LINEAR ABSTRACT DATA TYPES – LAB 03

EC 4070

DATA STRUCTURES AND ALGORITHMS

NAME : **WIJAYAWARDHANA W.A.H.A.**

REGISTRATION NO. : **2019/E/166**

SEMESTER : **SEMESTER 04**

DATE ASSIGNED : **10 MARCH 2022**

01.
a).

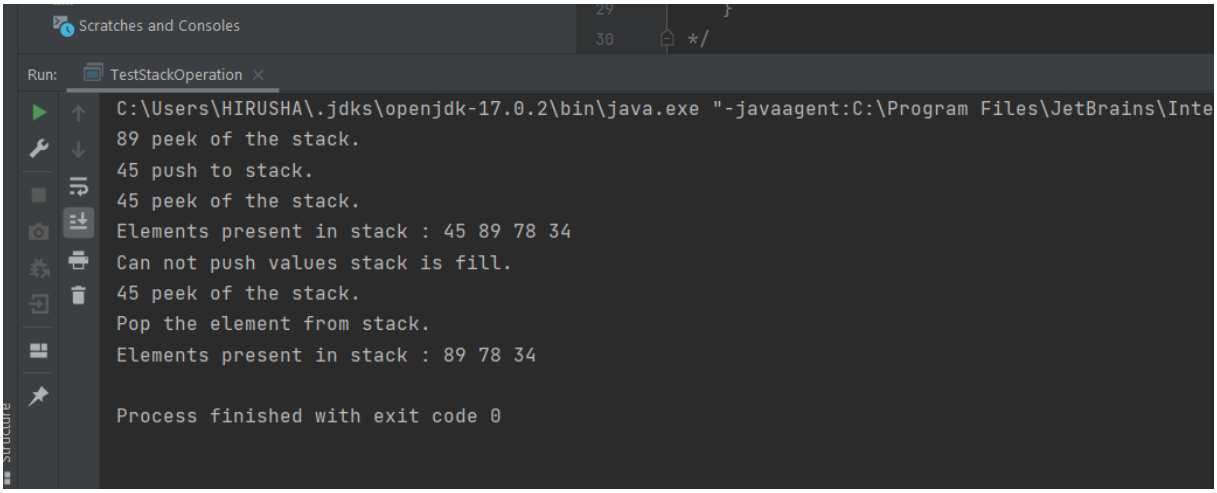
Code:-

```
public class StackOperation {
   int arraySize;  // Define arraySize.
   int[] stackElementArray = new int[arraySize]; // Define array.
   int topValue;
   int newElement;
   boolean stackEmpty;
   boolean stackFull;

   public void StackOperation() // Default constructor.
   {
      arraySize = 0;
      topValue = -1;
   }
   // StackOperation method for setting values.
   public void StackOperation(int arraySize , int[] stackElementArray,int topValue)
   {
      this.arraySize = arraySize;
      this.stackElementArray = stackElementArray;
      this.topValue = topValue;
      stackEmpty = false;
      stackFull = false;
   }
   // isEmpty method for checking the elements are empty or not.
   public void isEmpty()
   {
      if(topValue == -1)
      {
         stackEmpty = true;
      }
      else
      {
         stackEmpty = false;
      }

   }

   // isFull method for checking the elements are full or not.
   public void isFull()
   {
      if(topValue == stackElementArray.length)
      {
         stackFull =  true;
```

```
    }
    else
    {
      stackFull =  false;
    }
  }

  // peek method for output peek value of stack.
  public void peek()
  {
    isEmpty();
    if(stackEmpty == true)
    {
      System.out.println("Stack is empty.");
    }
    else
    {
      System.out.println(stackElementArray[topValue-1] + " peek of the stack.");
    }
  }

  // push method for adding new element at end.
  public void push(int newElement)
  {
    this.newElement = newElement;
    isFull();
    if(stackFull == true)
    {
      System.out.println("Can not push values stack is fill.");
    }
    else
    {
      stackElementArray[topValue] = newElement;
      topValue++;
      System.out.println(newElement + " push to stack.");
    }
  }

  // pop method for pop the element.
  public void pop()
  {
    if(stackEmpty == true)
    {
      System.out.println("Stack is empty can not pop values.");
    }
    else
    {
      stackElementArray[topValue-1] = 0;
      topValue--;
```

```
            System.out.println("Pop the element from stack.");
        }
    }

    // PrintElement method for print stack.
    public void printElement()
    {
        System.out.print("Elements present in stack : ");
        for(int i = topValue-1; i>=0; i--)
        {
            System.out.print(stackElementArray[i]+ " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        StackOperation newObject = new StackOperation();
        int arraySize = 4;
        int[] elementArray = new int[]{34, 78, 89, 0};
        newObject.StackOperation(arraySize, elementArray, 3);
        newObject.peek();
        newObject.push(45);
        newObject.peek();
        newObject.printElement();
        newObject.push(66);
        newObject.peek();
        newObject.pop();
        newObject.printElement();
    }

}
```

Outputs:-

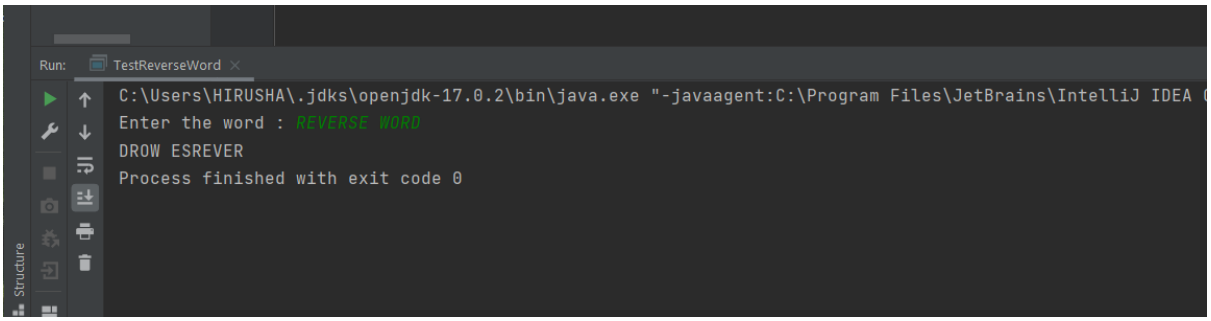


b). 1.

Code:-

```
import java.util.Scanner;  // Import scanner library.
public class ReverseWord {
   String word;     // Define word.
   Scanner scanner = new Scanner(System.in);  // Create object of scanner.
   public void ReverseWord()    // Default constructor.
   {
      word = "WORD";
   }
   public void ReverseWord(String word)   // Default constructor for setting values.
   {
      this.word = word;
      setCharactersArray(word);
   }
   // setWord method to set a word.
   public void setWord()
   {
      System.out.print("Enter the word : ");
      word = scanner.nextLine();
      setCharactersArray(word);
   }
   // setCharactersArray method to take characters to array.
   public void setCharactersArray(String word)
   {
      char[] characters = new char[word.length()];
      for(int i =0; i<word.length(); i++)
      {
         characters[i] = word.charAt(i);
      }
      printWord(characters);
   }

   // printWord method to reverse the word.
   public void printWord(char[] characters)
   {
      for(int j = characters.length-1; j>=0; j--)
      {
         System.out.print(characters[j]);
      }
   }

   public static void main(String[] args) {
      ReverseWord newObject = new ReverseWord();  // Crate an object of ReverseWord class.
      newObject.setWord();    // Calling setWord method.
   }
}
```

Output:-



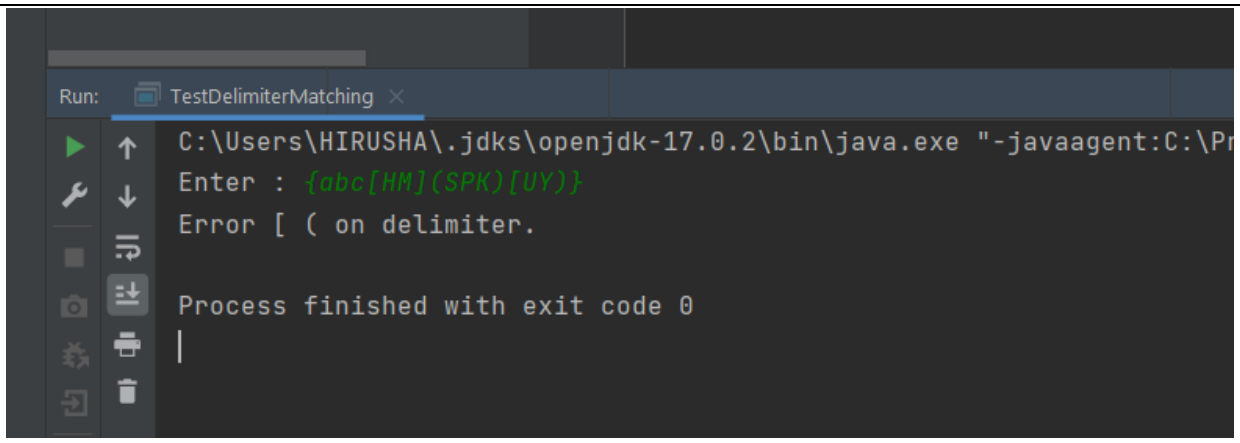
2.

Code:-

```
import java.util.Scanner;    // Import scanner library.
import java.util.ArrayList;  // Import ArrayList library.
public class DelimitersMatching {
   String delimiter;   // Define delimiter.
   Scanner scanner = new Scanner(System.in);    // Create object of scanner.
   public void DelimitersMatching()     // Default constructor.
   {
      delimiter = " ";
   }
   public void setDelimiter()       // setDelimiter method for setting elements.
   {
      System.out.print("Enter : ");     // Take the input from user.
      delimiter = scanner.nextLine();
   }

   // setCharacters method for passing to array.
   public void setCharacters()
   {
      int delimiterLength = delimiter.length();
      ArrayList<Character> myList = new ArrayList<Character>();
      char characters = '+';  // Default char value.
      int index = -1;
      for(int i =0; i<delimiter.length();i++)
      {
         characters = delimiter.charAt(i);
         if((characters == '{')||(characters == '[')||(characters == '('))  // Check the required
character available or not.
         {
            index++;
            myList.add(index,characters);

         }
```

```
        else if((characters == '}')||(characters == ']')||(characters == ')'))  // Check with the
elements at the last of list.
        {
           if(characters == '}')
              characters = '{';
           else if(characters == ')')
              characters = '(';
           else
              characters = '[';
           if(myList.get(index) != characters)  // If not equal this will print.
           {
              System.out.println("Error "+myList.get(index) +" "+characters + " on delimiter.");
              return;
           }
           else
           {
              index--;
           }
        }
     }
     System.out.println("Delimiter matching properly.");   // If delimiter match properly this will
come.
  }
  public static void main(String[] args) {
     DelimitersMatching newObject = new DelimitersMatching();  // Create an object.
     newObject.setDelimiter();    // Calling setDelimiter method.
     newObject.setCharacters();   // Calling setCharacters method.
  }
}
```

Output:-



02.

a).

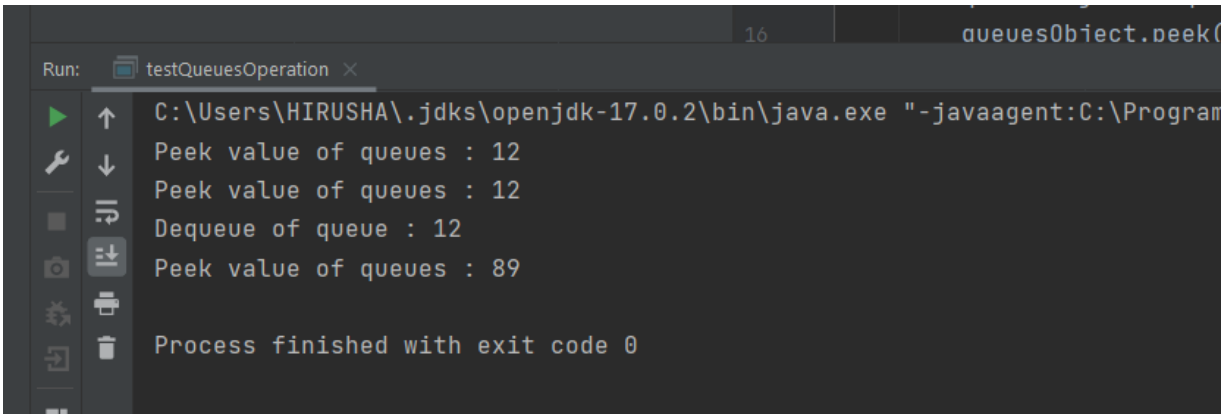
Code:-

```
import java.util.Scanner;  // Import scanner library.

public class QueuesOperation {
   int queuesFront;  // Define queuesFront.
   int queuesRear;   // Define queuesRear.
   int arraySize;    // Define arraySize.
   int[] queuesElement = new int[arraySize];
   Scanner scanner = new Scanner(System.in);  // Create an object of scanner.

   public void QueuesOperation() {     // Default constructor.
      queuesRear = -1;
      queuesFront = -1;
   }
   // Default constructor to set element
   public void QueuesOperation(int[] queuesElement, int arraySize, int rearValue) {
      this.queuesElement = queuesElement;
      this.arraySize = arraySize;
      queuesRear = rearValue;
      queuesFront = 0;
   }

   // setQueues method for setting values and elements.
   public void setQueues() {
      System.out.println("Enter size : ");
      arraySize = scanner.nextInt();   // Take queue size from user.
      boolean queuesEmpty = isEmpty();
      if (queuesEmpty == true) {
         System.out.println("Queues is empty.");
      }
```

```
        for (int i = queuesRear; i < arraySize; i++) {
            System.out.print("Enter element : ");
            queuesElement[i] = scanner.nextInt();
        }
        System.out.println("Queues is full.");
    }
    // isEmpty method for checking queue empty or not.
    public boolean isEmpty() {
        if (queuesRear == -1) {
            return true;
        } else {
            return false;
        }
    }

    // peek method for get peek value.
    public void peek() {
        boolean queuesEmpty = isEmpty();
        if (queuesEmpty == true) {
            System.out.println("Queues is empty.");
        } else {
            System.out.println("Peek value of queues : " + queuesElement[queuesFront]);
        }
    }
    // isFull method for checking queue full or not.
    public boolean isFull() {
        if (queuesRear == queuesElement.length - 1) {
            return true;
        } else {
            return false;
        }
    }

    // enqueue method for adding new element.
    public void enqueue(int newElement) {
        boolean queuesFull = isFull();
        if (queuesFull == true) {
            System.out.println("Queues is full.");
        } else {
            queuesElement[queuesRear - 1] = newElement;
            queuesRear++;
        }
    }

    // dequeue method for give front element.
    public void dequeue() {
        boolean queuesEmpty = isEmpty();
        if (queuesEmpty == true) {
            System.out.println("Queues is empty.");
```

```
        } else {
            System.out.println("Dequeue of queue : " + queuesElement[queuesFront]);
            queuesFront++;
        }
    }

    public static void main(String[] args) {
        QueuesOperation queuesObject = new QueuesOperation();  // Crate an object of
QueuesOperation class.
        int[] queuesArray = new int[10];     // Define an array for queue values.
        queuesArray[0] = 12;     // Assign values.
        queuesArray[1] = 89;     // Assign values.
        queuesArray[2] = 55;     // Assign values.
        queuesArray[3] = 69;     // Assign values.
        queuesArray[4] = 33;     // Assign values.
        queuesArray[5] = 84;     // Assign values.
        queuesObject.QueuesOperation(queuesArray, 10, 5); // Calling QueuesOperation method.
        queuesObject.peek();       // Calling peek method.
        queuesObject.enqueue(55);  // Calling enqueue method.
        queuesObject.peek();     // Calling peek method.
        queuesObject.dequeue();  // Calling dequeue method.
        queuesObject.peek();     // Calling peek method.
    }
}
```

Output:-



b)

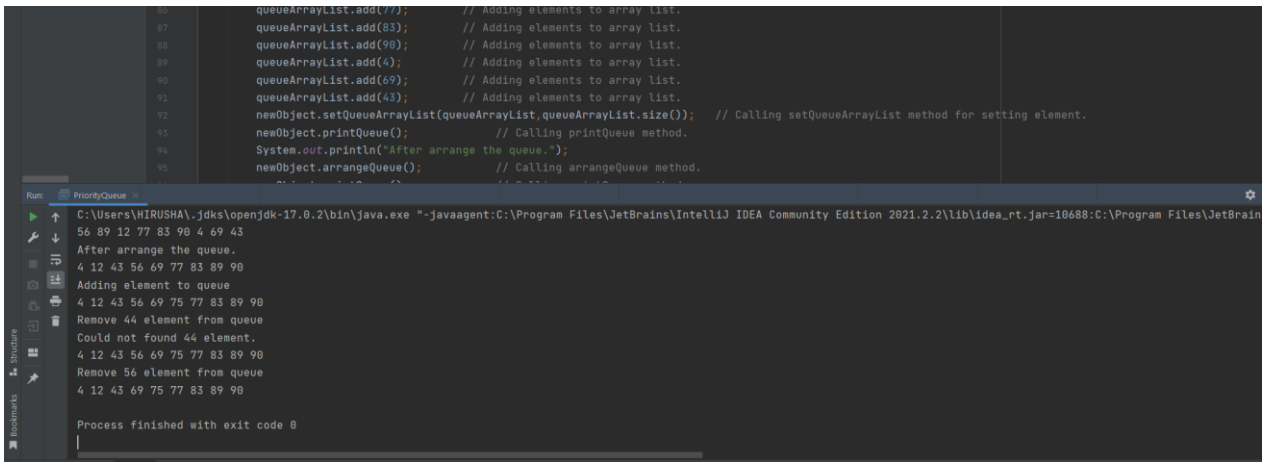
Code:-

```
import java.util.ArrayList;    // Import ArrayList library.
public class PriorityQueue {
   int queueSize;     // Define queue size.
   ArrayList<Integer> queueArrayList = new ArrayList<Integer>(queueSize);  // Define
queueArrayList.

   // setQueueArrayList method.
   public void setQueueArrayList(ArrayList<Integer> queueArrayList , int queueSize)
   {
      this.queueSize = queueSize;       // Assign queueSize when method calling.
      this.queueArrayList = queueArrayList;  // Assign queueArrayList when method calling.
   }

   // arrangeQueue method for prepare the order of queue.
   public void arrangeQueue()
   {
      for(int i = 0; i<queueSize; i++)      // Sorting the queue.
      {
         for(int j = i+1; j <queueSize; j++)
         {
            if(queueArrayList.get(i) > queueArrayList.get(j))   // Check the elements greater than or
less.
            {
               int temp = queueArrayList.get(i);
               queueArrayList.add(i,queueArrayList.get(j));
               queueArrayList.remove(i+1);
               queueArrayList.add(j,temp);
               queueArrayList.remove(j+1);
            }
         }
      }
   }

   // poll method for removing element.
   public void poll(int removingElement)
   {
      boolean isFound = false;       // Define to check element found or not.
      for (int i =0; i<queueArrayList.size(); i++)
      {
         if(queueArrayList.get(i) == removingElement)    // Check the element available or not.
         {
            queueArrayList.remove(i);       // Remove the element if it found.
            isFound = true;
         }
```

```
        }
        if(isFound == false)    // If nor found this part will run.
        {
            System.out.println("Could not found " + removingElement +" element.");
        }
    }

    // add method for adding element to queue.
    public void add(int addingElement)
    {
        for (int k = 0; k <queueArrayList.size(); k++)      // Check the suitable place for element.
        {
            if((addingElement <= queueArrayList.get(k))&&(k ==0))
            {
                queueArrayList.add(k,addingElement);
            }
            else if((addingElement>queueArrayList.get(k))&&(k == queueArrayList.size()-1))
            {
                queueArrayList.add(k+1,addingElement);
            }
            else if((addingElement>queueArrayList.get(k)) &&
(addingElement<=queueArrayList.get(k+1)))
            {
                queueArrayList.add(k+1,addingElement);
            }
        }
    }

    // Print method to print link list.
    public void printQueue()
    {
        for (int j =0; j < queueArrayList.size(); j++)
        {
            System.out.print(queueArrayList.get(j) + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        PriorityQueue newObject = new PriorityQueue();  // Creating an object of the PriorityQueue
class.
        ArrayList<Integer> queueArrayList = new ArrayList<Integer>(); // Define the array list.
        queueArrayList.add(56);       // Adding elements to array list.
        queueArrayList.add(89);       // Adding elements to array list.
        queueArrayList.add(12);       // Adding elements to array list.
        queueArrayList.add(77);       // Adding elements to array list.
        queueArrayList.add(83);       // Adding elements to array list.
        queueArrayList.add(90);       // Adding elements to array list.
        queueArrayList.add(4);        // Adding elements to array list.
```

```
        queueArrayList.add(69);       // Adding elements to array list.
        queueArrayList.add(43);       // Adding elements to array list.
        newObject.setQueueArrayList(queueArrayList,queueArrayList.size());   // Calling
setQueueArrayList method for setting element.
        newObject.printQueue();           // Calling printQueue method.
        System.out.println("After arrange the queue.");
        newObject.arrangeQueue();         // Calling arrangeQueue method.
        newObject.printQueue();           // Calling printQueue method.
        System.out.println("Adding element to queue");
        newObject.add(75);                // Calling add method.
        newObject.printQueue();           // Calling printQueue method.
        System.out.println("Remove 44 element from queue");
        newObject.poll(44);    // Calling poll method for remove item.
        newObject.printQueue();           // Calling printQueue method.
        System.out.println("Remove 56 element from queue");
        newObject.poll(56);    // Calling poll method for remove item.
        newObject.printQueue();           // Calling printQueue method.
    }

}
```

Output:-



03.

Code:-

```
import java.util.ArrayList;     // Import ArrayList library.
public class LinkList {
   ArrayList<Integer> linkListArrayList = new ArrayList<Integer>();  // Define an array list.
   int linkListIndex;     // Define array size.
   public void LinkList() // Default constructor.
   {
      linkListIndex = 0;
   }
   // Setting link list.
   public void setLinkList(ArrayList<Integer> linkListArrayList , int linkListIndex)
   {
      this.linkListArrayList = linkListArrayList;  // Assigning the link list to array list.
      this.linkListIndex = linkListIndex;          // Assigning link list size.
   }
   // Append new node method for adding mew element at end.
   public void appendNewNode(int newElement)
   {
      linkListArrayList.add(linkListIndex+1,newElement);  // Adding new element.
      linkListIndex++;
   }
   // Prepend new node method for adding mew element at front.
   public void prependNewNode(int newElement)
   {
      linkListArrayList.add((0),newElement);
   }
   // Delete the node at the front.
   public void deleteAtStart()
   {
      linkListArrayList.remove(0);
   }
   // Delete an element at specific place.
   public void deleteAtSpecificPosition(int indexForDelete)
   {
      linkListArrayList.remove(indexForDelete);
   }
   // Print method to print link list.
   public void printArrayList()
   {
      for (int j =0; j < linkListArrayList.size(); j++)
      {
         System.out.print(linkListArrayList.get(j) + " ");
      }
      System.out.println();
   }
```

```
    public static void main(String[] args) {
        ArrayList<Integer> arrayList = new ArrayList<>(); // Define the array list.
        LinkList newObject = new LinkList();    // Creating an object of the LinkList class.
        arrayList.add(0,55);       // Adding elements to array list.
        arrayList.add(1,67);       // Adding elements to array list.
        arrayList.add(2,90);       // Adding elements to array list.
        arrayList.add(3,19);       // Adding elements to array list.
        newObject.setLinkList(arrayList , arrayList.size()-1);   // Calling the setLinkList method.
        System.out.println("Link list at start.");
        newObject.printArrayList();              // Calling printArrayList method.
        newObject.appendNewNode(12);      // Calling appendNewNode method.
        System.out.println("After adding new element at end.");
        newObject.printArrayList();              // Calling printArrayList method.
        newObject.prependNewNode(99);     // Calling prependNewNode method.
        System.out.println("After adding new element at front.");
        newObject.printArrayList();              // Calling printArrayList method.
        newObject.deleteAtStart();               // Calling deleteAtStart method.
        System.out.println("Delete element at front.");
        newObject.printArrayList();              // Calling printArrayList method.
        newObject.deleteAtSpecificPosition(3); // Calling deleteAtSpecificPosition method.
        System.out.println("Delete element at 3 index.");
        newObject.printArrayList();              // Calling printArrayList method.
    }

}
```

Output:-