.println("Sort by GPA");
studentList.sortByGpa();
break;
}
case 10: {
System.out.println("Exit");
return;
}
}
}
}

public static void showMenu() {
System.out.println("==============================");
System.out.println("Student Data Management System");
System.out.println("==============================");
System.out.println("1. Add data from head");
System.out.println("2. Add data from tail");
System.out.println("3. Add data to specific index");
System.out.println("4. Remove data from head");
System.out.println("5. Remove data from tail");
System.out.println("6. Remove data from specific 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("==============================");
}
}
```



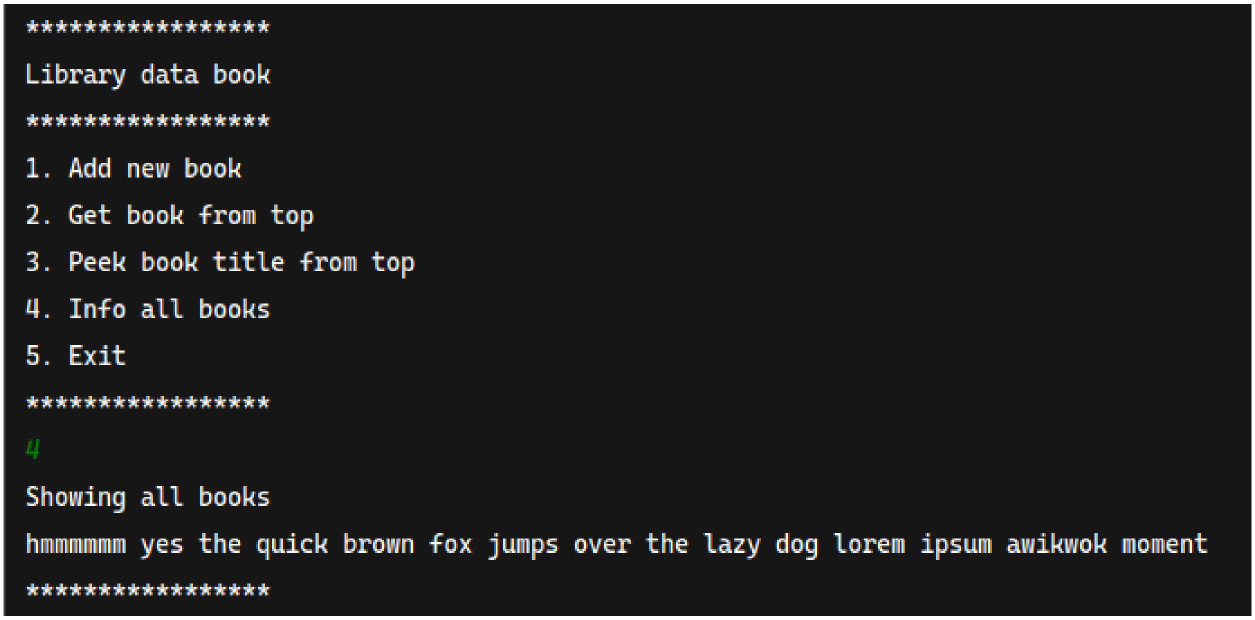
```
*****************
Library data book
*****************
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
*****************
1
Insert the title of the book: the quick brown fox jumps over the lazy dog
New book has been inserted successfully
*****************
```



```
*****************
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 name: hmmmm yes
*****************
```



```
*****************
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
Peeking the top book: hmmmmmm yes
*****************
```

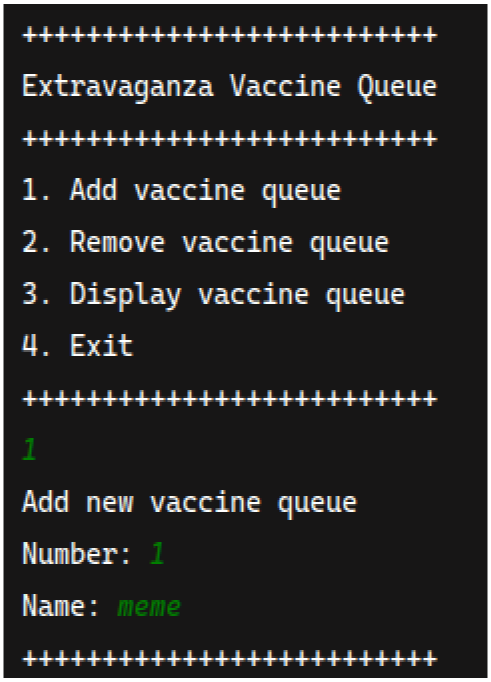


```
*****************
Library data book
*****************
1. Add new book
2. Get book from top
3. Peek book title from top
4. Info all books
5. Exit
*****************
4
Showing all books
hmmmmmm yes the quick brown fox jumps over the lazy dog lorem ipsum awikwok moment
*****************
```

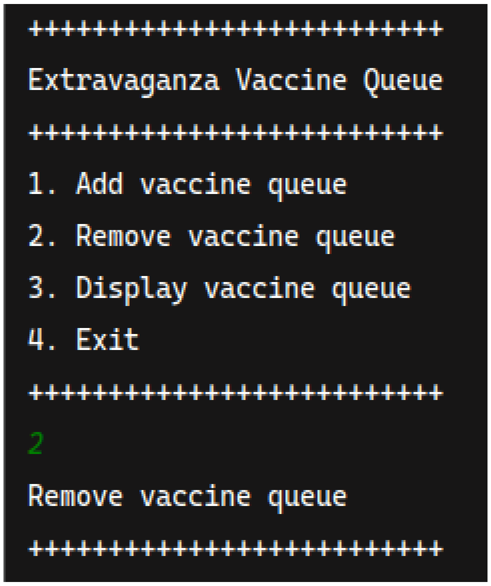
3. Code

```
public class Queue {
private final DoubleLinkedList<Patient> list;
private int maxCapacity;
private int currentCapacity;
public Queue(int maxQueue) {
this.maxCapacity = maxQueue;
this.currentCapacity = 0;
this.list = new DoubleLinkedList<>();
}
public void add(Patient data) {
if (currentCapacity >= maxCapacity) {
System.out.println("List is already maxed out");
return;
}
}
list.addLast(data);
maxCapacity--;
public void remove() throws Exception {
if (currentCapacity == 0) {
}
}
list.removeFirst();
maxCapacity++;
public void display() {
System.out.println("+++++++++++++++++++++++");
System.out.println("Current vaccine queue: ");
System.out.println("| No.\t\t | Name\t\t |");
Node<Patient> tmp = list.head;
while (tmp.next != null) {
System.out.printf("| %d\t\t | %s\t\t |\n", tmp.data.number, tmp.data.name);
tmp = tmp.next;
}
System.out.printf("Queue left: %d\n", maxCapacity);
System.out.println("+++++++++++++++++++++++");
}
}
```

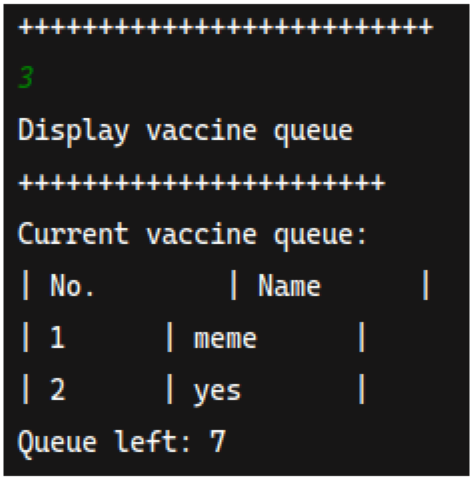
```
public class QueueMain {
public static void main(String[] args) throws Exception {
Scanner scanner = new Scanner(System.in);
Queue patientQueue = new Queue(10);
while (true) {
showMenu();
int chosenMenu = scanner.nextInt();
switch (chosenMenu) {
case 1: {
System.out.println("Add new vaccine queue");
System.out.print("Number: ");
int number = scanner.nextInt();
System.out.print("Name: ");
scanner.nextLine();
String name = scanner.nextLine();
patientQueue.add(new Patient(number, name));
break;
}
case 2: {
System.out.println("Remove vaccine queue");
patientQueue.remove();
break;
}
case 3: {
System.out.println("Display vaccine queue");
patientQueue.display();
break;
}
case 4: {
System.out.println("Exit");
return;
}
}
}
}
static void showMenu() {
System.out.println("++++++++++++++++++++++++++");
System.out.println("Extravaganza Vaccine Queue");
System.out.println("++++++++++++++++++++++++++");
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("++++++++++++++++++++++++++");
}
}
```



```
++++++++++++++++++++++++++
Extravaganza Vaccine Queue
++++++++++++++++++++++++++
1. Add vaccine queue
2. Remove vaccine queue
3. Display vaccine queue
4. Exit
++++++++++++++++++++++++++
1
Add new vaccine queue
Number: 1
Name: meme
++++++++++++++++++++++++++
```



```
++++++++++++++++++++++++++
Extravaganza Vaccine Queue
++++++++++++++++++++++++++
1. Add vaccine queue
2. Remove vaccine queue
3. Display vaccine queue
4. Exit
++++++++++++++++++++++++++
2
Remove vaccine queue
++++++++++++++++++++++++++
```



```
++++++++++++++++++++++++++
3
Display vaccine queue
+++++++++++++++++++++++
Current vaccine queue:
| No.         | Name      |
| 1       | meme      |
| 2       | yes       |
Queue left: 7
```

4. Code

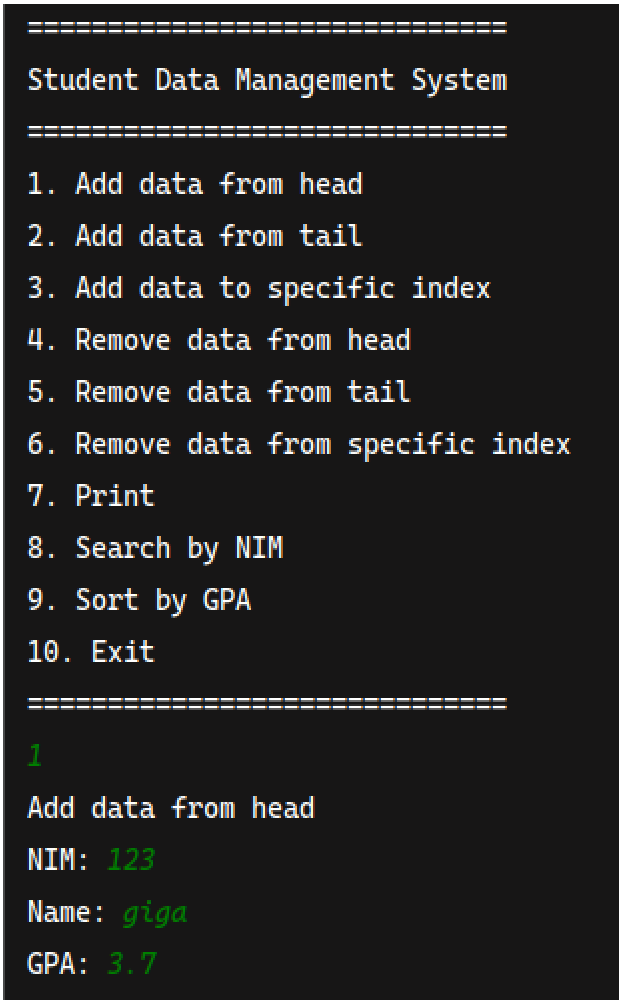
```
public class Student {
public String nim;
public String name;
public double gpa;
}
public Student(String nim, String name, double gpa) {
this.nim = nim;
this.name = name;
this.gpa = gpa;
}
}
```

```
public class StudentList {
private final DoubleLinkedList<Student> list;
public StudentList() {
list = new DoubleLinkedList<>();
}
// add data from head
public void addFirst(Student data) {
list.addFirst(data);
}
// add data from tail
public void addLast(Student data) {
list.addLast(data);
}
// add data in specific index from head
public void addFrom(Student data, int index) throws Exception {
list.addItem(data, index);
}
// remove data from head
public void removeFirst() throws Exception {
list.removeFirst();
}
// remove data from tail
public void removeLast() throws Exception {
list.removeLast();
}
// remove data in specific index
public void remove(int index) throws Exception {
list.remove(index);
}
// print
public void print() {
Node<Student> current = list.head;
while (current != null) {
System.out.println("| " + current.data.nim + " | " + current.data.name + "
current = current.next;
}
System.out.println();
}
// search by nim
public int search(String nim) {
if (list.isEmpty()) {
return -1;
}
}
Node<Student> current = list.head;
int i = 0;
while (current != null) {
if (current.data.nim.equals(nim)) {
return i;
}
current = current.next;
i++;
}
return -1;
// sort by gpa - desc
public void sortByGpa() {
if (list.head == null || list.head.next == null) {
return; // No need to sort if the list is empty or has only one element
}
boolean swapped;
Node<Student> current;
Node<Student> last = null;
do {
swapped = false;
current = list.head;
while (current.next != last) {
if (current.data.gpa > current.next.data.gpa) {
swap(current, current.next);
swapped = true;
}
current = current.next;
}
}
last = current;
} while (swapped);
}
static void swap(Node<Student> left, Node<Student> right) {
Student tmp = left.data;
left.data = right.data;
right.data = tmp;
}
}
```

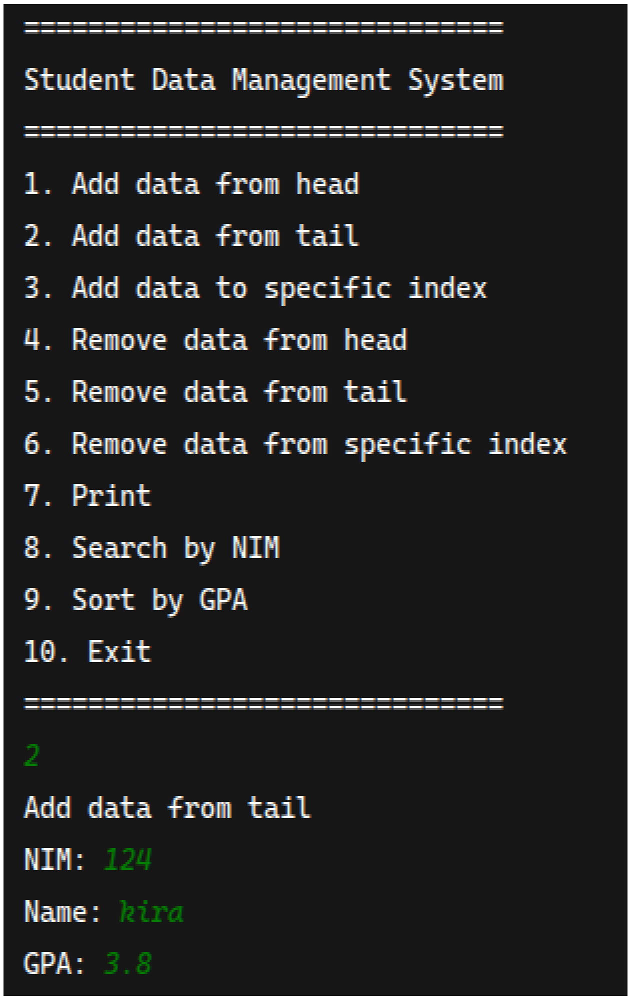
```
public class StudentMain {
public static void main(String[] args) throws Exception {
Scanner scanner = new Scanner(System.in);
StudentList studentList = new StudentList();
while (true) {
showMenu();
int chosenMenu = scanner.nextInt();
switch (chosenMenu) {
case 1: {
System.out.println("Add data from head");
System.out.print("NIM: ");
scanner.nextLine();
String nim = scanner.nextLine();
System.out.print("Name: ");
String name = scanner.nextLine();
System.out.print("GPA: ");
double gpa = scanner.nextDouble();
Student student = new Student(nim, name, gpa);
studentList.addFirst(student);
break;
```

```
}
case 2: {
System.out.println("Add data from tail");
System.out.print("NIM: ");
scanner.nextLine();
String nim = scanner.nextLine();
System.out.print("Name: ");
String name = scanner.nextLine();
System.out.print("GPA: ");
double gpa = scanner.nextDouble();
Student student = new Student(nim, name, gpa);
studentList.addLast(student);
break;
}
case 3: {
System.out.println("Add data to specific index");
System.out.print("Index: ");
int index = scanner.nextInt();
scanner.nextLine();
System.out.print("NIM: ");
String nim = scanner.nextLine();
System.out.print("Name: ");
String name = scanner.nextLine();
System.out.print("GPA: ");
double gpa = scanner.nextDouble();
Student student = new Student(nim, name, gpa);
studentList.addFrom(student, index);
break;
}
case 4: {
System.out.println("Remove data from head");
studentList.removeFirst();
break;
}
case 5: {
System.out.println("Remove data from tail");
studentList.removeLast();
break;
}
case 6: {
System.out.println("Remove data in specific index");
System.out.print("Index: ");
int index = scanner.nextInt();
studentList.remove(index);
break;
}
case 7: {
System.out.println("Print");
studentList.print();
break;
}
case 8: {
System.out.println("Search by NIM");
System.out.print("NIM: ");
```

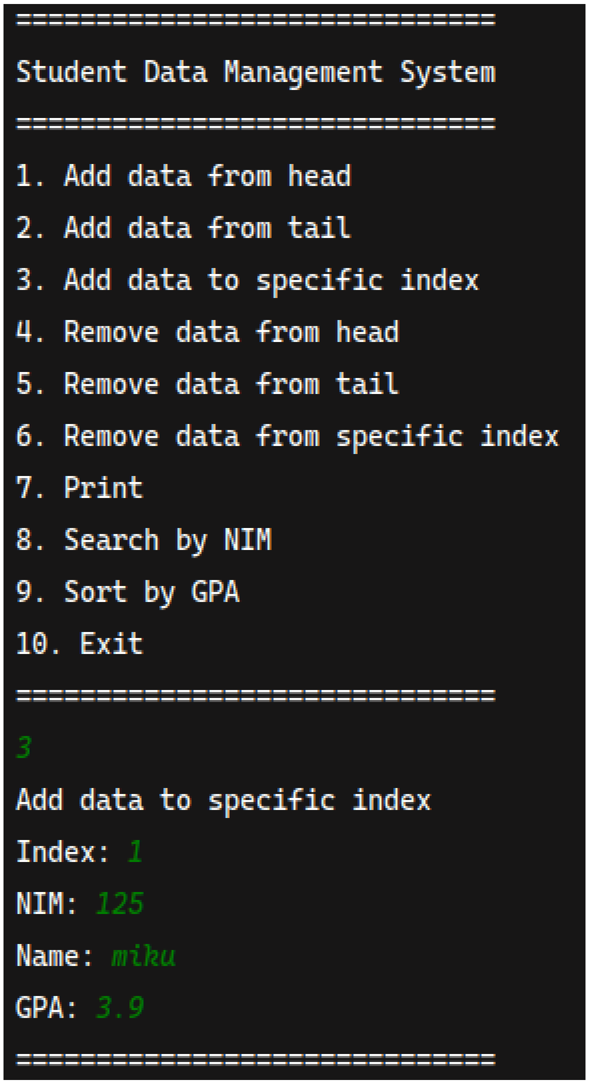
```
scanner.nextLine();
String nim = scanner.nextLine();
System.out.println("Index: " + studentList.search(nim));
break;
}
case 9: {
System.out.println("Sort by GPA");
studentList.sortByGpa();
break;
}
case 10: {
System.out.println("Exit");
return;
}
}
}
}
public static void showMenu() {
System.out.println("==============================");
System.out.println("Student Data Management System");
System.out.println("==============================");
System.out.println("1. Add data from head");
System.out.println("2. Add data from tail");
System.out.println("3. Add data to specific index");
System.out.println("4. Remove data from head");
System.out.println("5. Remove data from tail");
System.out.println("6. Remove data from specific 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("==============================");
}
}
```



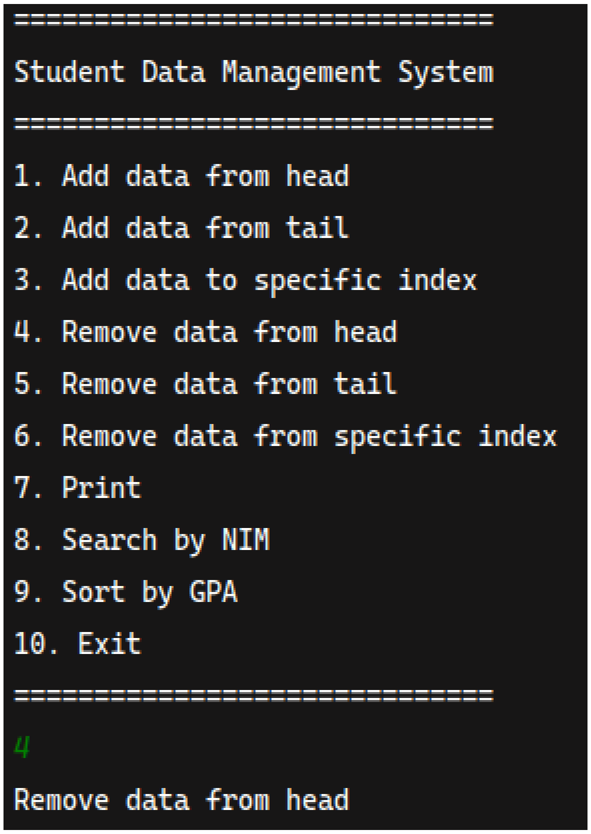
```
==============================
Student Data Management System
==============================
1. Add data from head
2. Add data from tail
3. Add data to specific index
4. Remove data from head
5. Remove data from tail
6. Remove data from specific index
7. Print
8. Search by NIM
9. Sort by GPA
10. Exit
==============================
1
Add data from head
NIM: 123
Name: giga
GPA: 3.7
```



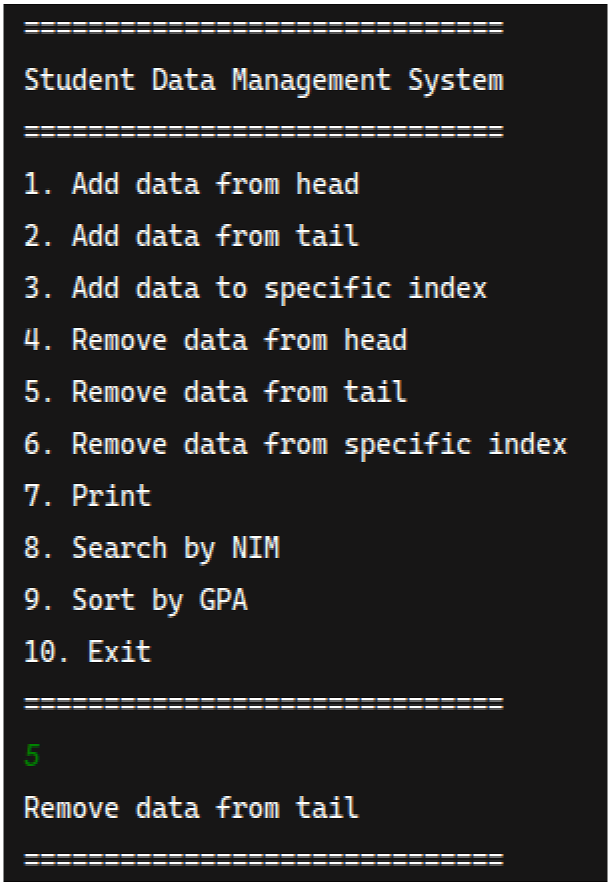
```
==============================
Student Data Management System
==============================
1. Add data from head
2. Add data from tail
3. Add data to specific index
4. Remove data from head
5. Remove data from tail
6. Remove data from specific index
7. Print
8. Search by NIM
9. Sort by GPA
10. Exit
==============================
2
Add data from tail
NIM: 124
Name: kira
GPA: 3.8
```



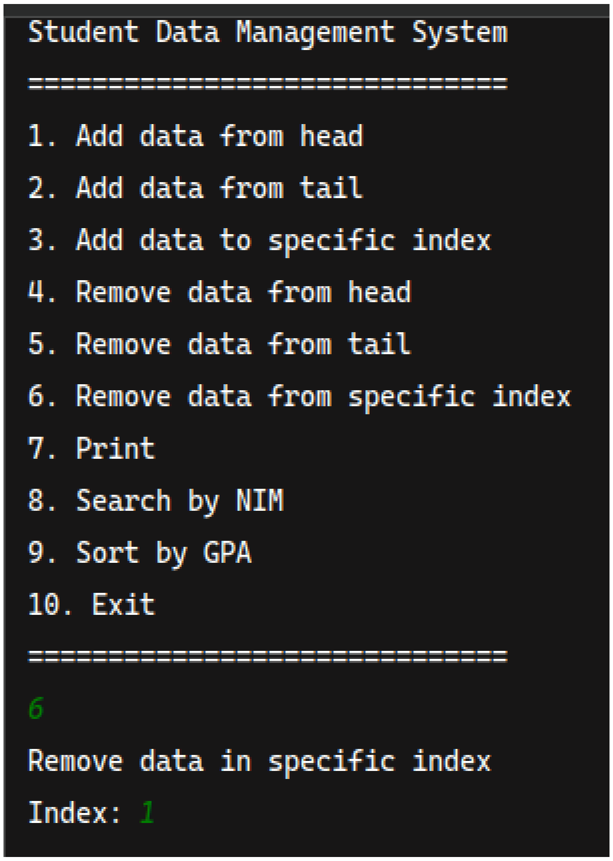
```
==============================
Student Data Management System
==============================
1. Add data from head
2. Add data from tail
3. Add data to specific index
4. Remove data from head
5. Remove data from tail
6. Remove data from specific index
7. Print
8. Search by NIM
9. Sort by GPA
10. Exit
==============================
3
Add data to specific index
Index: 1
NIM: 125
Name: miku
GPA: 3.9
==============================
```



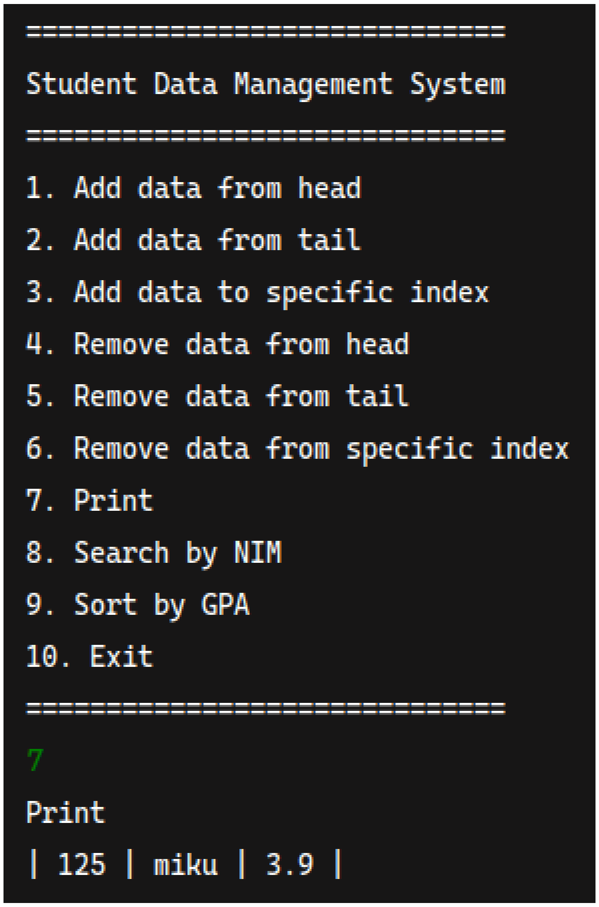
```
==============================
Student Data Management System
==============================
1. Add data from head
2. Add data from tail
3. Add data to specific index
4. Remove data from head
5. Remove data from tail
6. Remove data from specific index
7. Print
8. Search by NIM
9. Sort by GPA
10.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



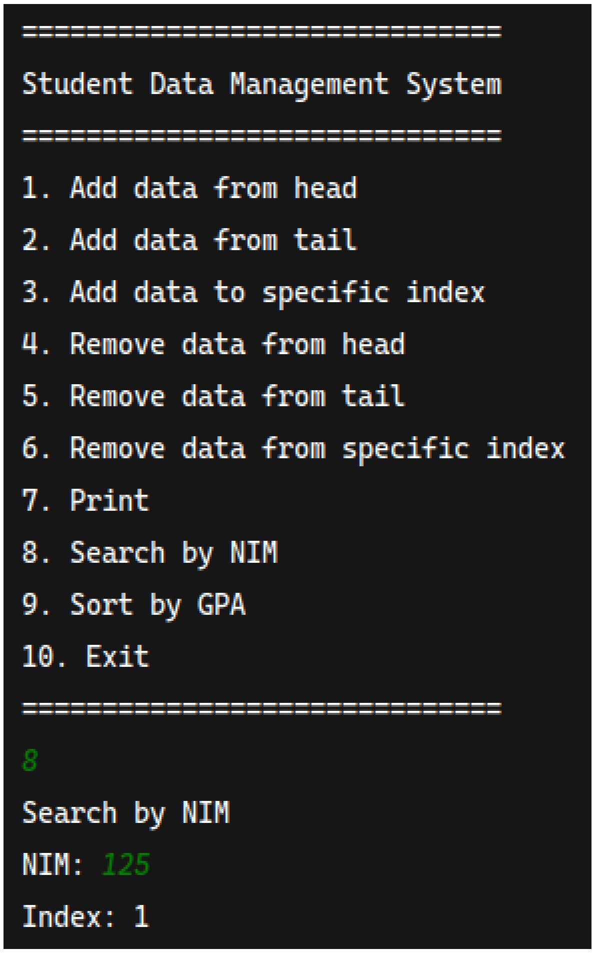
```
==============================
Student Data Management System
==============================
1. Add data from head
2. Add data from tail
3. Add data to specific index
4. Remove data from head
5. Remove data from tail
6. Remove data from specific index
7. Print
8. Search by NIM
9. Sort by GPA
10.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



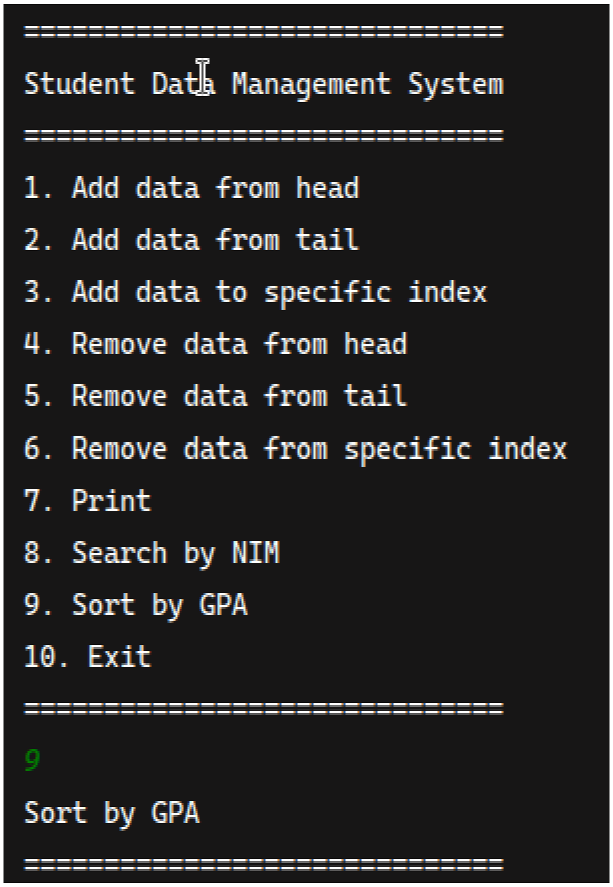
```
Student Data Management System
==============================
1. Add data from head
2. Add data from tail
3. Add data to specific index
4. Remove data from head
5. Remove data from tail
6. Remove data from specific index
7. Print
8. Search by NIM
9. Sort by GPA
10. Exit
==============================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



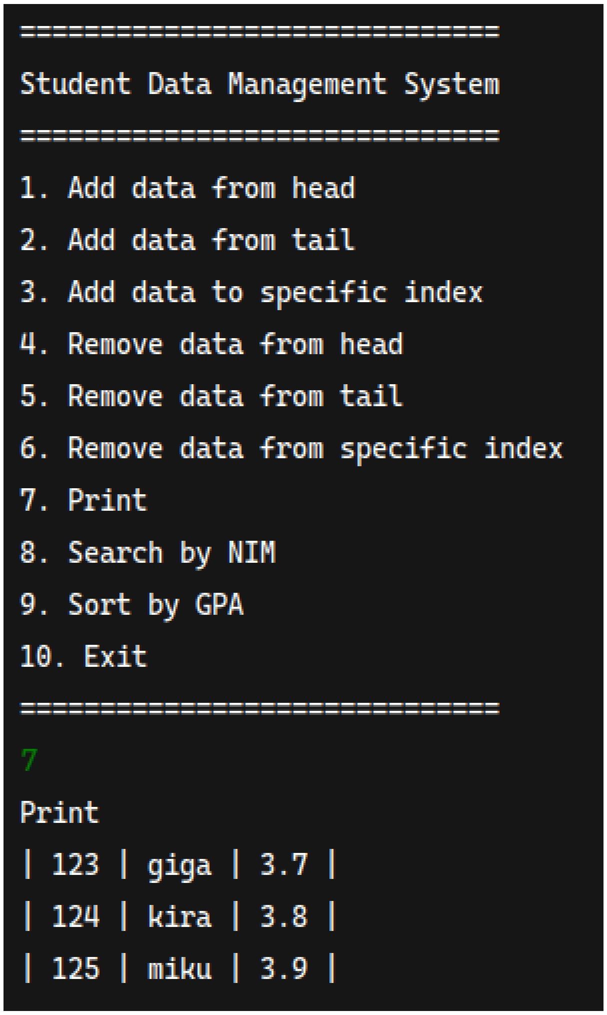
```
==============================
Student Data Management System
==============================
1. Add data from head
2. Add data from tail
3. Add data to specific index
4. Remove data from head
5. Remove data from tail
6. Remove data from specific index
7. Print
8. Search by NIM
9. Sort by GPA
10. Exit
==============================
7
Print
| 125 | miku | 3.9 |
```



```
==============================
Student Data Management System
==============================
1. Add data from head
2. Add data from tail
3. Add data to specific index
4. Remove data from head
5. Remove data from tail
6. Remove data from specific index
7. Print
8. Search by NIM
9. Sort by GPA
10. Exit
==============================
8
Search by NIM
NIM: 125
Index: 1
```



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



```
==============================
Student Data Management System
==============================
1. Add data from head
2. Add data from tail
3. Add data to specific index
4. Remove data from head
5. Remove data from tail
6. Remove data from specific index
7. Print
8. Search by NIM
9. Sort by GPA
10. Exit
==============================
7
Print
| 123 | giga | 3.7 |
| 124 | kira | 3.8 |
| 125 | miku | 3.9 |
```