Name : Faricha Aulia
Absen : 08
Class : 1I – IT

Lab Unit 1:

```java
package J15;

public class Node08 {
    int data;
    Node08 next;

    public Node08(int score, Node08 next){
        this.data=score;
        this.next=next;
    }
}
```

```java
package J15;

public class LinkedList08 {
    Node08 head;
    Node08 tail;

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

    public void print() {
        if (!isEmpty()) {
            Node08 tmp = head;
            System.out.println("Data on Linked List:\t");
            while (tmp != null) {
                System.out.println(tmp.data + "\t");
                tmp = tmp.next;
            }
            System.out.println("");
        } else {
            System.out.println("Linked List is empty");
        }
    }

    public void addFirst(int input) {
        Node08 ndInput = new Node08(input, null);
        if (!isEmpty()) {
            head = ndInput;
            tail = ndInput;
        } else {
            ndInput.next = head;
            head= ndInput;
        }
    }

    public void addLast(int input){
        Node08 ndInput = new Node08(input, null);
        if (isEmpty()){
            head = ndInput;
            tail = ndInput;
        } else {
            tail.next = ndInput;
            tail = ndInput;
        }
    }

    public void insertAfter(int key, int input){
        Node08 ndInput = new Node08(input, null);
        Node08 temp = head;
        do {
            if(temp.data == key){
                ndInput.next = temp.next;
                temp.next = ndInput;
```

```
            if(ndInput.next == null) tail=ndInput;
            break;
        }
        temp = temp.next;
    } while (temp != null);
}

public void insertAt(int index,int input){
    if(index<0){
        System.out.println("wrong index");
    } else if (index == 0){
        addFirst(input);
    } else {
        Node08 temp = head;
        for (int i=0; i<index-1; i++){
            temp = temp.next;
        }
        temp.next = new Node08(input, temp.next);
        if(temp.next.next==null){
            tail=temp.next;
        }
    }
}

public int getData (int index) {
    Node08 tmp = head;
    for (int i=0; i<index; i++){
        tmp = tmp.next;
    }
    return tmp.data;
}

public int indexOf (int key) {
    Node08 tmp = head;
    int index = 0;
    while (tmp != null && tmp.data != key){
        tmp = tmp.next;
        index++;
    }
    if (tmp == null){
        return -1;
    } else {
        return index;
    }
}

public void removeFirst() {
    if (isEmpty()){
        System.out.println("Linked list is empty, cannot delete data!");
    } else if (head == tail) {
        head = tail = null;
    } else {
        head = head.next;
    }
}

public void removeLast(){
    if (isEmpty()){
        System.out.println("Linked list is empty, cannot delete data!");
    } else if (head == tail) {
        head = tail = null;
    } else {
        Node08 temp = head;
        while (temp.next != tail){
            temp = temp.next;
        }
    }
}

public void remove (int key){
    if (isEmpty()){
        System.out.println("Linked List is empty, cannot delete data!");
    } else {
```

```
            Node08 temp = head;
            while (temp != null){
                if (temp.next==null){
                    System.out.println("data to be deleted was not found");
                    break;
                } else {
                    if ((temp.data == key) && (temp == head)){
                        this.removeFirst();
                        break;
                    }
                } temp = temp.next;
            }
        }
    }

    public void removeAt (int index){
        if(index == 0){
            removeFirst();
        } else {
            Node08 temp = head;
            for (int i=0; i<index-1; i++){
                temp = temp.next;
            }
            temp.next = temp.next.next;
            if (temp.next == null){
                tail = temp;
            }
        }
    }
}
```

```
package J15;

public class GraphMain08 {
    public static void main(String[] args) {
        Graph08 graph = new Graph08(6);
        graph.addEdge(0, 1);
        graph.addEdge(0, 4);
        graph.addEdge(1, 2);
        graph.addEdge(1, 3);
        graph.addEdge(1, 4);
        graph.addEdge(2, 3);
        graph.addEdge(3, 4);
        graph.addEdge(3, 0);
        graph.printGraph();
        graph.degree(2);
        graph.removeEdge(1,2);
        graph.removeEdge(1,3);
        graph.printGraph();
    }
}
```

```
package J15;

public class Graph08 {
    int vertex;
    LinkedList08 list[];

    public Graph08(int vertex){
        this.vertex = vertex;
        list = new LinkedList08[vertex];
        for(int i = 0; i<vertex; i++){
            list[i] = new LinkedList08();
        }
    }

    public void addEdge(int source, int destination){
        list[source].addFirst(destination);
        list[destination].addFirst(source);
    }

    public void degree(int source){
```

```
        System.out.println("degree of vertex" +source+ " : " +list[source].size());
        int k, totalIn = 0, totalOut = 0;
        for(int i=0; i<vertex; i++){
            for(int j=0; j<list[i].size(); j++){
                if(list[i].get(j)==source)
                    ++totalIn;
            }
            for(k=0; k<list[source].size();k++){
                list[source].get(k);
            }
            totalOut = k;
        }
        System.out.println("Indegree of vertex" +source+ ":" +totalIn);
        System.out.println("Outdegree of vertex" +source+ ":" +totalOut);
        System.out.println("Degree of vertex" +source+ " : " + (totalIn+totalOut));
    }

    public void removeEdge(int source, int destination){
        for (int i=0; i<vertex; i++){
            if(i==destination){
                list[source].remove(destination);
            }
        }
    }

    public void removeAllEdges(){
        for (int i=0;i<vertex;i++){
            list[i].clear();
        }
        System.out.println("Graph successfully emptied");
    }

    public void printGraph(){
        for (int i=0; i<vertex; i++){
            if(list[i].size()>0){
                System.out.println("Vertex " +i+ "connected with : ");
                for (int j=0, j<list[i].size(); j++){
                    System.out.println(list[i].get(j) + " ");
                }
                System.out.println(" ");
            }
        }
        System.out.println(" ");
    }


}
```

Lab Unit 2:

```
package J15;

import java.util.Scanner;

public class GraphArray08 {
    private final int vertices;
    private int[][] twoD_array;

    public GraphArray08(int v){
        vertices = v;
        twoD_array = new int[vertices + 1][vertices + 1];
    }

    public void makeEdge(int to, int from, int edge){
        try{
            twoD_array[to][from] = edge;
        } catch (ArrayIndexOutOfBoundsException index){
            System.out.println("Vertex does not exist");
        }
    }

    public int getEdge(int to, int from){
        try{
```

```
            return twoD_array[to][from];
        } catch (ArrayIndexOutOfBoundsException index){
            System.out.println("Vertex does not exist");
        }
        return -1;
    }

    public static void main(String[] args) {
        int v, e, count = 1, to = 0, from = 0;
        Scanner sc = new Scanner(System.in);
        GraphArray08 graph;
        try{
            System.out.println("How many vertices? : ");
            v = sc.nextInt();
            System.out.println("How many edges? : ");
            e = sc.nextInt();

            graph = new GraphArray08(v);

            System.out.println("Input edges: <to> <from>");
            while(count <= e){
                to = sc.nextInt();
                from = sc.nextInt();

                graph.makeEdge(to, from, 1);
                count++;
            }
            System.out.println("2D array as directed graph representation: ");
            System.out.println(" ");
            for(int i=1; i<=v; i++){
                System.out.println(i + " ");
                System.out.println();
            }

            for(int i=1; i<=v; i++){
                System.out.println(i+ "");
                for (int j=1; j<=v; j++){
                    System.out.println(graph.getEdge(i, j) + " ");
                    System.out.println();
                }
            }
        }
        catch (Exception E){
            System.out.println("Error. Check again");
        }
        sc.close();
    }
}
```

Question Lab Unit 1

1. Mention several types (at least 3) algorithms that use the Graph Concepts, and Explain the functions of these algorithms?

   Answer:
2. In the Graph class there is an array of type LinkedList: **LinkedList00 list[].** What is the purpose of creating this variable?
   Answer: store data in the form of objects
3. What is the reason for calling the **addFirst() method** to add data instead of another type of add method on a linked list when used in the addEdge method of the **Graph class?**
   Answer: to have nodes in a list
4. How to detect the prev pointer when deleting an edge in the graph?
   Answer: because every time we input the relationship between nodes, for example node A to B, then we also have to input from B to A with a value of 1 because the edge/relationship between nodes is 2-way.
5. Add the following program line to the main method. Are the results correct? if not, explain why this happened? What's the solution? Note : the output is incorrect when deleting data other than the first edge data
   Answer:

```
89          graph.removeEdge(1, 3);
90          graph.printGraph();
```

Question Lab Unit 2

1. What is the difference between the degrees in *directed* and *undirected graphs* ?

   Answer:

- directed graph : the edges are ordered pairs, where the consecutive pairs represent the direction of the edges connecting the two vertices
- undirected graph : an edge is an irregular pair, because there is no direction associated with an edge

2. In the implementation of the graph using an adjacency matrix. Why must the number ofvertices be added by 1 in the following array index?
   Answer: because it has a directed arc

```
8       public graphArray(int v)
9       {
10          vertices = v;
11          twoD_array = new int[vertices + 1][vertices + 1];
12      }
```

3. What is the function of the **getEdge() method** ?
   Answer: This method retrieves an edge.
4. What is the type of graph in the lab unit 2
   Answer: Directed graph
5. Why does the main method have to use *a try-catch exception* ?
   Answer: to catch the exception, so no need to throw that exception higher up.

Assigment

1. Add a **graphType**() method with a boolean type that will differentiate the type of graph: directed or undirected graph. Then update all methods related to the **graphType**() method (only executestatements according to the graph type) in lab unit 1
   Answer:

```
public Graph(List<Edge> edges)
  {
    // adjacency list memory allocation
    for (int i = 0; i < edges.size(); i++)
      adj_list.add(i, new ArrayList<>());

    // add edges to the graph
    for (Edge e : edges)
    {
      // allocate new node in adjacency List from src to dest
      adj_list.get(e.src).add(new Node(e.dest, e.weight));
    }
  }
```

2. Change the *vertex data type* on all graphs in lab unit 1 and lab unit 2 from Integer to generic type **<T> (check linked list PPT)** so that it can accept all primitive data types! For example, each *vertex* which was originally a number 0,1,2,3, and so on. then change it to a regional name such as Gresik, Bandung, Yogya, Malang, and so on.
   Answer:

```
public class Graph {
  int vertex;
  LinkedList list[];
```

```
    public Graph(int vertex) {
        this.vertex = vertex;
        list = new LinkedList[vertex];
        for(int i = 0; i < vertex; i++){
            list[i] = new LinkedList();
        }
    }

    public void addEdge(int source, int destination){
        list[source].addFirst(destination);

        list[destination].addFirst(source);
    }

    public void degree(int source) throws Exception {
        System.out.println("degree vertex "+ source +" : "+ list[source].size);


        int k, totalIn = 0, totalOut = 0;
        for(int i = 0; i < vertex; i++){
            for(int j = 0; j < list[i].size(); j++){
                if(list[i].get(j) == source)
                    ++totalIn;
            }


            for(k = 0; k < list[source].size(); k++){
                list[source].get(k);
            }
            totalOut = k;
        }
        System.out.println("Indegree dari vertex "+ source +" : "+ totalIn);
        System.out.println("Outdegree dari vertex "+ source +" : "+ totalOut);
        System.out.println("degree vertex "+ source +" : "+ (totalIn+totalOut));
    }

    public void removeEdge(int source, int destination) throws Exception {
        for(int i = 0; i < vertex; i++){
            if(i == destination){
                list[source].remove(destination);
            }
        }
    }

    public void removeAllEdges(){
        for(int i = 0; i < vertex; i++){
            list[i].clear();
        }
        System.out.println("Graph berhasil dikosongkan");
    }

    public void printGraph() throws Exception {
        for(int i = 0; i < vertex; i++){
            if(list[i].size() > 0){
                System.out.print("Vertex " + i + " terhubung dengan : ");
                for(int j = 0; j < list[i].size(); j ++){
                    System.out.print(list[i].get(j) + " ");
```

```
            }
            System.out.println("");
        }
    }
    System.out.println("");
  }
}
```