

Data Structure and Algorithm Practicum

Double Linked List



Name

Dicha Zelianivan Arkana

NIM

2241720002

Class

li

Department

Information Technology

Study Program

D4 Informatics Engineering

Lab Activity #1

Questions

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

Single linked list only keep tracks of its next node while the double linked list keeps both the previous and the next node making it capable of traversing backwards.

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

It's used to store the next and the previous pointer of the node in the linked list.

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

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

It sets the initial value of the head, which is null since we haven't added any value yet. It also sets the initial value of the linked list size, which is zero because there are no items yet.

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

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

Because we're dealing with DoubleLinkedList, we also have the previous node. Since we're adding the first node, there is no previous node for a first node, which is why we set it to null.

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);
```

The last element have its **next** value as null since it's the last element of the list. If the next node is not null then it won't be the last element of the list.

Lab Activities #2

Questions

1. What's the meaning of these statements in `removeFirst()` method?
2. How do we detect the position of the data that are in the last index in method `removeLast()`?

We do that by checking if the `node.next.next` is null. If that conditions returns true, then we set the `node.next` to be null.

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;
```

It doesn't completely remove the current node since the current node will keep the reference of the head.

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

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

It's used to remove the current node by moving the `next` reference of the previous node into the `next` reference of the current node and then connecting the `prev` reference of the `next` node into the `prev` reference of the current node

Lab Activities #3

Questions

1. What is the function of method `size()` in `DoubleLinkedList` class?
It's to get the size of 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?

We need to set the first index as 1 in the constructor. This will make the consequent index to start from 1.

-
3. Please explain the difference between method `add()` in double linked list and single linked list!

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

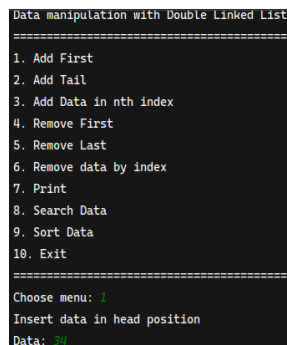
```
public boolean isEmpty() {  
    if (size == 0) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

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

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. 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 choose any sorting approach you prefer (bubble sort, selection sort, insertion sort, or merge sort)



```
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 head position  
Data: 
```

Figure 1: Inserting data from head

```
=====
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: 89
Item added successfully
```

Figure 2: Inserting data from tail

```
=====
Choose menu: 3
Insert data in nth position
Position: 1
Data: 90
Item added successfully
```

Figure 3: Inserting data in nth position

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

Figure 4: Remove the first 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
=====
Choose menu: 5
Last item has been removed successfully
=====

```

Figure 5: Remove the data in nth position

```

=====
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
=====

```

Figure 6: Remove the last data

```

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

```

Figure 7: Remove the data in nth position

```
=====
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
Data: 123
Data was found on index: 0
=====
```

Figure 8: Remove the data in nth position

Node.java

```
public class Node<T> {
    T data;
    Node<T> prev, next;

    public Node(Node<T> prev, T data, Node<T> next) {
        this.prev = prev;
        this.data = data;
        this.next = next;
    }
}
```

DoubleLinkedListInteger.java

```
public class DoubleLinkedListInteger {
    Node<Integer> head;
    int size;

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

    boolean isEmpty() {
        return size == 0;
    }

    void addFirst(Integer item) {
        if (isEmpty()) {
            head = new Node<>(null, item, null);
        } else {
```

```

        Node<Integer> newNode = new Node<>(null, item, head);
        head.prev = newNode;
        head = newNode;
    }
    size++;
}

void addLast(Integer item) {
    if (isEmpty()) {
        addFirst(item);
        return;
    }

    Node<Integer> current = head;
    while (current.next != null) {
        current = current.next;
    }
    current.next = new Node<Integer>(current, item, null);
    size++;
}

void addItem(Integer item, int index) throws Exception {
    if (isEmpty()) {
        addFirst(item);
        return;
    }

    if (index < 0 || index > size) {
        throw new Exception("Index out of bound");
    }

    Node<Integer> current = head;
    int i = 0;
    while (i < index) {
        current = current.next;
        i++;
    }

    if (current.next == null) {
        Node<Integer> newNode = new Node<>(null, item, current);
        current.prev = newNode;
        head = newNode;
    } else {
        Node<Integer> 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()) {
        System.out.println("Linked list is empty");
        return;
    }

    Node<Integer> tmp = head;
    while (tmp != null) {
        System.out.print(tmp.data + "\t");
        tmp = tmp.next;
    }
    System.out.println();
}

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

    if (size == 1) {
        removeLast();
        return;
    }

    head = head.next;
    head.prev = null;
    size--;
}

```

```

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

    if (head.next == null) {
        head = null;
    } else {
        Node<Integer> 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");
    }

    if (index == 0) {
        removeFirst();
        return;
    }

    Node<Integer> current = head;
    int i = 0;
    while (i < index - 1) {
        current = current.next;
        i++;
    }
    current.next = current.next.next;
    size--;
}

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

int getLast() throws Exception {

```

```

        if (isEmpty()) {
            throw new Exception("Linked list is still empty");
        }

        Node<Integer> 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 is still empty");
        }

        Node<Integer> tmp = head;
        for (int i = 0; i < index; i++) {
            tmp = tmp.next;
        }

        return tmp.data;
    }

    int search(int data) {
        if (isEmpty()) {
            return -1;
        }

        Node<Integer> current = head;
        int i = 0;
        while (current.next != null) {
            if (current.data == data) {
                return i;
            }
            i++;
            current = current.next;
        }

        return -1;
    }

    void bubbleSort() {
        if (head == null || head.next == null) {
            return; // No need to sort if the list is empty or has only one element

```

```

    }

    boolean swapped;
    Node<Integer> current;
    Node<Integer> last = null;

    do {
        swapped = false;
        current = head;

        while (current.next != last) {
            if (current.data > current.next.data) {
                swap(current, current.next);
                swapped = true;
            }
            current = current.next;
        }

        last = current;
    } while (swapped);
}

void swap(Node<Integer> left, Node<Integer> right) {
    int temp = left.data;
    left.data = right.data;
    right.data = temp;
}
}

```

DoubleLinkedListMain

```

public class DoubleLinkedListMain {

    public static void main(String[] args) throws Exception {
        Scanner scanner = new Scanner(System.in);
        DoubleLinkedListInteger list = new DoubleLinkedListInteger();

        while (true) {
            showMenu();
            int chosenMenu = scanner.nextInt();
            switch (chosenMenu) {
                case 1: {
                    System.out.println("Insert data in head position");
                    System.out.print("Data: ");
                    int data = scanner.nextInt();

```

```

        list.addFirst(data);
        System.out.println("Item added successfully");
        break;
    }
    case 2: {
        System.out.println("Insert data in tail position");
        System.out.print("Data: ");
        int data = scanner.nextInt();
        list.addLast(data);
        System.out.println("Item added successfully");
        break;
    }
    case 3: {
        System.out.println("Insert data in nth position");
        System.out.print("Position: ");
        int position = scanner.nextInt();
        System.out.print("Data: ");
        int data = scanner.nextInt();
        list.addItem(data, position);
        System.out.println("Item added successfully");
        break;
    }
    case 4: {
        list.removeFirst();
        System.out.println("First item has been removed successfully");
        break;
    }
    case 5: {
        list.removeLast();
        System.out.println("Last item has been removed successfully");
        break;
    }
    case 6: {
        System.out.println("Remove data in nth position");
        System.out.print("Position: ");
        int position = scanner.nextInt();
        list.remove(position);
        System.out.printf("Item in index %s has been removed successfully\n", position);
        break;
    }
    case 7: {
        list.print();
        break;
    }
    case 8: {

```

```

        System.out.print("Data: ");
        int data = scanner.nextInt();
        int index = list.search(data);
        if (index == -1) {
            System.out.println("No data was found on the list");
            break;
        }
        System.out.printf("Data was found on index: %s\n", index);
        break;
    }
    case 9: {
        list.bubbleSort();
        break;
    }
    case 10: {
        System.out.println("Exiting...");
        break;
    }
    default: {
        System.out.println("Invalid input");
        return;
    }
}
}

}

static void showMenu() {
    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("=====");
    System.out.print("Choose menu: ");
}
}

```

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

```
*****
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
*****
```

Figure 9: Add item

```
*****
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: hmmm yes
*****
```

Figure 10: Get book from top of 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
Peeking the top book: hmmmmmm yes
*****

```

Figure 11: Peek book title from top

```

*****
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
*****

```

Figure 12: Info all books

StudentList.java

```

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;
}
}

```

StudentMain.java

```

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("=====");
    }
}
```

-
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**)!

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

Figure 13: Add to queue

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

Figure 14: Remove from queue

```

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

```

Figure 15: Display vaccine queue

Queue.java

```

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("+++++++");
    }
}

```

QueueMain.java

```

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("+++++");
}
}

```

-
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!

```
=====
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
```

Figure 16: Add 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
=====
2
Add data from tail
NIM: 124
Name: hiru
GPA: 3.8
```

Figure 17: Add 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
=====
3
Add data to specific index
Index: 1
NIM: 123
Name: wibu
GPA: 3.0
=====

```

Figure 18: Add data to specific index

```

=====
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

```

Figure 19: 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
=====
```

Figure 20: 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
```

Figure 21: Remove data from specific index

```
=====
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 |
```

Figure 22: Display data

```
=====
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
```

Figure 23: Search data

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

Figure 24: Sort data

```
=====
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 |
```

Figure 25: Sort data

Student.java

```
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;
    }
}
```

StudentList.java

```
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;
}
}

```

StudentMain.java

```

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("=====");
}
}

```