TABLE OF CONTENTS

CLOs PLOs Mapping and List of Experiments

SE-331L Data Structure Algorithm Lab

Course Code	CS-331
Credit Hours	3+1
Contact Hours	48
Prerequisite	CS-214 Object Oriented Programming

Course Learning Outcomes				
S#	**CLO**	**Domain**	**Taxonomy level**	**PLO**
1	***Implement*** various data structures and their algorithms.	Cognitive	2,3	5
2	***Design*** new data structures and algorithms to solve problems.	Cognitive	6	3,5

COURSE LEARNING OUTCOMES:

Upon successful completion of the course, the student will be able to:

RELEVANT PROGRAM LEARNING OUTCOMES (PLOs):

The course is designed so that students will achieve the following PLOs:

Relevant Program Learning Outcomes (PLOs)					
1	Engineering Knowledge:	☐	7	Environment and Sustainability:	☐
2	Problem Analysis:	☐	8	Ethics:	☐
3	Design/Development of Solutions:	☐	9	Individual and Team Work:	☐
4	Investigation:	☐	10	Communication:	☐
5	Modern Tool Usage:	☐	11	Project Management:	☐
6	The Engineer and Society	☐	12	Lifelong Learning:	☐

S#	Experiment Descriptions	Deliverables	Tools	(CLO,PLO)
1	Basic Array Operations	Reestablishment	Eclipse	CLO 1, PLO 5
2	Array Utility	Arrays	Eclipse	CLO 2, PLO 3
3	Array Sorting		Eclipse	CLO 2, PLO 3
4	Stack using Array	Stack	Eclipse	CLO 2, PLO 3
5	Postfix to Infix		Eclipse	CLO 2, PLO 3
6	Queue Using Array	Queue	Eclipse	CLO 2, PLO 3
7	Simple Linked List	Linked List	Eclipse	CLO 1, PLO 5
8	Double Ended Linked List		Eclipse	CLO 1, 2, PLO 3
9	Sorted Linked List		Eclipse	CLO 1, 2, PLO 3
10	Doubly Linked List		Eclipse	CLO 1, 2, PLO 3
11	Stack using Linked List	Stack	Eclipse	CLO 2, PLO 3
12	Queue using Linked List	Queue	Eclipse	CLO 2, PLO 3
13	Tree	Tree	Eclipse	CLO 1, PLO 5
14	Insertion in Tree		Eclipse	CLO 1, 2, PLO 3
15	Deletion in Tree		Eclipse	CLO 1, 2, PLO 3
16	Graph	Graph	Eclipse	CLO 1, PLO 5

Assessment Criteria		
Teaching Methodology	✚	Demonstration, Practical tasks, Feedback on Practical tasks
Sessional (25 Marks)	✚	Lab Tasks
Mid Term (25 Marks)	✚	Objective +subjective paper
Final Term (50 Marks)	✚	Objective +subjective paper

Note: The above list of experiments is common for this subject in all concerned departments of FICT, however subject instructor shall choose the relevant experiments among this list according to the program requirements

DOCUMENT VERSION DETAILS

Version	Date	Author	Rationale
1.0	**3rd July, 2021**	**Assistant Prof. Muhammad Atif**	**First Draft (Lab Format Design)**

LAB NO. 1

Basic Array Operations

1. Lab outcomes:

After completing this lab, students will be able to;
- Re-establishing arrays and its operations.
- Understanding basic operations for arrays.

2. Outline:

- Arrays
- Basic Operations
- Algorithms

3. Corresponding CLO and PLO:

- CLO 1, PLO 5

4. Background:

As this lab focuses on re-establishment of arrays from previous course, no background is provided.

5. Equipment:

- Eclipse

6. Procedure:

Question 01:

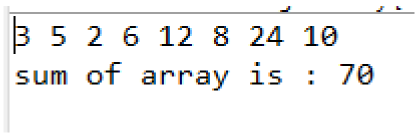
Write a program that displays sum of integer array.

Code:

```
public class SumOfIntegerInArray {

	public static void main(String[] args) {

		int [] arr= {3, 5, 2, 6, 12, 8, 24 ,10};
		int sum=0 ;

		for(int i= 0; i < arr.length; i++) {
			System.out.print (arr[i]+" ");
			sum+=arr[i];
		}
```

```
        System.out.print ("\nsum of array is : "+sum);

    }

}//end class SumOfIntegerInArray
```

Output:



```
3 5 2 6 12 8 24 10
sum of array is : 70
```

Question 2:

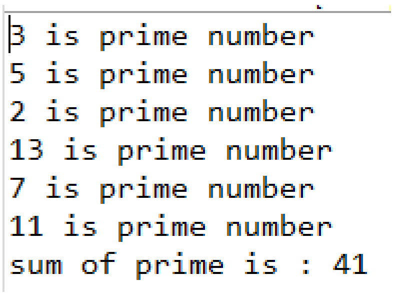
Write a program that find out prime no from integer array. Also, find the sum of prime no then delete these numbers from array.

Code:

```
public class PrimeNumber {
    private static int nElement;
    public static void main(String[] args) {
        int [] arr= {3, 5, 2, 6, 12, 13 ,7 ,8 ,4 ,10 ,11};
        nElement = arr.length;
        int sum=0;
        for(int i=0; i<nElement; i++) {
            if(prime(arr[i])) {
                sum+=delete(arr,i);
                i--;
            }
        }
        System.out.print("sum of prime is : "+ sum);
    }//end main()

    public static Integer delete(int [] arr,int index) {
        if(arr.length==0)
            return 0;
        if(index >= arr.length && index < 0)
            return 0;
        Integer num = arr[index];
        for ( int i=index ; i < nElement-1; i++)
            arr[i] = arr[i+1];

        nElement--;
        return num;
    }//end delete()

    public static boolean prime(int num) {
        int j=0;
```

```
		for (int i=1; i <= num; i++) {
			if(num % i==0)
				j++;
		}
		if(j==2)
			return true;
		else
			return false;
	}//end prime( int )

}
```

Output:



```
3 is prime number
5 is prime number
2 is prime number
13 is prime number
7 is prime number
11 is prime number
sum of prime is : 41
```

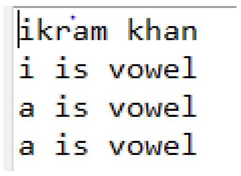
Question 3:

Write a program that find out vowel in character array.

Code:

```
public class VowelInArray {

	public static void main(String[] args) {
		char [] ch = {'i','k','r','a','m','
','k','h','a','n'};
		String vowels="aeiou";
		for(int i=0; i < ch.length; i++) {
			if(contains(vowels,ch[i]))
				System.out.println(ch[i] + " is vowel");
		}
	}//end main()

	public static boolean contains(String vowels, char c) {
		for(int i=0; i < vowels.length() ;i++) {
			if(vowels.charAt(i) == c)
				return true;
		}
		return false;
	}

}
```

Output:



```
ikram khan
i is vowel
a is vowel
a is vowel
```

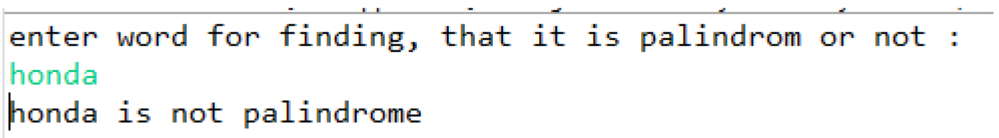
Question 04:

1. Create Character Stack Class, that must include pop, push and peak methods.
2. By using this class Reverse the String (Take string as a input).
3. After reversing String Compare your output string to input string to find out whether it's Palindrome or not.

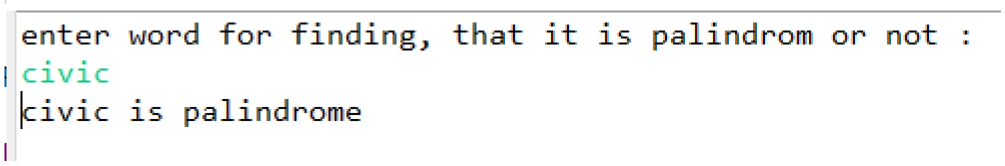
Code:

```
import java.util.Scanner;

class Palindrome {

	public static void main(String[] args) {
		System.out.print("enter word for finding, that it is 
palindrom or not : ");
		Stack s = new Stack(20);
		Scanner sc = new Scanner(System.in);
		String word = sc.next();

		for(int i =0 ; i < word.length(); i++ )
			s.push(word.charAt(i));

		String newWord="";
		for(int i =0 ; i < word.length(); i++ )
			newWord+=s.pop();

		if(word.equals(newWord))
			System.out.print(word + " is palindrome");
		else
			System.out.print(word + " is not palindrome");
	}

}

class Stack{
	private char arr [];
      private int top;
	public Stack(int size) {
		arr=new char[size];
		top=-1;
	}

	public void push(char item) {
		if(top==arr.length)
			return;
		arr[++top]=item;
	}
	public char pop() {
		if(top==-1)
			return 0;
		return arr[top--];
	}

	public char peak() {
		if(top==-1)
			return 0;

		return arr[top];
	}
    }
```

<u>Output</u>:



```
enter word for finding, that it is palindrom or not :
honda
honda is not palindrome
```



```
enter word for finding, that it is palindrom or not :
civic
civic is palindrome
```

7. Observations:

At the end of this lab, I was able to reestablish fundamentals of arrays and it’s most basic algorithms.

Rubrics

Demonstration	Absent	Student is unable to follow the provided instructions properly. The student can name the hardware or simulation platform, but unable to implement anything practically or on the software	Student can understand the provided laboratory instructions and familiar with the lab environment (Trainer/ software/ IDE), but cannot implement on the platform practically or on the software	Student has followed instructions to construct the fundamental schematic/ block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code, and have successfully executed the program/ run circuit on software platform	Student perfectly implemented a working model/ logic/ circuit/ block diagram/ code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory Reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recorded along with detailed procedure	Appropriate computations or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all respects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

LAB NO. 2

ARRAY UTILITY

1. Objectives:

After completing this lab, students will be able to;

- Gain understanding arrays.
- Understand and Implement different components of arrays.

2. Outline:

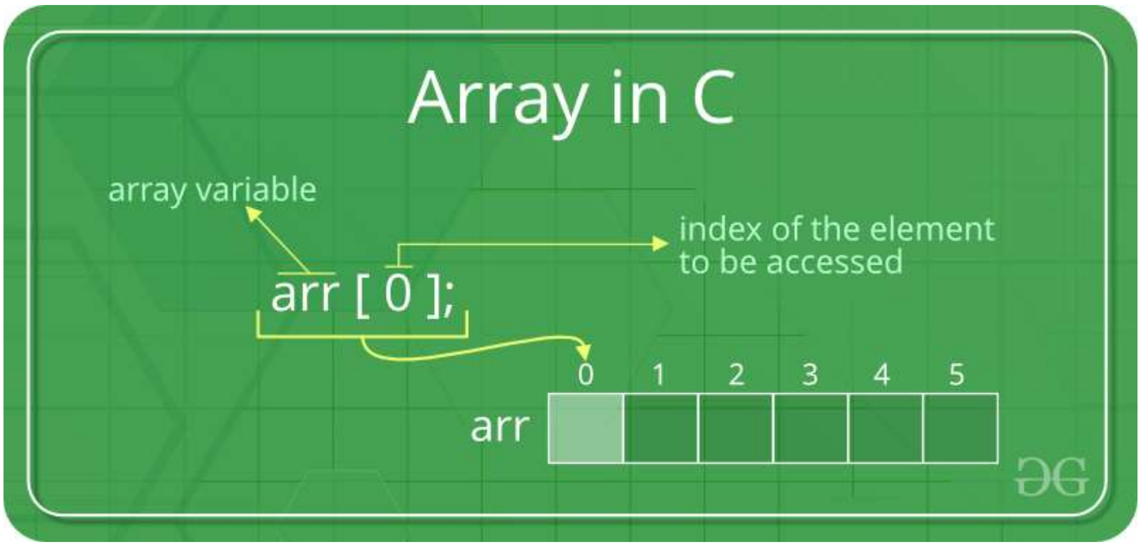
- Arrays.
- Algorithms for arrays
- Practical Implementation
- Examples.

3. Corresponding CLO and PLO:

- CLO 2, PLO 3

4. Background:

The array is the most commonly used data storage structure; it's built into most programming languages. e Because they are so well known, arrays offer a convenient jumping off place for introducing data structures and for seeing how object-oriented programming and data structures relate to each other.



Basic Array Algorithm:

Traversing:

Algorithm 4.1: (Traversing a linear Array) Here **LA** is a linear array with lower bound **LB** and Upper bound **UB**. This algorithm traverses **LA** applying an operation **PROCESS** to each element of **LA**.

```
1.  [Initialize counter.] Set K:=LB.
2.  Repeat Step 3 and 4 while K ≤UB.
3.          [Visit elements.]   Apply PROCESS to LA[K].
4.          [Increase Counter.]  Set K:= K+1
    [End of Sep 2 Loop]
5.  Exit
```

Insertion:

Algorithm 4.2:

(Inserting into a Linear Array) INSERT(LA,N,K,ITEM) Here LA is the linear array with N elements and K is a positive integer such that K≤N. This algorithm inserts an element ITEM into the Kth position in LA.

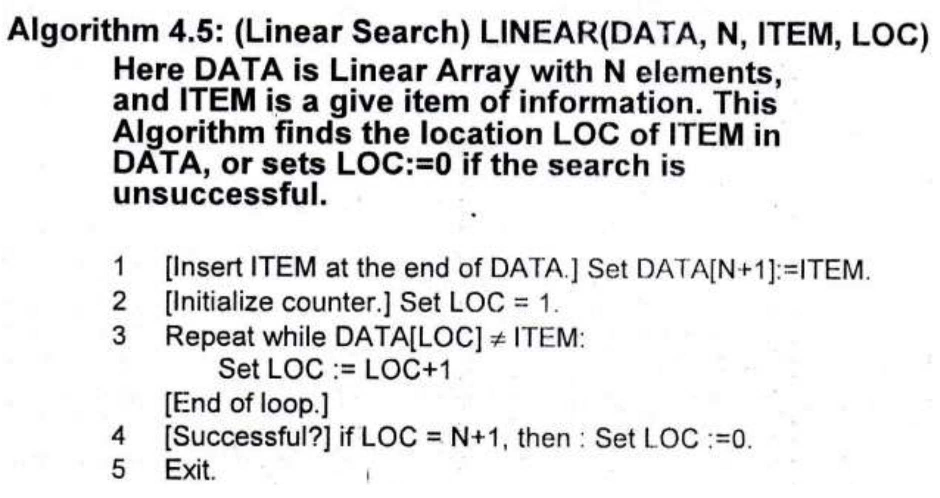
```
1.  [initialize counter.]  Set J:=N
2.  Repeat Step 3 and 4 while J≥K
3.      [Move Jth element downward.] Set LA[j+1]:=LA[j].
4.      [Decrease counter.] Set J:= j-1.
    [End of Step 2 Loop]
5.  [Insert element.] Set LA[K] := ITEM.
6.  [Reset N.]   Set N:=N+1.
7.  Exit
```

Deletion Array:

Algorithm 4.3: (Deleting from a Linear Array) DELETE(LA, N, K, ITEM) Here LA is a linear array with N elements and K is a positive integer such that K≤N. This algorithm deletes the Kth element from LA.

```
1.  Set ITEM :=LA[K].
2.  Repeat for J=K to N-1:
        [Move J+1st Element upward.] Set LA[J] = LA[J+1].
    [End of loop]
3.  [Reset number N of elements in LA.] Set N:=N-1.
4.  Exit
```

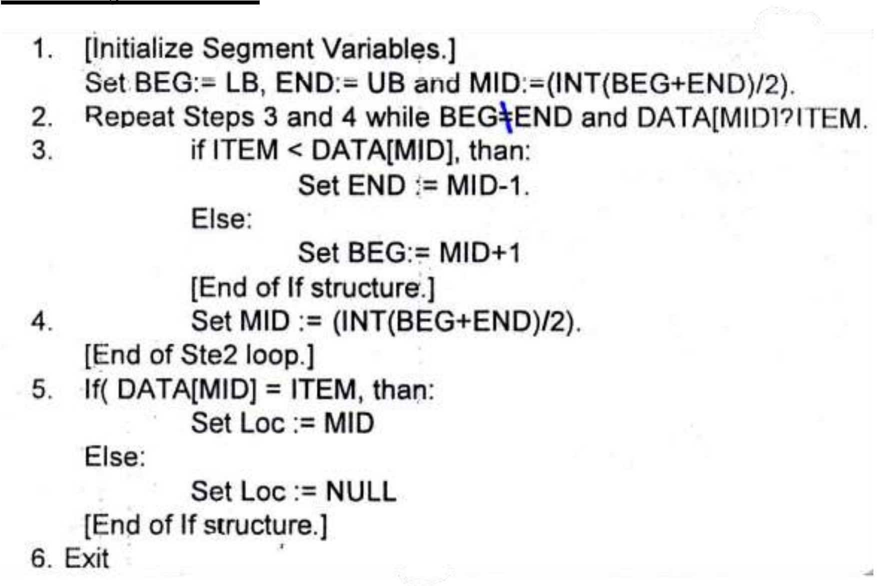
Linear Search:



Algorithm 4.5: (Linear Search) LINEAR(DATA, N, ITEM, LOC)
Here DATA is Linear Array with N elements, and ITEM is a give item of information. This Algorithm finds the location LOC of ITEM in DATA, or sets LOC:=0 if the search is unsuccessful.

```
1   [Insert ITEM at the end of DATA.] Set DATA[N+1]:=ITEM.
2   [Initialize counter.] Set LOC = 1.
3   Repeat while DATA[LOC] ≠ ITEM:
        Set LOC := LOC+1
    [End of loop.]
4   [Successful?] if LOC = N+1, then : Set LOC :=0.
5   Exit.
```

Bineary Search:



```
1.  [Initialize Segment Variables.]
    Set BEG:= LB, END:= UB and MID:=(INT(BEG+END)/2).
2.  Repeat Steps 3 and 4 while BEG≤END and DATA[MID]?ITEM.
3.          if ITEM < DATA[MID], than:
                    Set END := MID-1.
            Else:
                    Set BEG:= MID+1
            [End of If structure.]
4.          Set MID := (INT(BEG+END)/2).
    [End of Ste2 loop.]
5.  If( DATA[MID] = ITEM, than:
            Set Loc := MID
    Else:
            Set Loc := NULL
    [End of If structure.]
6. Exit
```

5. Equipment:

Eclipse.

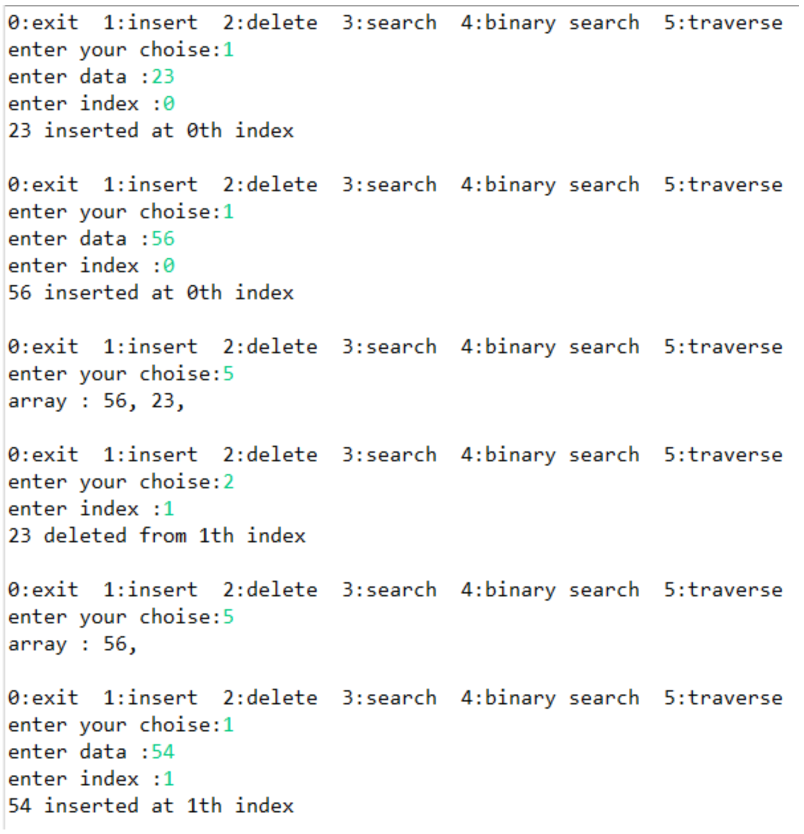
6. Procedure:

Code:

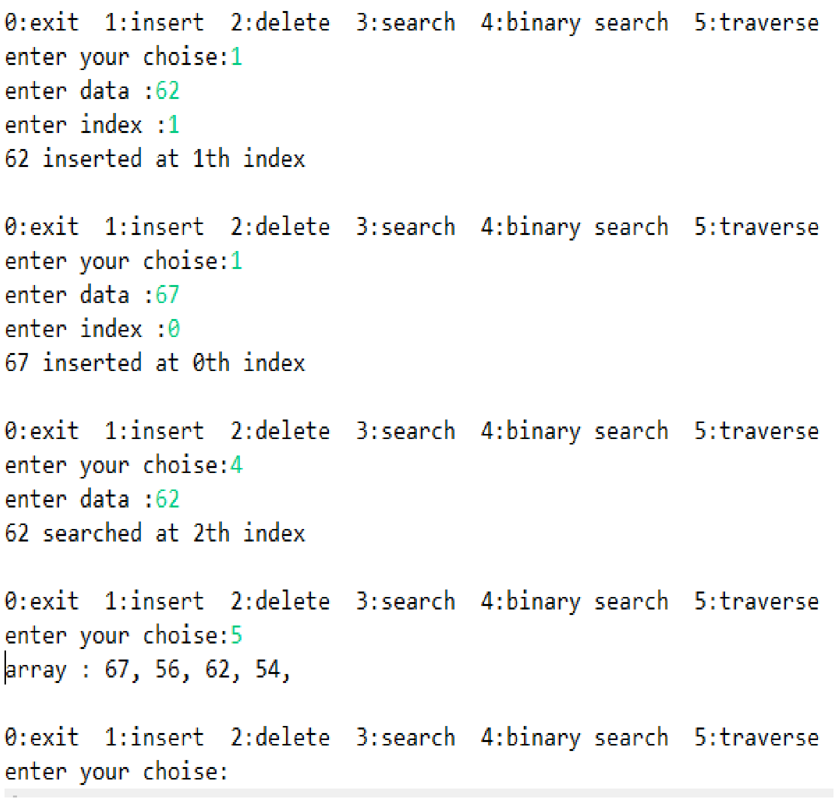
```java
import java.util.Scanner;

public class Lab2 {
	public static void main(String args[]) {

		Array a = new Array(10);
		Scanner sc = new Scanner(System.in);
		boolean run = true;
		while (run) {
			System.out.print(
					"0:exit  1:insert  2:delete  3:search
4:binary search  5:traverse" + "\nenter your choise:");
			int ch = 0;
			ch = sc.nextInt();

			switch (ch) {
			case 0:

				run = false;

				break;
			case 1:
				System.out.print("enter data :");
				int data = sc.nextInt();
				System.out.print("enter index :");
				System.out.println(a.insert(data, sc.nextInt()));
				break;
			case 2:

				System.out.print("enter index :");
				System.out.println(a.delete(sc.nextInt()));

				break;
			case 3:
				System.out.print("enter data :");
				System.out.println(a.search(sc.nextInt()));

				break;
			case 4:

				System.out.print("enter data :");
				System.out.println(a.bSearch(sc.nextInt()));

				break;
			case 5:

				System.out.println(a.traverse());

				break;
			default:
				System.out.println("invalid option");
			}// end switch
		} // end while loop
	} // end void main()

} // end Lab class


class Array{
    private int arr[];
    int nElement;
    int size;

public Array(int s) {
        size = s;
        arr = new int [s];
        nElement = 0;
}

public String traverse() {
         if(nElement == 0)
           return    "array is empty   \n" ;
        String output= "array : ";
        for (int i=0; i<nElement; i++)
                output += arr[i] + ", ";
        return output+"\n";
}//end traverse()

public String insert( int data , int i) {

       if(nElement >= size)
            return    "array is full   \n";
       if(i > nElement || i < 0)
          return    "invalid index   \n";


       for(int j = nElement-1 ; j >= i ; j--)
          arr[j+1]=arr[j];
       arr[i]=data;
       nElement++;
       return data +   " inserted at " +i+   "th index \n  ";
   }//end insert()


   public String delete(int i) {
       if(nElement == 0)
          return    "array is empty   \n" ;
       if(i > nElement-1|| i < 0)
          return    "invalid index \n  " ;

       int data = arr[i];
       for(int j = i ; j < nElement-1 ;j++)
          arr[j] = arr[j+1];
       nElement--;
       return data +   " deleted from "+i+"th index   \n";
   }//end delete()


   public String search(int data ) {
           if(nElement == 0)
                  return    "array is empty  \n " ;

         Integer index=null;
         for(int i = 0 ; i < nElement ;i++) {
            if(data == arr[i]){
              index = i;
```

```
                        break;
                         }
                }
                   if(index == null)
                    return     "data is not available \n  ";

                return data +   " searched at "+index +   "th index
\n";
              }//end search()

              public String bSearch( int item) {
                     if(nElement == 0)
                           return     "array is empty  \n " ;

                 int beg = 0, end = nElement-1, mid = (beg + end) / 2;
                 while (beg < end && arr[mid] != item) {
                    if ( item < arr[mid])
                       end = mid - 1;

                    else
                       beg = mid + 1;

                    mid = (beg + end) / 2;
                 } // end while loop
                 if ( arr[mid] != (item))
                          return     "data is not available \n  ";
                 else
                    return arr[mid]+   " searched at "+mid +   "th
index\n";
              }// end bSearch()

}
```

Output:



```
0:exit  1:insert  2:delete  3:search  4:binary search  5:traverse
enter your choise:1
enter data :23
enter index :0
23 inserted at 0th index

0:exit  1:insert  2:delete  3:search  4:binary search  5:traverse
enter your choise:1
enter data :56
enter index :0
56 inserted at 0th index

0:exit  1:insert  2:delete  3:search  4:binary search  5:traverse
enter your choise:5
array : 56, 23,

0:exit  1:insert  2:delete  3:search  4:binary search  5:traverse
enter your choise:2
enter index :1
23 deleted from 1th index

0:exit  1:insert  2:delete  3:search  4:binary search  5:traverse
enter your choise:5
array : 56,

0:exit  1:insert  2:delete  3:search  4:binary search  5:traverse
enter your choise:1
enter data :54
enter index :1
54 inserted at 1th index
```



```
0:exit  1:insert  2:delete  3:search  4:binary search  5:traverse
enter your choise:1
enter data :62
enter index :1
62 inserted at 1th index

0:exit  1:insert  2:delete  3:search  4:binary search  5:traverse
enter your choise:1
enter data :67
enter index :0
67 inserted at 0th index

0:exit  1:insert  2:delete  3:search  4:binary search  5:traverse
enter your choise:4
enter data :62
62 searched at 2th index

0:exit  1:insert  2:delete  3:search  4:binary search  5:traverse
enter your choise:5
array : 67, 56, 62, 54,

0:exit  1:insert  2:delete  3:search  4:binary search  5:traverse
enter your choise:
```

7. Observations:

At the end of this lab, I was able to reestablish fundamentals of arrays and it's most basic algorithms.

Rubrics

Demonstration	Absent	Student is unable to follow the provided instructions properly. The student can name the hardware or simulation platform, but unable to implement anything practically or on the software	Student can understand the provided laboratory instructions and familiar with the lab environment (Trainer/ software/ IDE), but cannot implement on the platform practically or on the software	Student has followed instructions to construct the fundamental schematic/ block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code, and have successfully executed the program/ run circuit on software platform	Student perfectly implemented a working model/ logic/ circuit/ block diagram/ code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory Reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recorded along with detailed procedure	Appropriate computations or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all respects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

LAB NO. 3

Array Sorting

1. Lab outcomes:

After completing this lab, students will be able to;

- Re-establishing arrays and its operations.
- Understanding basic operations for arrays.

2. Outline:

- Arrays
- Basic Operations
- Algorithms

3. Corresponding CLO and PLO:

- CLO 2, PLO 3

4. Background:

As soon as you create a significant database, you'll probably think of reasons to sort it in various ways. You need to arrange names in alphabetical order, students by grade, customers by zip code, home sales by price, cities in increasing population, and stars by magnitude, and so on.

Sorting data may also be a preliminary step to searching it. As we saw in the last lecture, a binary search, which can be applied only to sorted data, is much faster than a linear search. Because sorting is so important and potentially so time-consuming, it has been the subject of extensive research in computer science, and some very sophisticated methods have been developed.

Sorting by Computer:

A computer program isn't able to glance over the data in this way. It can only compare two players at once, because that's how the comparison operators work. Things may seem simple to us humans, out the algorithm can't see the big picture and must, therefore, concentrate on the details and follow some simple rules.

The three algorithms in this session all involve two steps, executed over and over until the data is sorted:

1. Compare two items.
2. Swap two items or copy one item. .

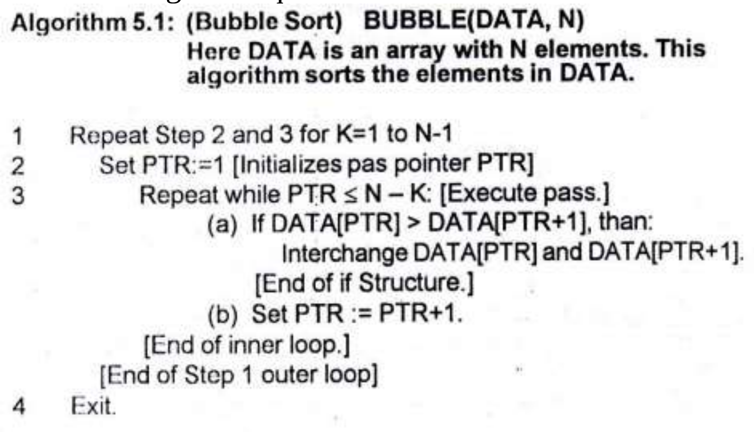
However, each algorithm handles the details in a different way.

ln this session we'll look at three of the simpler algorithms:

Bubble Sort
Selection Sort
The insertion sort
They are discussed below:

Bubble Sort:

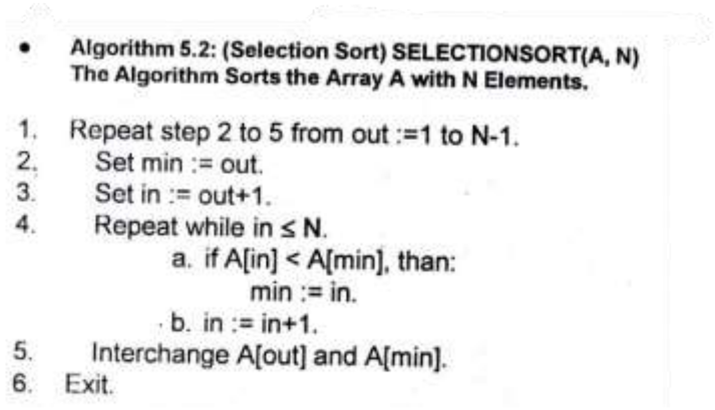
The bubble sort is notoriously slow, but it's conceptually the simplest of the sorting algorithms, and for that reason is a good beginning for our exploration of sorting techniques.



Algorithm 5.1: (Bubble Sort) BUBBLE(DATA, N)
Here DATA is an array with N elements. This algorithm sorts the elements in DATA.

```
1   Repeat Step 2 and 3 for K=1 to N-1
2     Set PTR:=1 [Initializes pas pointer PTR]
3         Repeat while PTR ≤ N – K: [Execute pass.]
                (a) If DATA[PTR] > DATA[PTR+1], than:
                        Interchange DATA[PTR] and DATA[PTR+1].
                    [End of if Structure.]
                (b) Set PTR := PTR+1.
          [End of inner loop.]
      [End of Step 1 outer loop]
4   Exit.
```

Selection Sort:



- **Algorithm 5.2: (Selection Sort) SELECTIONSORT(A, N)**
  The Algorithm Sorts the Array A with N Elements.

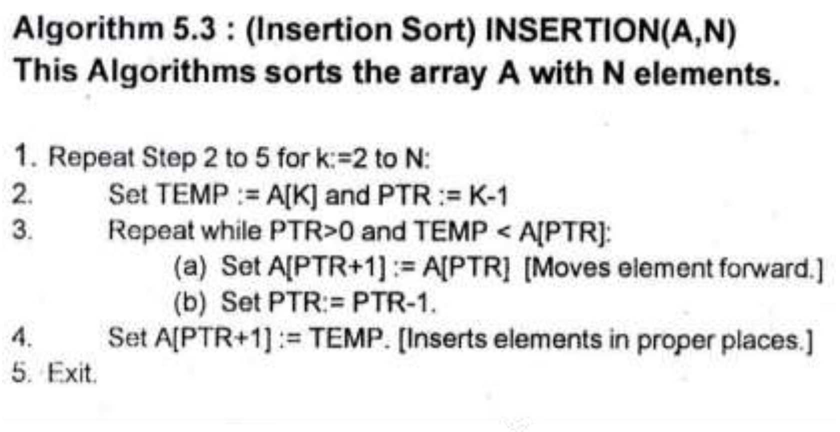
```
1.  Repeat step 2 to 5 from out :=1 to N-1.
2.    Set min := out.
3.    Set in := out+1.
4.    Repeat while in ≤ N.
            a. if A[in] < A[min], than:
                   min := in.
            b. in := in+1.
5.    Interchange A[out] and A[min].
6.  Exit.
```

Insertion Sort:

Insertion sort is an efficient sorting algorithm for sorting a small number of elements. Insertion sort works the way many people sort a hand of playing cards:

We start with an empty left hand and the cards face down on the table. We then remove one card at a time from the table and insert it into the correct position in the left hand.

To find the correct position for a card, we compare it with each of the cards already in the hand, from right to left, as illustrated in Figure. At all times, the cards held in the left hand are sorted, and these cards were originally the top cards of the pile on the table.



Algorithm 5.3 : (Insertion Sort) INSERTION(A,N)
This Algorithms sorts the array A with N elements.

```
1. Repeat Step 2 to 5 for k:=2 to N:
2.      Set TEMP := A[K] and PTR := K-1
3.      Repeat while PTR>0 and TEMP < A[PTR]:
             (a) Set A[PTR+1] := A[PTR] [Moves element forward.]
             (b) Set PTR:= PTR-1.
4.      Set A[PTR+1] := TEMP. [Inserts elements in proper places.]
5. Exit.
```

5. Equipment:

- Eclipse

6. Procedure:

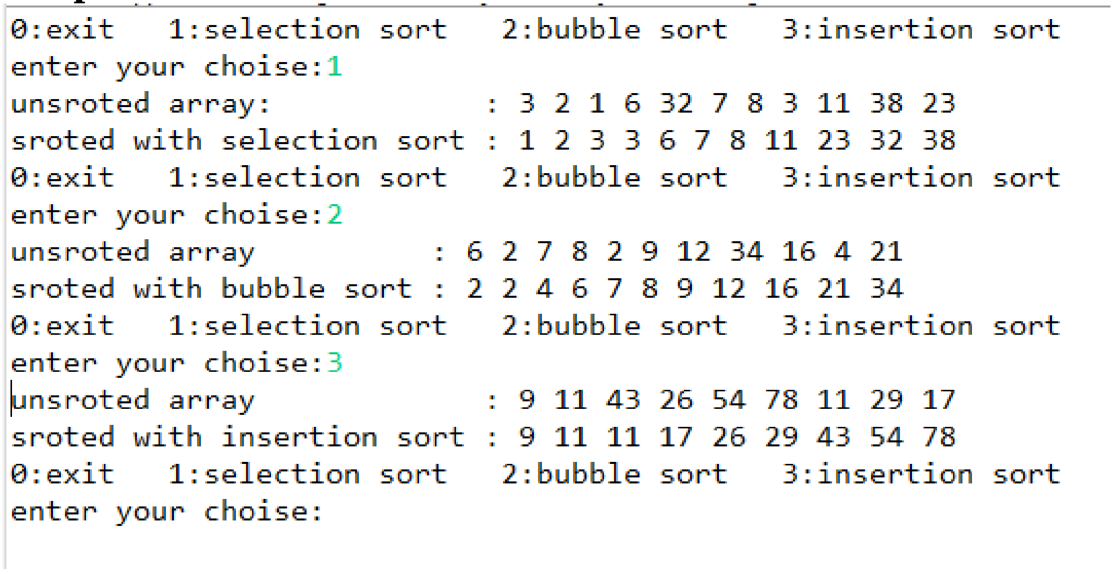
Code:

```
import java.util.Scanner;

public class Lab3 {

	public static void main(String args[]) {

		int arr1[] = { 3, 2, 1, 6, 32, 7, 8, 3, 11, 38, 23 };
		int arr2[] = { 6, 2, 7, 8, 2, 9, 12, 34, 16, 4, 21 };
		int arr3[] = { 9, 11, 43, 26, 54, 78, 11, 29, 17 };

		Scanner sc = new Scanner(System.in);
		boolean run = true;
		while (run) {
			System.out.print("0:exit   1:selection sort   2:bubble
sort   3:insertion sort"
		+ "\nenter your choise:");
			int ch = 0;
			ch = sc.nextInt();

			switch (ch) {
			case 0:

				run = false;
				System.out.println(".....exit.....");
				break;
			case 1:

				System.out.println("unsroted array:            :
" + arrayToString(arr1, arr1.length));
				selectionSort(arr1, arr1.length);
				System.out.println("sroted with selection sort :
" + arrayToString(arr1, arr1.length));

				break;
```

```
                    case 2:

                        System.out.println("unsroted array          : " +
arrayToString(arr2, arr2.length));
                        bubbleSort(arr2, arr2.length);
                        System.out.println("sroted with bubble sort : " +
arrayToString(arr2, arr2.length));

                        break;
                    case 3:

                        System.out.println("unsroted array             :
" + arrayToString(arr3, arr3.length));
                        insertionSort(arr3, arr3.length);
                        System.out.println("sroted with insertion sort :
" + arrayToString(arr3, arr3.length));

                        break;

                    default:
                        System.out.println("invalid option");
                    }// end switch
            } // end while loop
        } // end void main()

        public static String arrayToString(int[] arr, int nElement) {
            String stringArray = "";
            for (int i = 0; i < nElement; i++)
                stringArray += arr[i] + " ";
            return stringArray;
        }

        public static void selectionSort(int[] arr, int nElement) {
            for (int i = 0; i < nElement - 1; i++) {
                int imin = i;
                for (int j = i + 1; j < nElement; j++) {
                    if (arr[j] < arr[imin])
                        imin = j;
                } // end in for loop
                int temp = arr[imin];
                arr[imin] = arr[i];
                arr[i] = temp;
            } // end out for loop
        }// end selectionSort()

        public static void insertionSort(int[] arr, int nElement) {
            for (int i = 1; i < nElement; i++) {

                int temp = arr[i];
                int j = i - 1;
                while (j >= 0 && temp < arr[j]) {
                    arr[j + 1] = arr[j];
                    j--;
                } // end in loop
                arr[j + 1] = temp;
            } // end out loop
        }// end insert()
```

```
    public static void bubbleSort(int[] arr, int nElement) {
        for (int i = 0; i < nElement - 1; i++) {
            for (int j = 0; j < nElement - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                } // end if
            } // end in for
        } // end out for
    }// end bubbleSort()

} // end Lab class
```

Output:



```
0:exit   1:selection sort   2:bubble sort   3:insertion sort
enter your choise:1
unsroted array:              : 3 2 1 6 32 7 8 3 11 38 23
sroted with selection sort : 1 2 3 3 6 7 8 11 23 32 38
0:exit   1:selection sort   2:bubble sort   3:insertion sort
enter your choise:2
unsroted array             : 6 2 7 8 2 9 12 34 16 4 21
sroted with bubble sort : 2 2 4 6 7 8 9 12 16 21 34
0:exit   1:selection sort   2:bubble sort   3:insertion sort
enter your choise:3
unsroted array               : 9 11 43 26 54 78 11 29 17
sroted with insertion sort : 9 11 11 17 26 29 43 54 78
0:exit   1:selection sort   2:bubble sort   3:insertion sort
enter your choise:
```

7. Observations:

At the end of this lab, I was able to reestablish fundamentals of sortion of arrays and it’s algorithms.

Rubrics

Demonstration	Absent	Student is unable to follow the provided instructions properly. The student can name the hardware or simulation platform, but unable to implement anything practically or on the software	Student can understand the provided laboratory instructions and familiar with the lab environment (Trainer/ software/ IDE), but cannot implement on the platform practically or on the software	Student has followed instructions to construct the fundamental schematic/ block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code, and have successfully executed the program/ run circuit on software platform	Student perfectly implemented a working model/ logic/ circuit/ block diagram/ code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory Reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recorded along with detailed procedure	Appropriate computations or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all respects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

LAB NO. 4

Stack using Array

1. Objectives:

After completing this lab, students will be able to;
- Introduction to Stacks.
- Implementation of stack in arrays.
- Procedures and Algorithm

2. Outline:

- Array
- Stack
- Practical Implementation
- Example

3. Corresponding CLO and PLO:

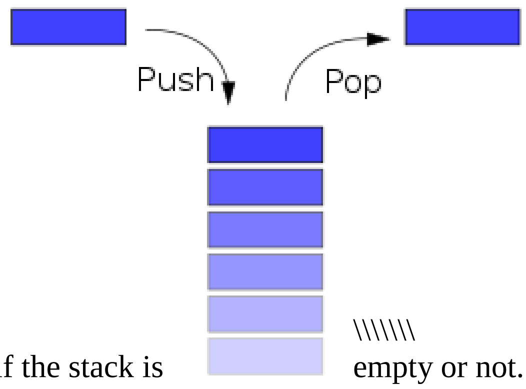
- CLO 2, PLO 3

4. Background:

A stack is a linear data structure that follows the LIFO (Last–In, First–Out) principle. That means the objects can be inserted or removed only at one end of it, also called a top.

The stack supports the following operations:

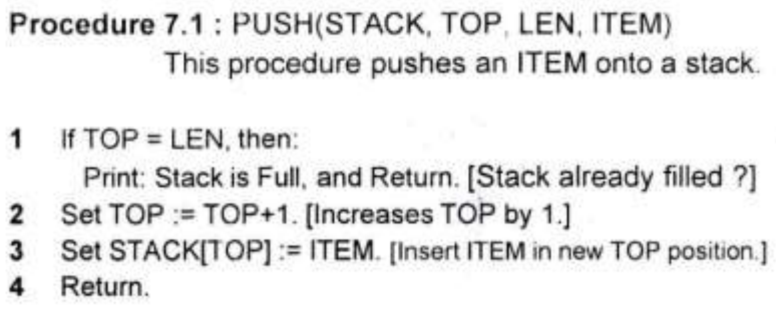
- **Push** inserts an item at the top of the stack (i.e., above its current top element).
- **Pop** removes the object at the top of the stack and returns that object from the function. The stack size will be decremented by one.



\\\\\\\

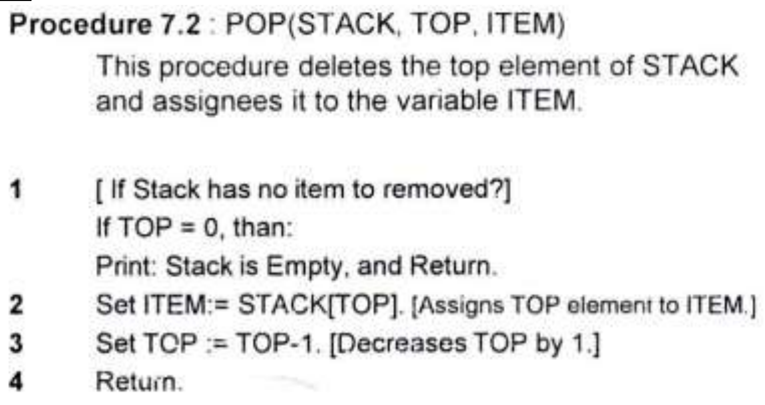
- **isEmpty** tests if the stack is empty or not.
- **isFull** tests if the stack is full or not.
- **Peek** returns the object at the top of the stack without removing it from the stack or modifying the stack in any way.
- **Size** returns the total number of elements present in the stack.

Push:



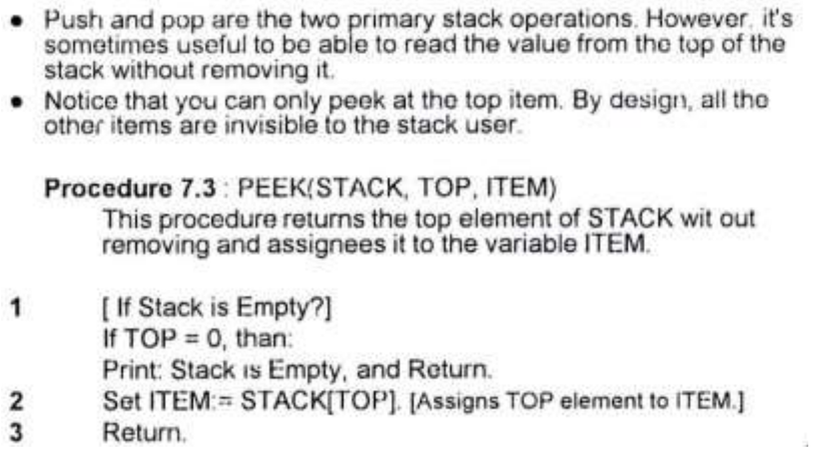
```
Procedure 7.1 : PUSH(STACK, TOP, LEN, ITEM)
        This procedure pushes an ITEM onto a stack.

1   If TOP = LEN, then:
      Print: Stack is Full, and Return. [Stack already filled ?]
2   Set TOP := TOP+1. [Increases TOP by 1.]
3   Set STACK[TOP] := ITEM. [Insert ITEM in new TOP position.]
4   Return.
```

Pop:



```
Procedure 7.2 : POP(STACK, TOP, ITEM)
      This procedure deletes the top element of STACK
      and assignees it to the variable ITEM.

1     [ If Stack has no item to removed?]
      If TOP = 0, than:
      Print: Stack is Empty, and Return.
2     Set ITEM:= STACK[TOP]. [Assigns TOP element to ITEM.]
3     Set TOP := TOP-1. [Decreases TOP by 1.]
4     Return.
```

Peek:



- Push and pop are the two primary stack operations. However, it's sometimes useful to be able to read the value from the top of the stack without removing it.
- Notice that you can only peek at the top item. By design, all the other items are invisible to the stack user.

```
Procedure 7.3 : PEEK(STACK, TOP, ITEM)
      This procedure returns the top element of STACK wit out
      removing and assignees it to the variable ITEM.

1     [ If Stack is Empty?]
      If TOP = 0, than:
      Print: Stack is Empty, and Return.
2     Set ITEM:= STACK[TOP]. [Assigns TOP element to ITEM.]
3     Return.
```

5. Equipment:

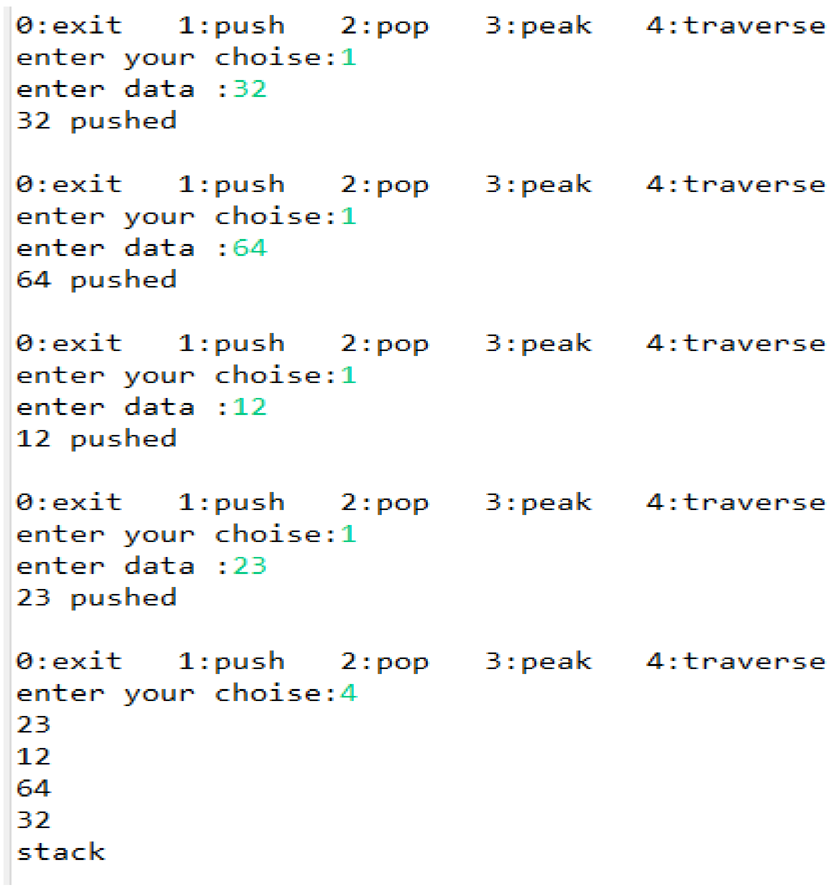
Eclipse

6. Procedure:

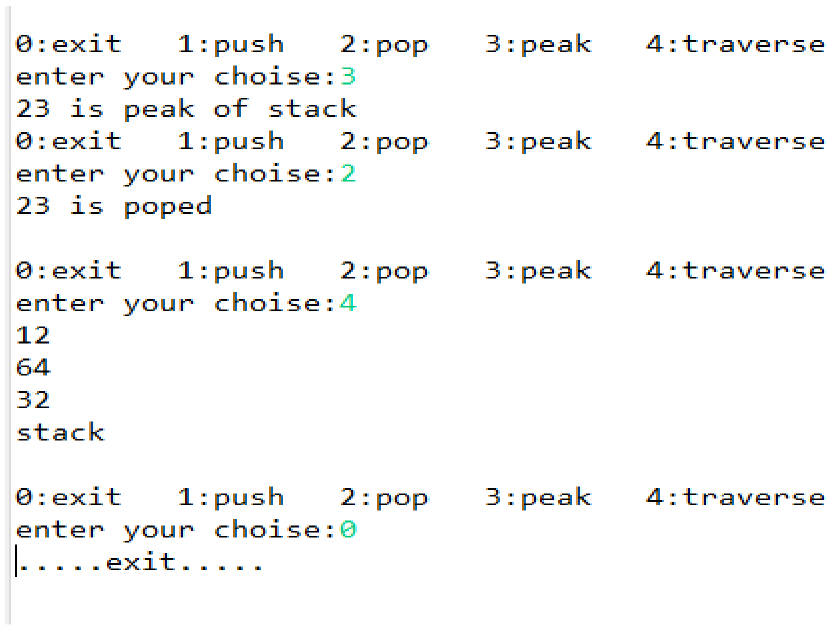
Code:

```java
import java.util.Scanner;

public class Lab4 {

	public static void main(String args[]) {
			StackArray s = new StackArray(10);
			Scanner sc = new Scanner(System.in);
			boolean run = true;
			while (run) {
				System.out.print("0:exit   1:push   2:pop   3:peak   
4:traverse" + "\nenter your choise:");
				int ch = 0;
				ch = sc.nextInt();

				switch (ch) {
				case 0:

					run = false;
					System.out.println(".....exit.....");
					break;
				case 1:

					System.out.print("enter data :");
					System.out.println(s.push(sc.nextInt()));

					break;
				case 2:
					System.out.println(s.pop());

					break;
				case 3:
					System.out.print(s.peak());

					break;
				case 4:
					System.out.print(s.traverse());
					break;

				default:
					System.out.println("invalid option");
				}// end switch
			} // end while loop
		} // end void main()

} // end Lab class

class StackArray {
	private int arr[];
	private int top;

	public StackArray(int size) {
		arr = new int[size];
		top = -1;
	}

	public String traverse() {
		if (top == -1)
			return "Stack overflow\n";
		String output = "stack \n ";
		for (int i = 0; i <= top; i++)
			output = arr[i] + "\n" + output;
		return output + "\n";
	}// end traverse()
	public String push(int item) {
		if (top == arr.length)
			return "Stack overflow\n";
		arr[++top] = item;
		return item + " pushed\n";
	}// end push()

	public String pop() {
		if (top == -1)
			return "stack underflow\n";
		return arr[top--] + " is poped\n";
	}// end pop()

	public String peak() {
		if (top == -1)
			return "stack underflow\n";

		return arr[top] + " is peak of stack\n";
	}
}// end stack
```

Output:



```
0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:1
enter data :32
32 pushed

0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:1
enter data :64
64 pushed

0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:1
enter data :12
12 pushed

0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:1
enter data :23
23 pushed

0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:4
23
12
64
32
stack
```



```
0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:3
23 is peak of stack
0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:2
23 is poped

0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:4
12
64
32
stack

0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:0
.....exit.....
```

7. Observations:

At the end of this lab, I was able to reestablish fundamentals of stack with arrays and it's most basic algorithms.

Rubrics

Demonstration	Absent	Student is unable to follow the provided instructions properly. The student can name the hardware or simulation platform, but unable to implement anything practically or on the software	Student can understand the provided laboratory instructions and familiar with the lab environment (Trainer/ software/ IDE), but cannot implement on the platform practically or on the software	Student has followed instructions to construct the fundamental schematic/ block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code, and have successfully executed the program/ run circuit on software platform	Student perfectly implemented a working model/ logic/ circuit/ block diagram/ code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory Reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recorded along with detailed procedure	Appropriate computations or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all respects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

LAB NO. 5

Postfix to Infix using Stack

1. Lab outcomes:

After completing this lab, students will be able to;
- Re-establishing arrays and its operations.
- Understanding basic operations for arrays.

2. Outline:

- Arrays
- Basic Operations
- Algorithms

3. Corresponding CLO and PLO:

- CLO 2, PLO 3

4. Background:

Another very important application of stack is parsing (that is, analyzing) arithmetic expressions like

2+3
or 2. (3++)
or ((2+4)*7)+3*(9-5)

Parsing Arithmetic Expression:

As it turns out, it's fairly difficult, at least for a computer algorithm, to evaluate an arithmetic expression directly. It's easier for the algorithm to use a two-step process: 1. Transform the arithmetic expression into a different format, called postfix notation. 2. Evaluate the postfix expression.

Step 1 is a bit involved, but step 2 is easy. In any case, this two-step approach results in a simpler algorithm than trying to parse the arithmetic expression directly. Of course, for a human it's easier to parse the ordinary arithmetic expression. We'll return to the difference between the human and computer approaches in a moment. Before we delve into the details of steps 1 and 2, we'll introduce postfix notation.

Infix Notation:

Everyday arithmetic expressions are written with an operator (+, *, or /) placed between two operands (numbers, or symbols that stand for numbers). This is called infix notation, because the operator is written inside the operands.

For Example A+B C-D E*F G/H

In Infix Notation we must distinguish between (A+B)*C and A+(B*C)

The order of the operators and operands in arithmetic expression doesn't uniquely determine the order in which the operations are to be performed but by using parenthesis and operator-precedence level we determine the order evaluation.

Polish Notation (Prefix Notation):

Polish notation (Infix notation) named after the polish mathematician Jan Jukasiewicz, refers to the notation in which the operator symbol is placed before its two operands. For Example +AB -CD *EF /GH

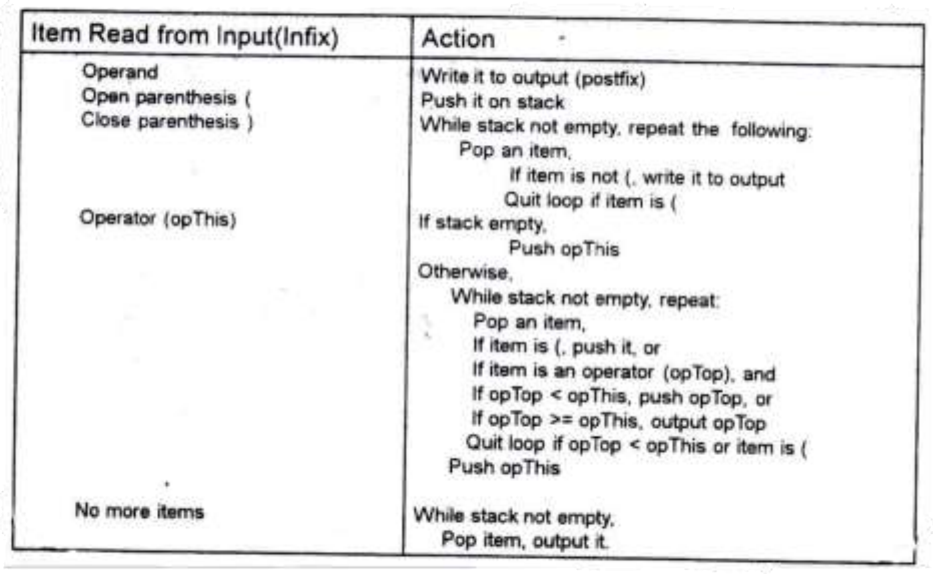
The Fundamental Property of Polish notation is that the order in which the operations are to be performed is completely determined bye the positions of the operators and operands in expression. Accordingly we never need parenthesis when writing expression in Polish notation.

Reverse Polish Notation:

In Postfix notation (which is also called Reverse polish Notation, or RFN, the operator follows the two operands.

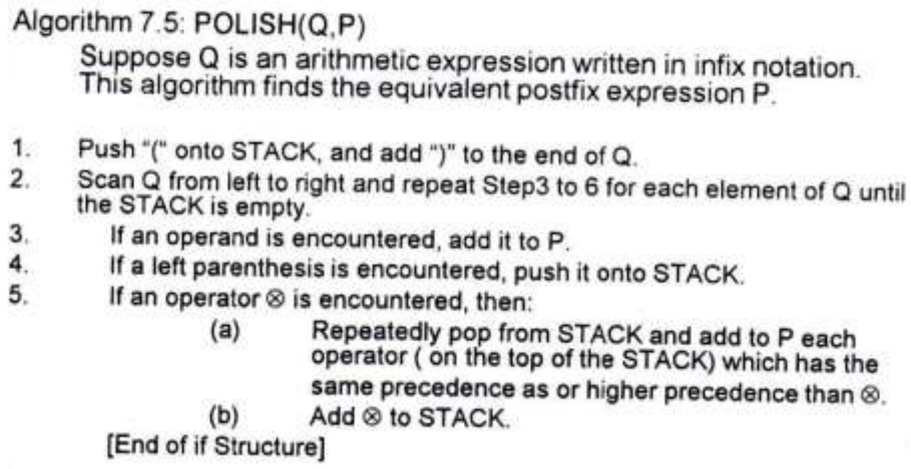
For Example AB+ CD- EF* GH/

Translation Rules:

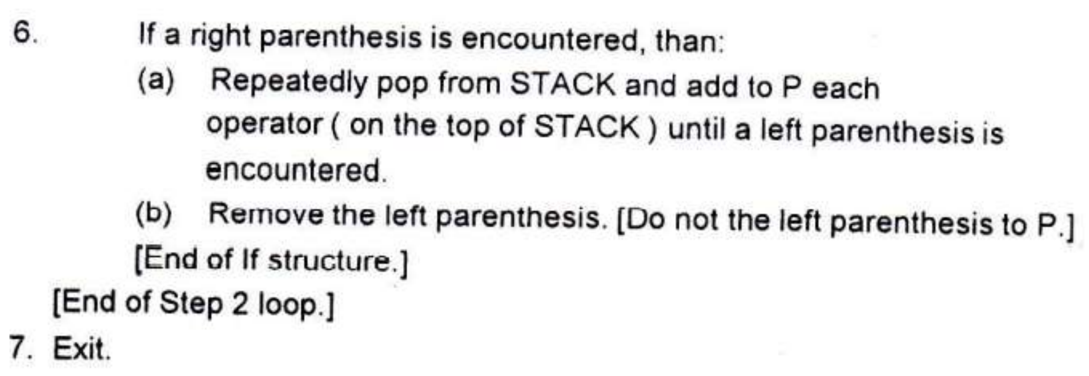


Item Read from Input(Infix)	Action
Operand	Write it to output (postfix)
Open parenthesis (	Push it on stack
Close parenthesis)	While stack not empty, repeat the following: Pop an item, If item is not (, write it to output Quit loop if item is (
Operator (opThis)	If stack empty, Push opThis Otherwise, While stack not empty, repeat: Pop an item, If item is (, push it, or If item is an operator (opTop), and If opTop < opThis, push opTop, or If opTop >= opThis, output opTop Quit loop if opTop < opThis or item is (Push opThis
No more items	While stack not empty, Pop item, output it.

Algorithm:



```
Algorithm 7.5: POLISH(Q,P)
        Suppose Q is an arithmetic expression written in infix notation.
        This algorithm finds the equivalent postfix expression P.

1.      Push "(" onto STACK, and add ")" to the end of Q.
2.      Scan Q from left to right and repeat Step3 to 6 for each element of Q until
        the STACK is empty.
3.          If an operand is encountered, add it to P.
4.          If a left parenthesis is encountered, push it onto STACK.
5.          If an operator ⊗ is encountered, then:
                    (a)     Repeatedly pop from STACK and add to P each
                            operator ( on the top of the STACK) which has the
                            same precedence as or higher precedence than ⊗.
                    (b)     Add ⊗ to STACK.
            [End of if Structure]
```



```
6.        If a right parenthesis is encountered, than:
          (a)  Repeatedly pop from STACK and add to P each
               operator ( on the top of STACK ) until a left parenthesis is
               encountered.
          (b)  Remove the left parenthesis. [Do not the left parenthesis to P.]
          [End of If structure.]
     [End of Step 2 loop.]
7.  Exit.
```

5. Equipment:

- Eclipse

6. Procedure:

Code:

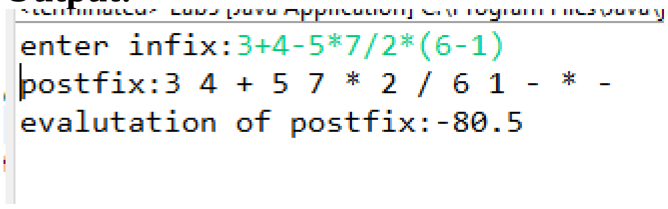
```
import java.util.Scanner;

public class Lab5 {
        static public Scanner sc = new Scanner(System.in);

        public static void main(String[] args) {

                System.out.print("enter infix:");
                String infix = sc.next();
                String postfix = postfix(infix);
                System.out.println("postfix:" + postfix);
                System.out.println("evalutation of postfix:" +
evaluate(postfix));
        }// end main()

        static public String postfix(String infix) {
                infix += ')';
                int size = infix.length();
                char ch = ' ';
                String operator = "+-*^/";
                Stack <Character> s1 = new Stack <Character>(size);
                s1.push('(');
                String postfix = "";

                for (int i = 0; !s1.isEmpty(); i++) {
                        ch = infix.charAt(i);

                        if (!operator.contains(String.valueOf(ch)) && !(ch ==
'(' || ch == ')'))
                                postfix += ch;

                        if (ch == '(')// check for left parenthesis
                                s1.push(ch); // push to stack

                        if (operator.contains(String.valueOf(ch)))// check for
operator
                        {
```

```
				postfix += ' ';
				char opthis = ch;
				char optop = s1.peak();

				while (!(optop == '(') && prio(optop) >=
prio(opthis)) { // pop high and same precedence operator and

									// add to output

					postfix += s1.pop();
					postfix += ' ';
					optop = s1.peak();
				} // end while

				s1.push(ch);
			} // end if
			if (ch == ')') {
				while (!(s1.peak() == '(')) {
					postfix += ' ';
					postfix += s1.pop();
				}
				s1.pop();
			} // end if
		} // end for
		return postfix;
	}

	static public String evaluate(String postfix) {
		String operator = "+-*^/";
		String eq[] = postfix.split(" ");
		int size = eq.length;
		Stack <String> s1 = new Stack <String>(size);
		String ch = "";
		for (int i = 0; i < size; i++) {
			ch = eq[i];
			// check for operand
			if (!operator.contains(ch)) {
				if (ch.charAt(0) >= 'a' && ch.charAt(0) <= 'z') {
					System.out.println("assaign value in
varible " + ch);

					ch = sc.next();
				}
				s1.push(ch);
			}
			if (operator.contains(ch))// check for operator
			{
				String ans = "";
				String x1 = s1.pop();
				String x2 = s1.pop();
				ans = compute(x2, ch, x1);
				s1.push(ans);

			} // end if
		} // end for
		return s1.pop();
	}// end infix()

	static public String compute(String x2, String ch, String x1) {
		double ans = 0.0;
```

```
                String sAns = "";
                if (ch.equals("+"))
                        ans = Double.valueOf(x2) + Double.valueOf(x1);
                else if (ch.equals("-"))
                        ans = Double.valueOf(x2) - Double.valueOf(x1);
                else if (ch.equals("*"))
                        ans = Double.valueOf(x2) * Double.valueOf(x1);
                else if (ch.equals("/"))
                        ans = Double.valueOf(x2) / Double.valueOf(x1);
                else if (ch.equals("^"))
                        ans = Math.pow(Double.valueOf(x2), Double.valueOf(x1));
                else
                        System.out.println("operator is not found");
                sAns = String.valueOf(ans);
                return sAns;
        }// end compute()

        static public int prio(char op)// return the priority of operators
        {
                if (op == '+' || op == '-')
                        return 1;
                else if (op == '*' || op == '/')
                        return 2;
                else if (op == '^')
                        return 3;
                else
                        return 0;
        }
} // end prio()

class Stack<T> {
        private int top;
        private int cap;
        private T[] ch;

        public Stack(int size) {
                top = -1;
                cap = size;
                ch = (T[]) new Object[size];
        } // end constructor

        public void push(T c) {
                if (top >= cap)
                        System.out.println("stack is overflow");
                else {
                        top++;
                        ch[top] = c;
                } // end else
        }// end insert

        public T pop() {
                if (isEmpty()) {
                        System.out.println("stack is underflow");
                        return null;
                }
                T temp = ch[top];
                top--;
                return temp;
        }
```

```
	public boolean isEmpty() {
		if (top == -1)
			return true;
		else
			return false;
	}

	public T peak() {
		if (isEmpty()) {
			System.out.println("stack is underflow");
			return null;
		}
		return ch[top];
	}
}// end Stack Output:
```

Output:



```
enter infix:3+4-5*7/2*(6-1)
postfix:3 4 + 5 7 * 2 / 6 1 - * -
evalutation of postfix:-80.5
```

Observations*:*

At the end of this lab, I was able to implement folish notation and it's algorithms.

7. Rubrics

Demonstration	Absent	Student is unable to follow the provided instructions properly. The student can name the hardware or simulation platform, but unable to implement anything practically or on the software	Student can understand the provided laboratory instructions and familiar with the lab environment (Trainer/ software/ IDE), but cannot implement on the platform practically or on the software	Student has followed instructions to construct the fundamental schematic/ block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code, and have successfully executed the program/ run circuit on software platform	Student perfectly implemented a working model/ logic/ circuit/ block diagram/ code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory Reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recorded along with detailed procedure	Appropriate computations or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all respects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

LAB NO. 6

Queue using Array

1. Objectives:

After completing this lab, students will be able to;
- Introduction to Queue.
- Implementation of Queue in Arrays.
- Procedure and Algorithm

2. Outline:

- Array
- Queue
- Practical Implementation
- Example

3. Corresponding CLO and PLO:

- CLO 2, PLO 3

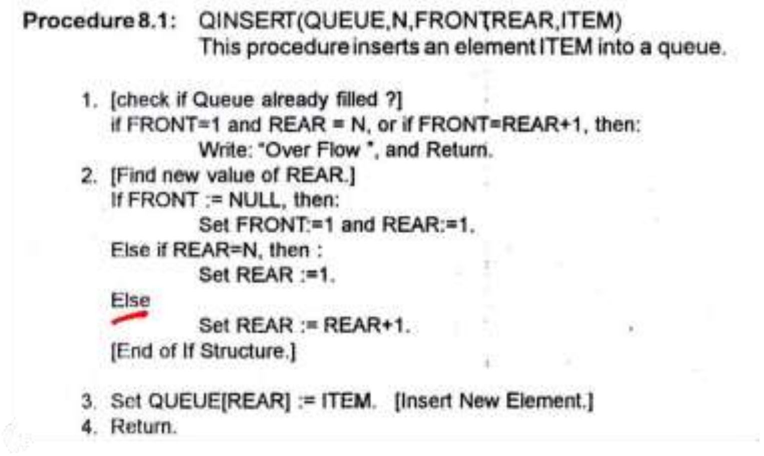
4. Background:

This article covers queue implementation in Java. A queue is a linear data structure that follows the FIFO (First–In, First–Out) principle. That means the object inserted first will be the first one out, followed by the object inserted next.

The queue supports the following core operations:

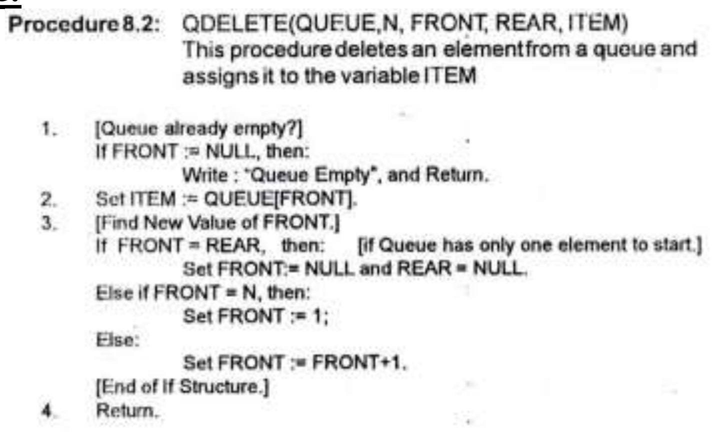
1. **Enqueue**: Inserts an item at the rear of the queue.
2. **Dequeue**: Removes the object from the front of the queue, thereby decrementing queue size by one.
3. **Peek**: Returns the object at the front of the queue without removing it.
4. **IsEmpty**: Tests if the queue is empty or not.
5. **Size**: Returns the total number of elements present in the queue.

<u>Insert:</u>



```
Procedure 8.1:  QINSERT(QUEUE,N,FRONT,REAR,ITEM)
                This procedure inserts an element ITEM into a queue.

  1. [check if Queue already filled ?]
     if FRONT=1 and REAR = N, or if FRONT=REAR+1, then:
                Write: "Over Flow ", and Return.
  2. [Find new value of REAR.]
     If FRONT := NULL, then:
                Set FRONT:=1 and REAR:=1.
     Else if REAR=N, then :
                Set REAR :=1.
     Else
                Set REAR := REAR+1.
     [End of If Structure.]

  3. Set QUEUE[REAR] := ITEM.   [Insert New Element.]
  4. Return.
```

Delete:



```
Procedure 8.2:  QDELETE(QUEUE,N, FRONT, REAR, ITEM)
                This procedure deletes an element from a queue and
                assigns it to the variable ITEM

1.  [Queue already empty?]
    If FRONT := NULL, then:
          Write : "Queue Empty", and Return.
2.  Set ITEM := QUEUE[FRONT].
3.  [Find New Value of FRONT.]
    If  FRONT = REAR,  then:     [If Queue has only one element to start.]
          Set FRONT:= NULL and REAR = NULL.
    Else if FRONT = N, then:
          Set FRONT := 1;
    Else:
          Set FRONT := FRONT+1.
    [End of If Structure.]
4.  Return.
```

5. Equipment:

Eclipse.

6. Procedure:

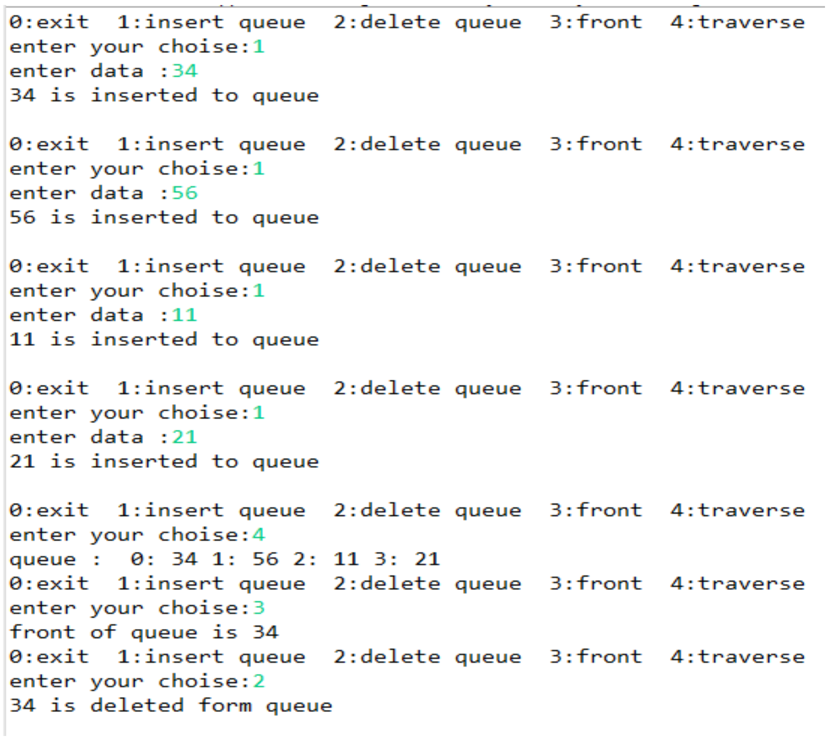
Code:

```java
import java.util.Scanner;

public class Lab6 {

	public static void main(String args[]) {

		QueueArray q = new QueueArray(10);
		Scanner sc = new Scanner(System.in);
		boolean run = true;
		while (run) {
			System.out.print("0:exit  1:insert queue  2:delete 
queue  3:front  4:traverse" + "\nenter your choise:");
			int ch = 0;
			ch = sc.nextInt();

			switch (ch) {
			case 0:
				run = false;
				System.out.println(".....exit.....");
				break;
			case 1:
				System.out.print("enter data :");
				System.out.println(q.enqueue(sc.nextInt()));
				break;
			case 2:
				System.out.println(q.dequeue());
				break;
			case 3:
				System.out.print(q.front());
				break;
			case 4:
				System.out.print(q.traverse());
				break;

			default:
```

```
                        System.out.println("invalid option");
                    }// end switch
                } // end while loop
        } // end void main()

} // end Lab class

class QueueArray {
        private int size;
        private int front, rear;
        private int[] arr;

        public QueueArray(int size) {
                this.size = size;
                rear = front = -1;
                arr = new int[size];
        }

        public String enqueue(int item) {
                if (isFull())
                        return "queue is full \n";
                else {

                        if (isEmpty())
                                rear = front = 0;
                        else if (rear == size - 1)
                                rear = 0;
                        else
                                rear++;
                        arr[rear] = item;
                        return item + " is inserted to queue \n";
                } // else
        }// end enqueue()

        public String dequeue() {
                if (isEmpty())
                        return "queue is empty \n";

                int item = arr[front];
                if (rear == front)
                        rear = front = -1;
                else if (front == size - 1)
                        front = 0;
                else
                        front++;
                return item + " is deleted form queue \n";
        }

        public boolean isEmpty() {
                return (front == -1);
        }

        public boolean isFull() {
                return (front == 0 && rear == size - 1 || front == rear + 1);
        }

        public String front() {
                if (isEmpty())
                        return "queue is empty \n ";
```

```
		return "front of queue is " + arr[front] + "\n";
	}

	public String traverse() {
		if (isEmpty())
			return "queue is empty";
		String output = "queue : ";

		int j = front;
		for (int i = 0; i < nElements(); i++) {
			if (j == size)
				j = 0;
			output += " " + j + ": " + arr[j];
			j++;

		} // end for
		return output + "\n";
	}// end traverse()

	public int nElements() {
		return (size - front + rear + 1) % size;
	}// end nElements()
}// end Queue class
```

Output:



```
0:exit   1:insert queue   2:delete queue   3:front   4:traverse
enter your choise:1
enter data :34
34 is inserted to queue

0:exit   1:insert queue   2:delete queue   3:front   4:traverse
enter your choise:1
enter data :56
56 is inserted to queue

0:exit   1:insert queue   2:delete queue   3:front   4:traverse
enter your choise:1
enter data :11
11 is inserted to queue

0:exit   1:insert queue   2:delete queue   3:front   4:traverse
enter your choise:1
enter data :21
21 is inserted to queue

0:exit   1:insert queue   2:delete queue   3:front   4:traverse
enter your choise:4
queue :  0: 34 1: 56 2: 11 3: 21
0:exit   1:insert queue   2:delete queue   3:front   4:traverse
enter your choise:3
front of queue is 34
0:exit   1:insert queue   2:delete queue   3:front   4:traverse
enter your choise:2
34 is deleted form queue
```

7. Observations:

At the end of this lab, I was able to reestablish fundamentals of queue with arrays and it’s most basic algorithms.

Rubrics

Demonstration	Absent	Student is unable to follow the provided instructions properly. The student can name the hardware or simulation platform, but unable to implement anything practically or on the software	Student can understand the provided laboratory instructions and familiar with the lab environment (Trainer/ software/ IDE), but cannot implement on the platform practically or on the software	Student has followed instructions to construct the fundamental schematic/ block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code, and have successfully executed the program/ run circuit on software platform	Student perfectly implemented a working model/ logic/ circuit/ block diagram/ code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory Reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recorded along with detailed procedure	Appropriate computations or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all respects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

LAB NO. 7

Simple Linked List

1. Objective:

After completing this lab, students will be able to;

- Introduction to Linked List
- Establishment of Linked List Concept
- Procedure and Implementation

2. Outline:

- Linked List
- Simple Linked List
- Practical Implementation
- Example

3. Corresponding CLO and PLO:

- CLO 1, PLO 5

4. Background:

Linked lists are probably the second most commonly used general-purpose storage structures after arrays.

The linked list is a versatile mechanism suitable for use in many kinds of general-purpose databases.

It can also replace an array as the basis for other storage structures such as stacks and queues. In fact, you can use a linked list in many cases where you use an array (unless you need frequent random access to individual items using an index).

In a linked list, each data item is embedded in a *link.*

A link is an object of a class called something like Link. Because there are many similar links in a list, it makes sense to use a separate class for them, distinct from the linked list itself. Each link object contains a reference (usually called next) to the next link in the list.

A field in the list itself contains a reference to the first link.

Here's part of the definition of a class Link. It contains some data and a reference to the next link.

```
class Link
{
        public int iData;               // data
        public double dData;  // data
        public Link next;               // reference to next link
}
```

This kind of class definition is sometimes called ***self-referential*** because it contains a field—called next in this case—of the same type as itself.

Below are some of the algorithms of linked list.

Traversing a linked List:

```
Algorithm 9.1: (Traversing a Linked List) TRAVERSE(LIST,START)
               Let LIST be a linked list in memory. This algorithm
               traverses LIST, applying an operation PRECESS to
               each element of LIST. The variable CURRENT points
               to the node currently being processed.

1. Set CURRENT:=START. [Initializes pointer CURRENT.]
2. Repeat Step 3 and 4 while CURRENT!=NULL.
3.          Apply PROCESS to DATA[CURRENT].
4.          Set CURRENT:= NEXT[CURRENT].
            [CURRENT now points to the next node.]
   [End of step 2 loop.]
5. Exit
```

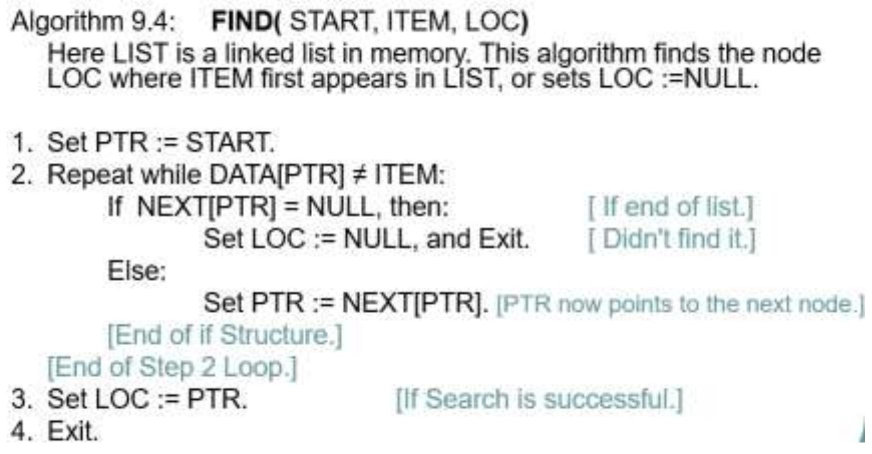
Inserting at beginning of a linked list:

```
Algorithm 9.2: INSERTFIRST( START, ITEM)
This algorithm inserts ITEM as the first node in the LIST.

    1.  Set DATA[NEW] = ITEM;
        [Copies new data into new node.]
    2.  Set NEXT[NEW] = START.
        [New node now points to the original first node.]
    3.  Set START := NEW.
        [Changes START so it points to the new node.]
    4.  Exit
```

Deleting at beginning of linked list:

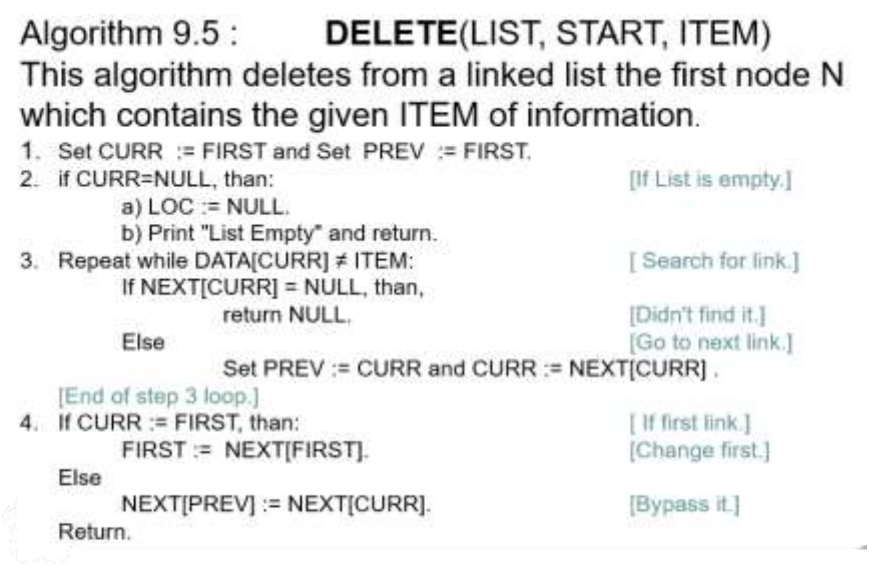
```
Algorithm 9.3: DELETEFIRST( START, ITEM)
    This algorithm deletes a node ITEM from the
    beginning of the LIST.
  1.  If START = NULL, than:  [If List is already Empty.]
          a)  Set ITEM := NULL.     [Nothing to delete.]
          b)  Print "LIST EMPTY" and Return.
  2.  Set ITEM := DATA[FIRST]. [Save data to ITEM.]
  3.  Set FIRST:= NEXT[FIRST].  [Delete it.]
  4.  Return.
```

Finding specified Links:



```
Algorithm 9.4:   FIND( START, ITEM, LOC)
    Here LIST is a linked list in memory. This algorithm finds the node
    LOC where ITEM first appears in LIST, or sets LOC :=NULL.

1. Set PTR := START.
2. Repeat while DATA[PTR] ≠ ITEM:
        If NEXT[PTR] = NULL, then:           [ If end of list.]
            Set LOC := NULL, and Exit.       [ Didn't find it.]
        Else:
            Set PTR := NEXT[PTR]. [PTR now points to the next node.]
        [End of If Structure.]
    [End of Step 2 Loop.]
3. Set LOC := PTR.            [If Search is successful.]
4. Exit.
```

Deleting a specified link:



```
Algorithm 9.5 :    DELETE(LIST, START, ITEM)
This algorithm deletes from a linked list the first node N
which contains the given ITEM of information.
1. Set CURR := FIRST and Set PREV := FIRST.
2. if CURR=NULL, than:                          [If List is empty.]
        a) LOC := NULL.
        b) Print "List Empty" and return.
3. Repeat while DATA[CURR] ≠ ITEM:              [ Search for link.]
        If NEXT[CURR] = NULL, than,
              return NULL.                      [Didn't find it.]
        Else                                    [Go to next link.]
              Set PREV := CURR and CURR := NEXT[CURR] .
   [End of step 3 loop.]
4. If CURR := FIRST, than:                      [ If first link.]
        FIRST := NEXT[FIRST].                   [Change first.]
   Else
        NEXT[PREV] := NEXT[CURR].               [Bypass it.]
   Return.
```

5. Equipment:

Eclipse

6. Procedure:

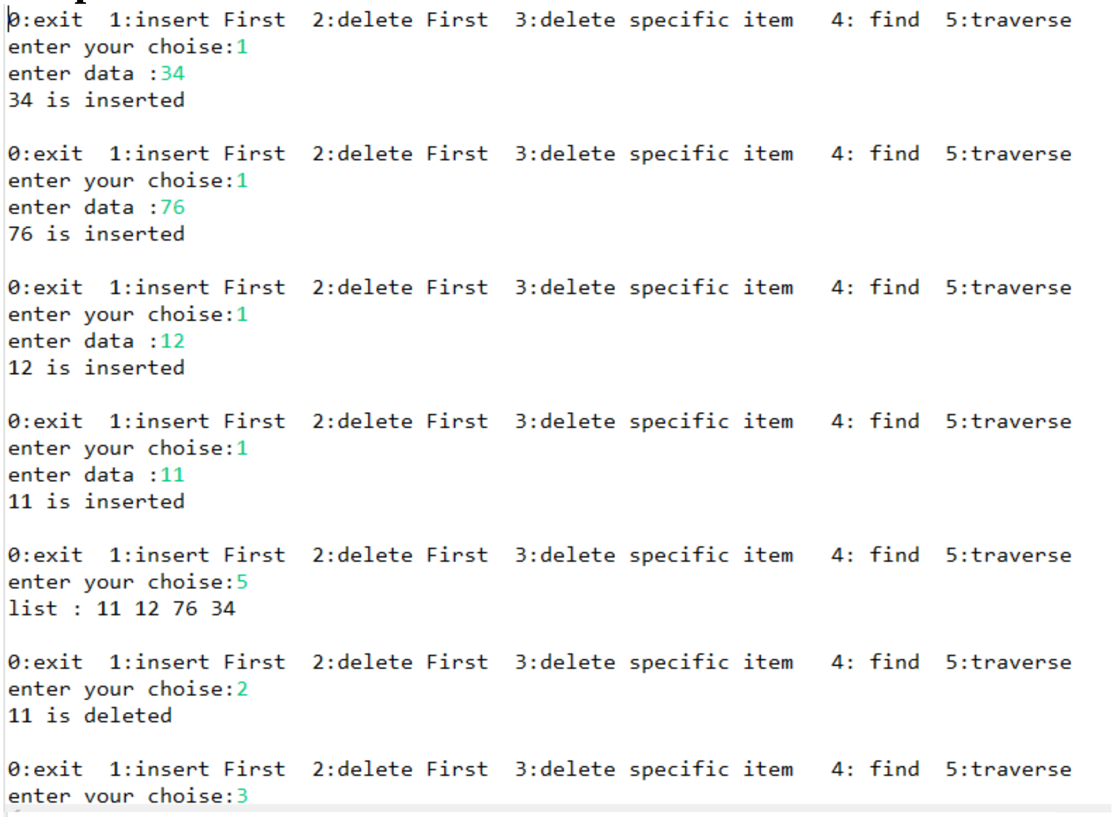
Code:

```
import java.util.Scanner;

public class Lab7 {

	public static void main(String args[]) {

		LinkedList l = new LinkedList();
		Scanner sc = new Scanner(System.in);
		boolean run = true;
		while (run) {
			System.out.print("0:exit  1:insert First  2:delete 
First  3:delete specific item  4: find  5:traverse"
					+ "\nenter your choise:");
```

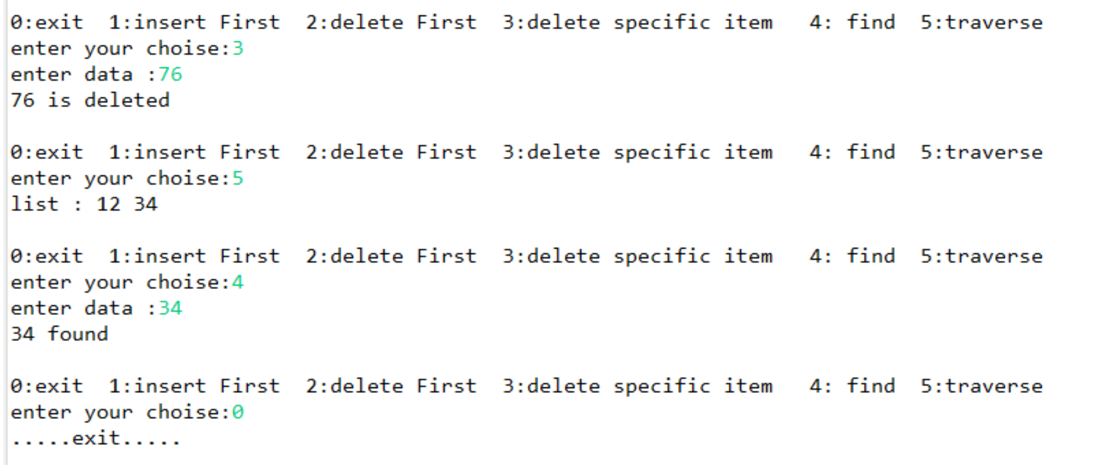
```
                    int ch = 0;
                    ch = sc.nextInt();

                    switch (ch) {
                    case 0:

                            run = false;
                            System.out.println(".....exit.....");
                            break;
                    case 1:
                            System.out.print("enter data :");
                            System.out.println(l.insertFirst(sc.nextInt()));
                            break;
                    case 2:

                            System.out.println(l.deleteFirst());

                            break;
                    case 3:
                            System.out.print("enter data :");
                            System.out.println(l.delete(sc.nextInt()));

                            break;
                    case 4:

                            System.out.print("enter data :");
                            System.out.println(l.find(sc.nextInt()));

                            break;
                    case 5:

                            System.out.println(l.traverse());

                            break;
                    default:
                            System.out.println("invalid option");
                    }// end switch
            } // end while loop
        } // end void main()

} // end Lab class

class node {
        public int data;
        public node next;
}// end node

class LinkedList {
        private node start;

        public LinkedList() {

                start = null;
        }// end List()

        public String insertFirst(int item) {

                node temp = new node();
                temp.data = item;
        temp.next = start;
        start = temp;
        return item + " is inserted \n";
    }// end insert()

    public String delete(int element) {
        if (start == null)
            return "list is empty\n";

        node curr = start;
        node prev = start;

        while (curr.data != element) {
            if (curr.next == null)
                return "didn't find \n";
            else {
                prev = curr;
                curr = curr.next;
            }
        }
        int data = prev.next.data;
        if (curr == start)
            start = start.next;
        else
            prev.next = curr.next;
        curr = null;
        return data + " is deleted \n";
    }// end delete ()

    public String deleteFirst() {
        if (start == null)
            return "list is empty\n";
        int data = start.data;
        start = start.next;
        return data + " is deleted \n";
    }// end delete ()

    public String traverse() {
        if (isEmpty())
            return "list is empty \n";

        String output = "list : ";
        node temp = start;
        while (temp != null) {
            output += temp.data + " ";
            temp = temp.next;
        } // end while
        return output + "\n";
    }// end display()

    public String find(int item) {
        node temp = start;
        while (temp.data != item) {
            if (temp.next == null)
                return item + " didn't find \n";

            else
                temp = temp.next;
        } // end while()
```

```
            return temp.data + " found \n";
        }// end find()

        public boolean isEmpty() {
            return (start == null);
        }// end isEmpty()

}// end List
```

Output:



```
0:exit  1:insert First  2:delete First  3:delete specific item   4: find  5:traverse
enter your choise:1
enter data :34
34 is inserted

0:exit  1:insert First  2:delete First  3:delete specific item   4: find  5:traverse
enter your choise:1
enter data :76
76 is inserted

0:exit  1:insert First  2:delete First  3:delete specific item   4: find  5:traverse
enter your choise:1
enter data :12
12 is inserted

0:exit  1:insert First  2:delete First  3:delete specific item   4: find  5:traverse
enter your choise:1
enter data :11
11 is inserted

0:exit  1:insert First  2:delete First  3:delete specific item   4: find  5:traverse
enter your choise:5
list : 11 12 76 34

0:exit  1:insert First  2:delete First  3:delete specific item   4: find  5:traverse
enter your choise:2
11 is deleted

0:exit  1:insert First  2:delete First  3:delete specific item   4: find  5:traverse
enter your choise:3
```



```
0:exit  1:insert First  2:delete First  3:delete specific item   4: find  5:traverse
enter your choise:3
enter data :76
76 is deleted

0:exit  1:insert First  2:delete First  3:delete specific item   4: find  5:traverse
enter your choise:5
list : 12 34

0:exit  1:insert First  2:delete First  3:delete specific item   4: find  5:traverse
enter your choise:4
enter data :34
34 found

0:exit  1:insert First  2:delete First  3:delete specific item   4: find  5:traverse
enter your choise:0
.....exit.....
```

Observations:

At the end of this lab, I was able to reestablish fundamentals of simple linked list and it’s most basic algorithms.

Rubrics

Demonstration	Absent	Student is unable to follow the provided instructions properly. The student can name the hardware or simulation platform, but unable to implement anything practically or on the software	Student can understand the provided laboratory instructions and familiar with the lab environment (Trainer/ software/ IDE), but cannot implement on the platform practically or on the software	Student has followed instructions to construct the fundamental schematic/ block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code, and have successfully executed the program/ run circuit on software platform	Student perfectly implemented a working model/ logic/ circuit/ block diagram/ code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory Reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recorded along with detailed procedure	Appropriate computations or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all respects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

LAB NO. 8

Double Ended Linked List

1. Objectives:

After completing this lab, students will be able to;

- Deep understanding of Linked List
- Introduction to Double Ended Linked List
- Procedure and Algorithm

2. Outline:

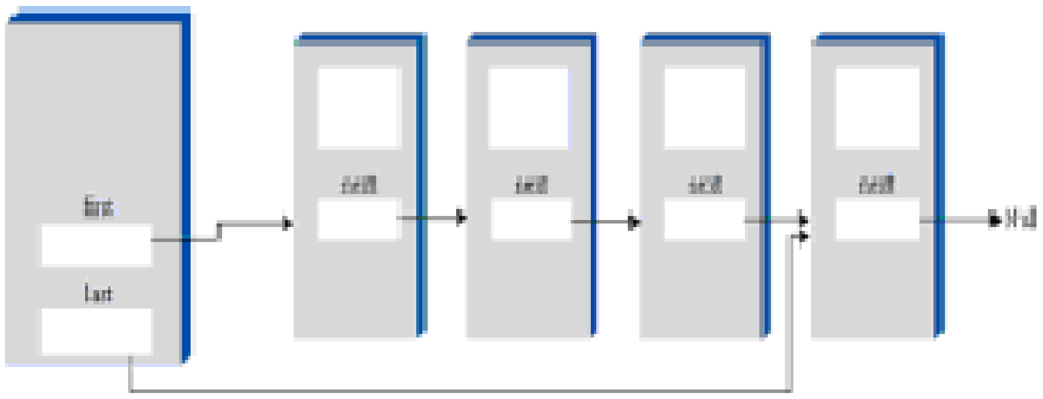
- Linked List
- Double Ended Linked List
- Practical Implementation
- Examples.

3. Corresponding CLO and PLO:

- CLO 1, 2, PLO 3

4. Background:

A double-ended list is similar to an ordinary linked list, but it has one additional feature: a reference to the last link as well as to the first.



The reference to the last link permits us to insert a new link directly at the end of the list as well as at the beginning.

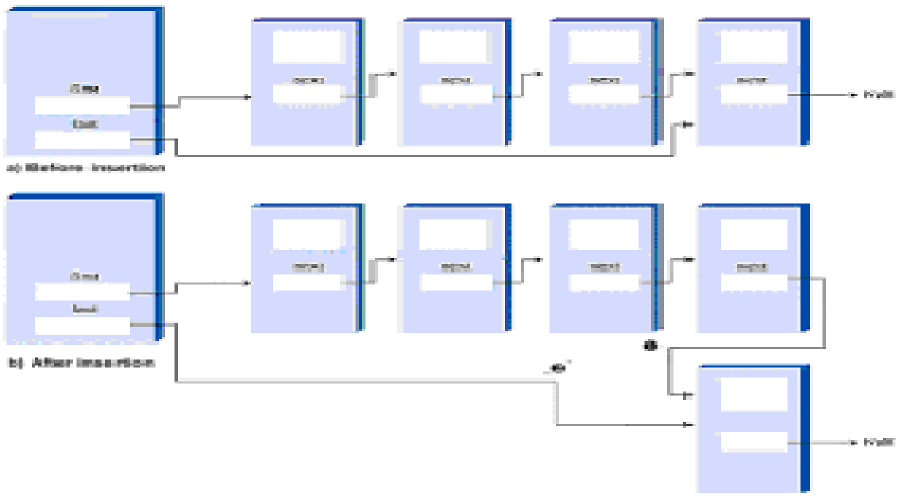
We can insert a new link at the end of an ordinary single-ended list by iterating through the entire list until we reach the end, but this is inefficient. Access to the end of the list as well as the beginning makes the double-ended list suitable for certain situations that a single-ended list can't handle efficiently. One such situation is implementing a queue.

Inserting at beginning of double ended linked list:

Algorithm 9.6: **INSERTFIRST(** START, LAST, ITEM**)**

This algorithm inserts ITEM as the first node in the LIST.

```
1.  Set  DATA[NEW] = ITEM;
    [Copies new data into new node.]
2.  IF START=NULL, than:            [If empty list.]
            Set LAST := NEW.        [NewLink <-- last.]
3.  Set NEXT[NEW] = START.
    [New node now points to the original first node.]
4.  Set START := NEW.
    [Changes START so it points to the new node.]
5.  Return.
```



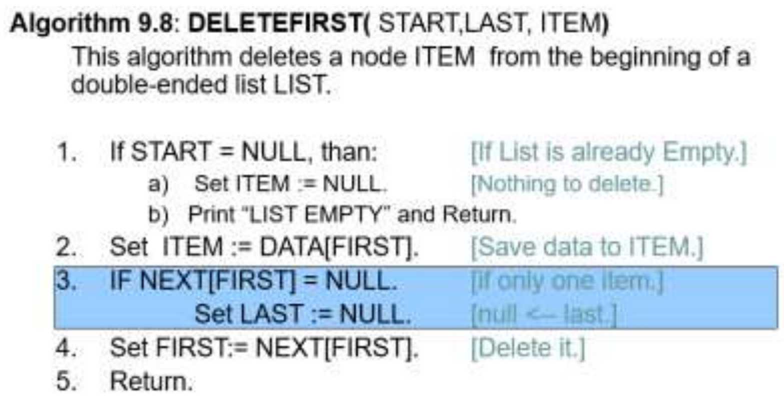
Inserting at end of double ended linked list:

Algorithm 9.7: **INSERTLAST(** START, LAST, ITEM**)**

This algorithm inserts ITEM as the LAST node in the LIST.

```
1.  Set  DATA[NEW] = ITEM;
    [Copies new data into new node.]
2.  IF START=NULL, than:         [If empty list.]
          Set FIRST := NEW.      [first --> newLink.]
    Else
          Set NEXT[LAST] := NEW.
3.  Set LAST := NEW.
    [Changes LAST so it points to the new node.]
4.  Set NEXT[NEW] := NULL.
5.  Return.
```

Deleting at beginning of double ended linked list:



```
Algorithm 9.8: DELETEFIRST( START,LAST, ITEM)
    This algorithm deletes a node ITEM from the beginning of a
    double-ended list LIST.

  1.  If START = NULL, than:            [If List is already Empty.]
          a)  Set ITEM := NULL.         [Nothing to delete.]
          b)  Print "LIST EMPTY" and Return.
  2.  Set ITEM := DATA[FIRST].          [Save data to ITEM.]
  3.  IF NEXT[FIRST] = NULL.            [If only one item.]
            Set LAST := NULL.           [null <-- last.]
  4.  Set FIRST:= NEXT[FIRST].          [Delete it.]
  5.  Return.
```

5. Equipment:

Eclipse

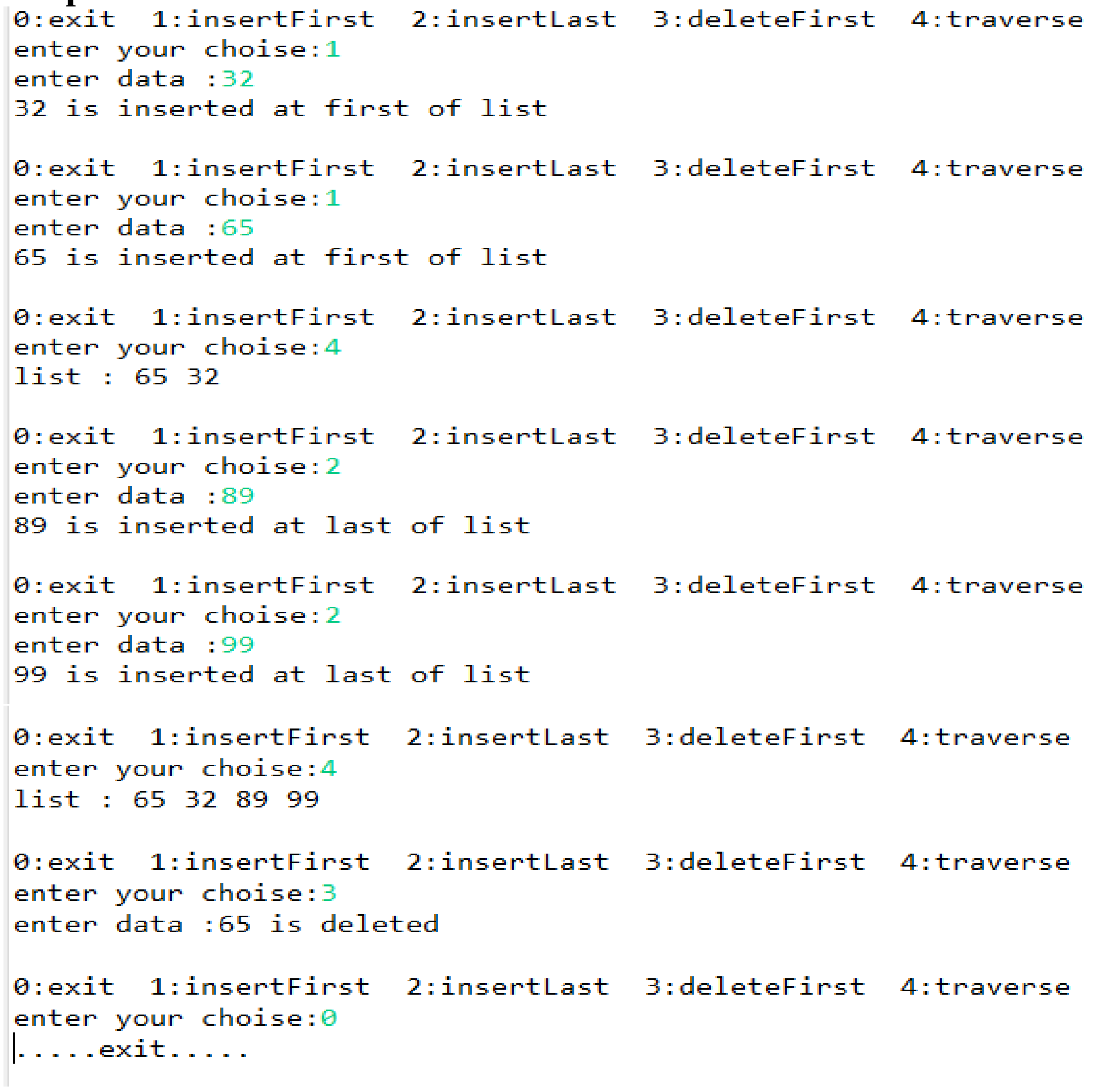
6. Procedure:

Code:

```
import java.util.Scanner;
public class Lab8 {

	public static void main(String args[]) {
		DoubleEndedLinkedList l = new DoubleEndedLinkedList();
		Scanner sc = new Scanner(System.in);
		boolean run = true;
		while (run) {
			System.out.print("0:exit  1:insertFirst  2:insertLast
3:deleteFirst  4:traverse" + "\nenter your choise:");
			int ch = 0;
			ch = sc.nextInt();

			switch (ch) {
			case 0:

				run = false;
				System.out.println(".....exit.....");
				break;

			case 1:
				System.out.print("enter data :");
				System.out.println(l.insertFirst(sc.nextInt()));
				break;

			case 2:
				System.out.print("enter data :");
				System.out.println(l.insertLast(sc.nextInt()));
				break;

			case 3:
				System.out.print("enter data :");
				System.out.println(l.deleteFirst());
				break;
```

```
                    case 4:
                            System.out.println(l.traverse());
                            break;

                    default:
                            System.out.println("invalid option");
                    }// end switch
            } // end while loop
    } // end void main()

} // end Lab class

class node {
        public int data;
        public node next;
}// end node


class DoubleEndedLinkedList {
        private node start;
        private node end;

        public DoubleEndedLinkedList() {
                start = null;
                end = null;
        }// end List()

        public String insertFirst(int item) {

                node temp = new node();
                temp.data = item;
                if (start == null)
                        end = temp;
                else
                        temp.next = start;
                start = temp;
                return item + " is inserted at first of list \n";
        }// end insert()

        public String insertLast(int item) {
                node temp = new node();

                temp.data = item;
                if (start == null)
                        start = temp;
                else
                        end.next = temp;
                end = temp;
                return item + " is inserted at last of list \n";
        }// end insertLast(int)

        public String deleteFirst() {
                if (start == null)
                        return "list is empty \n";
                int data = start.data;
                if (start.next == null)
                        end = null;
                start = start.next;
                return data + " is deleted\n";
        }// end delete ()

        public String traverse() {
                if (start == null)
                        return "list is empty \n";
                node temp = start;
                String output = "list : ";
                while (temp != null) {
                        output += temp.data + " ";
                        temp = temp.next;
                } // end while
                return output + "\n";
        }// end display()
}// end DoubleEndedLinkedList
```

Output:



```
0:exit   1:insertFirst   2:insertLast   3:deleteFirst   4:traverse
enter your choise:1
enter data :32
32 is inserted at first of list

0:exit   1:insertFirst   2:insertLast   3:deleteFirst   4:traverse
enter your choise:1
enter data :65
65 is inserted at first of list

0:exit   1:insertFirst   2:insertLast   3:deleteFirst   4:traverse
enter your choise:4
list : 65 32

0:exit   1:insertFirst   2:insertLast   3:deleteFirst   4:traverse
enter your choise:2
enter data :89
89 is inserted at last of list

0:exit   1:insertFirst   2:insertLast   3:deleteFirst   4:traverse
enter your choise:2
enter data :99
99 is inserted at last of list

0:exit   1:insertFirst   2:insertLast   3:deleteFirst   4:traverse
enter your choise:4
list : 65 32 89 99

0:exit   1:insertFirst   2:insertLast   3:deleteFirst   4:traverse
enter your choise:3
enter data :65 is deleted

0:exit   1:insertFirst   2:insertLast   3:deleteFirst   4:traverse
enter your choise:0
.....exit.....
```

Observations:

At the end of this lab, I was able to reestablish fundamentals of double ended linkedlist and it’s most basic algorithms.

7. Rubrics

Demonstration	Absent	Student is unable to follow the provided instructions properly. The student can name the hardware or simulation platform, but unable to implement anything practically or on the software	Student can understand the provided laboratory instructions and familiar with the lab environment (Trainer/ software/ IDE), but cannot implement on the platform practically or on the software	Student has followed instructions to construct the fundamental schematic/ block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code, and have successfully executed the program/ run circuit on software platform	Student perfectly implemented a working model/ logic/ circuit/ block diagram/ code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor’s Signature**		

Laboratory Reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recorded along with detailed procedure	Appropriate computations or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all respects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor’s Signature**		

LAB NO. 9

Sorted Linked List

1. Objectives:

After completing this lab, students will be able to;

2. Deep understanding to Linked List
3. Introduction to Sorted Linked List
4. Procedure and Algorithm

2. Outline:

- Linked List
- Sorted Linked List
- Practical Implementation
- Examples

3. Corresponding CLO and PLO:

- CLO 1, 2, PLO 3

4. Background:

In linked lists we've seen thus far, there was no requirement that data be stored in order.

However, for certain applications it's useful to maintain the data in sorted order within the list. A list with this characteristic is called a *sorted list*.

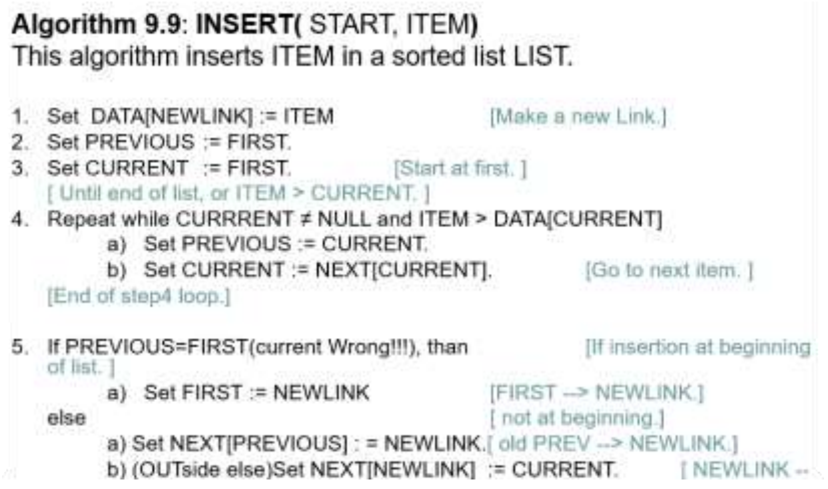
In a sorted list, the items are arranged in sorted order by key value.

Deletion is often limited to the smallest (or the largest) item in the list, which is at the start of the list, although sometimes find() and delete() methods, which search through the list for specified links, are used as well.

In general you can use a sorted list in most situations where you use a sorted array.

The advantages of a sorted list over a sorted array are speed of insertion (because elements don't need to be moved) and the fact that a list can expand to fill available memory, while an array is limited to a fixed size. However, a sorted list is somewhat more difficult to implement than a sorted array.

<u>Inserting in sorted linked list:</u>



Algorithm 9.9: **INSERT(** START, ITEM**)**
This algorithm inserts ITEM in a sorted list LIST.

```
1.  Set  DATA[NEWLINK] := ITEM                 [Make a new Link.]
2.  Set PREVIOUS := FIRST.
3.  Set CURRENT  := FIRST.             [Start at first. ]
    [ Until end of list, or ITEM > CURRENT. ]
4.  Repeat while CURRRENT ≠ NULL and ITEM > DATA[CURRENT]
          a)  Set PREVIOUS := CURRENT.
          b)  Set CURRENT := NEXT[CURRENT].            [Go to next item. ]
    [End of step4 loop.]

5.  If PREVIOUS=FIRST(current Wrong!!!), than          [If insertion at beginning
    of list. ]
          a)  Set FIRST := NEWLINK             [FIRST --> NEWLINK.]
    else                                       [ not at beginning.]
          a) Set NEXT[PREVIOUS] : = NEWLINK.[ old PREV --> NEWLINK.]
          b) (OUTside else)Set NEXT[NEWLINK]  := CURRENT.       [ NEWLINK --
```

Efficiency of Sorted Liked List:

Insertion and deletion of arbitrary items in the sorted linked list require O(N) comparisons (N/2 on the average) because the appropriate location must be found by stepping through the list.

However, the minimum value can be found, or deleted, in O(1) time because it's at the beginning of the list. If an application frequently accesses the minimum item and fast insertion isn't critical, then a sorted linked list is an effective choice.

A sorted list can be used as a fairly efficient sorting mechanism.

Suppose you have an array of unsorted data items. If you take the items from the array and insert them one by one into the sorted list, they'll be placed in sorted order automatically. If you then remove them from the list and put them back in the array, they array will be sorted.

It turns out this is substantially more efficient than the more usual insertion sort within an array, described in lecture 5. This is because fewer copies are necessary. It's still an O(N^2) process, because inserting each item into the sorted list involves comparing a new item with an average of half the items already in the list, and there are N items to insert, resulting in about $N^2/4$ comparisons.

However, each item is only copied twice: once from the array to the list, and once from the list to the array. N*2 copies compare favorably with the insertion sort within an array, where there are about N^2 copies.

5. Equipment:

Eclipse

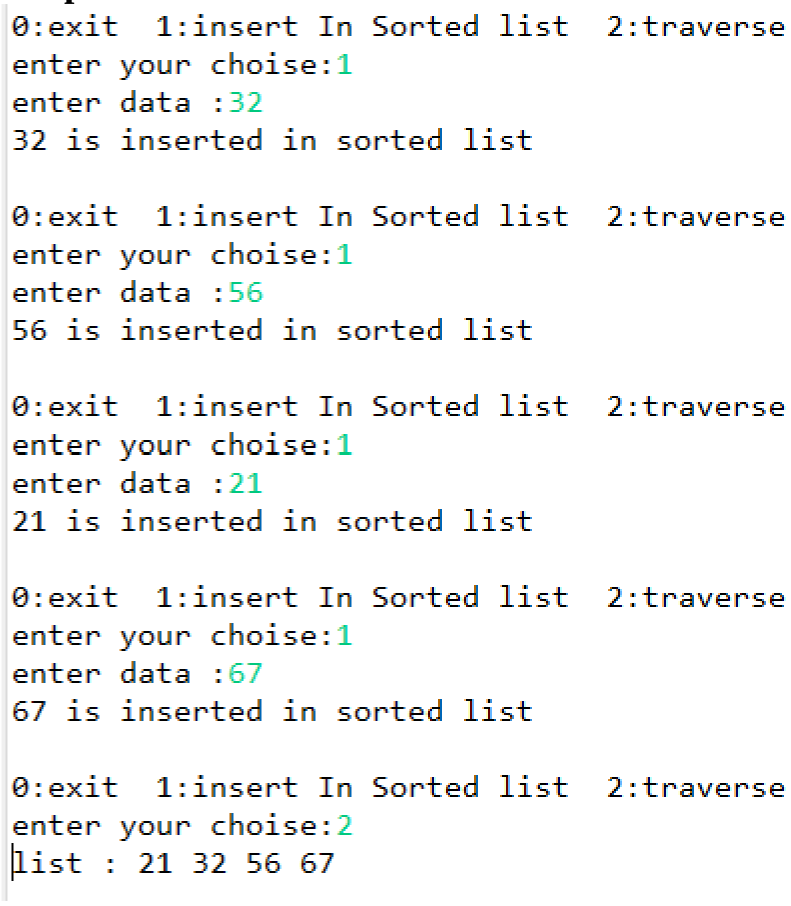
6. Procedure:

Code:

```
import java.util.Scanner;

public class Lab9 {

	public static void main(String args[]) {

		// Array a1 = new Array(10);
		Scanner sc = new Scanner(System.in);
		SortedLinkedList l = new SortedLinkedList();
		boolean run = true;
		while (run) {
			System.out.print("0:exit  1:insert In Sorted list 
2:traverse" + "\nenter your choise:");
			int ch = 0;
			ch = sc.nextInt();

			switch (ch) {
			case 0:

				run = false;
				System.out.println(".....exit.....");
				break;
```

```
                case 1:
                        System.out.print("enter data :");

        System.out.println(l.insertInSorted(sc.nextInt()));
                        break;

                case 2:
                        System.out.println(l.traverse());
                        break;
                default:
                        System.out.println("invalid option");
                }// end switch
            } // end while loop
        } // end void main()

} // end Lab class

class node {
        public int data;
        public node next;
}// end node


class SortedLinkedList {

        node start, end;

        public SortedLinkedList() {
                start = null;
                end = null;
        }

        public String insertInSorted(int item) {

                node temp = new node();
                temp.data = item;
                node curr = start;
                node previous = start;
                while (curr != null && item >= curr.data) {
                        previous = curr;
                        curr = curr.next;
                }
                if (curr == start)
                        start = temp;
                else if (previous == start)
                        start.next = temp;
                else
                        previous.next = temp;
                temp.next = curr;

                return item + " is inserted in sorted list \n";
        }// end insertInSorted()

        public String traverse() {
                if (start == null)
                        return "list is empty \n";
                node temp = start;
                String output = "list : ";
                while (temp != null) {
                        output += temp.data + " ";
                        temp = temp.next;
                } // end while
                return output + "\n";
        }// end display()

}
```

Output:



```
0:exit  1:insert In Sorted list  2:traverse
enter your choise:1
enter data :32
32 is inserted in sorted list

0:exit  1:insert In Sorted list  2:traverse
enter your choise:1
enter data :56
56 is inserted in sorted list

0:exit  1:insert In Sorted list  2:traverse
enter your choise:1
enter data :21
21 is inserted in sorted list

0:exit  1:insert In Sorted list  2:traverse
enter your choise:1
enter data :67
67 is inserted in sorted list

0:exit  1:insert In Sorted list  2:traverse
enter your choise:2
list : 21 32 56 67
```

7. Observations:

At the end of this lab, I was able to reestablish fundamentals of sorted linked list and it’s most basic algorithms.

Rubrics

Demonstration	Absent	Student is unable to follow the provided instructions properly. The student can name the hardware or simulation platform, but unable to implement anything practically or on the software	Student can understand the provided laboratory instructions and familiar with the lab environment (Trainer/ software/ IDE), but cannot implement on the platform practically or on the software	Student has followed instructions to construct the fundamental schematic/ block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code, and have successfully executed the program/ run circuit on software platform	Student perfectly implemented a working model/ logic/ circuit/ block diagram/ code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory Reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recorded along with detailed procedure	Appropriate computations or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all respects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

LAB NO. 10

Doubly Linked List

1. Objectives:

After completing this lab, students will be able to;
- Deep Understanding of Linked List
- Introduction to Doubly Linked List
- Procedure and Algorithm

2. Outline:

- Linked List
- Doubly Linked List
- Practical Implementation
- Examples

3. Corresponding CLO and PLO:

- CLO 1, 2, PLO 3

4. Background:

Let's examine another variation on the linked list: the *doubly linked* list (not to be confused with the double-ended list).

What's the advantage of a doubly linked list?

A potential problem with ordinary linked lists is that it's difficult to traverse backward along the list. A statement like

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

steps conveniently to the next link, but there's no corresponding way to go to the previous link. Depending on the application, this could pose problems.

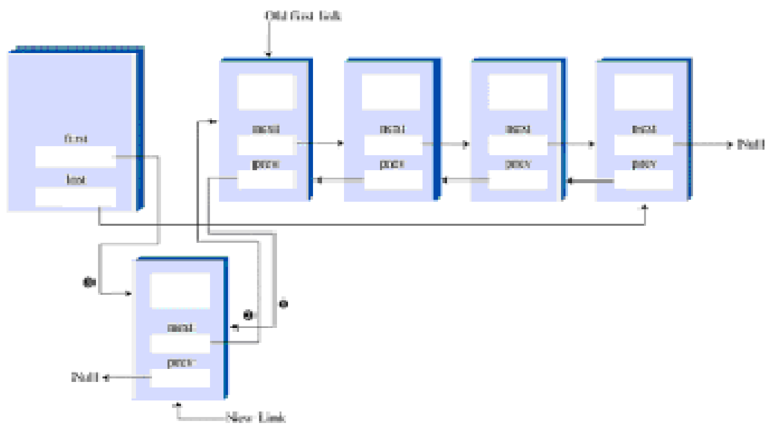
For example, imagine a text editor in which a linked list is used to store the text. Each text line on the screen is stored as a String object embedded in a link. When the editor's user moves the cursor downward on the screen, the program steps to the next link to manipulate or display the new line. But what happens if the user moves the cursor upward? In an ordinary linked list, you'd need to return current (or its equivalent) to the start of the list and then step all the way down again to the new current link. This isn't very efficient. You want to make a single step upward. The doubly linked list provides this capability. It allows you to traverse backward as well as forward through the list. The secret is that each link has two references to other links instead of one. The first is to the next link, as in ordinary lists. The second is to the previous link.

For example, imagine a text editor in which a linked list is used to store the text. Each text line on the screen is stored as a String object embedded in a link. When the editor's user moves the cursor downward on the screen, the program steps to the next link to manipulate or display the new line. But what happens if the user moves the cursor upward? In an ordinary linked list, you'd need to return current (or its equivalent) to the start of the list and then step all the way down again to the new current link. This isn't very efficient. You want to make a single step upward. The doubly linked list provides this capability. It allows you to traverse backward as well as forward through the list. The secret is that each link has two references to other

links instead of one. The first is to the next link, as in ordinary lists. The second is to the previous link.

Inserting at beginning of linked list:

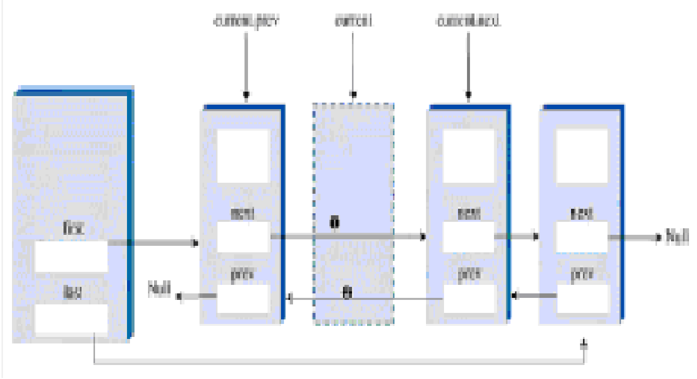
```
Algorithm 9.10: INSERTFIRST( FIRST, LAST, ITEM)
This algorithm inserts ITEM as the first node in a doubly
linked list  LIST.

1.   Set  DATA[NEWLINK] = ITEM;           [create new link.]
2.   if FIRST = NULL, than:               [If Empty List.]
        Set LAST := NEWLINK.              [newLink <-- last.]
     Else
        Set PREVIOUS[FIRST] := NEWLINK.  [NEWLINK <-- OLD FIRST.]
     [End of If Structure.]

3.   Set NEXT[NEWLINK] := FIRST.          [ NEWLINK --> OLD FIRST.]
4.   Set FIRST = NEWLINK.                 [ FIRST --> NEWLINK.]
5.   Exit.
```

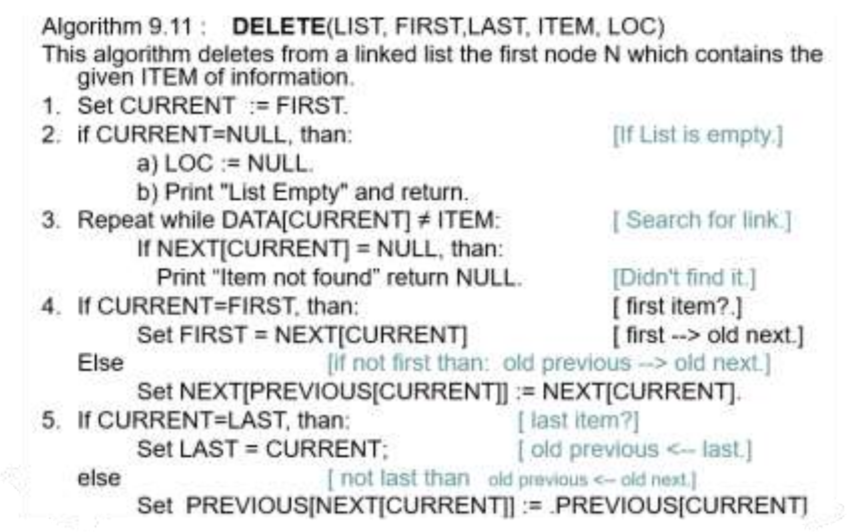


Deleting in doubly linked list:

There are three deletion routines: deleteFirst(), deleteLast(), and deleteKey(). The first two are fairly straightforward. In deleteKey(), the key being deleted is current.

Assuming the link to be deleted is neither the first nor the last one in the list, then the next field of PREVIOUS[CURRENT] (the link before the one being deleted) is set to point to NEXT[CURRENT] (the link following the one being deleted), and the PREVIOUS field of NEXT[CURRENT] is set to point to PREVIOUS[CURRENT]. This disconnects the current link from the list.





```
Algorithm 9.11 :   DELETE(LIST, FIRST,LAST, ITEM, LOC)
This algorithm deletes from a linked list the first node N which contains the
   given ITEM of information.
1.  Set CURRENT  := FIRST.
2.  if CURRENT=NULL, than:                         [If List is empty.]
        a) LOC := NULL.
        b) Print "List Empty" and return.
3.  Repeat while DATA[CURRENT] ≠ ITEM:             [ Search for link.]
        If NEXT[CURRENT] = NULL, than:
          Print "Item not found" return NULL.      [Didn't find it.]
4.  If CURRENT=FIRST, than:                        [ first item?.]
        Set FIRST = NEXT[CURRENT]                  [ first --> old next.]
    Else                 [If not first than:  old previous --> old next.]
        Set NEXT[PREVIOUS[CURRENT]] := NEXT[CURRENT].
5.  If CURRENT=LAST, than:                 [ last item?]
        Set LAST = CURRENT;                [ old previous <-- last.]
    else                 [ not last than   old previous <-- old next.]
        Set  PREVIOUS[NEXT[CURRENT]] := .PREVIOUS[CURRENT]
```

5. Equipment:

Eclipse

6. Procedure:

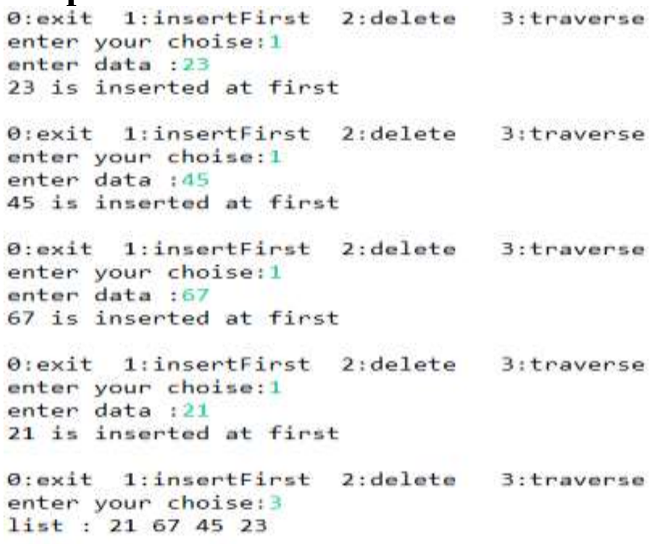
Code:

```
import java.util.Scanner;

public class Lab10 {

	public static void main(String args[]) {

		DoublyLinkedList l = new DoublyLinkedList();
		Scanner sc = new Scanner(System.in);
		boolean run = true;
		while (run) {
			System.out.print("0:exit  1:insertFirst  2:delete
3:traverse" + "\nenter your choise:");
			int ch = 0;
			ch = sc.nextInt();

			switch (ch) {
			case 0:

				run = false;
				System.out.println(".....exit.....");
				break;
			case 1:
				System.out.print("enter data :");
				System.out.println(l.insertFirst(sc.nextInt()));

				break;
			case 2:
				System.out.print("enter data :");
				System.out.println(l.delete(sc.nextInt()));

				break;

			case 3:
				System.out.println(l.traverse());
```

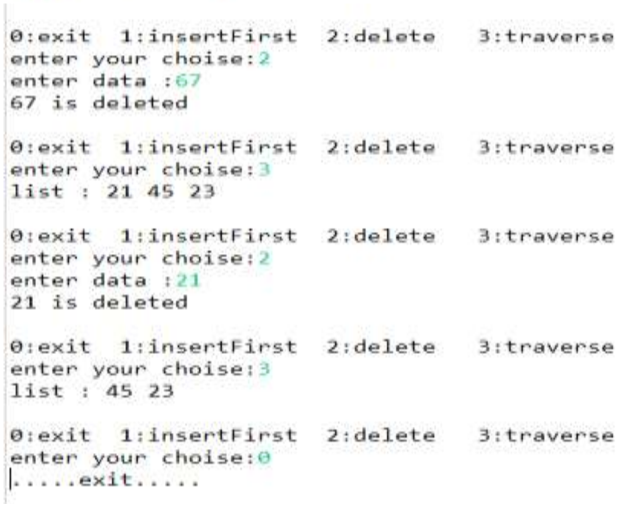
```
                        break;
                default:
                        System.out.println("invalid option");
                }// end switch
        } // end while loop
    } // end void main()

} // end Lab class


class node {
        public int data;
        public node next;
        public node prev;
        public node() {}
        public node(int item) {
                data = item;
        }
}// end node

class DoublyLinkedList {

        node start, end;

        public DoublyLinkedList() {
                start = null;
                end = null;
        }

        public String insertFirst(int item) {
                node temp = new node(item);
                if (start == null)
                        end = temp;
                else
                        start.prev = temp;

                temp.next = start;
                start = temp;
                return item + " is inserted at first  \n";
        }// end insertFirst()

        public String delete(int item) {
                if (start == null)
                        return "list is empty  \n ";
                node current = start;
                while (current.data != item) {
                        if (current.next == null)
                                return item + " is not found  \n ";
                        current = current.next;
                }
                if (current == start)
                        start = current.next;
                else
                        current.prev.next = current.next;
                if (current == end)
                        end = current.prev;
                else
                        current.next.prev = current.prev;
```

```
            return item + " is deleted  \n ";
        }// end delete()

        public String traverse() {
            if (start == null)
                return "list is empty \n";
            node temp = start;
            String output = "list : ";
            while (temp != null) {
                output += temp.data + " ";
                temp = temp.next;
            } // end while
            return output + "\n";
        }// end display()

}
```

Output:



```
0:exit  1:insertFirst  2:delete   3:traverse
enter your choise:1
enter data :23
23 is inserted at first

0:exit  1:insertFirst  2:delete   3:traverse
enter your choise:1
enter data :45
45 is inserted at first

0:exit  1:insertFirst  2:delete   3:traverse
enter your choise:1
enter data :67
67 is inserted at first

0:exit  1:insertFirst  2:delete   3:traverse
enter your choise:1
enter data :21
21 is inserted at first

0:exit  1:insertFirst  2:delete   3:traverse
enter your choise:3
list : 21 67 45 23
```



```
0:exit  1:insertFirst  2:delete   3:traverse
enter your choise:2
enter data :67
67 is deleted

0:exit  1:insertFirst  2:delete   3:traverse
enter your choise:3
list : 21 45 23

0:exit  1:insertFirst  2:delete   3:traverse
enter your choise:2
enter data :21
21 is deleted

0:exit  1:insertFirst  2:delete   3:traverse
enter your choise:3
list : 45 23

0:exit  1:insertFirst  2:delete   3:traverse
enter your choise:0
.....exit.....
```

7. Observations:

At the end of this lab, I was able to reestablish fundamentals of doubly linked list and it's most basic algorithms.

Rubrics

Demonstration	Absent	Student is unable to follow the provided instructions properly. The student can name the hardware or simulation platform, but unable to implement anything practically or on the software	Student can understand the provided laboratory instructions and familiar with the lab environment (Trainer/ software/ IDE), but cannot implement on the platform practically or on the software	Student has followed instructions to construct the fundamental schematic/ block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code, and have successfully executed the program/ run circuit on software platform	Student perfectly implemented a working model/ logic/ circuit/ block diagram/ code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory Reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recorded along with detailed procedure	Appropriate computations or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all respects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

LAB NO. 11

Stack Using Linked List

1. Objectives:

After completing this lab, students will be able to;

- Deep Understanding of Linked List
- Deep Understanding of Stack.
- Implementation of Stack in Linked List
- Procedure and Algorithm

2. Outline:

- Linked List
- Stack
- Practical Implementation
- Examples

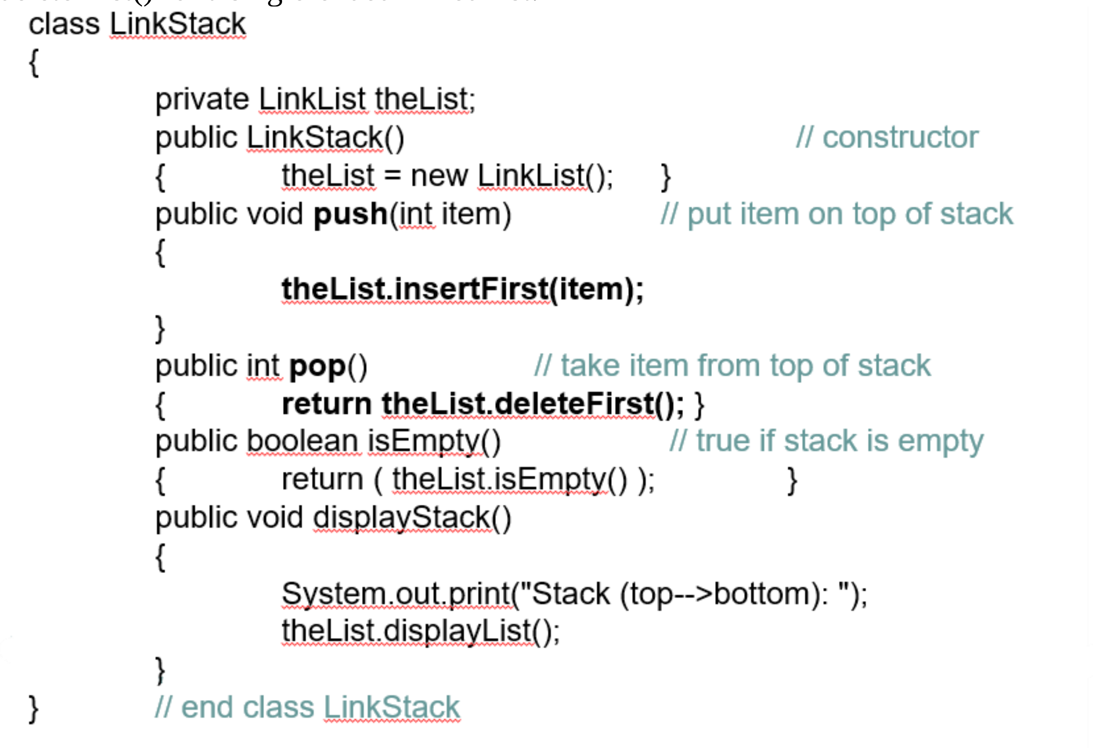
3. Corresponding CLO and PLO:

- CLO 2, PLO 3

4. Background:

When we created a stack in the previous chapter, we used an ordinary array to hold the stack's data. The stack's push() and pop() operations were actually carried out by array operations, which insert data into, and take it out of, an array.

We can also use a linked list to hold a stack's data. In this case the push() and pop() operations would be carried out by calling the methods of insertFirst() and deleteFirst() for a single-ended linkedList.



```
class LinkStack
{
        private LinkList theList;
        public LinkStack()                              // constructor
        {       theList = new LinkList();     }
        public void push(int item)              // put item on top of stack
        {
                theList.insertFirst(item);
        }
        public int pop()               // take item from top of stack
        {       return theList.deleteFirst(); }
        public boolean isEmpty()               // true if stack is empty
        {       return ( theList.isEmpty() );           }
        public void displayStack()
        {
                System.out.print("Stack (top-->bottom): ");
                theList.displayList();
        }
}       // end class LinkStack
```

5. Equipment:

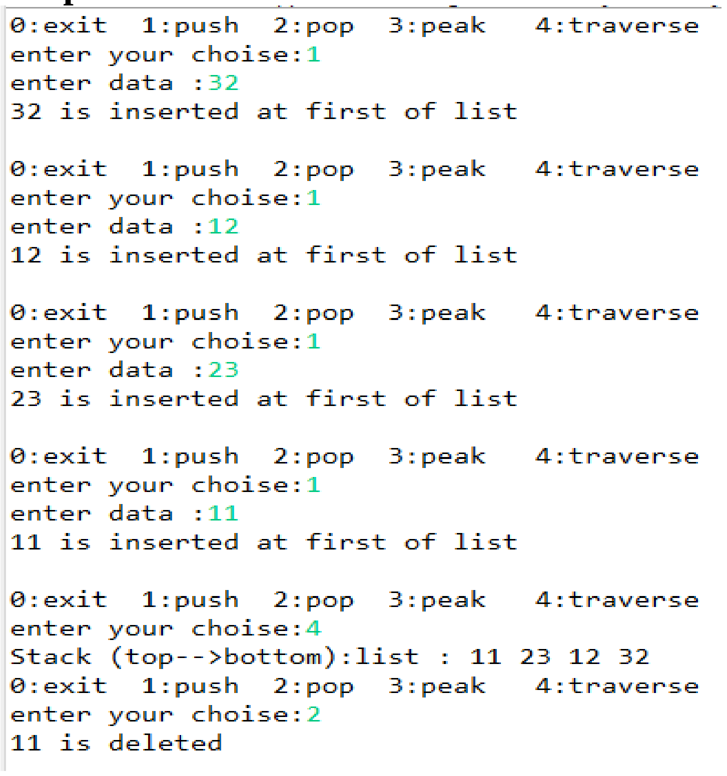
Eclipse

6. Procedure:

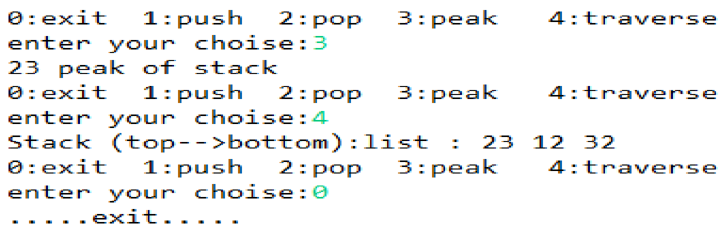
Code:

```
import java.util.Scanner;

public class Lab11 {

	public static void main(String args[]) {

		LinkStack ls = new LinkStack();
		Scanner sc = new Scanner(System.in);
		boolean run = true;
		while (run) {
			System.out.print("0:exit  1:push  2:pop  3:peak  
4:traverse" + "\nenter your choise:");
			int ch = 0;
			ch = sc.nextInt();

			switch (ch) {
			case 0:

				run = false;
				System.out.println(".....exit.....");
				break;
			case 1:

				System.out.print("enter data :");
				System.out.println(ls.push(sc.nextInt()));

				break;
			case 2:
				System.out.println(ls.pop());

				break;
			case 3:
				System.out.print(ls.peak());

				break;
			case 4:
				ls.traverse();
				break;

			default:
				System.out.println("invalid option");
			}// end switch
		} // end while loop
	} // end void main()

} // end Lab class

class LinkStack {
	private DoubleEndedLinkedList list;

	public LinkStack() {
		list = new DoubleEndedLinkedList();
	}

	public String push(int item) {
		return list.insertFirst(item);

	}

	public String pop() {
		return list.deleteFirst();
	}

	public String peak() {
		return list.start.data + " peak of stack \n";
	}

	public void traverse() {
		System.out.print("Stack (top-->bottom):");
		System.out.print(list.traverse());
	}
}// end LinkStack
```

Output:



```
0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:1
enter data :32
32 is inserted at first of list

0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:1
enter data :12
12 is inserted at first of list

0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:1
enter data :23
23 is inserted at first of list

0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:1
enter data :11
11 is inserted at first of list

0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:4
Stack (top-->bottom):list : 11 23 12 32
0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:2
11 is deleted
```



```
0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:3
23 peak of stack
0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:4
Stack (top-->bottom):list : 23 12 32
0:exit   1:push   2:pop   3:peak   4:traverse
enter your choise:0
.....exit.....
```

Observations

At the end of this lab, I was able to reestablish fundamentals of stack with linked list and it’s most basic algorithms.

Rubrics

Demonstration	Absent	Student is unable to follow the provided instructions properly. The student can name the hardware or simulation platform, but unable to implement anything practically or on the software	Student can understand the provided laboratory instructions and familiar with the lab environment (Trainer/ software/ IDE), but cannot implement on the platform practically or on the software	Student has followed instructions to construct the fundamental schematic/ block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code, and have successfully executed the program/ run circuit on software platform	Student perfectly implemented a working model/ logic/ circuit/ block diagram/ code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory Reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recorded along with detailed procedure	Appropriate computations or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all respects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

LAB NO. 12

Queue Using Linked List

1. Objectives:

- Deep understanding of Linked List
- Deep understanding of Queue
- Implementation of Queue in Linked List

2. Outline:

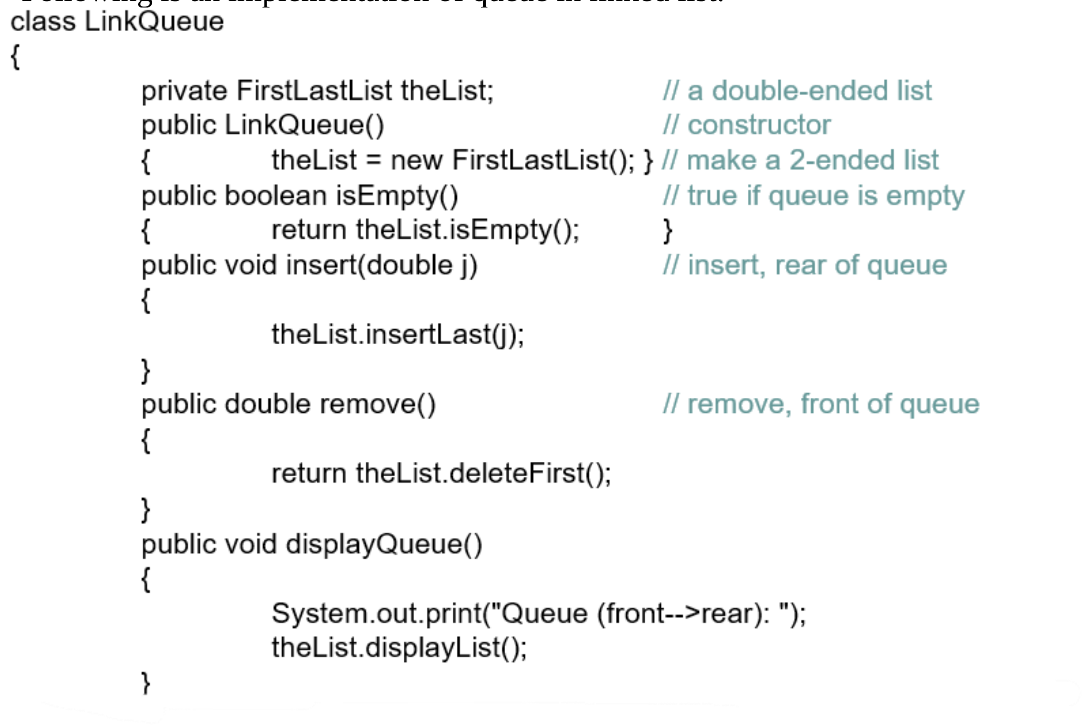
- Linked List
- Queue
- Practical Implementation
- Example

3. Corresponding CLO and PLO:

- CLO 2, PLO 3

4. Background:

Linked Lists can also be used to develop queue ADT.
Following is an implementation of queue in linked list.



```
class LinkQueue
{
          private FirstLastList theList;              // a double-ended list
          public LinkQueue()                          // constructor
          {         theList = new FirstLastList(); } // make a 2-ended list
          public boolean isEmpty()                    // true if queue is empty
          {         return theList.isEmpty();         }
          public void insert(double j)                // insert, rear of queue
          {
                    theList.insertLast(j);
          }
          public double remove()                      // remove, front of queue
          {
                    return theList.deleteFirst();
          }
          public void displayQueue()
          {
                    System.out.print("Queue (front-->rear): ");
                    theList.displayList();
          }
```

5. Equipment:

Eclipse

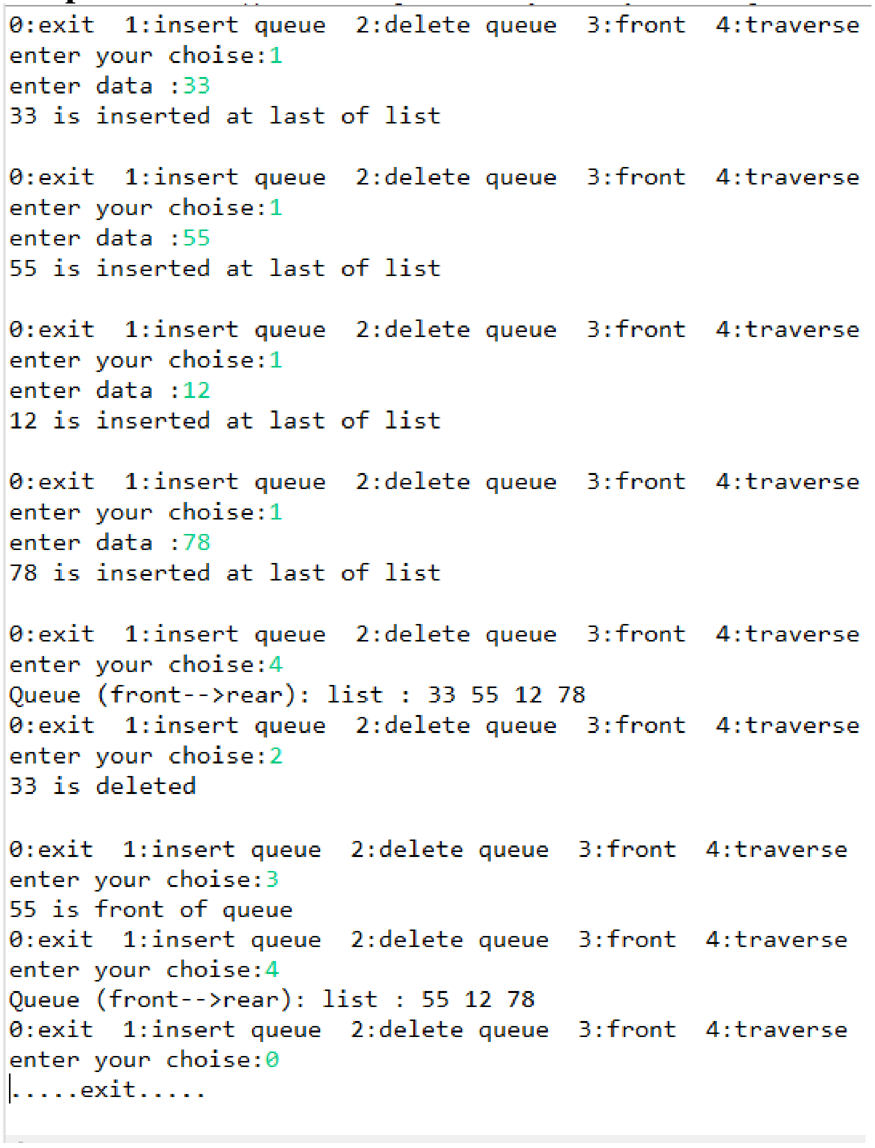
6. Procedure:

Code:

```
import java.util.Scanner;

public class Lab12 {

        public static void main(String args[]) {

                LinkQueue lq = new LinkQueue();
                Scanner sc = new Scanner(System.in);
                boolean run = true;
                while (run) {
                        System.out.print("0:exit  1:insert queue  2:delete queue
3:front  4:traverse" + "\nenter your choise:");
                        int ch = 0;
                        ch = sc.nextInt();

                        switch (ch) {
                        case 0:

                                run = false;
                                System.out.println(".....exit.....");
                                break;
                        case 1:

                                System.out.print("enter data :");
                                System.out.println(lq.enqueue(sc.nextInt()));

                                break;
                        case 2:
                                System.out.println(lq.dequeue());

                                break;
                        case 3:
                                System.out.print(lq.front());

                                break;
                        case 4:
                                lq.traverse();
                                break;
                        default:
                                System.out.println("invalid option");
                        }// end switch
                } // end while loop
        } // end void main()

} // end Lab class

class LinkQueue {
        private DoubleEndedLinkedList list;

        public LinkQueue() {
                list = new DoubleEndedLinkedList();
        }

        public String enqueue(int item) {
                return list.insertLast(item);
        }

        public String dequeue() {
                return list.deleteFirst();
        }

        public String front() {
                if (list.start == null)
                        return "queue is empty \n";
```

```
            return list.start.data + " is front of queue \n";
        }

        public void traverse() {
            System.out.print("Queue (front-->rear): ");
            System.out.print(list.traverse());
        }
} // end LinkQueue
```

Output:



```
0:exit  1:insert queue  2:delete queue  3:front  4:traverse
enter your choise:1
enter data :33
33 is inserted at last of list

0:exit  1:insert queue  2:delete queue  3:front  4:traverse
enter your choise:1
enter data :55
55 is inserted at last of list

0:exit  1:insert queue  2:delete queue  3:front  4:traverse
enter your choise:1
enter data :12
12 is inserted at last of list

0:exit  1:insert queue  2:delete queue  3:front  4:traverse
enter your choise:1
enter data :78
78 is inserted at last of list

0:exit  1:insert queue  2:delete queue  3:front  4:traverse
enter your choise:4
Queue (front-->rear): list : 33 55 12 78
0:exit  1:insert queue  2:delete queue  3:front  4:traverse
enter your choise:2
33 is deleted

0:exit  1:insert queue  2:delete queue  3:front  4:traverse
enter your choise:3
55 is front of queue
0:exit  1:insert queue  2:delete queue  3:front  4:traverse
enter your choise:4
Queue (front-->rear): list : 55 12 78
0:exit  1:insert queue  2:delete queue  3:front  4:traverse
enter your choise:0
.....exit.....
```

7. Observations:

At the end of this lab, I was able to reestablish fundamentals of stack using linked list and it’s most basic algorithms.

Rubrics

Demonstration	Absent	Student is unable to follow the provided instructions properly. The student can name the hardware or simulation platform, but unable to implement anything practically or on the software	Student can understand the provided laboratory instructions and familiar with the lab environment (Trainer/ software/ IDE), but cannot implement on the platform practically or on the software	Student has followed instructions to construct the fundamental schematic/ block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code, and have successfully executed the program/ run circuit on software platform	Student perfectly implemented a working model/ logic/ circuit/ block diagram/ code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory Reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recorded along with detailed procedure	Appropriate computations or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all respects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

LAB NO. 13

Tree

1. Objectives:

- Introduction to Tree
- Re-establishment of OOP
- Procedure and Algorithm

2. Outline:

- Tree
- Practical Implementation
- Examples

3. Corresponding CLO and PLO:

- CLO 1, PLO 5

4. Background:

Binary trees are one of the fundamental data storage structures used in programming. They provide advantages that the data structures we've seen so far cannot.

Tree is one of the most important nonlinear data structures in computing. It allows us to implement faster algorithms (compared with algorithms using linear data structures).

Definition:

A Tree is a set of nodes storing elements in a parent-child relationship with the following properties:

It has a special node called root.

Each node other than the root has a Parent node.

Terms:

Parent:

The parent of a node is the node linked above it

Sibling:

Two nodes are siblings if they have the same parent.

Leaf:

A node which has no child.

Subtree:

Any node and its descendants form a subtree of the original tree.

Path of two nodes:

A path that begins at the starting node and goes from node to node along the edges that join them until the ending node.

Length of a path:

The number of the edges that compose it.

Depth of a node:

The length of the path between the root and the node.

Height of a tree:

The maximum depth of a leaf node.

Tree Types:

Binary tree:

Each node has at most two children

n- ary tree:

Each node has at most n children

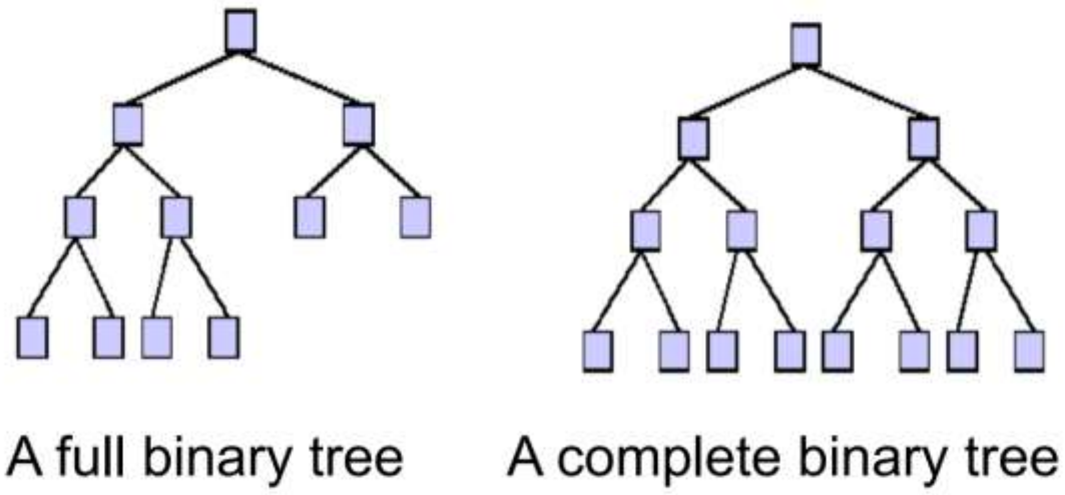
General tree:

Each node can have an arbitrary number of children.

A binary tree is full if every non-leaf node in the tree has exactly two children

A binary tree is complete if every level except the deepest contains as many nodes as possible, and all the nodes at the deepest level are as far left as possible. A complete binary tree of depth n has Maximum number of leaves: 2n

Maximum number of total nodes: 2^(n+1)-1.



One commonly encountered tree is the hierarchical file structure in a computer system.

The root directory of a given device (designated with the backslash, as in C:\, on many systems) is the tree's root. The directories one level below the root directory are its children. There may be many levels of subdirectories. Files represent leaves; they have no children of their own.

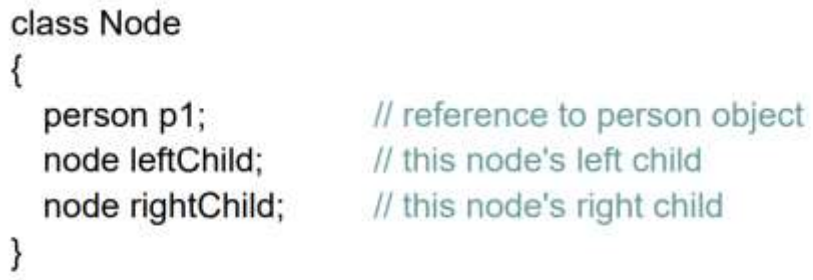
Clearly a hierarchical file structure is not a binary tree, because a directory may have many children.

Traverse In Tree:

Traversing a tree means visiting each node in a specified order. This process is not as commonly used as finding, inserting, and deleting nodes. One reason for this is that traversal is not particularly fast. But traversing a tree is useful in some circumstances and the algorithm is interesting.

There are three simple ways to traverse a tree. They're called preorder, inorder, and postorder.

The order most commonly used for binary search trees is inorder, so let's look at that first, and then return briefly to the other two



```
class Node
{
   person p1;           // reference to person object
   node leftChild;      // this node's left child
   node rightChild;     // this node's right child
}
```

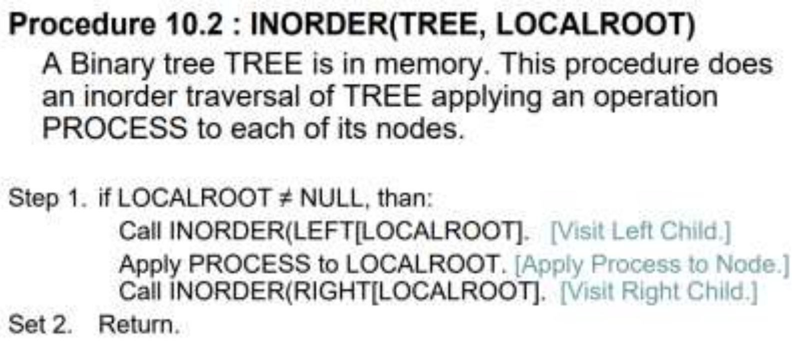
A tree can be traversed by running through leftChild and rightChild respectively. There are three main ways to do so.

1. Inorder Traverse:

An inorder traversal of a binary search tree will cause all the nodes to be visited in ascending order, based on their key values. If you want to create a sorted list of the data in a binary tree, this is one way to do it.

The simplest way to carry out a traversal is the use of recursion. A recursive method to traverse the entire tree is called with a node as an argument. Initially, this node is the root. The method needs to do only three things:

1. Call itself to traverse the node's left subtree.
2. Visit the node.
3. Call itself to traverse the node's right subtree.

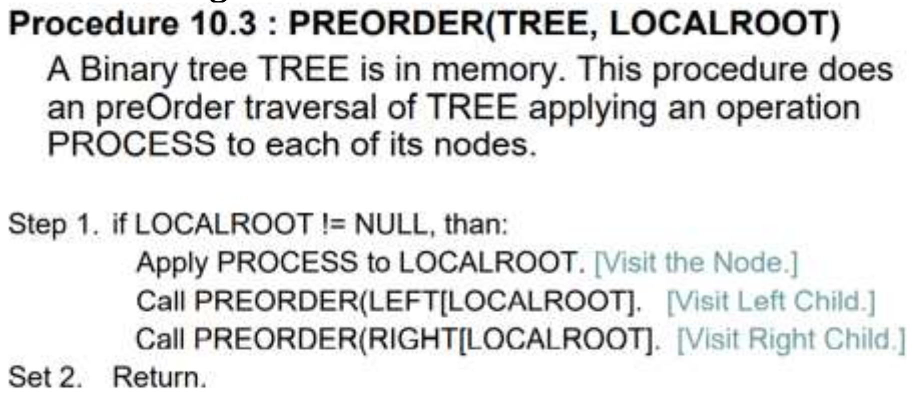


Procedure 10.2 : INORDER(TREE, LOCALROOT)

A Binary tree TREE is in memory. This procedure does an inorder traversal of TREE applying an operation PROCESS to each of its nodes.

```
Step 1.  if LOCALROOT ≠ NULL, than:
             Call INORDER(LEFT[LOCALROOT].   [Visit Left Child.]
             Apply PROCESS to LOCALROOT. [Apply Process to Node.]
             Call INORDER(RIGHT[LOCALROOT].  [Visit Right Child.]
Set 2.   Return.
```

2. Preorder Traversing:

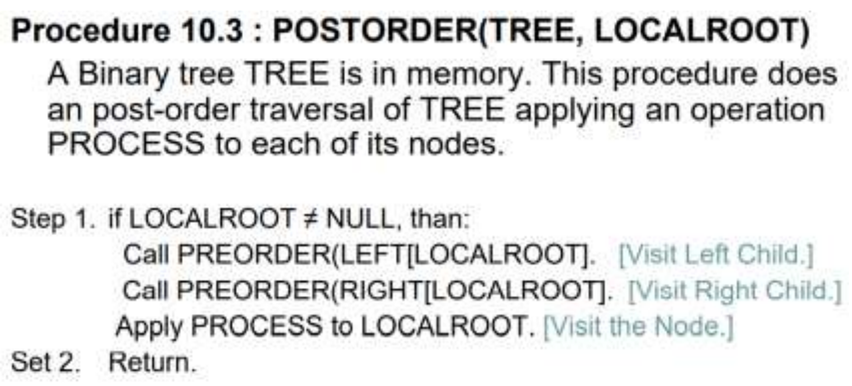


Procedure 10.3 : PREORDER(TREE, LOCALROOT)

A Binary tree TREE is in memory. This procedure does an preOrder traversal of TREE applying an operation PROCESS to each of its nodes.

```
Step 1.  if LOCALROOT != NULL, than:
             Apply PROCESS to LOCALROOT. [Visit the Node.]
             Call PREORDER(LEFT[LOCALROOT].   [Visit Left Child.]
             Call PREORDER(RIGHT[LOCALROOT].  [Visit Right Child.]
Set 2.   Return.
```

3. PostOrder Traversing:



Procedure 10.3 : POSTORDER(TREE, LOCALROOT)

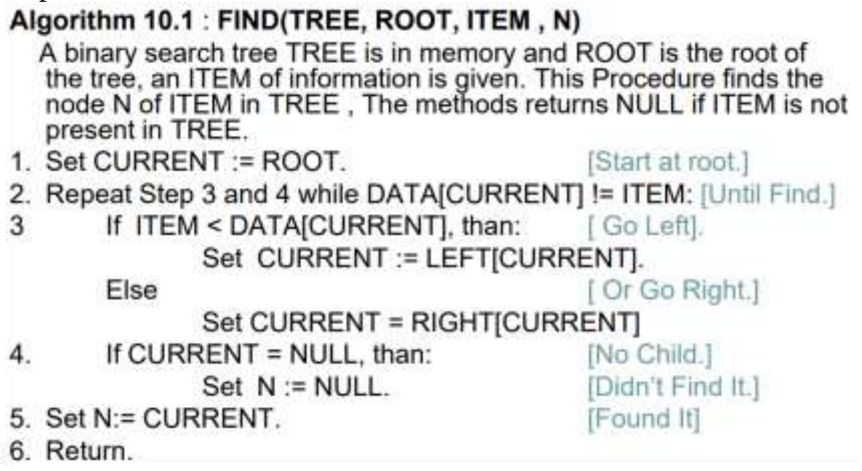
A Binary tree TREE is in memory. This procedure does an post-order traversal of TREE applying an operation PROCESS to each of its nodes.

```
Step 1. if LOCALROOT ≠ NULL, than:
            Call PREORDER(LEFT[LOCALROOT].   [Visit Left Child.]
            Call PREORDER(RIGHT[LOCALROOT].  [Visit Right Child.]
           Apply PROCESS to LOCALROOT. [Visit the Node.]
Set 2.  Return.
```

To insert a node we must first find the place to insert it. This is much the same process as trying to find a node that turns out not to exist, as described in the section on Find. We follow the path from the root to the appropriate node, which will be the parent of the new node. Once this parent is found, the new node is connected as its left or right child, depending on whether the new node's key is less than or greater than that of the parent

Thus, first step is to understand how to find a specific node.Finding a node with a specific key is the simplest of the major tree operations, so let's start with that.

Remember that the nodes in a binary search tree correspond to objects containing information. They could be person objects, with an employee number as the key and also perhaps name, address, telephone number, salary, and other fields. Or they could represent car parts, with a part number as the key value and fields for quantity on hand, price, and so on.



Algorithm 10.1 : FIND(TREE, ROOT, ITEM , N)

A binary search tree TREE is in memory and ROOT is the root of the tree, an ITEM of information is given. This Procedure finds the node N of ITEM in TREE , The methods returns NULL if ITEM is not present in TREE.

```
1. Set CURRENT := ROOT.                          [Start at root.]
2. Repeat Step 3 and 4 while DATA[CURRENT] != ITEM: [Until Find.]
3        If  ITEM < DATA[CURRENT], than:         [ Go Left].
                 Set  CURRENT := LEFT[CURRENT].
         Else                                    [ Or Go Right.]
                 Set CURRENT = RIGHT[CURRENT]
4.       If CURRENT = NULL, than:                [No Child.]
                 Set  N := NULL.                 [Didn't Find It.]
5. Set N:= CURRENT.                              [Found It]
6. Return.
```

5. Equipment:

Eclipse

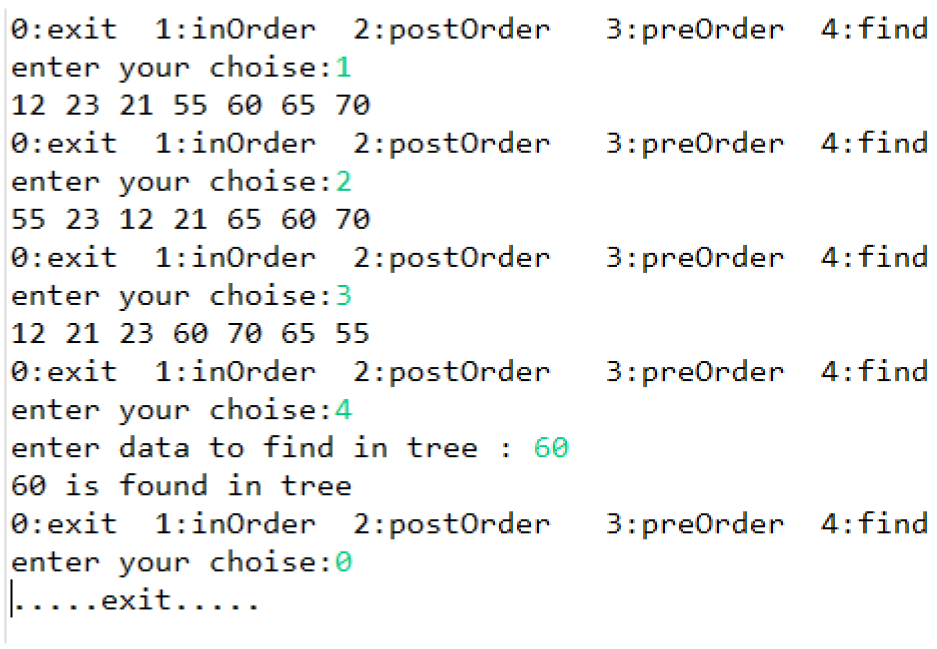
6. Procedure:

Code:

```
import java.util.Scanner;

class Node {
	int data;
	Node left;
	Node right;
	boolean isLeft = false;
	public Node() {
	}

	public Node(int item) {
		data = item;
	}
}

public class Lab13 {

	public static void main(String args[]) {

		Node root = new Node(55);

		root.left = new Node(23);
		root.left.left = new Node(12);
		root.left.right = new Node(21);
		root.right = new Node(65);
		root.right.left = new Node(60);
		root.right.right = new Node(70);
		Scanner sc = new Scanner(System.in);
		boolean run = true;
		while (run) {
			System.out.print("0:exit  1:inOrder  2:postOrder
3:preOrder  4:find" + "\nenter your choise:");
			int ch = 0;
			ch = sc.nextInt();

			switch (ch) {
			case 0:

				run = false;
				System.out.println(".....exit.....");
				break;
			case 1:
				inOrder(root);
				System.out.print("\n");

				break;
			case 2:

				postOrder(root);
				System.out.print("\n");

				break;
			case 3:

				preOrder(root);
				System.out.print("\n");

				break;
			case 4:
				System.out.print("enter data to find in tree :
");

				System.out.print(find(root, sc.nextInt()));
				break;
			case 5:
```

```
                        break;
                    default:
                        System.out.println("invalid option");
                    }// end switch
                } // end while loop
        } // end void main()

        public static void postOrder(Node root) {
                if (root == null)
                        return;
                System.out.print(root.data + " ");
                postOrder(root.left);
                postOrder(root.right);

        }// end postorder()

        public static void inOrder(Node root) {
                if (root == null)
                        return;

                inOrder(root.left);
                System.out.print(root.data + " ");
                inOrder(root.right);

        }// end postorder()

        public static void preOrder(Node root) {
                if (root == null)
                        return;

                preOrder(root.left);
                preOrder(root.right);
                System.out.print(root.data + " ");

        }// end postorder()

        public static String find(Node root, int item) {

                Node cur = root;
                while (cur.data != item) {

                        if (item < cur.data)
                                cur = cur.left;
                        else
                                cur = cur.right;

                        if (cur == null)
                                return item + " is not found in tree  \n";
                } // end while()
                return item + " is found in tree   \n";
        }// end search()

} // end Lab class
```

Output:


```
0:exit  1:inOrder  2:postOrder   3:preOrder  4:find
enter your choise:1
12 23 21 55 60 65 70
0:exit  1:inOrder  2:postOrder   3:preOrder  4:find
enter your choise:2
55 23 12 21 65 60 70
0:exit  1:inOrder  2:postOrder   3:preOrder  4:find
enter your choise:3
12 21 23 60 70 65 55
0:exit  1:inOrder  2:postOrder   3:preOrder  4:find
enter your choise:4
enter data to find in tree : 60
60 is found in tree
0:exit  1:inOrder  2:postOrder   3:preOrder  4:find
enter your choise:0
.....exit.....
```

7. Observations:

At the end of this lab, I was able to reestablish fundamentals of Tree and it's most basic algorithms.

Rubrics

Demonstration	Absent	Student is unable to follow the provided instructions properly. The student can name the hardware or simulation platform, but unable to implement anything practically or on the software	Student can understand the provided laboratory instructions and familiar with the lab environment (Trainer/ software/ IDE), but cannot implement on the platform practically or on the software	Student has followed instructions to construct the fundamental schematic/ block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code, and have successfully executed the program/ run circuit on software platform	Student perfectly implemented a working model/ logic/ circuit/ block diagram/ code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory Reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recorded along with detailed procedure	Appropriate computations or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all respects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

LAB NO. 14

Insertion in Tree

1. Lab outcomes:

After completing this lab, students will be able to;

- Deep Understanding of Tree
- Understanding of Insertion in Tree

2. Outline:

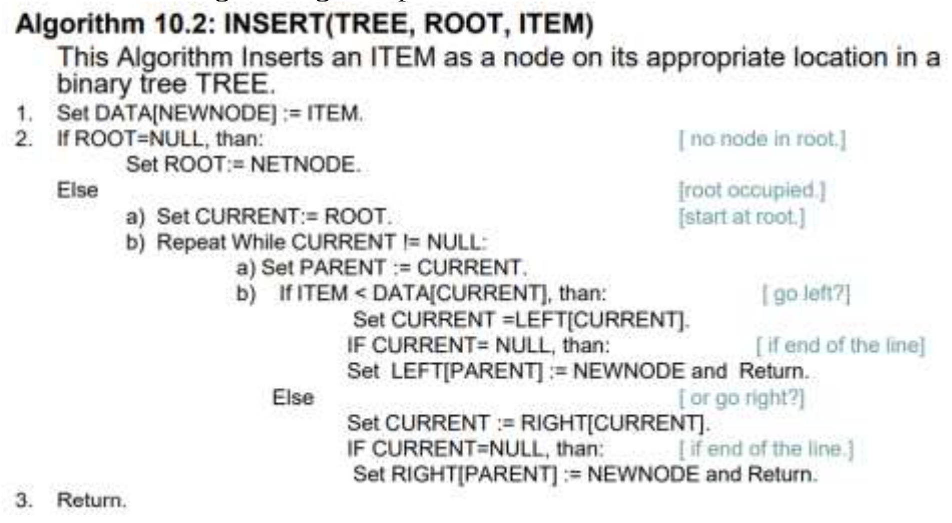
- Tree
- Insertion In Tree
- Algorithms

3. Corresponding CLO and PLO:

- CLO 1, 2, PLO 5

4. Background:

After establishing finding the specific node, we can insert a value to that node.



```
Algorithm 10.2: INSERT(TREE, ROOT, ITEM)
    This Algorithm Inserts an ITEM as a node on its appropriate location in a
    binary tree TREE.
1.  Set DATA[NEWNODE] := ITEM.
2.  If ROOT=NULL, than:                                  [ no node in root.]
            Set ROOT:= NETNODE.
    Else                                                 [root occupied.]
            a) Set CURRENT:= ROOT.                       [start at root.]
            b) Repeat While CURRENT != NULL:
                    a) Set PARENT := CURRENT.
                    b)   If ITEM < DATA[CURRENT], than:          [ go left?]
                              Set CURRENT =LEFT[CURRENT].
                             IF CURRENT= NULL, than:            [ if end of the line]
                             Set  LEFT[PARENT] := NEWNODE and  Return.
                        Else                             [ or go right?]
                             Set CURRENT := RIGHT[CURRENT].
                             IF CURRENT=NULL, than:      [ if end of the line.]
                              Set RIGHT[PARENT] := NEWNODE and Return.
3.  Return.
```

5. Equipment:

- Eclipse

6. Procedure:

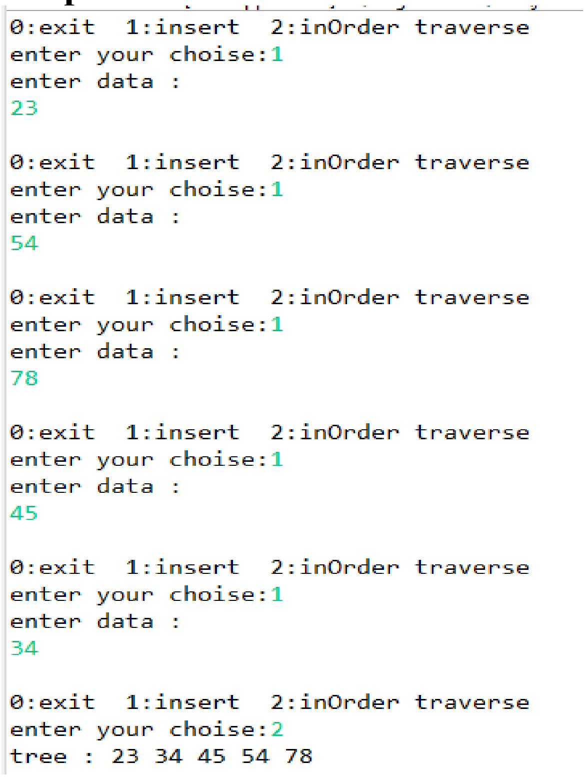
Code :

```
import java.util.Scanner;

public class Lab14 {

	public static void main(String args[]) {

		InsertionInTree t = new InsertionInTree();
		Scanner sc = new Scanner(System.in);
```

```
                boolean run = true;
                while (run) {
                        System.out.print("0:exit  1:insert  2:inOrder traverse "
+ "\nenter your choise:");
                        int ch = 0;
                        ch = sc.nextInt();

                        switch (ch) {
                        case 0:

                                run = false;
                                System.out.println(".....exit.....");
                                break;
                        case 1:
                                System.out.println("enter data : ");
                                t.insert(sc.nextInt());
                                System.out.print("\n");

                                break;
                        case 2:
                                System.out.print("tree : ");
                                t.inOrder(t.root);
                                System.out.println("");
                                break;

                        default:
                                System.out.println("invalid option");
                        }// end switch
                } // end while loop
        } // end void main()

} // end Lab class

class InsertionInTree {
        public Node root;

        public InsertionInTree() {
                root = null;
        }

        public String insert(int item) {
                Node temp = new Node(item);
                if (root == null) {
                        root = temp;
                        return item + " is inserted  \n";
                }

                Node cur = root;
                while (cur != null) {
                        Node parent = cur;
                        if (item < cur.data) {
                                cur = cur.left;

                                if (cur == null) {
                                        parent.left = temp;
                                        parent.left.isLeft = true;
                                }
                        } // end if
                        else {
```

```
                        cur = cur.right;

                        if (cur == null) {
                              parent.right = temp;
                              parent.right.isLeft = false;
                        }
                  } // end else
            } // end while()
            return item + " is inserted \n ";
      }// end insert()

      public void inOrder(Node root) {
            if (root == null)
                  return;

            inOrder(root.left);
            System.out.print(root.data + " ");
            inOrder(root.right);

      }// end inOrder()

}// end Tree
```

Output:



```
0:exit  1:insert  2:inOrder traverse
enter your choise:1
enter data :
23

0:exit  1:insert  2:inOrder traverse
enter your choise:1
enter data :
54

0:exit  1:insert  2:inOrder traverse
enter your choise:1
enter data :
78

0:exit  1:insert  2:inOrder traverse
enter your choise:1
enter data :
45

0:exit  1:insert  2:inOrder traverse
enter your choise:1
enter data :
34

0:exit  1:insert  2:inOrder traverse
enter your choise:2
tree : 23 34 45 54 78
```

7. Observations:

At the end of this lab, I was able to implement insertion function of tree and it's most basic algorithms.

Rubrics

Demonstration	Absent	Student is unable to follow the provided instructions properly. The student can name the hardware or simulation platform, but unable to implement anything practically or on the software	Student can understand the provided laboratory instructions and familiar with the lab environment (Trainer/ software/ IDE), but cannot implement on the platform practically or on the software	Student has followed instructions to construct the fundamental schematic/ block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code, and have successfully executed the program/ run circuit on software platform	Student perfectly implemented a working model/ logic/ circuit/ block diagram/ code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor’s Signature**		

Laboratory Reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recorded along with detailed procedure	Appropriate computations or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all respects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor’s Signature**		

LAB NO. 15

Deletion In Tree

1. Lab outcomes:

After completing this lab, students will be able to;
- Deep understanding of Tree
- Deletion in Tree
- Algorithm and Code

2. Outline:

- Tree
- Deletion in Queue
- Algorithms

3. Corresponding CLO and PLO:

- CLO 2, PLO 5

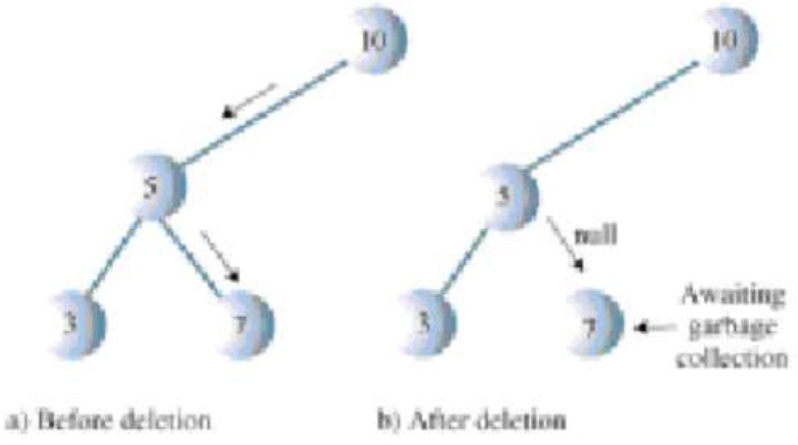
4. Background:

Deleting a node is the most complicated common operation required for binary search trees. However, deletion is important in many tree applications. λ You start by finding the node you want to delete, using the same approach we saw in find() and insert(). Once you've found the node, there are three cases to consider. 1. The node to be deleted is a leaf (has no children). 2. The node to be deleted has one child. 3. The node to be deleted has two children. λ We'll look at these three cases in turn. The first is easy, the second almost as easy, and the third quite complicated

Case 1: The node to be deleted has no Children:

To delete a leaf node, you simply change the appropriate child field in the node's parent to point to null instead of to the node. The node will still exist, but it will no longer be part of the tree.

Because of Java's garbage collection feature, we don't need to worry about explicitly deleting the node itself. When Java realizes that nothing in the program refers to the node, it will be removed from memory. (In C and C++ you would need to execute free() or delete() to remove the node from memory.)

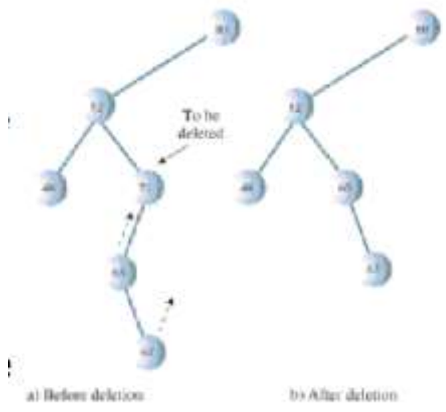


Algorithm:

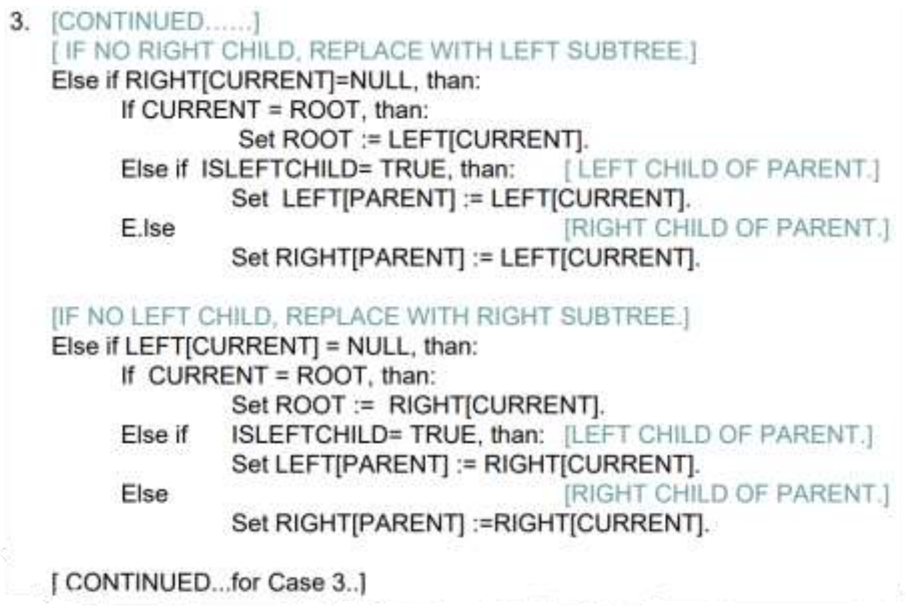
```
Procedure 10.aaa  DELETE(TREE, ROOT, KEY)
    This Method deletes the Node N of which has key KEY from a binary
    tree TREE.
1.  [Initialize Pointers.]
        SET     CURRENT := ROOT and
                PARENT  := ROOT and
                ISLEFTCHILD = TRUE.                    [Is it the Left Leaf.?]
2.  Repeat until DATA[CURRENT] != KEY)         [ Search for Node.]
    a)  Set PARENT := CURRENT.
    b)  If KEY <DATA[CURRENT], than:           [ GO LEFT?]
            i)   Set ISLEFTCHILD := TRUE.
            ii)  Set CURRENT:= LEFT[CURRENT].
        Else                                   [ OR GO RIGHT?]
            i)   Set ISLEFTCHILD := FALSE.
            ii)  Set CURRENT := RIGHT[CURRENT].
   c)   If CURRENT = NULL, than:               [ END OF THE LINE?]
        Print "NODE not Found " and RETURN.    [ DIDN'T FIND IT].
                            (continue...)

   [ If node is found and it has no left or right child . CASE 1.]
3.  IF LEFT[CURRENT]  =NULL and RIGHT[CURRENT] =NULL, than:
        a)      If CURRENT = ROOT, than                [ IF ROOT?]
                        Set ROOT := NULL.              [ TREE IS EMPTY.]
                Else If ISLEFTCHILD= TRUE, than:       [if it is a left child?]
                        Set LEFT[PARENT] := NULL.      [ disconnect from parent.]
                Else
                        Set RIGHT[PARENT] := NULL.

[CONTINUES...to Case 2 and 3].
```

Case 2: The node has one children:

This case isn't so bad either. The node has only two connections: to its parent and to its only child. You want to "snip" the node out of this sequence by connecting its parent directly to its child. This involves changing the appropriate reference in the parent (leftChild or rightChild) to point to the deleted node's child.



a) Before deletion b) After deletion



```
3. [CONTINUED......]
   [ IF NO RIGHT CHILD, REPLACE WITH LEFT SUBTREE.]
   Else if RIGHT[CURRENT]=NULL, than:
          If CURRENT = ROOT, than:
                    Set ROOT := LEFT[CURRENT].
          Else if  ISLEFTCHILD= TRUE, than:     [ LEFT CHILD OF PARENT.]
                    Set  LEFT[PARENT] := LEFT[CURRENT].
          E.lse                                 [RIGHT CHILD OF PARENT.]
                    Set RIGHT[PARENT] := LEFT[CURRENT].

   [IF NO LEFT CHILD, REPLACE WITH RIGHT SUBTREE.]
   Else if LEFT[CURRENT] = NULL, than:
          If  CURRENT = ROOT, than:
                    Set ROOT :=  RIGHT[CURRENT].
          Else if   ISLEFTCHILD= TRUE, than:  [LEFT CHILD OF PARENT.]
                    Set LEFT[PARENT] := RIGHT[CURRENT].
          Else                                [RIGHT CHILD OF PARENT.]
                    Set RIGHT[PARENT] :=RIGHT[CURRENT].

   [ CONTINUED...for Case 3..]
```

Case 3: The node has 3 children:

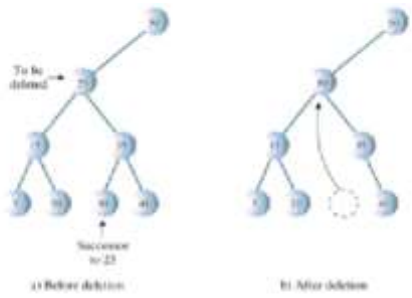
Now the fun begins. If the deleted node has two children, you can't just replace it with one of these children, at least if the child has its own children.

We need another approach.

The good news is that there's a trick.

The bad news is that, even with the trick, there are a lot of special cases to consider. Remember that in a binary search tree the nodes are arranged in order of ascending keys. For each node, the node with the next-highest key is called its inorder successor, or simply its successor.

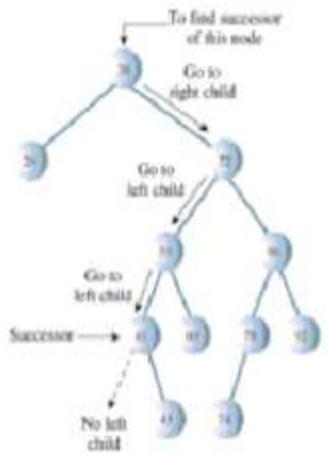
Here's the trick: To delete a node with two children, replace the node with its inorder successor. Figure shows a deleted node being replaced by its successor. Notice that the nodes are still in order. (There's more to it if the successor itself has children; we'll look at that possibility in a moment.)



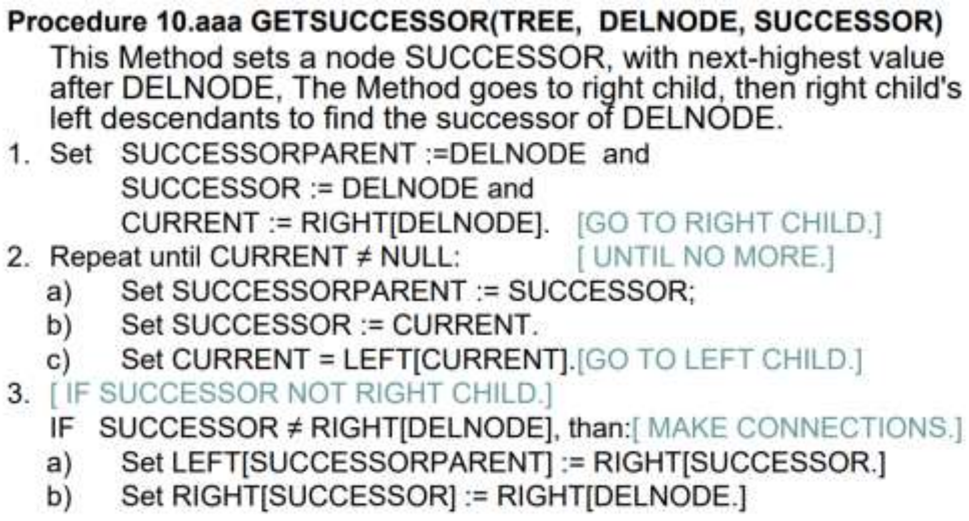
Finding Successor:

First the program goes to the original node's right child, which must have a key larger than the node.

Then it goes to this right child's left child (if it has one), and to this left child's left child, and so on, following down the path of left children. The last left child in this path is the successor of the original node.

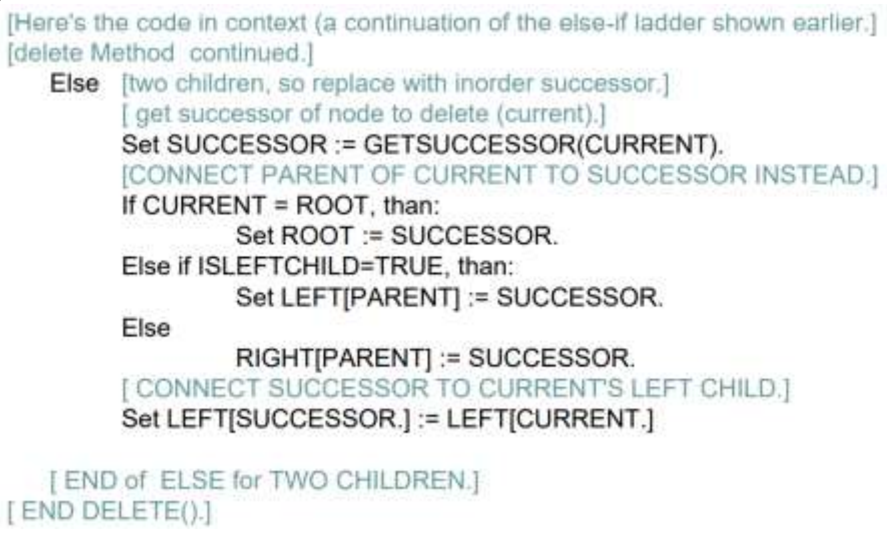


Algorithm to find Successor:



```
Procedure 10.aaa GETSUCCESSOR(TREE,  DELNODE, SUCCESSOR)
   This Method sets a node SUCCESSOR, with next-highest value
   after DELNODE, The Method goes to right child, then right child's
   left descendants to find the successor of DELNODE.
1. Set   SUCCESSORPARENT :=DELNODE  and
         SUCCESSOR := DELNODE and
         CURRENT := RIGHT[DELNODE].   [GO TO RIGHT CHILD.]
2. Repeat until CURRENT ≠ NULL:       [ UNTIL NO MORE.]
   a)    Set SUCCESSORPARENT := SUCCESSOR;
   b)    Set SUCCESSOR := CURRENT.
   c)    Set CURRENT = LEFT[CURRENT].[GO TO LEFT CHILD.]
3. [ IF SUCCESSOR NOT RIGHT CHILD.]
   IF   SUCCESSOR ≠ RIGHT[DELNODE], than:[ MAKE CONNECTIONS.]
   a)    Set LEFT[SUCCESSORPARENT] := RIGHT[SUCCESSOR.]
   b)    Set RIGHT[SUCCESSOR] := RIGHT[DELNODE.]
```

Algorithm for Case 3:



```
[Here's the code in context (a continuation of the else-if ladder shown earlier.]
[delete Method  continued.]
   Else  [two children, so replace with inorder successor.]
         [ get successor of node to delete (current).]
         Set SUCCESSOR := GETSUCCESSOR(CURRENT).
         [CONNECT PARENT OF CURRENT TO SUCCESSOR INSTEAD.]
         If CURRENT = ROOT, than:
                  Set ROOT := SUCCESSOR.
         Else if ISLEFTCHILD=TRUE, than:
                  Set LEFT[PARENT] := SUCCESSOR.
         Else
                  RIGHT[PARENT] := SUCCESSOR.
         [ CONNECT SUCCESSOR TO CURRENT'S LEFT CHILD.]
         Set LEFT[SUCCESSOR.] := LEFT[CURRENT.]

   [ END of  ELSE for TWO CHILDREN.]
[ END DELETE().]
```

5. Equipment:

- Eclipse

6. Procedure:

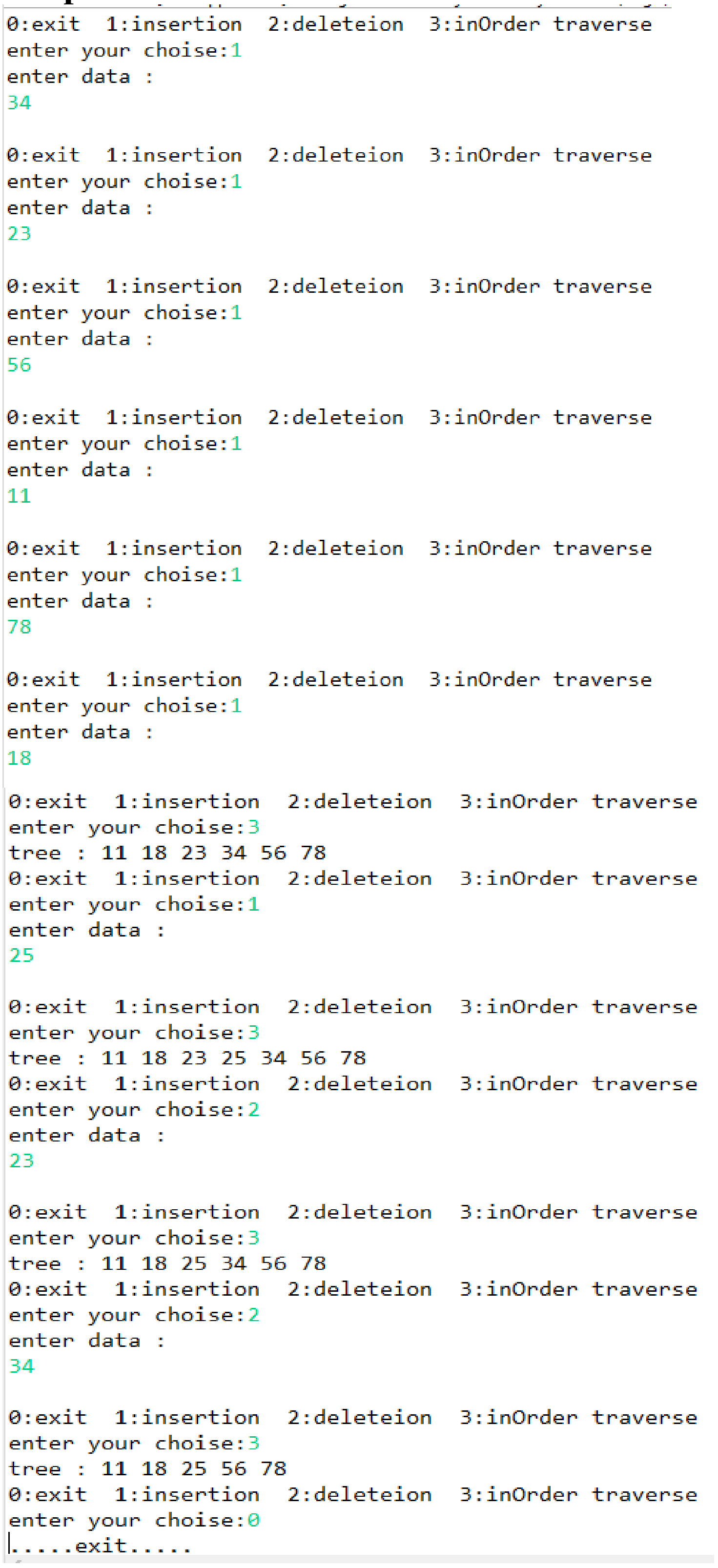
Code:

```
import java.util.Scanner;

public class Lab15 {

	public static void main(String args[]) {

		DeletionInTree t = new DeletionInTree();
		Scanner sc = new Scanner(System.in);
		boolean run = true;
		while (run) {
			System.out.print("0:exit  1:insertion  2:deleteion 
3:inOrder traverse" + "\nenter your choise:");
			int ch = 0;
			ch = sc.nextInt();

			switch (ch) {
			case 0:

				run = false;
				System.out.println(".....exit.....");
				break;
			case 1:
				System.out.println("enter data : ");
				t.insert(sc.nextInt());
				System.out.print("\n");
				break;

			case 2:
				System.out.println("enter data : ");
				t.delete(sc.nextInt());
				System.out.print("\n");
				break;

			case 3:
				System.out.print("tree : ");
				t.inOrder(t.root);
				System.out.println("");
				break;

			default:
				System.out.println("invalid option");
			}// end switch
		} // end while loop
	} // end void main()

} // end Lab class

class DeletionInTree {
	public Node root;

	public DeletionInTree() {
		root = null;
	}

	public String insert(int item) {
		Node temp = new Node(item);
		if (root == null) {
```

```
                root = temp;
                return item + " is inserted  \n";
            }

            Node cur = root;
            while (cur != null) {
                Node parent = cur;
                if (item < cur.data) {
                    cur = cur.left;

                    if (cur == null) {
                        parent.left = temp;
                        parent.left.isLeft = true;
                    }
                } // end if
                else {
                    cur = cur.right;

                    if (cur == null) {
                        parent.right = temp;
                        parent.right.isLeft = false;
                    }
                } // end else
            } // end while()
            return item + " is inserted \n ";
        }// end insert()

        public void inOrder(Node root) {
            if (root == null)
                return;

            inOrder(root.left);
            System.out.print(root.data + " ");
            inOrder(root.right);

        }// end inOrder()

        public String delete(int item) {
            if (root == null)
                return "Tree is empty   ";

            Node curr = root;
            Node parent = root;
            boolean isLeftChild = true;
            while (curr.data != item) {
                parent = curr;
                if (item < curr.data) {
                    curr = curr.left;
                    isLeftChild = true;
                } else {
                    curr = curr.right;
                    isLeftChild = false;
                }
                if (curr == null)
                    return "node not found   ";

            } // end while()
            // case 1
            if (curr.left == null && curr.right == null) {
```

```
                        if (curr == root)
                                root = null;
                        else if (isLeftChild)
                                parent.left = null;
                        else
                                parent.right = null;
                } // end case 1

                // case 2 right
                if (curr.left == null && curr.right != null) {
                        if (curr == root)
                                root = curr.right;
                        else if (isLeftChild)
                                parent.left = curr.right;
                        else
                                parent.right = curr.right;
                } // end case 2 right
                // case 2 left
                if (curr.left != null && curr.right == null) {
                        if (curr == root)
                                root = curr.left;
                        else if (isLeftChild)
                                parent.left = curr.left;
                        else
                                parent.right = curr.left;
                } // end case 2 left
                // case 3
                if (curr.left != null && curr.right != null) {
                        Node min = getMin(curr.right);
                        curr.data = min.data;

                        Node temp = curr;
                        if (temp.right == min)
                                temp.right = min.right;
                        else {
                                temp = temp.right;
                                while (temp.left != min)
                                        temp = temp.left;

                                temp.left = min.right;

                        }
                } // end case 3
                return item + " is deleted   ";
        }// end delete()

        public static Node getMin(Node curr) {

                while (curr.left != null)
                        curr = curr.left;

                return curr;
        }// end getMin()
}// end Tree
```

Output:



```
0:exit  1:insertion  2:deleteion  3:inOrder traverse
enter your choise:1
enter data :
34

0:exit  1:insertion  2:deleteion  3:inOrder traverse
enter your choise:1
enter data :
23

0:exit  1:insertion  2:deleteion  3:inOrder traverse
enter your choise:1
enter data :
56

0:exit  1:insertion  2:deleteion  3:inOrder traverse
enter your choise:1
enter data :
11

0:exit  1:insertion  2:deleteion  3:inOrder traverse
enter your choise:1
enter data :
78

0:exit  1:insertion  2:deleteion  3:inOrder traverse
enter your choise:1
enter data :
18

0:exit  1:insertion  2:deleteion  3:inOrder traverse
enter your choise:3
tree : 11 18 23 34 56 78
0:exit  1:insertion  2:deleteion  3:inOrder traverse
enter your choise:1
enter data :
25

0:exit  1:insertion  2:deleteion  3:inOrder traverse
enter your choise:3
tree : 11 18 23 25 34 56 78
0:exit  1:insertion  2:deleteion  3:inOrder traverse
enter your choise:2
enter data :
23

0:exit  1:insertion  2:deleteion  3:inOrder traverse
enter your choise:3
tree : 11 18 25 34 56 78
0:exit  1:insertion  2:deleteion  3:inOrder traverse
enter your choise:2
enter data :
34

0:exit  1:insertion  2:deleteion  3:inOrder traverse
enter your choise:3
tree : 11 18 25 56 78
0:exit  1:insertion  2:deleteion  3:inOrder traverse
enter your choise:0
.....exit.....
```

7. Observations:

At the end of this lab, I was able to implement deletion function of tree and it’s most basic algorithms.

Rubrics

Demonstration	Absent	Student is unable to follow the provided instructions properly. The student can name the hardware or simulation platform, but unable to implement anything practically or on the software	Student can understand the provided laboratory instructions and familiar with the lab environment (Trainer/ software/ IDE), but cannot implement on the platform practically or on the software	Student has followed instructions to construct the fundamental schematic/ block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code, and have successfully executed the program/ run circuit on software platform	Student perfectly implemented a working model/ logic/ circuit/ block diagram/ code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

Laboratory Reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recorded along with detailed procedure	Appropriate computations or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all respects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor's Signature**		

LAB NO. 16

Graph

1. Objectives:

- Introduction to Graph
- Procedure and Implementation

2. Outline:

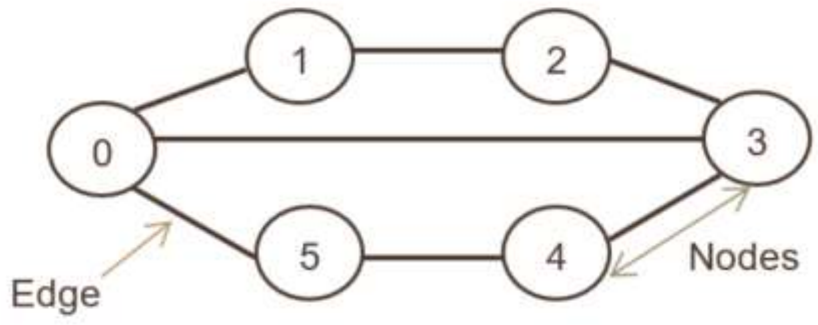
- Graph
- Practical Implementation
- Examples

3. Corresponding CLO and PLO:

- CLO 1, PLO 5

4. Background:

A Graph in the data structure can be termed as a data structure consisting of data that is stored among many groups of edges(paths) and vertices (nodes), which are interconnected. Graph data structure (N, E) is structured with a collection of Nodes and Edges. Both nodes and vertices need to be finite.



More precisely, a graph is a data structure (V, E) that consists of

- A collection of vertices V
- A collection of edges E, represented as ordered pairs of vertices (u,v)

Graph Terminology

- **Adjacency:**
  A vertex is said to be adjacent to another vertex if there is an edge connecting them. Vertices 2 and 3 are not adjacent because there is no edge between them.
- **Path:**
  A sequence of edges that allows you to go from vertex A to vertex B is called a path. 0-1, 1-2 and 0-2 are paths from vertex 0 to vertex 2.
- **Directed Graph:**

A graph in which an edge (u,v) doesn't necessarily mean that there is an edge (v, u) as well. The edges in such a graph are represented by arrows to show the direction of the edge.

Graph Representation:

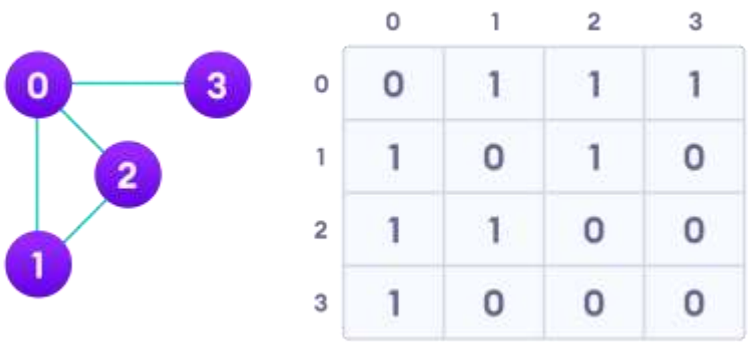
Graphs are commonly represented in two ways:

1. Adjacency Matrix:

An adjacency matrix is a 2D array of V x V vertices. Each row and column represent a vertex.

If the value of any element a[i][j] is 1, it represents that there is an edge connecting vertex i and vertex j.

The adjacency matrix for the graph we created above is



	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0

since it is an undirected graph, for edge (0,2), we also need to mark edge (2,0); making the adjacency matrix symmetric about the diagonal.

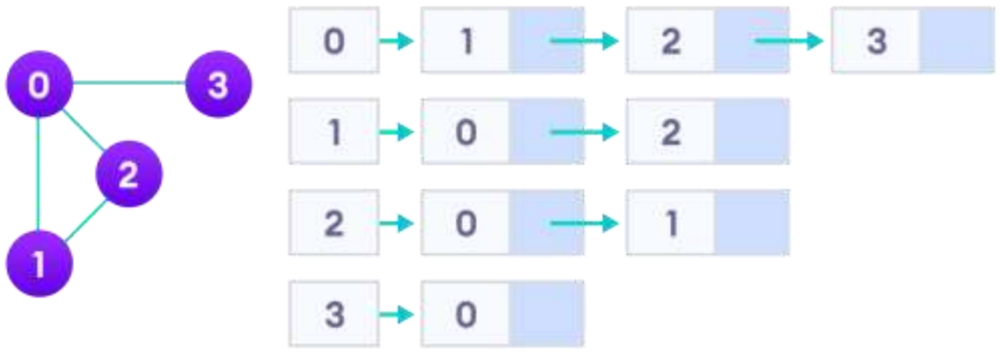
Edge lookup(checking if an edge exists between vertex A and vertex B) is extremely fast in adjacency matrix representation but we have to reserve space for every possible link between all vertices(V x V), so it requires more space.

2. Adjacency List:

An adjacency list represents a graph as an array of linked lists.

The index of the array represents a vertex and each element in its linked list represents the other vertices that form an edge with the vertex.

The adjacency list for the graph we made in the first example is as follows:



An adjacency list is efficient in terms of storage because we only need to store the values for the edges. For a graph with millions of vertices, this can mean a lot of saved space.

Graph Operations:

The most common graph operations are:

- Check if the element is present in the graph
- Graph Traversal
- Add elements(vertex, edges) to graph
- Finding the path from one vertex to another

5. Equipment:

6. Procedure:

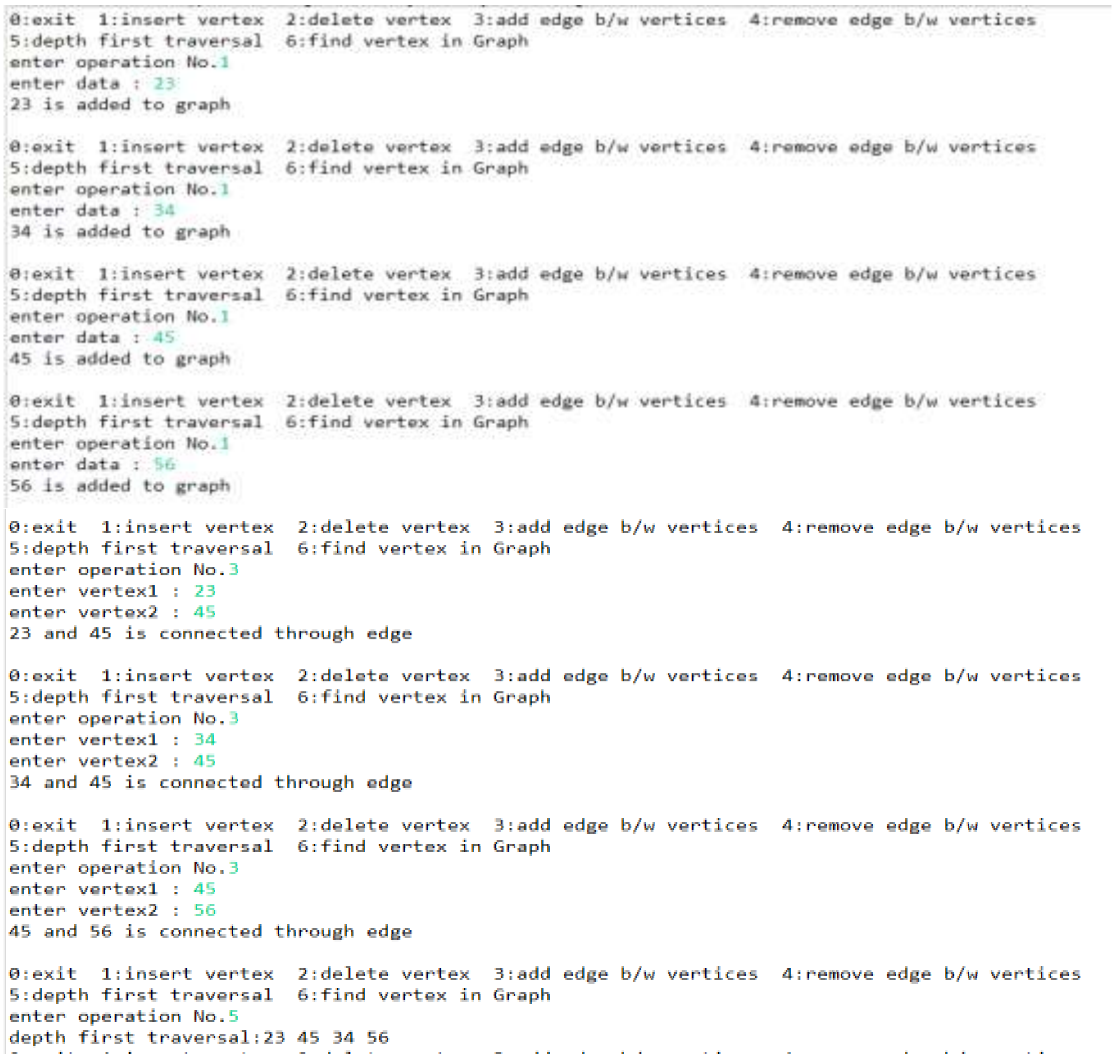
Code:

```
import java.util.Scanner;
import java.util.ArrayList;
import java.util.LinkedList;

public class Lab16 {

    public static void main(String args[]) {

        Graph g = new Graph();
        Scanner sc = new Scanner(System.in);
        boolean run = true;
        while (run) {
            System.out.print(
                    "0:exit  1:insert vertex  2:delete vertex  3:add edge b/w verti
ces  4:remove edge b/w vertices"
                            + "\n5:depth first traversal  6:find vertex in Graph  "
 + "\nenter operation No.");
            int ch = 0;
            ch = sc.nextInt();

            switch (ch) {
            case 0:

                run = false;
                System.out.println(".....exit.....");
                break;
            case 1:
                System.out.print("enter data : ");
                System.out.println(g.addVertex(sc.nextInt()));
                break;
            case 2:
                System.out.print("enter data : ");
                System.out.println(g.delVert(sc.nextInt()));
                break;
            case 3:
                System.out.print("enter vertex1 : ");
                int vert1 = sc.nextInt();
                System.out.print("enter vertex2 : ");
                System.out.println(g.addEdge(vert1, sc.nextInt()));
                break;
            case 4:
                System.out.print("enter vertex1 : ");
                int vert = sc.nextInt();
                System.out.print("enter vertex2 : ");
                System.out.println(g.removeEdge(vert, sc.nextInt()));

                break;
            case 5:
                System.out.print("depth first traversal:");
                g.dft();
                System.out.println("");
                break;
            case 6:
                System.out.print("enter data to find:");
                System.out.print(g.dfs(sc.nextInt()));
                System.out.println("");
                break;

            default:
                System.out.println("invalid option");
            }// end switch
        } // end while loop
    } // end void main()

} // end Lab class

class Vertex {
    public int data;
    public LinkedList<Vertex> adjacency;
    public boolean visited;

    public Vertex(int data) {
        visited = false;
        adjacency = new LinkedList<Vertex>();
        this.data = data;
    }// end Vertex constructor
}// end Vertex class

class Graph {
    private ArrayList<Vertex> vertices;
```

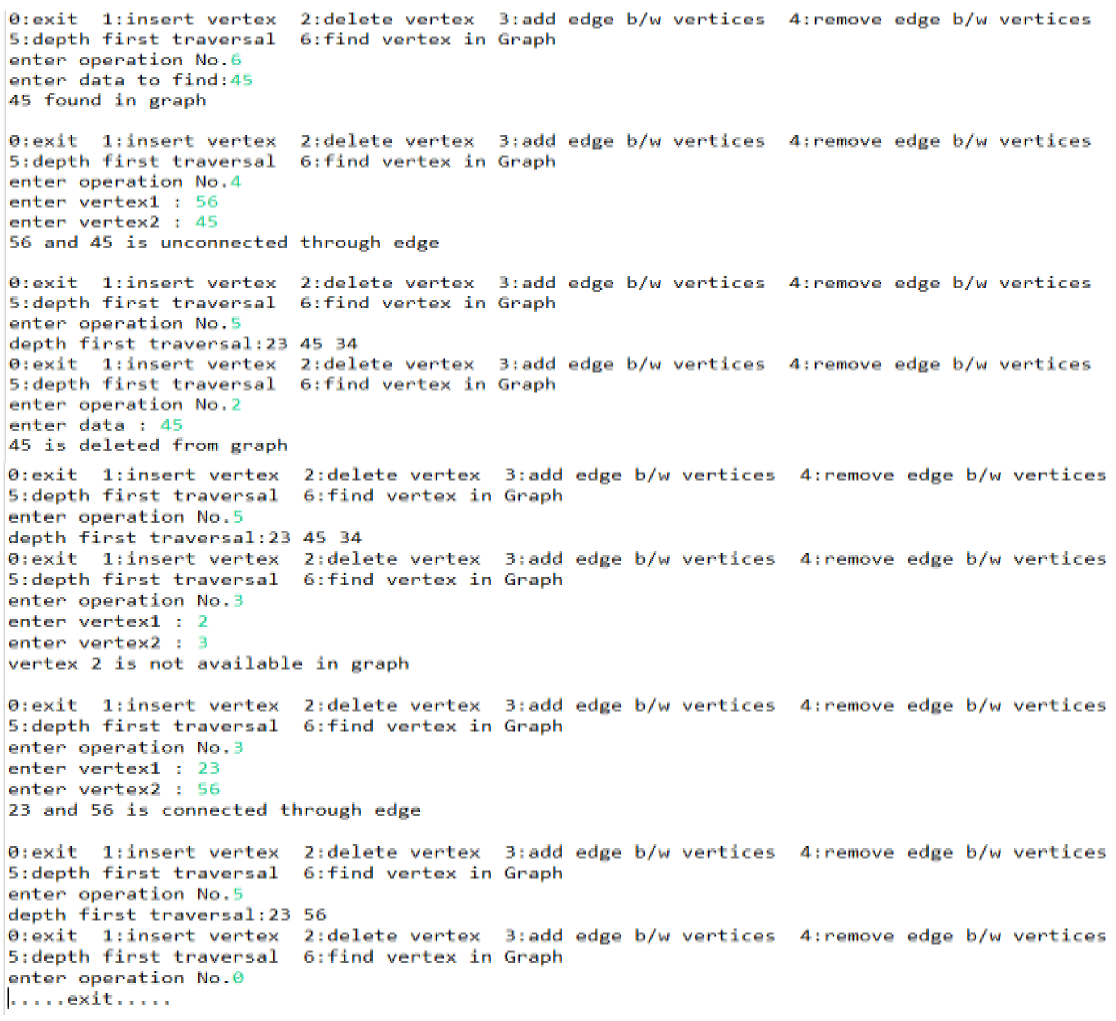
```java
public Graph() {
    vertices = new ArrayList<Vertex>();
}// end Graph constructor

public String addVertex(int v) {
    Vertex vert = new Vertex(v);
    for (int i = 0; i < vertices.size(); i++)
        if (vertices.get(i).data == vert.data)
            return "same vertex is not allowed  \n ";

    vertices.add(vert);
    return vert.data + " is added to graph   \n";
}// end addVertex()

public String delVert(Integer vert) {

    for (int i = 0; i < vertices.size(); i++) {

        if (vertices.get(i).data == vert) {
            vertices.remove(i);
            return vert + " is deleted from graph  \n ";
        } // end if

    } // end for
    return vert + " is not found in graph   \n";
}// end delVert()

public String addEdge(int u, int v) {
    int i = 0, j = 0;
    Vertex vertU = null, vertV = null;

    for (i = 0; i < vertices.size(); i++) {
        if (vertices.get(i).data == u) {
            vertU = vertices.get(i);
            break;
        }
    }

    for (j = 0; j < vertices.size(); j++) {
        if (vertices.get(j).data == v) {
            vertV = vertices.get(j);
            break;
        }
    }

    if (vertU != null && vertV != null) {
        vertU.adjacency.add(vertV);
```

```
            vertV.adjacency.add(vertU);
            return u + " and " + v + " is connected through edge   \n";
        } else if (vertU == null)
            return "vertex " + u + " is not available in graph   \n";
        else if (vertV == null)
            return "vertex " + v + " is not available in graph   \n";
        return "there is no edge between " + u + " and    " + v + "\n";

    }// end addEdge()

    public String removeEdge(int u, int v) {
        int i = 0, j = 0;
        Vertex vertU = null, vertV = null;
        for (i = 0; i < vertices.size(); i++) {
            if (vertices.get(i).data == u) {
                vertU = vertices.get(i);
                break;
            }
        }

        for (j = 0; j < vertices.size(); j++) {
            if (vertices.get(j).data == v) {
                vertV = vertices.get(j);
                break;
            }
        }

        if (vertU != null && vertV != null) {
            vertU.adjacency.remove(vertV);
            vertV.adjacency.remove(vertU);
            return u + " and " + v + " is unconnected through edge  \n";
        }
        if (vertU == null)
            return "vertex " + u + " is not available in graph  \n";
        if (vertV == null)
            return "vertex " + v + " is not available in graph  \n";
        return "there is no edge between " + u + " and    " + v + "\n";

    }// end removeEdge()

    public void dft() {
        Stack<Vertex> s = new Stack<Vertex>(10);

        Vertex v = vertices.get(0);
        System.out.print(v.data + " ");
        v.visited = true;
        s.push(v);
        while (!s.isEmpty()) {
            v = getUnVisitedAdjacency(s.peak());
            if (v == null)
                s.pop();
            else {
                System.out.print(v.data + " ");
                v.visited = true;
                s.push(v);
            }

        }
        for (int i = 0; i < vertices.size(); i++)
            vertices.get(i).visited = false;

    }// end dft()

    public String dfs(int item) {
        Stack<Vertex> s = new Stack<Vertex>(10);

        Vertex v = vertices.get(0);
        v.visited = true;
        if (v.data == item)
            return item + " found in graph\n";
        s.push(v);
        while (!s.isEmpty()) {
            v = getUnVisitedAdjacency(s.peak());
            if (v == null)
                s.pop();
            else {
                if (v.data == item)
                    return item + " found in graph\n";
                v.visited = true;
                s.push(v);
            }

        }
        for (int i = 0; i < vertices.size(); i++)
            vertices.get(i).visited = false;

        return item + " didn't find in graph\n";
    }// end dfs()

    public Vertex getUnVisitedAdjacency(Vertex v) {
        for (int i = 0; i < v.adjacency.size(); i++) {
            if (v.adjacency.get(i).visited == false)
                return v.adjacency.get(i);
        }
        return null;
    }// end getUnVisitedAdjacency()

}// end Graph

class Stack<T> {
    private T arr[];
    private int top;

    public Stack(int size) {
        arr = (T[]) new Object[size];
        top = -1;
    }

    public void push(T item) {
        if (top == arr.length)
            return;
        arr[++top] = item;
    }

    public T pop() {
        if (top == -1)
            return null;
        return arr[top--];
    }

    public T peak() {
        if (top == -1)
            return null;

        return arr[top];
    }

    public boolean isEmpty() {
        return top == -1;
    }
}
```

Output:



```
0:exit  1:insert vertex  2:delete vertex  3:add edge b/w vertices  4:remove edge b/w vertices
5:depth first traversal  6:find vertex in Graph
enter operation No.1
enter data : 23
23 is added to graph

0:exit  1:insert vertex  2:delete vertex  3:add edge b/w vertices  4:remove edge b/w vertices
5:depth first traversal  6:find vertex in Graph
enter operation No.1
enter data : 34
34 is added to graph

0:exit  1:insert vertex  2:delete vertex  3:add edge b/w vertices  4:remove edge b/w vertices
5:depth first traversal  6:find vertex in Graph
enter operation No.1
enter data : 45
45 is added to graph

0:exit  1:insert vertex  2:delete vertex  3:add edge b/w vertices  4:remove edge b/w vertices
5:depth first traversal  6:find vertex in Graph
enter operation No.1
enter data : 56
56 is added to graph

0:exit  1:insert vertex  2:delete vertex  3:add edge b/w vertices  4:remove edge b/w vertices
5:depth first traversal  6:find vertex in Graph
enter operation No.3
enter vertex1 : 23
enter vertex2 : 45
23 and 45 is connected through edge

0:exit  1:insert vertex  2:delete vertex  3:add edge b/w vertices  4:remove edge b/w vertices
5:depth first traversal  6:find vertex in Graph
enter operation No.3
enter vertex1 : 34
enter vertex2 : 45
34 and 45 is connected through edge

0:exit  1:insert vertex  2:delete vertex  3:add edge b/w vertices  4:remove edge b/w vertices
5:depth first traversal  6:find vertex in Graph
enter operation No.3
enter vertex1 : 45
enter vertex2 : 56
45 and 56 is connected through edge

0:exit  1:insert vertex  2:delete vertex  3:add edge b/w vertices  4:remove edge b/w vertices
5:depth first traversal  6:find vertex in Graph
enter operation No.5
depth first traversal:23 45 34 56
```



```
0:exit  1:insert vertex  2:delete vertex  3:add edge b/w vertices  4:remove edge b/w vertices
5:depth first traversal  6:find vertex in Graph
enter operation No.6
enter data to find:45
45 found in graph

0:exit  1:insert vertex  2:delete vertex  3:add edge b/w vertices  4:remove edge b/w vertices
5:depth first traversal  6:find vertex in Graph
enter operation No.4
enter vertex1 : 56
enter vertex2 : 45
56 and 45 is unconnected through edge

0:exit  1:insert vertex  2:delete vertex  3:add edge b/w vertices  4:remove edge b/w vertices
5:depth first traversal  6:find vertex in Graph
enter operation No.5
depth first traversal:23 45 34
0:exit  1:insert vertex  2:delete vertex  3:add edge b/w vertices  4:remove edge b/w vertices
5:depth first traversal  6:find vertex in Graph
enter operation No.2
enter data : 45
45 is deleted from graph
0:exit  1:insert vertex  2:delete vertex  3:add edge b/w vertices  4:remove edge b/w vertices
5:depth first traversal  6:find vertex in Graph
enter operation No.5
depth first traversal:23 45 34
0:exit  1:insert vertex  2:delete vertex  3:add edge b/w vertices  4:remove edge b/w vertices
5:depth first traversal  6:find vertex in Graph
enter operation No.3
enter vertex1 : 2
enter vertex2 : 3
vertex 2 is not available in graph

0:exit  1:insert vertex  2:delete vertex  3:add edge b/w vertices  4:remove edge b/w vertices
5:depth first traversal  6:find vertex in Graph
enter operation No.3
enter vertex1 : 23
enter vertex2 : 56
23 and 56 is connected through edge

0:exit  1:insert vertex  2:delete vertex  3:add edge b/w vertices  4:remove edge b/w vertices
5:depth first traversal  6:find vertex in Graph
enter operation No.5
depth first traversal:23 56
0:exit  1:insert vertex  2:delete vertex  3:add edge b/w vertices  4:remove edge b/w vertices
5:depth first traversal  6:find vertex in Graph
enter operation No.0
.....exit.....
```

7. Observations:

At the end of this lab, I was able to implement Graph and it’s most basic algorithms.

Rubrics

Demonstration	Absent	Student is unable to follow the provided instructions properly. The student can name the hardware or simulation platform, but unable to implement anything practically or on the software	Student can understand the provided laboratory instructions and familiar with the lab environment (Trainer/ software/ IDE), but cannot implement on the platform practically or on the software	Student has followed instructions to construct the fundamental schematic/ block diagram/ code/ model on the protoboard/ trainer/ simulation software.	Student has constructed the functional/ working schematic/ model/ block diagram/ code, and have successfully executed the program/ run circuit on software platform	Student perfectly implemented a working model/ logic/ circuit/ block diagram/ code and successfully executed the lab objective in Realtime or in a simulation environment and produced the desired results
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor’s Signature**		

Laboratory Reports	Report not submitted	Plagiarized content presented or incomplete submission	Requirements are listed and experimental procedure is presented	Observations are recorded along with detailed procedure	Appropriate computations or numerical analysis is performed	Correctly drawn conclusion with exact results and complete report in all respects
Category	**Ungraded**	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Percentage	[0]	[1-20]	[21-40]	[41-60]	[61-80]	[81-100]
Marks	0.0	0.01 - 0.20	0.21 - 0.40	0.41 - 0.60	0.61 - 0.80	0.81 - 1.0
Date		**Total Marks**		**Instructor’s Signature**		