

|  |  |
| --- | --- |
| **Data Structures** | |
| Lab Manual | |
| **Department of Computer Science and Engineering**  **The NorthCap University, Gurugram** | |
|  |

**Data Structures**

**Lab Manual**

**CSL 209**

**Dr. Anuradha Dhull**



Department of Computer Science and Engineering

NorthCap University, Gurugram- 122001, India

Session 2021-22

*Published by:*

**School of Engineering and Technology**

**Department of Computer Science & Engineering**

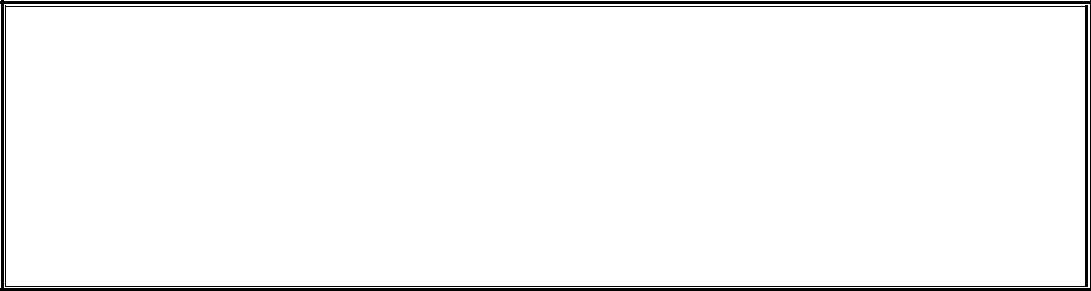
**The NorthCap University Gurugram**

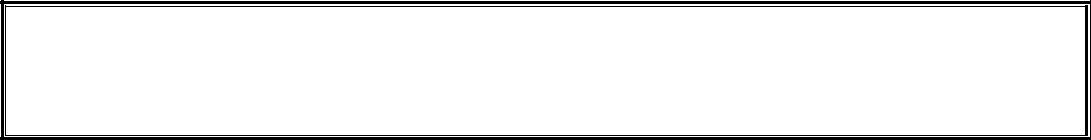
**• Laboratory Manual is for Internal Circulation only**

**© Copyright Reserved**

*No part of this Practical Record Book may be*

*reproduced, used, stored without prior permission of The NorthCap University*



Copying or facilitating copying of lab work comes under cheating and is considered as use of unfair means. Students indulging in copying or facilitating copying shall be awarded zero marks for that particular experiment. Frequent cases of copying may lead to disciplinary action. Attendance in lab classes is mandatory.

Labs are open up to 7 PM upon request. Students are encouraged to make full use of labs beyond normal lab hours.

**PREFACE**

Data Structures Lab Manual is designed to meet the course and program requirements of NCU curriculum for B.Tech II year students of CSE branch. The concept of the lab work is to give brief practical experience for basic lab skills to students. It provides the space and scope for self-study so that students can come up with new and creative ideas.

The Lab manual is written on the basis of “teach yourself pattern” and expected that students who come with proper preparation should be able to perform the experiments without any difficulty. Brief introduction to each experiment with information about self-study material is provided. The laboratory exercises will include familiarization with linear and non-linear data structures starting with the basics of Arrays, Linked Lists to advanced data structures like Trees & Graphs along with their implementation in Java programming language that covers all the basic operations. Students would attain the practical understanding of different kinds of data structures along with their implementation and operations performed. Finally, the students would require applying the practical concepts of data structures learned in solving real time problems that involves their basic usage and applicability. At the start of each experiment a question bank for preparation and practice is suggested which may be used to test the basic understanding of the students about the experiment. Students are expected to come thoroughly prepared for the lab. General disciplines, safety guidelines and report writing are also discussed.

The lab manual is a part of curriculum for the TheNorthCap University, Gurugram. Teacher’s copy of the experimental results and answer for the questions are available as sample guidelines.

We hope that lab manual would be useful to students of CSE, IT, ECE and BSc branches and author requests the readers to kindly forward their suggestions / constructive criticism for further improvement of the work book.

Author expresses deep gratitude to Members, Governing Body-NCU for encouragement and motivation.

**Authors**

**The NorthCap University**

**Gurugram, India**

|  |  |  |  |
| --- | --- | --- | --- |
| **CONTENTS** | | | |
| S.N. | **Details** | Page No. | |
|  | **Syllabus** |  | |
| 1 | **Introduction** |  | |
| 2 | **Lab Requirement** |  |
| 3 | **General Instructions** |  |
| 4 | **List of Experiments** |  |
| 5 | **List of Flip Assignment** |  |
| 6 | **List of Projects** |  |
| 7 | **Rubrics** |  |
| 8 | **Annexure 1 (Format of Lab Report)** |  |
| 9 | **Annexure 2 (Format of Project Report)** |  |

**COURSE TEMPLATE**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1. **Department:** | | **Department of Computer Science and Engineering** | | | | | |
| 1. **Course Name: Data Structures** | | | | 1. **Course Code** | 1. **L-T-P** | | 1. **Credits** |
| CSL209 | 3-0-2 | | 4 |
| 1. **Type of Course (Check one):** | | Programme Core **✔** Programme Elective Open Elective | | | | | |
| 1. **Pre-requisite(s), if any: FOCP-II** | | | | | | | |
| 1. **Frequency of offering (check one):** Odd **✔** Even Either semester Every semester | | | | | | | |
| 1. **Brief Syllabus:**   Solving computational problems requires the knowledge of efficient data organization and the ability to make effective choices among multiple solutions. In this course, we will explore several fundamental data structures in computer science and learn to implement them in Java. The course aims to teach the fundamentals of data structures, their design, implementation and effective use in problem solving approach. With the knowledge of data structures and practical experience in implementing them, students can become much more effective designer and developer. The course will start with the basic introduction of linear as well as non-linear data structures and further proceeds with the programming intensive task of implementing them. This course will also cover file organization and different hashing techniques in its last module. | | | | | | | |
| **Total lecture, Tutorial and Practical Hours for this course (Take 15 teaching weeks per semester): 75** | | | | | | | |
| **Lectures: 45 hours** | | | **Practice** | | | | |
| **Tutorials: 0 hours** | | | **Lab Work: 30 hours** | |
| 1. **Course Outcomes (COs)**   Possible usefulness of this course after its completion i.e. how this course will be practically useful to him once it is completed | | | | | | | |
| **CO 1** | Infer the role of various data structures and apply complexity analysis on different algorithms. | | | | | | |
| **CO 2** | Differentiate between Arrays and Linked Lists, and implement numerous operations using Java. | | | | | | |
| **CO 3** | Demonstrate and contrast stack and Queue operations in real world applications. | | | | | | |
| **CO 4** | Describe Non-linear data structures (Trees and Graphs) and their applications in various domains. | | | | | | |
| **CO 5** | Perform and compare several searching and sorting algorithms, and develop basic understanding about File organization. | | | | | | |
| 1. **UNIT WISE DETAILS No. of Units: 5** | | | | | | | |
| **Unit Number: 1 Title: Object Oriented Programming and Data Structure No. of hours:9**  **Content Summary:**  Java Software Development Tools, Interfaces, Data & its meaning, Types of data structures, Linear and Non-linear data structures, Introduction to Algorithms and analyzing time complexities | | | | | | | |
| **Unit Number: 2 Title: Arrays and Linked Lists No. of hours: 9**  **Content Summary:**  Introduction to Arrays, Representation in Row major and Column major order, Array traversal, Wrapper Classes, The ArrayList, Linear and Binary search, Linked list introduction, Insertion and Deletion, Doubly Linked list, Circular Linked list and their implementation | | | | | | | |
| **Unit Number: 3 Title: Stacks & Queues No. of hours: 9**  **Content Summary:**  Stack operations Push, pop, peek, Implementation using arrays and linked list, Stack applications, Polish Notations, Introduction to Queues, Implementation using Arrays and Linked Lists, Circular Queues, Doubly ended queues (Dqueues), Priority Queues applications and implementation | | | | | | | |
| **Unit Number: 4 Title: Trees & Graphs No. of hours: 12**  **Content Summary:**  Introduction to Binary Trees, Expression trees, Tree traversal: Inorder, Preorder & Postorder, Tree implementation using Array and Linked List, Introduction to Binary Search Tree, Insertion and Deletion in BST, AVL Trees, Insertion and Deletion, Introduction to Graphs, Representation, Graph traversal: Breadth First Search & Depth First Search, Applications | | | | | | | |
| **Unit Number: 5 Title: Sorting Algorithms & File Organization No. of hours: 6**  **Content Summary:**  Sorting introduction, Quick Sort, Merge Sort and Heap Sort Implementation and analysis, File Organization: File representation on hard disks, Indexing and Hashing techniques, Linear probing & Quadratic probing | | | | | | | |
| 1. **Brief Description of Self-learning components by students (through books/resource material etc.):**   Object-Oriented Design with UML  GUI Programming | | | | | | | |
| **Books Recommended :**  **Text Books:**   1. Robert Lafore, “Data Structures and Algorithms in Java”, SAMS publications, 2nd Edition, 2002 2. Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser, “Data Structures and Algorithms in Java”, Wiley Publications, 6th Edition, 2014   **Reference Books:**   1. Seymour Lipschutz, “Data Structures”, Schaum’s Outlines, McGraw Hill Education, 1st Edition, 2014 2. Aaron M. Tannenbaum, Yedidyah Langsam, Moshe J.Augenstein, “Data Structure using C”, Pearson India, 1st Edition, 2018   **Electronic Textbooks available through the ASU Library**:   |  | | --- | | **JAVA For Everyone: Late Objects, 2nd Edition, Cay S. Horstmann.** |   (https://bcs.wiley.com/he-bcs/Books?action=index&bcsId=6907&itemId=1118063317)  **Reference Websites: (nptel, swayam, coursera, edx, udemy, lms, official documentation weblink)**  <https://canvas.asu.edu/courses/58080>  <https://lms.ncuindia.edu/course/view.php?id=323>  <https://nptel.ac.in/courses/106102064>  <https://www.coursera.org/learn/data-structures?specialization=data-structures-algorithms>  <https://www.edx.org/course/data-structures-an-active-learning-approach> | | | | | | | |

**Practice (Case Studies/Practice Drills) Content**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Topic** | **Unit covered** |
|  | Applications of different data structures in real world applications | 1 |
|  | Time complexities of algorithms | 1 |
|  | Conversion of infix to postfix and prefix, postfix to prefix and infix, prefix to postfix and infix | 2 |
|  | Priority Queue implementation using Heap data structure | 2 |

**Practical Content**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No.** | **Title of the Experiment** | **Software/Hardware based** | **Unit covered** | **Time Required** |
|  | (i) Write a java program to make an array of specified length asking from the user.  (ii) Write a java program to write various operations on arrays such as finding elements in an array, find duplicate values in an array, sort an array etc.  (iii) Write a java program to write linear and binary search using arrays. | **Software** | **1** | 4 |
|  | (i) Write a java program to create a singly linked list of specified length asking from the user.  (ii) Write a java program to write various operations on linked lists such as add an element in a linked list, remove or sort a linked list etc.  (iii) Write a java program to create a circular linked list and perform various operations on it.  (iv) Write a java program to create a doubly linked list and perform various operations on it.  (v) Write a java program to create a circular doubly linked list and perform various operations on it. | **Software** | **2** | 8 |
|  | (i) Write a java program to represent stack using arrays and perform operations such as pop, push, peek etc.  (ii) Write a java program to represent stack using linked list perform operations such as pop, push, peek etc. | **Software** | **3** | 4 |
|  | (i) Write a java program to represent a queue using arrays and perform operations such as add, remove, search in queue using front, rear pointers etc.  (ii) Write a java program to represent queue using linked list perform operations such as add, remove, search in queue using front and rear pointers etc.  (iii) Write a java program to implement stack using queue.  (iv) Write a java program to implement queue using stack. | **Software** | **3** | 4 |
|  | (i) Write a java program to create a binary search tree using arrays and perform add, remove, search operations in tree.  (ii) Write a java program to create a binary search tree using linked lists and perform add, remove, search operations in tree.  (iii) Write a java program to create binary tree for inorder traversal using arrays/linked list.  (iv) Write a java program to create binary tree for postorder traversal using arrays/linked list.  (v) Write a java program to create binary tree for preorder traversal using arrays/linked list. | **Software** | **4** | 4 |
|  | (i) Write a java program to implement breadth-first search using arrays/linked list.  (ii) Write a java program to implement depth-first search using arrays/linked list. | **Software** | **4** | 2 |
|  | Write a java program to implement bubble sort. | **Software** | **5** | 1 |
|  | Write a java program to implement merge sort. | **Software** | **5** | 1 |
|  | Write a java program to implement quick sort. | **Software** | **5** | 1 |
|  | Write a java program to implement heap sort. | **Software** | **5** | 1 |
| **Value Added Experiments** | | | | |
|  | Tower of Hanoi | **Software** | **2** | 1 |
|  | Stack implementation using Queue | **Software** | **2** | 1 |
|  | Queue implementation using Stack | **Software** | **2** | 1 |
|  | Merge sort implementation using Linked List | **Software** | **5** | 1 |
|  | Dijkstra’s Algorithm to find shortest path | **Software** | **5** | 1 |

|  |
| --- |
| **Project (To be done as individual/in group): YES** |

**Evaluation Scheme (Choose one related to the course)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S. No.** | **TYPE OF COURSE** | **PARTICULAR** | **ALLOTTED RANGE OF MARKS** | **PASS CRITERIA** |
| 1 | Theory+ Practical  (L-T-P/L-0-P) | Minor Test | 15% | Must Secure 30% Marks Out of Combined Marks of Major Test Plus Minor Test with Overall 40% Marks in Total. |
| Major Test | 35% |
| Continuous Evaluation Through Class Tests/Practice/Assignments/Presentation/Quiz | 10% |
| Online Quiz | 5% |
| Lab Work | 35% |

**Mapping of PO’s and CO’s**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO10** | **PO11** | **PO12** | **PSO1** | **PSO2** | **PSO3** |
| **CO1** | **3** | **3** | **1** | **2** | **-** | **1** | **-** | **2** | **3** | **3** | **2** | **3** | **3** | **2** | **3** |
| **CO2** | **2** | **2** | **2** | **2** | **3** | **1** | **1** | **2** | **3** | **3** | **2** | **3** | **3** | **2** | **3** |
| **CO3** | **2** | **2** | **2** | **2** | **3** | **1** | **1** | **2** | **3** | **3** | **2** | **3** | **3** | **2** | **3** |
| **CO4** | **2** | **2** | **2** | **2** | **3** | **1** | **1** | **2** | **3** | **3** | **2** | **3** | **3** | **2** | **3** |
| **CO5** | **2** | **2** | **1** | **2** | **3** | **1** | **1** | **2** | **3** | **3** | **2** | **3** | **3** | **2** | **3** |

1. **INTRODUCTION**

That ‘learning is a continuous process’ cannot be over emphasized. The theoretical knowledge gained during lecture sessions need to be strengthened through practical experimentation. Thus, practical makes an integral part of a learning process.

The purpose of conducting experiments can be stated as follows:

* To understand and implement various data structures like Arrays, Linked Lists, Stack, Queues, Tree and Graphs
* Reporting and analysing their complexities.
* To illustrate the basic functionality and operations performed
* To use sound development principles to implement various non-linear data structures of varying complexities.
* Hands on experience on the experimental setup and software tools

1. **LAB REQUIREMENTS**

|  |  |
| --- | --- |
| **Requirements** | **Details** |
| **Software Requirements** | Any programming editor (Sublime text/Notepad++) |
| **Operating System** | Any Operating System |
| **Hardware Requirements** | Windows and Linux:  8 GB RAM (Recommended)  80 GB hard disk space |
| **Required Bandwidth** | NA |

1. **GENERAL INSTRUCTIONS** 
   1. **General discipline in the lab**
   * Students must turn up in time and contact concerned faculty for the experiment they are supposed to perform.
   * Students will not be allowed to enter late in the lab.
   * Students will not leave the class till the period is over.
   * Students should come prepared for their experiment.
   * Experimental results should be entered in the lab report format and certified/signed by concerned faculty/ lab Instructor.
   * Students must get the connection of the hardware setup verified before switching on the power supply.
   * Students should maintain silence while performing the experiments. If any necessity arises for discussion amongst them, they should discuss with a very low pitch without disturbing the adjacent groups.
   * Violating the above code of conduct may attract disciplinary action.
   * Damaging lab equipment or removing any component from the lab may invite penalties and strict disciplinary action.
   1. **Attendance**

* Attendance in the lab class is compulsory.
* Students should not attend a different lab group/section other than the one assigned at the beginning of the session.
* On account of illness or some family problems, if a student misses his/her lab classes, he/she may be assigned a different group to make up the losses in consultation with the concerned faculty / lab instructor. Or he/she may work in the lab during spare/extra hours to complete the experiment. No attendance will be granted for such case**.**
  1. **Preparation and Performance**
* Students should come to the lab thoroughly prepared on the experiments they are assigned to perform on that day. Brief introduction to each experiment with information about self-study reference is provided on LMS.
* Students must bring the lab report during each practical class with written records of the last experiments performed complete in all respect.
* Each student is required to write a complete report of the experiment he has performed and bring to lab class for evaluation in the next working lab. Sufficient space in work book is provided for independent writing of theory, observation, calculation and conclusion.
* Students should follow the Zero tolerance policy for copying / plagiarism. Zero marks will be awarded if found copied. If caught further, it will lead to disciplinary action.
* Refer **Annexure 1** for Lab Report Format.

1. **LIST OF EXPERIMENTS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr. No.** | **Title of the Experiment** | **Software used** | **Unit covered** | **CO**  **Covered** | **Time Required** |
| **1** | Create an array of integer with size n. Return the difference between the largest and the smallest value inside that array. | Sublime Text/Notepad++ | 2 | CO1 | * + - 1. hrs |
| 2 | 1. Write a program that initializes an array with ten random integers and then prints four lines of output, containing: 2. Every element at an even index 3. Every odd element 4. All elements in reverse order 5. Only the first and last element | Sublime Text/Notepad++ | 2 | CO2 |  |
| 3 | 1. Write a program to read numbers in an integer array of size 5 and display the following:  * Sum of all the elements * Sum of alternate elements in the array * Second highest element in the array | Sublime Text/Notepad++ | 2 | CO1 |  |
| 4 | Write a program to create a singly linked list of n nodes and perform:  • Insertion   * At the beginning * At the end * At a specific location   • Deletion   * At the beginning * At the end * At a specific location | Sublime Text/Notepad++ | 1 | CO1 |  |
| 5 | Write a program to create a doubly linked list of n nodes and perform:  • Insertion   * At the beginning * At the end * At a specific location   • Deletion   * At the beginning * At the end  1. At a specific location | Sublime Text/Notepad++ | 1 | CO2 |  |
| 6 | Write a program to create a circular linked list of n nodes and perform:  • Insertion   * At the beginning * At the end * At a specific location   • Deletion   * At the beginning * At the end   At a specific location | Sublime Text/Notepad++ | 1 | CO2 |  |
| 7 | Write a program to implement stack using arrays and linked lists. | Sublime Text/Notepad++ | 2 | CO3 |  |
| 8 | Write a program to reverse a sentence using stack. | Sublime Text/Notepad++ | 2 | CO3 |  |
| 9 | Write a program to check for balanced parenthesis in a given expression. | Sublime Text/Notepad++ | 2 | CO3 |  |
| 10 | Write a program to convert infix expression to prefix and postfix expression. | Sublime Text/Notepad++ | 2 | CO3 |  |
| 11 | Write a program to implement Tower of Hanoi using stacks | Sublime Text/Notepad++ | 3 | CO3 |  |
| 12 | Write a program to implement Linear Queue using Array and Linked Lists. | Sublime Text/Notepad++ | 3 | CO3 |  |
| 13 | Write a program to implement Circular Queue using Array and Linked Lists. | Sublime Text/Notepad++ | 4 | CO4 |  |
| 14 | Write a program to implement Doubly Ended Queue using Array and Linked Lists. | Sublime Text/Notepad++ | 4 | CO4 |  |
| 15 | Write a Program to implement Binary Search Tree operations. | Sublime Text/Notepad++ | 4 | CO4 |  |
| 16 | Write a program to implement Bubble Sort, Selection Sort, Quick Sort, Merge Sort and Insertion Sort algorithm. | Sublime Text/Notepad++ | 4 | CO6 |  |

1. **LIST OF FLIP EXPERIMENTS**
2. Write a Java program to swap the first and last elements of an array (length must be at least 5) and create a new array.
3. Write a Java program to segregate all 0s on left side and all 1s on right side of a given array of 0s and 1s.
4. Write a program to rearrange a given list such that it consists of alternating minimum maximum elements
5. Write a program for calculating the frequency of different values in a linked list of integers with multiple entries.
6. Write a program in java to Check if stack elements are pairwise consecutive or not.
7. Write a program to reverse the first m elements from the queue, where m < n, n is the total number of elements in queue.
8. Write a Java program to construct a binary tree using the given In-Order and Pre-Order traversal
   1. Example:
   2. Input :
      1. in[] = {4, 8, 2, 5, 1, 6, 3, 7}
      2. pre[] = {1, 2, 4, 8, 5, 3, 6, 7}
9. Write a program to implement Dijkstra's to calculate shortest path in graphs.
10. **RUBRICS**

|  |  |
| --- | --- |
| **Marks Distribution** | |
| **Continuous Evaluation(50 Marks)** | **End Semester Exam (20 Marks)** |
| Each experiment shall be evaluated for 10 marks and at the end of the semester proportional marks shall be awarded out of 50. | End semester practical evaluation including Mini project (if any) carries 20 marks. |
| Following is the breakup of 10 marks for each  **4 Marks**: Observation & conduct of experiment. Teacher may ask questions about experiment.  **3 Marks:** For report writing  **3 Marks:** For the 15 minutes quiz to be conducted in every lab. |

**Annexure 1**

**(Student Lab Report)**

**Data Structures**

**(CSL 209)**

**Lab Practical Report**



Faculty name: **Amandeep Kaur** Student name: **Piyush Gambhir**

Roll No.: **21CSU349**

Semester: **III**

Group: **AIML-B**

**Department of Computer Science and Engineering**

**The NorthCap University, Gurugram- 122001, India**

**Session 2022-23**

**INDEX**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **S.No** | **Experiment** | **Page No.** | **Date of Experiment** | **Date of Submission** | **Marks** | **CO Covered** | **Sign** |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

**EXPERIMENT NO. 1**

|  |
| --- |
| **Student Name and Roll Number:** Piyush Gambhir |
| **Semester /Section:** Semester-III – AIML-B |
| **Link to Code:** [Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III (github.com)](https://github.com/Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III) |
| **Date:** |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective(s):**  To familiarize the students with linear data structure array and its basic operations |
| **Outcome:**  The students will be able to implement and use arrays for solving various problems |
| **Problem Statement:**  Create an array of integer with size n. Return the difference between the largest and the smallest value inside that array. |
| **Background Study:**  An Array is a data structure consisting of a collection of elements (values or variables), each identified by at least one array index or key. An array is stored such that the position of each element can be computed from its index tuple by a mathematical formula. The simplest type of data structure is a linear array, also called one-dimensional array. |
| **Algorithm (Student Work Area):**   1. Create a variable mini/maxi and initialize it with the value at index zero of the array. 2. Iterate over the array and compare if the current element is greater than the maxi or less than the mini. 3. Update the mini/maxi element with the current element so that the minimum/maximum element is stored in the mini/maxi variable. 4. Return the maxi – mini. |
| **Code (Student Work Area):**  /\*   \* Problem Statement:   \* Create an array of integer with size n. Return the difference between the largest and the smallest value inside that array.   \*/  import java.util.Scanner;  /\*\*   \* problem\_statement   \*/  public class problem\_statement\_experiment\_1 {      // Method enterArrayElements(int[] arr)      // Method for taking input for array elements      public static void enterArrayElements(int[] arr) {          Scanner sc = new Scanner(System.in);          System.out.println("""                  ====================================                  Enter Array Elements                  ====================================                  """);          for (int i = 0; i < arr.length; i++) {              System.out.print("Enter the Value of Array Element " + (i + 1) + ": ");              int temp = sc.nextInt();              arr[i] = temp;          }          sc.close();      }      // Method printArrayElements(int[] arr)      // Method for printing array elements      public static void printArrayElements(int[] arr) {          System.out.print("""                  ====================================                  Printing Array Elements                  ====================================                  """);          for (int i = 0; i < arr.length; i++) {              System.out.println("Value of Array Element " + (i + 1) + ": " + arr[i]);          }      }      // Method getLargestElement(int[] arr)      // Method to get largest value in the array      public static int getLargestElemnt(int[] arr) {          int largestElement = 0;          for (int i = 0; i < arr.length; i++) {              if (arr[i] > largestElement) {                  largestElement = arr[i];              }          }          return largestElement;      }      // Method getSmallestElement(int[] arr)      // Method to get smallest value in the array      public static int getSmallestElemnt(int[] arr) {          int smallestElement = 0;          for (int i = 0; i < arr.length; i++) {              if (arr[i] < smallestElement) {                  smallestElement = arr[i];              }          }          return smallestElement;      }      public static void main(String[] args) {          Scanner sc = new Scanner(System.in);          System.out.print("Enter the Size of The Array: ");          int n = sc.nextInt();          int[] arr = new int[n];          System.out.println("\n");          enterArrayElements(arr);          System.out.println("\n");          printArrayElements(arr);          System.out.println("\n");          System.out.println("Largest Value in the Array = " + getLargestElemnt(arr));          System.out.println("Smallest Value in the Array = " + getSmallestElemnt(arr));          System.out.println("\n");          System.out.println("The Difference Between Largest and Smallest Value in the Array = "                  + (getLargestElemnt(arr) - getSmallestElemnt(arr)));      }  } |
| **Output – Screenshots (Student Work Area):** |
| **Question Bank:**   1. What is Data Structure?   Data structures is a particular way of storing data in a computer so that it can be used efficiently.  For Example: We can store a list of items of same data type in array data structure.   1. Why is Array called as Linear Data Structure?   Arrays are called linear data structure because data in array is store in sequential order and each element is connected to its previous and next element.   1. What type of Indexing is used in Java?   Substring Index - Substring indexes are used for searches on three-character groups. Three-character groups are stored in the index.  Indexing in java starts from 0 and goes till n-1.   1. How to find the missing number in integer array of 1 to 100? 2. Create a temp array temp[] of size n + 1 with all initial values as 0. 3. Traverse the input array arr[], and do following for each arr[i] 4. if(temp[arr[i]] == 0) temp[arr[i]] = 1 5. Traverse temp[] and output the array element having value as 0 (This is the missing element). 6. How to find the second-highest value in a numeric array. 7. START 8. Initialize maxone=0, maxtwo=0,len,dif,i,j 9. Find the length of the array 10. Run a loop i=0 till i<len with i=i+1 11. Check if(maxone<array[i]) 12. And set maxtwo=maxone;maxone=array[i] 13. Check Else-if(maxtwo<Array[i]) 14. And set maxtwo=array[i] 15. Dif=maxone-maxtwo 16. Print maxone,maxtwo and dif 17. END 18. How to swap the first and last elements of an array. 19. START 20. Initialize array num[]={20,30,40} and x,len as variable 21. Find the length of the array (len) 22. Run loop till i=0;i<len;i=i+1 23. Print num[i] 24. Print (“Before swap”) 25. Set x= num[0] And num[0]=num[len-1] And num[len-1]=x 26. Run loop till i=0;i<len;i=i+1 27. Print num[i] 28. Print (“After Swap”) 29. END 30. Write a Java Program to check if see if Array contains a specific value. (Linear Search)   public class question\_bank\_8 {      public static void main(String[] args) {          int[] arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };          int searchValue = 5;          boolean isFound = false;          for (int i = 0; i < arr.length; i++) {              if (arr[i] == searchValue) {                  isFound = true;                  break;              }          }          System.out.println("Is " + searchValue + " Found in Array: " + isFound);      }  } |

**EXPERIMENT NO. 2**

|  |
| --- |
| **Student Name and Roll Number:** Piyush Gambhir |
| **Semester /Section:** Semester-III – AIML-B |
| **Link to Code:** [Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III (github.com)](https://github.com/Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III) |
| **Date:** |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective(s):**  To familiarize the students with linear data structure array and its basic operations |
| **Outcome:**  The students will be able to implement and use arrays for solving various problems |
| **Problem Statement:**   1. Write a program that initializes an array with ten random integers and then prints four lines of output, containing: 2. Every element at an even index 3. Every odd element 4. All elements in reverse order 5. Only the first and last element |
| **Background Study:**  An Array is a data structure consisting of a collection of elements (values or variables), each identified by at least one array index or key. An array is stored such that the position of each element can be computed from its index tuple by a mathematical formula. The simplest type of data structure is a linear array, also called one-dimensional array. |
| **Algorithm (Student Work area):**  **Print every element at an even index**   1. START 2. Initiate arr[] with 10 elements and i,m,j as variables 3. Run a loop i=0 till i<10 with i=i+2 4. Print elements at even index of the array 5. END   **Print every odd element**   1. START 2. Run a loop i=0 till i<10 with i=i+1 3. Check if((arr[i]%2)!=0); 4. Print every odd elements of the array 5. END   **Print all elements in reverse order**   1. START 2. Run a loop j=9 till j>=0 with j=j-1 3. Print elements of the array in reversed order 4. END   **Print only the first and last element**   1. START 2. Print arr[0] and arr[9] as first and last element respectively 3. END |
| **Code (Student Work Area):**  /\*   \* Problem Statement:   \* Write a program that initializes an array with ten random integers and then prints four lines of output, containing:   \* • Every element at an even index   \* • Every odd element   \* • All elements in reverse order   \* • Only the first and last elemenarray.   \*/  import java.util.Scanner;  /\*\*   \* problem\_statement\_experiment\_2   \*/  public class problem\_statement\_experiment\_2 {      // Method enterArrayElements(int[] arr)      // Method for taking input for array elements      public static void enterArrayElements(int[] arr) {          Scanner sc = new Scanner(System.in);          System.out.print("""                  ====================================                  Enter Array Elements                  ====================================                  """);          for (int i = 0; i < arr.length; i++) {              System.out.print("Enter the Value of Array Element " + (i + 1) + ": ");              int temp = sc.nextInt();              arr[i] = temp;          }          sc.close();      }      // Method printArrayElements(int[] arr)      // Method for printing array elements      public static void printArrayElements(int[] arr) {          System.out.print("""                  ====================================                  Printing Array Elements                  ====================================                  """);          for (int i = 0; i < arr.length; i++) {              System.out.println("Value of Array Element " + (i + 1) + ": " + arr[i]);          }      }      // Method printArrayElements(int[] arr)      // Method for printing array elements      public static void printReversedArrayElements(int[] arr) {          System.out.print("""                  ====================================                  Printing Array Elements in Reverse                  ====================================                  """);          for (int i = arr.length - 1; i >= 0; i--) {              System.out.println("Value of Array Element " + (i + 1) + ": " + arr[i]);          }      }      // Method printOddIndexArrayElements(int[] arr)      // Method for printing array elements at odd index      public static void printOddIndexArrayElements(int[] arr) {          System.out.print("""                  =======================================                  Printing Array Elements at Odd Indices                  =======================================                  """);          for (int i = 0; i < arr.length; i++) {              if (i % 2 != 0) {                  System.out.println("Value of Array Element at Index: " + (i) + ": " + arr[i]);              }          }      }      // Method printEvenIndexArrayElements(int[] arr)      // Method for printing array elements at even index      public static void printEvenIndexArrayElements(int[] arr) {          System.out.print("""                  ========================================                  Printing Array Elements at Even Indices                  ========================================                  """);          for (int i = 0; i < arr.length; i++) {              if (i % 2 == 0) {                  System.out.println("Value of Array Element at Index: " + (i) + ": " + arr[i]);              }          }      }      public static void printFirstLastArrayElement(int[] arr) {          System.out.print("""                  ========================================                  Printing First and Last Array Element                  ========================================                  """);          System.out.println("The First Element of Array; " + arr[0]);          System.out.println("The Last Element of Array; " + arr[arr.length - 1]);      }      public static void main(String[] args) {          Scanner sc = new Scanner(System.in);          System.out.print("Enter the Size of The Array: ");          int n = sc.nextInt();          int[] arr = new int[n];          System.out.println("\n");          enterArrayElements(arr);          System.out.println("\n");          printArrayElements(arr);          System.out.println("\n");          printReversedArrayElements(arr);          System.out.println("\n");          printOddIndexArrayElements(arr);          System.out.println("\n");          printEvenIndexArrayElements(arr);          System.out.println("\n");          printFirstLastArrayElement(arr);      }  } |
| **Output – Screenshots (Student Work Area):** |
| **Question Bank:**   1. How we can segregate all 0s on left side and all 1s on right side of a given array of 0s and 1s. 2. START 3. Initiate arr[]={1,2,3,4,5} and len,x =0as a variable 4. Find the length of the array(len) 5. Run a loop i=0 till i<len with i=i+1 6. If (arr[i[==0) 7. {x=x+1} 8. Print zeros to the front x times 9. The remaining number of 1s will be 1- (x) 10. Print the remaining elements 11. END 12. How to reverse the array elements. 13. START 14. Initiate arr[] with 10 elements and i,m,j as variables 15. Run a loop j=9 till j>=0 with j=j-1   Print elements of the array in reversed order   1. END 2. How to find the index of an array element 3. START 4. Initiate arr[]={1,2,3,4,5} and len,x as a variable 5. Find the length of the array(len) 6. Input the character who’s index is needed and store it in x 7. Run loop till i=0;i<len;i=i+1   Check if (arr[i]==x)  Print (“Index of character is:”+i)   1. END 2. How to remove a specific element from an array. 3. START 4. Initiate arr[]={1,2,3,4,5} and len,x as a variable 5. Find the length of the array(len) 6. Input which element need to be deleted and store it in x 7. Input the character who’s index is needed and store it in x 8. Run loop till i=x;i<len-1;i=i+1   arr[i]=arr[i+1]   1. Run a loop i=0 till i<len-1 with i=i+1   Print elements of the array   1. End 2. How to insert an element (specific position) into an array. 3. START 4. Initiate arr[] = {1, 2, 3, 4, 5} and, and len x,n,pos as a variable 5. Get the the variables n and pos as inputs for element and position respectively 6. Initialise a new array arr\_new[] with size one greater than arr[] 7. Copy all the elements from previous array into the new array till the position pos 8. Insert the element n at position pos 9. Insert the rest of the elements from the previous array into the new array after the pos 10. END |

**EXPERIMENT NO. 3**

|  |
| --- |
| **Student Name and Roll Number:** Piyush Gambhir |
| **Semester /Section:** Semester-III – AIML-B |
| **Link to Code:** [Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III (github.com)](https://github.com/Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III) |
| **Date:** |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective(s):**  To familiarize the students with linear data structure array and its basic operations |
| **Outcome:**  The students will be able to implement and use arrays for solving various problems |
| **Problem Statement:**   1. Write a program to read numbers in an integer array of size 5 and display the following:  * Sum of all the elements * Sum of alternate elements in the array * Second highest element in the array |
| **Background Study:**  An Array is a data structure consisting of a collection of elements (values or variables), each identified by at least one array index or key. An array is stored such that the position of each element can be computed from its index tuple by a mathematical formula. The simplest type of data structure is a linear array, also called one-dimensional array. |
| **Algorithm (Student Work Area):**  **Sum of all Elements of an Array**   1. START 2. Declare and initialize an array. 3. The variable sum will be used to calculate the sum of the elements. Initialize it to 0. 4. Loop through the array and add each element of array to variable sum as sum = sum + arr[i]. 5. END |
| **Code (Student Work Area):**  /\*   \* Problem Statement:   \* Write a program that initializes an array with ten random integers and then prints four lines of output, containing:   \* • Every element at an even index   \* • Every odd element   \* • All elements in reverse order   \* • Only the first and last elemenarray.   \*/  import java.util.Scanner;  /\*\*   \* problem\_statement\_experiment\_2   \*/  public class problem\_statement\_experiment\_2 {      // Method enterArrayElements(int[] arr)      // Method for taking input for array elements      public static void enterArrayElements(int[] arr) {          Scanner sc = new Scanner(System.in);          System.out.print("""                  ====================================                  Enter Array Elements                  ====================================                  """);          for (int i = 0; i < arr.length; i++) {              System.out.print("Enter the Value of Array Element " + (i + 1) + ": ");              int temp = sc.nextInt();              arr[i] = temp;          }          sc.close();      }      // Method printArrayElements(int[] arr)      // Method for printing array elements      public static void printArrayElements(int[] arr) {          System.out.print("""                  ====================================                  Printing Array Elements                  ====================================                  """);          for (int i = 0; i < arr.length; i++) {              System.out.println("Value of Array Element " + (i + 1) + ": " + arr[i]);          }      }      // Method printArrayElements(int[] arr)      // Method for printing array elements      public static void printReversedArrayElements(int[] arr) {          System.out.print("""                  ====================================                  Printing Array Elements in Reverse                  ====================================                  """);          for (int i = arr.length - 1; i >= 0; i--) {              System.out.println("Value of Array Element " + (i + 1) + ": " + arr[i]);          }      }      // Method printOddIndexArrayElements(int[] arr)      // Method for printing array elements at odd index      public static void printOddIndexArrayElements(int[] arr) {          System.out.print("""                  =======================================                  Printing Array Elements at Odd Indices                  =======================================                  """);          for (int i = 0; i < arr.length; i++) {              if (i % 2 != 0) {                  System.out.println("Value of Array Element at Index: " + (i) + ": " + arr[i]);              }          }      }      // Method printEvenIndexArrayElements(int[] arr)      // Method for printing array elements at even index      public static void printEvenIndexArrayElements(int[] arr) {          System.out.print("""                  ========================================                  Printing Array Elements at Even Indices                  ========================================                  """);          for (int i = 0; i < arr.length; i++) {              if (i % 2 == 0) {                  System.out.println("Value of Array Element at Index: " + (i) + ": " + arr[i]);              }          }      }      public static void printFirstLastArrayElement(int[] arr) {          System.out.print("""                  ========================================                  Printing First and Last Array Element                  ========================================                  """);          System.out.println("The First Element of Array; " + arr[0]);          System.out.println("The Last Element of Array; " + arr[arr.length - 1]);      }      public static void main(String[] args) {          Scanner sc = new Scanner(System.in);          System.out.print("Enter the Size of The Array: ");          int n = sc.nextInt();          int[] arr = new int[n];          System.out.println("\n");          enterArrayElements(arr);          System.out.println("\n");          printArrayElements(arr);          System.out.println("\n");          printReversedArrayElements(arr);          System.out.println("\n");          printOddIndexArrayElements(arr);          System.out.println("\n");          printEvenIndexArrayElements(arr);          System.out.println("\n");          printFirstLastArrayElement(arr);      }  } |
| **Code (Student Work Area):** |
| **3Question Bank:**   1. How we can count occurrence of a given number in the array and its frequency.    * + 1. Start        2. Initiate arr[]={1,2,3,4,5} and oc=0 as a variable        3. Enter the element whose frequency you want to know and store in x        4. Run loop int i=0i<=4;i=i+1)   If(arr[i]=x)  Oc=oc+1   * + - 1. Print the occurrence of each element (oc)       2. End  1. How we can print the following in 2-D integer array with each element of maximum 2 digits Elements of the entered array. 2. Elements of the entered array. 3. START 4. input of no of rows and column as r and m respectively 5. Run i=0 till i<r with i=i+1   Another nested loop j=0 tillvj<r with j=j+1  { Print arr[i][j]}}   1. END 2. Elements of the array after each element is multiplied by 2 if it is an odd number. 3. START 4. input of no of rows and column as r and m respectively 5. Run i=0 till i<r with i=i+1   Another nested loop j=0 tillvj<r with j=j+1  { check if a[i][j]%2!=0  a[i][j]=a[i][j]\*2  print a[i][j]  else   * + 1. print a[i][j] }}  1. END 2. Elements of the array after each element is multiplied by 2 if it is an odd number. 3. START 4. Initialize arr[] = {1, 2, 3, 4, 5, 6, 6, 5, 6, 6, 7}, variable count6= 0, count7= 0 and variable,x as length of arr 5. Run i = 0 till i &lt; x with i=i+1)   if(arr[i] == 6 and i+ 1 != x)  if (arr[i+1] == 7)  count7 = count7 + 1  if (arr[i+1] == 6)  count6 = count6 + 1   1. Print count6,count7 2. END 3. Given an array of integers, return the number of times that two 6's are next to each other in the array. Also count instances where the second element is 7 4. START 5. Initialize arr[] = {10, 20, 30, 40, 50}, variable n, and variable x as length of arr 6. run i = 0 till i &lt; x with i=i+1)    1. If(i+1 != x)       1. n = arr[i]       2. arr[i] = arr[i+1]       3. arr[i+1] = n 7. Print arr 8. END 9. Write a method called swapPairs() that accepts an array of integers and swaps the elements at adjacent indexes. That is, elements 0 and 1 are swapped, elements 2 and 3 are swapped, and so on. If the array has an odd length, the final element should be left unmodified. For example, the list {10, 20, 30, 40, 50} should become {20, 10, 40, 30, 50} after a call to your method. 10. Start 11. Initialize minimum value min to 12. Traverse the array to find the minimum element in the array 13. While traversing if any element smaller than min\_idx is found then swap both the 14. values. 15. Then, increment min\_idx to point to next element 16. Repeat until array is sorted 17. Then, check if the number of elements present in the array is even or odd 18. If odd, then simply return the mid value of the array 19. Else, the median is the average of the two middle values 20. END 21. Write a method called *median*() that accepts an array of integers as its argument and returns the median of the numbers in the array. The median is the number that will appear in the middle if you arrange the elements in order.   public static int median(int[] arr) {          int median = 0;          int n = arr.length;          if (n % 2 == 0) {              median = (arr[n / 2] + arr[(n / 2) - 1]) / 2;          } else {              median = arr[n / 2];          }          return median;      } |

**EXPERIMENT NO. 4**

|  |
| --- |
| **Student Name and Roll Number:** Piyush Gambhir |
| **Semester /Section:** Semester-III – AIML-B |
| **Link to Code:** [Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III (github.com)](https://github.com/Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III) |
| **Date:** |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective(s):**  To familiarize the students with linear data structure Linked List and its basic operations |
| **Outcome:**  The students will be able to implement and use singly linked list for solving various problems |
| **Problem Statement:**  Write a program to create a singly linked list of n nodes and perform:  • Insertion   * At the beginning * At the end * At a specific location   • Deletion   * At the beginning * At the end * At a specific location |
| **Background Study:** **Insertion Operation** Adding a new node in linked list is a more than one step activity. We shall learn this with diagrams here. First, create a node using the same structure and find the location where it has to be inserted.  Linked List Insertion  Imagine that we are inserting a node **B** (NewNode), between **A** (LeftNode) and **C** (RightNode). Then point B.next to C −  NewNode.next −> RightNode;  It should look like this −  Linked List Insertion  Now, the next node at the left should point to the new node.  LeftNode.next −> NewNode;  Linked List Insertion  This will put the new node in the middle of the two. The new list should look like this −  Linked List Insertion  Similar steps should be taken if the node is being inserted at the beginning of the list. While inserting it at the end, the second last node of the list should point to the new node and the new node will point to NULL. **Deletion Operation** Deletion is also a more than one step process. We shall learn with pictorial representation. First, locate the target node to be removed, by using searching algorithms.  Linked List Deletion  The left (previous) node of the target node now should point to the next node of the target node −  LeftNode.next −> TargetNode.next;  Linked List Deletion  This will remove the link that was pointing to the target node. Now, using the following code, we will remove what the target node is pointing at.  TargetNode.next −> NULL;  Linked List Deletion  We need to use the deleted node. We can keep that in memory otherwise we can simply deallocate memory and wipe off the target node completely.  Linked List Deletion |
| **Algorithm (Student Work Area):**  **Insertion at the beginning:**   1. START 2. Allocate a new node. 3. Make the next of the new node as head. 4. Make the new node as head. 5. END   **Insertion at the end:**   1. START 2. Allocate a new node. 3. Traverse till the last node. 4. Make the new node as the next node of the last node; 5. END   **Insertion at a specific position:**   1. START 2. Firstly, check if the given previous node is NULL or not. 3. Then, allocate a new node and 4. Assign the data to the new node 5. And then make the next of new node as the next of previous node. 6. Finally, move the next of the previous node as a new node. 7. END   **Deletion at the beginning:**   1. START 2. Make a temp node equal to head. 3. Make temp equal to the next element. 4. Make temp as head. 5. END   **Deletion at the end:**   1. START 2. Make a temp node equal to the head node. 3. Traverse temp node till the second last node. 4. Make the next node of the temp null. 5. END   **Deletion at a specific position:**   1. START 2. Firstly, check if the given previous node is NULL or not. 3. Finally, move the next of the previous node as a new node. 4. END |
| **Code (Student Work Area):**  /\*  Problem Statement  Write a program to create a singly linked list of n nodes and perform:  • Insertion      o At the beginning      o At the end      o At a specific location  • Deletion      o At the beginning      o At the end      o At a specific location   \*/  /\*\*   \* problem\_statement\_experiment\_4   \*/  public class problem\_statement\_experiment\_4 {      static Node head;      static class Node {          int data;          Node next;          Node(int data) {              this.data = data;              this.next = null;          }      }      public static void printLinkedList() {          Node currentNode = head;          while (currentNode != null) {              System.out.print(currentNode.data + " ");              currentNode = currentNode.next;          }      }      public static void insertNodeAtBeginning(int data) {          Node newNode = new Node(data);          newNode.next = head;          head = newNode;      }      public static void insertNodeAtEnd(int data) {          Node newNode = new Node(data);          if (head == null) {              head = newNode;          }          Node currentNode = head;          while (currentNode.next != null) {              currentNode = currentNode.next;          }          currentNode.next = newNode;      }      public static void insertNode(int data, int position) {          Node newNode = new Node(data);          Node currentNode = head;          int size = 0;          while (currentNode.next != null) {              currentNode = currentNode.next;              size++;          }          if (position > size) {              System.out.println("Invalid position");          } else {              currentNode = head;              while (position > 1) {                  currentNode = currentNode.next;                  position--;              }              newNode.next = currentNode.next;              currentNode.next = newNode;          }      }      public static void deleteNodeAtBeginning() {          if (head == null) {              System.out.println("List is empty");          } else {              head = head.next;          }      }      public static void deleteNodeAtEnd() {          if (head == null) {              System.out.println("List is Empty.");          } else {              Node currentNode = head;              while (currentNode.next.next != null) {                  currentNode = currentNode.next;              }              currentNode.next = null;          }      }      public static void deleteNode(int position) {          if (head == null) {              System.out.println("List is Empty.");          } else {              Node currentNode = head;              int size = 0;              while (currentNode.next != null) {                  currentNode = currentNode.next;                  size++;              }              if (position > size) {                  System.out.println("Invalid position");              } else {                  currentNode = head;                  while (position > 1) {                      currentNode = currentNode.next;                      position--;                  }                  currentNode.next = currentNode.next.next;              }          }      }      public static void main(String[] args) {          Node first = new Node(1);          Node second = new Node(2);          Node third = new Node(3);          Node fourth = new Node(4);          Node fifth = new Node(5);          head = first;          first.next = second;          second.next = third;          third.next = fourth;          fourth.next = fifth;          System.out.println("Linked List before insertion:");          printLinkedList();          System.out.println();          System.out.println("Linked List after insertion at beginning:");          insertNodeAtBeginning(0);          printLinkedList();          System.out.println();          System.out.println("Linked List after insertion at end:");          insertNodeAtEnd(6);          printLinkedList();          System.out.println();          System.out.println("Linked List after insertion at specific location:");          insertNode(7, 3);          printLinkedList();          System.out.println();          System.out.println("Linked List after deletion at beginning:");          deleteNodeAtBeginning();          printLinkedList();          System.out.println();          System.out.println("Linked List after deletion at end:");          deleteNodeAtEnd();          printLinkedList();          System.out.println();          System.out.println("Linked List after deletion at specific location:");          deleteNode(2);          printLinkedList();          System.out.println();      }  } |
| **Output – Screenshots (Student Work Area):** |
| **Question Bank:**   1. How Linked List id different from Arrays?   Array: Arrays store elements in contiguous memory locations, resulting in easily calculable addresses for the elements stored and this allows faster access to an element at a specific index.  Linked List: Linked lists are less rigid in their storage structure and elements are usually not stored in contiguous locations; hence they need to be stored with additional tags giving a reference to the next element.   1. How to perform the following set of operations on a singly linked list (SLL):  * Swapping the first and last node of a singly linked list   1. Start   2. Swapping the first and last node of a singly linked list   3. Create a class Node which has two attributes: data and next. Next is a pointer to the next node in the list.   4. Create another class Swap which has two attributes: head and tail.   5. addNode() will add a new node to the list:   6. Create a new node.   7. It first checks, whether the head is equal to null which means the list is empty.   8. If the list is empty, both head and tail will point to a newly added node.   9. End * Pairwise swap elements of a given linked list   1. Start   2. Initialize prev and curr pointers.   3. Traverse the list, store in temp node the value of curr->next and change next of curr as of the prev node.   4. If temp is NULL or temp is the last node then change prev->next to NULL and break the iteration. (Above mentioned corner conditions).   5. Else we have to change next of prev to next of next of curr.   6. Update prev and curr nodes for next iterate.   7. End * Get the location of the first and last occurrence of an element in a single LinkedList  1. Start 2. Run a for loop and for i = 0 to n-1 3. Take first = -1 and last = -1 4. When we find an element first time then we update first = i 5. We always update last=i whenever we find the element. 6. We print first and last 7. End  * Remove duplicates from an unsorted linked list  1. Start 2. Create a function ‘getResult()’ that will accept one parameter, i.e., one head pointer of the linked list. 3. Initialize two variables: ‘temp1’ will keep track of the element whose duplicates are being checked, and ‘temp2’ will keep track of the node that is being checked for the duplicate. 4. Assign the value of ‘head’ to ‘temp1’ and assign the null value to ‘temp2’. 5. Make an iteration using the ‘while’ loop, which will terminate if the value of ‘temp1’ or the ‘next’ pointer of ‘temp1’ is equal to null. 6. Store value of ‘temp1’ in ‘temp2’. 7. Make one nested iteration using the ‘while’ loop, which will terminate if the value of the ‘next’ pointer of ‘temp2’ is equal to null. 8. Check if the value of both the ‘temp1’ and ‘temp2’ are equal or not, if they are equal, delete that node and increment the next pointer of ‘temp2’ to its next node’s next pointer and if not equal, then only increment it once. 9. Increment the ‘next’ pointer of the ‘temp1’ node to its next node. 10. End  * Delete alternate nodes of a Linked List  1. Start 2. Write struct with data and next pointer. 3. Write a function to insert the node into the singly linked list. 4. Initialize the singly linked list with dummy data. 5. Iterate over the singly linked list. 6. Delete alternate node by maintaining the previous node. 7. Write a function to delete the node. Consider the following three cases while deleting the node. 8. If the node is head node, then move the head to next node. 9. If the node is middle node, then link the next node to the previous node 10. If the node is end node, then remove the previous node link. 11. End |

**EXPERIMENT NO. 5**

|  |
| --- |
| **Student Name and Roll Number:** Piyush Gambhir |
| **Semester /Section:** Semester-III – AIML-B |
| **Link to Code:** [Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III (github.com)](https://github.com/Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III) |
| **Date:** |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective(s):**  To familiarize the students with linear data structure Linked List and its basic operations |
| **Outcome:**  The students will be able to implement and use doubly linked list for solving various problems |
| **Problem Statement:**  Write a program to create a doubly linked list of n nodes and perform:  • Insertion   * At the beginning * At the end * At a specific location   • Deletion   * At the beginning * At the end * At a specific location |
| **Background Study:**  A Doubly Linked List (DLL) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in singly linked list. **Insertion Operation**A node can be added in three ways  **1)**At the front of the DLL  **2)** After a given node.  **3)** At the end of the DLL  **1) Add a node at the front:**dll_add_front **2) Add a node after a given node.:**  dll_add_middle **3) Add a node at the end:** dll_add_end **Deletion Operation** The deletion of a node in a doubly-linked list can be divided into three main categories:  Suppose we have a double-linked list with elements **1**, **2**, and **3**.  Original doubly linked list 1. Delete the First Node of Doubly Linked List **Reset value node after the del\_node (i.e. node two)**  Reorganize the pointers  ***Reorganize the pointers***  Finally, free the memory of del\_node. And, the linked will look like this  Final list  ***Final list*** 2. Deletion of the Inner Node If del\_node is an inner node (second node), we must have to reset the value of next and prev of the nodes before and after the del\_node.  **For the node before the del\_node (i.e. first node)**  Assign the value of next of del\_node to the next of the first node.  **For the node after the del\_node (i.e. third node)**  Assign the value of prev of del\_node to the prev of the third node.  Reorganize the pointers  ***Reorganize the pointers***  Finally, we will free the memory of del\_node. And, the final doubly linked list looks like this.  Final list  ***Final list*** 3. Delete the Last Node of Doubly Linked List In this case, we are deleting the last node with value **3** of the doubly linked list.  Here, we can simply delete the del\_node and make the next of node before del\_node point to NULL.  Reorganize the pointers  ***Reorganize the pointers***  The final doubly linked list looks like this.  Final list  ***Final list*** |
| **Algorithm (Student Work Area):**  **Insertion at the beginning:**   1. START 2. Allocate a new node. 3. Firstly, check if the given previous node is NULL or not, if null make new node as head. 4. Make the next of the new node as current head. 5. Make the new node as head. 6. Make the previous of the next of new node as new node. 7. END   **Insertion at the end:**   1. START 2. Allocate a new node. 3. Traverse till the last node. 4. Make the new node as the next node of the last node. 5. Make the previous of the new node as current node. 6. END |
| **Code (Student Work Area):**  /\*  Problem Statement  Write a program to create a doubly linked list of n nodes and perform:  • Insertion      o At the beginning      o At the end      o At a specific location  • Deletion      o At the beginning      o At the end      o At a specific location   \*/  /\*\*   \* problem\_statement\_experiment\_4   \*/  public class problem\_statement\_experiment\_5 {      static Node head;      static class Node {          int data;          Node next;          Node prev;          Node(int data) {              this.data = data;              this.next = null;              this.prev = null;          }      }      public static void printLinkedList() {          Node currentNode = head;          while (currentNode != null) {              System.out.print(currentNode.data + " ");              currentNode = currentNode.next;          }      }      public static void insertNodeAtBeginning(int data) {          Node newNode = new Node(data);          newNode.next = head;          head = newNode;          newNode.prev = null;          if (newNode.next != null) {              newNode.next.prev = newNode;          }      }      public static void insertNodeAtEnd(int data) {          Node newNode = new Node(data);          if (head == null) {              head = newNode;          }          Node currentNode = head;          while (currentNode.next != null) {              currentNode = currentNode.next;          }          currentNode.next = newNode;          newNode.prev = currentNode;      }      public static void insertNodeAtSpecificLocation(int data, int location) {          Node newNode = new Node(data);          Node currentNode = head;          int count = 1;          while (currentNode != null) {              if (count == location) {                  newNode.next = currentNode.next;                  currentNode.next = newNode;                  newNode.prev = currentNode;                  if (newNode.next != null) {                      newNode.next.prev = newNode;                  }                  break;              }              currentNode = currentNode.next;              count++;          }      }      public static void deleteNodeAtBeginning() {          if (head == null) {              System.out.println("List is empty");          } else {              head = head.next;              head.prev = null;          }      }      public static void deleteNodeAtEnd() {          if (head == null) {              System.out.println("List is empty");          } else {              Node currentNode = head;              while (currentNode.next != null) {                  currentNode = currentNode.next;              }              currentNode.prev.next = null;          }      }      public static void deleteNodeAtSpecificLocation(int location) {          Node currentNode = head;          int count = 1;          while (currentNode != null) {              if (count == location) {                  currentNode.prev.next = currentNode.next;                  currentNode.next.prev = currentNode.prev;                  break;              }              currentNode = currentNode.next;              count++;          }      }      public static void main(String[] args) {          Node first = new Node(1);          Node second = new Node(2);          Node third = new Node(3);          Node fourth = new Node(4);          Node fifth = new Node(5);          head = first;          first.next = second;          second.next = third;          second.prev = first;          third.next = fourth;          third.prev = second;          fourth.next = fifth;          fourth.prev = third;          fifth.prev = fourth;          System.out.println("Linked List before insertion:");          printLinkedList();          System.out.println();          System.out.println("Linked List after insertion at beginning:");          insertNodeAtBeginning(0);          printLinkedList();          System.out.println();          System.out.println("Linked List after insertion at end:");          insertNodeAtEnd(6);          printLinkedList();          System.out.println();          System.out.println("Linked List after insertion at specific location:");          insertNodeAtSpecificLocation(7, 3);          printLinkedList();          System.out.println();          System.out.println("Linked List after deletion at beginning:");          deleteNodeAtBeginning();          printLinkedList();          System.out.println();          System.out.println("Linked List after deletion at end:");          deleteNodeAtEnd();          printLinkedList();          System.out.println();          System.out.println("Linked List after deletion at specific location:");          deleteNodeAtSpecificLocation(3);          printLinkedList();          System.out.println();      }  } |
| **Output – Screenshots (Student Work Area):** |
| **Question Bank:**   1. What is Doubly Linked List?   Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer).   1. What is the complexity of Traversal, Insertion and deletion operations in doubly linked list?   In a doubly-linked list, the time complexity for inserting and deleting an element is O(1). Whereas, traversal is O(n). |

**EXPERIMENT NO. 6**

|  |
| --- |
| **Student Name and Roll Number:** Piyush Gambhir |
| **Semester /Section:** Semester-III – AIML-B |
| **Link to Code:** [Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III (github.com)](https://github.com/Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III) |
| **Date:** |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective(s):**  To familiarize the students with linear data structure Linked List and its basic operations |
| **Outcome:**  The students will be able to implement and use Circular linked list for solving various problems |
| **Problem Statement:**  Write a program to create a Circular linked list of n nodes and perform:  • Insertion   * At the beginning * At the end * At a specific location   • Deletion   * At the beginning * At the end * At a specific location |
| **Background Study:**  **Circular linked list** is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list.   **Insertion** We can insert a node in a circular linked list either as a first node (empty list), in the beginning, in the end, or in between the other nodes. Let us see each of these insertion operations using a pictorial representation below.  **#1) Insert in an empty list**  [Insert in an empty list](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2019/06/Insert-in-an-empty-list.png)  When there are no nodes in circular list and the list is empty, the last pointer is null, then we insert a new node N by pointing the last pointer to the node N as shown above. The next pointer of N will point to the node N itself as there is only one node. Thus N becomes the first as well as last node in the list.  **#2) Insert at the beginning of the list**  [Insert at the beginning of the list](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2019/06/Insert-at-the-beginning-of-the-list.png)  As shown in the above representation, when we add a node at the beginning of the list, the next pointer of the last node points to the new node N thereby making it a first node.  **N->next = last->next**  **Last->next = N**  **#3) Insert at the end of the list**  [last node points to the new node](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2019/06/last-node-points-to-the-new-node.png)  **To insert a new node at the end of the list, we follow these steps:**  **N-> next = last ->next; last ->next = N last = N**  **#4) Insert in between the list**  [Insert in between the list](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2019/06/To-insert-a-new-node-at-the-end-of-the-list.png)  Suppose we need to insert a new node N between N3 and N4, we first need to traverse the list and locate the node after which the new node is to be inserted, in this case, its N3.  **After the node is located, we perform the following steps.**  **N -> next = N3 -> next; N3 -> next = N**  This inserts a new node N after N3. **Deletion** The deletion operation of the circular linked list involves locating the node that is to be deleted and then freeing its memory.  For this we maintain two additional pointers curr and prev and then traverse the list to locate the node. The given node to be deleted can be the first node, the last node or the node in between. Depending on the location we set the curr and prev pointers and then delete the curr node.  **A pictorial representation of the deletion operation is shown below.**  [deletion operation](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2019/06/deletion.png) |
| **Algorithm (Student Work Area):** |
| **Code (Student Work Area):**  /\*  Problem Statement:  Write a program to create a Circular linked list of n nodes and perform:  •   Insertion      o   At the beginning      o   At the end      o   At a specific location  •   Deletion      o   At the beginning      o   At the end      o   At a specific location   \*/  /\*\*   \* problem\_statement\_experiment\_6   \*/  public class problem\_statement\_experiment\_6 {      static Node head;      static class Node {          int data;          Node next;          Node(int data) {              this.data = data;              this.next = null;          }      }      public static void printList() {          Node temp = head;          if (head != null) {              do {                  System.out.print(temp.data + " ");                  temp = temp.next;              } while (temp != head);          }          System.out.println();      }      public static void insertAtBeginning(int data) {          Node newNode = new Node(data);          if (head == null) {              head = newNode;              newNode.next = head;          } else {              Node temp = head;              while (temp.next != head) {                  temp = temp.next;              }              temp.next = newNode;              newNode.next = head;              head = newNode;          }      }      public static void insertAtEnd(int data) {          Node newNode = new Node(data);          if (head == null) {              head = newNode;              newNode.next = head;          } else {              Node temp = head;              while (temp.next != head) {                  temp = temp.next;              }              temp.next = newNode;              newNode.next = head;          }      }      public static void insertAtSpecificLocation(int data, int location) {          Node newNode = new Node(data);          if (head == null) {              head = newNode;              newNode.next = head;          } else {              Node temp = head;              int count = 1;              while (count < location - 1) {                  temp = temp.next;                  count++;              }              newNode.next = temp.next;              temp.next = newNode;          }      }      public static void deleteAtBeginning() {          if (head == null) {              System.out.println("List is empty");          } else {              Node temp = head;              while (temp.next != head) {                  temp = temp.next;              }              temp.next = head.next;              head = head.next;          }      }      public static void deleteAtEnd() {          if (head == null) {              System.out.println("List is empty");          } else {              Node temp = head;              while (temp.next.next != head) {                  temp = temp.next;              }              temp.next = head;          }      }      public static void deleteAtSpecificLocation(int location) {          if (head == null) {              System.out.println("List is empty");          } else {              Node temp = head;              int count = 1;              while (count < location - 1) {                  temp = temp.next;                  count++;              }              temp.next = temp.next.next;          }      }      public static void main(String[] args) {          Node first = new Node(1);          Node second = new Node(2);          Node third = new Node(3);          Node fourth = new Node(4);          head = first;          first.next = second;          second.next = third;          third.next = fourth;          fourth.next = head;          System.out.println("Circular Linked List before insertion");          printList();          insertAtBeginning(0);          System.out.println("Circular Linked List after insertion at beginning");          printList();          insertAtEnd(5);          System.out.println("Circular Linked List after insertion at end");          printList();          insertAtSpecificLocation(6, 3);          System.out.println("Circular Linked List after insertion at specific location");          printList();          deleteAtBeginning();          System.out.println("Circular Linked List after deletion at beginning");          printList();          deleteAtEnd();          System.out.println("Circular Linked List after deletion at end");          printList();          deleteAtSpecificLocation(3);          System.out.println("Circular Linked List after deletion at specific location");          printList();      }  } |
| **Output – Screenshots (Student Work Area):** |
| **Question Bank:**   1. How Circular Linked List id different from Singly Linked List?   A circular linked list is a variation of a singly linked list. The only difference between the singly linked list and a circular linked list is that the last node does not point to any node in a singly linked list, so its link part contains a NULL value.   1. Analyze the complexity of Traversal, insertion, and Deletion operations in Circular Linked List?   The time complexity of traversal is O(n). Whereas, for insertion and deletion is O(1). |

**EXPERIMENT NO. 7**

|  |
| --- |
| **Student Name and Roll Number:** Piyush Gambhir |
| **Semester /Section:** Semester-III – AIML-B |
| **Link to Code:** [Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III (github.com)](https://github.com/Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III) |
| **Date:** |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective(s):**  To familiarize the students with linear data structure Stacks and its basic operations |
| **Outcome:**  The students will be able to implement and use Stacks for solving various problems |
| **Problem Statement:**  Write a program to create a stack and perform:   1. POP 2. PUSH 3. PEEK 4. ISEMPTY 5. ISFULL 6. Use Arrays for Implementation 7. Use Linked List for Implementation |
| **Background:**  Stacks are dynamic data structures that follow the **Last In First Out (LIFO)** principle. The last item to be inserted into a stack is the first one to be deleted from it.  For example, you have a stack of trays on a table. The tray at the top of the stack is the first item to be moved if you require a tray from that stack.  **Inserting and deleting elements**  Stacks have restrictions on the insertion and deletion of elements. Elements can be inserted or deleted only from one end of the stack i.e. from the top. The element at the top is called the top element. The operations of inserting and deleting elements are called push() and pop() respectively.  When the top element of a stack is deleted, if the stack remains non-empty, then the element just below the previous top element becomes the new top element of the stack.  For example, in the stack of trays, if you take the tray on the top and do not replace it, then the second tray automatically becomes the top element (tray) of that stack.  **Features of stacks**   * Dynamic data structures * Do not have a fixed size * Do not consume a fixed amount of memory * Size of stack changes with each push() and pop() operation. Each push() and pop() operation increases and decreases the size of the stack by 1, respectively.   A stack can be visualized as follows:  enter image description here |
| **Algorithm (Student Work Area):**  **Algorithm For Push:**  begin  if stack is full  return  endif  else  increment top  stack[top] assign value  end else  end procedure  **Algorithm For Pop**  begin  if stack is empty  return  endif  else  store value of stack[top]  decrement top  return value  end else  end procedure  **Algorithm For Top**  begin  return stack[top]  end procedure  **Algorithm For isEmpty**  begin  if top < 1  return true  else  return false  end procedure |
| **Code (Student Work Area):**  /\*  Problem Statement  Write a program to create a stack and perform:  • POP  • PUSH  • PEEK  • ISEMPTY  • ISFULL  1) Use Arrays for Implementation  2) Use Linked List for Implementation  \*/  // Implementing Stack Using Arrays  class stackArray {      public int stackSize = 5;      int[] stack = new int[stackSize];      int top = -1;      // Method POP()      // Method to delete top most element of the stack.      public int POP() {          if (top < 0) {              System.out.println("Stack Underflow!");              return 0;          }          else {              top = stack[top--];              return top;          }      }      // Method PUSH()      // Method to push element into stack.      public void PUSH(int data) {          if (top >= stackSize - 1) {              System.out.println("Stack OverFlow!");          }          else {              stack[++top] = data;              System.out.println(data + " pushed into stack.");          }      }      // Method PEEK()      // Method to get the top most elemnt of the stack.      public int PEEK() {          if (top < 0) {              System.out.println("Stack Underflow!");              return -1;          }          else {              return top;          }      }      // Method ISEMPTY()      // Method to check if the stack is empty or not.      public boolean ISEMPTY() {          if (top < 0) {              return true;          }          else {              return false;          }      }      // Method ISFULL()      // Method to check if the stack is full or not.      public boolean ISFULL() {          if (top == stackSize) {              return true;          } else {              return false;          }      }  }  // Implementing Stack Using Linked List  class stackLinkedList {      stackNode top;      class stackNode {          int data;          stackNode next;          stackNode(int data) {              this.data = data;          }      }      // Method PUSH()      // Method to push element into stack.      public void PUSH(int data) {          stackNode newNode = new stackNode(data);          if (top == null) {              top = newNode;          }          else {              newNode.next = top;              top = newNode;              System.out.println(newNode.data + " pushed into stack.");          }      }      // Method POP()      // Method to delete top most element of the stack.      public int POP() {          int popValue = -1;          if (top == null) {              System.out.println("Stack is Empty!");          }          else {              popValue = top.data;              top = top.next;          }          return popValue;      }      // Method PEEK()      // Method to get the top most elemnt of the stack.      public int PEEK() {          if (top == null) {              System.out.println("Stack is Empty!");              return -1;          }          else {              return top.data;          }      }      // Method ISEMPTY()      // Method to check if the stack is empty or not.      public boolean ISEMPTY() {          if (top == null) {              return true;          }          else {              return false;          }      }  }  /\*\*   \* problem\_statement\_experiment\_7   \*/  public class problem\_statement\_experiment\_7 {      public static void main(String[] args) {          // Using Array Implementation of Stack          stackArray stackA = new stackArray();          System.out.println("""                  ===============================                  Array Implementation of Stack                  ===============================                  """);          stackA.PUSH(1);          stackA.PUSH(2);          stackA.PUSH(3);          stackA.PUSH(4);          stackA.PUSH(5);          System.out.println("Top --> " + stackA.PEEK());          System.out.println("Popped --> " + stackA.POP());          System.out.println("Stack is Empty: " + stackA.ISEMPTY());          System.out.println("Stack is Full: " + stackA.ISFULL());          System.out.println("\n");          // Using Linked List Implementation of Stack          stackLinkedList stackLL = new stackLinkedList();          System.out.println("""                  ===================================                  Linked List Implementation of Stack                  ===================================                  """);          stackLL.PUSH(1);          stackLL.PUSH(2);          stackLL.PUSH(3);          stackLL.PUSH(4);          stackLL.PUSH(5);          System.out.println("Top --> " + stackLL.PEEK());          System.out.println("Popped --> " + stackLL.POP());          System.out.println("Stack is Empty: " + stackLL.ISEMPTY());      }  } |
| **Output – Screenshots (Student Work Area):** |
| **Question Bank:**   1. What are Stacks?   It is a linear data structure that follows a particular order in which the operations are performed. Stacks follows the order of LIFO (Last in First Out).   1. What are the applications of stacks? 2. A Stack can be used for evaluating expressions consisting of operands and operators. 3. Stacks can be used for Backtracking, i.e., to check parenthesis matching in an expression. 4. It can also be used to convert one form of expression to another form. 5. It can be used for systematic Memory Management. |

**EXPERIMENT NO. 8**

|  |
| --- |
| **Student Name and Roll Number:** Piyush Gambhir |
| **Semester /Section:** Semester-III – AIML-B |
| **Link to Code:** [Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III (github.com)](https://github.com/Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III) |
| **Date:** |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective(s):**  To familiarize the students with linear data structure Stacks and its applications. |
| **Outcome:**  The students will be able to implement and use Stacks for solving various problems |
| **Problem Statement:**  Write a program to create a stack and perform:  Reversal of a sentence using stack.  **Given a string str consisting of a sentence, the task is to reverse the entire sentence word by word.**  **Examples:**  **Input: str = “data structures and algorithms” Output:  algorithms and structures data** |
| **Background:**  Stacks are dynamic data structures that follow the **Last In First Out (LIFO)** principle. The last item to be inserted into a stack is the first one to be deleted from it.  For example, you have a stack of trays on a table. The tray at the top of the stack is the first item to be moved if you require a tray from that stack.  **Inserting and deleting elements**  Stacks have restrictions on the insertion and deletion of elements. Elements can be inserted or deleted only from one end of the stack i.e. from the top. The element at the top is called the top element. The operations of inserting and deleting elements are called push() and pop() respectively.  When the top element of a stack is deleted, if the stack remains non-empty, then the element just below the previous top element becomes the new top element of the stack.  For example, in the stack of trays, if you take the tray on the top and do not replace it, then the second tray automatically becomes the top element (tray) of that stack.  **Features of stacks**   * Dynamic data structures * Do not have a fixed size * Do not consume a fixed amount of memory * Size of stack changes with each push() and pop() operation. Each push() and pop() operation increases and decreases the size of the stack by 1, respectively.   A stack can be visualized as follows:  enter image description here |
| **Algorithm (Student Work Area):**   1. The first step would be, creating an empty stack. 2. Then pick the characters from the string one by one and put them to the stack, so that the last character of the string comes at the top of the stack. 3. Now, the last step, pop the stack and put the popped characters back in the empty string. |
| **Code (Student Work Area):**  import java.util.\*;  /\*  Problem Statement  Write a program to create a stack and perform:  1) Reversal of a sentence using stack.  2) Given a string str consisting of a sentence, the task is to reverse the entire     sentence word by word.       Examples:     Input: str = “data structures and algorithms”     Output: algorithms and structures data  \*/  /\*\*   \* problem\_statement\_experiment\_8   \*/  public class problem\_statement\_experiment\_8 {      public static String reverseSentence(String str) {          Stack<String> stack = new Stack<String>();          String[] strArray = str.split(" ");          String result = "";          for (int i = 0; i < strArray.length; i++) {              stack.push(strArray[i]);          }          while (!stack.isEmpty()) {              result += stack.pop() + " ";          }          return result;      }      public static void main(String[] args) {          String str = "data structures and algorithms";          System.out.println(reverseSentence(str));      }  } |
| **Output – Screenshots (Student Work Area):** |
| **Question Bank:**   1. What are Stacks?   It is a linear data structure that follows a particular order in which the operations are performed. Stacks follows the order of LIFO (Last in First Out).   1. How can we split a sentence and push it into the stack? 2. Create an empty stack. 3. Traverse the entire string, while traversing add the characters of the string into a temporary variable until you get a space(‘ ‘) and push that temporary variable into the stack. 4. Repeat the above step until the end of the string. |

**EXPERIMENT NO. 9**

|  |
| --- |
| **Student Name and Roll Number:** Piyush Gambhir |
| **Semester /Section:** Semester-III – AIML-B |
| **Link to Code:** [Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III (github.com)](https://github.com/Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III) |
| **Date:** |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective(s):**  To familiarize the students with linear data structure Stacks and its applications. |
| **Outcome:**  The students will be able to implement and use Stacks for solving various problems |
| **Problem Statement:**  Write a program to check whether the parenthesis in the expression are balanced or not.  **Given a string str consisting of an expression**  **Examples:**  **Input: str = (a+b)\*c**  **Output: Parenthesis Balanced** |
| **Background:**  Stacks are dynamic data structures that follow the **Last In First Out (LIFO)** principle. The last item to be inserted into a stack is the first one to be deleted from it.  For example, you have a stack of trays on a table. The tray at the top of the stack is the first item to be moved if you require a tray from that stack.  **Inserting and deleting elements**  Stacks have restrictions on the insertion and deletion of elements. Elements can be inserted or deleted only from one end of the stack i.e. from the top. The element at the top is called the top element. The operations of inserting and deleting elements are called push() and pop() respectively.  When the top element of a stack is deleted, if the stack remains non-empty, then the element just below the previous top element becomes the new top element of the stack.  For example, in the stack of trays, if you take the tray on the top and do not replace it, then the second tray automatically becomes the top element (tray) of that stack.  **Features of stacks**   * Dynamic data structures * Do not have a fixed size * Do not consume a fixed amount of memory * Size of stack changes with each push() and pop() operation. Each push() and pop() operation increases and decreases the size of the stack by 1, respectively.   A stack can be visualized as follows:  enter image description here |
| **Algorithm (Student Work Area):**   1. Declare a character stack (say temp). 2. Now traverse the string exp. 3. If the current character is a ‘(‘ then push it to stack. 4. If the current character is a closing bracket ‘)’ then pop from stack. 5. If stack is empty then, parenthesis are Not Balanced. 6. After complete traversal, if there is some starting parenthesis left in stack then Not balanced, else Balanced. |
| **Code (Student Work Area):**  import java.util.\*;  /\*  Problem Statement  Write a program to check whether the parenthesis in the expression are balanced or not.  Given a string str consisting of an expression  Examples:  Input: str = (a+b)\*c  Output: Parenthesis Balanced  \*/  /\*\*   \* problem\_statement\_experiment\_9   \*/  public class problem\_statement\_experiment\_9 {      public static void checkParenthesisBalanced(String expression) {          Stack<Character> stack = new Stack<Character>();          boolean isBalanced = true;          for (int i = 0; i < expression.length(); i++) {              if (expression.charAt(i) == '(') {                  stack.push(expression.charAt(i));              } else if (expression.charAt(i) == ')') {                  if (stack.isEmpty()) {                      isBalanced = false;                      break;                  } else {                      stack.pop();                  }              }          }          if (isBalanced) {              System.out.println("Parenthesis Balanced");          } else {              System.out.println("Parenthesis Not Balanced");          }      }      public static void main(String[] args) {          String expression = "(a+b)\*c";          checkParenthesisBalanced(expression);      }  } |
| **Output – Screenshots (Student Work Area):** |
| 1. How a stack helps in syntax analysis or compilation of a program?   It checks the syntactical structure of the given input, i.e. whether the given input is in the correct syntax (of the language in which the input has been written) or not. |

**EXPERIMENT NO. 10**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Student Name and Roll Number:** Piyush Gambhir |
| **Semester /Section:** Semester-III – AIML-B |
| **Link to Code:** [Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III (github.com)](https://github.com/Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III) |
| **Date:** |
| **Faculty Signature:** |

|  |
| --- |
| **Objective(s):**  To familiarize the students with linear data structure Stacks and its applications. |
| **Outcome:**  The students will be able to implement and use Stacks for solving various problems |
| **Problem Statement:**  Write a program to convert Infix expression into Postfix.  **Given a string str consisting of an infix expression, convert it into Postfix**  **Examples:**  **Input: str = (a+b)\*c**  **Output: ab+\*** |
| **Background:**  Any expression can be represented using three types of expressions (Infix, Postfix, and Prefix). We can also convert one type of expression to another type of expression like Infix to Postfix, Infix to Prefix, Postfix to Prefix and vice versa.  **Infix to postfix conversion** Scan through an expression, getting one token at a time.  1 Fix a priority level for each operator. For example, from high to low:      3.    - (unary negation)     2.    \* /     1.    + - (subtraction)  Thus, high priority corresponds to high number in the table.  2 If the token is an operand, do not stack it. Pass it to the output.  3 If token is an operator or parenthesis, do the following:     -- Pop the stack until you find a symbol of lower priority number than the current one. An incoming left parenthesis will be considered to have higher priority than any other symbol. A left parenthesis on the stack will not be removed unless an incoming right parenthesis is found. The popped stack elements will be written to output.     --Stack the current symbol.     -- If a right parenthesis is the current symbol, pop the stack down to (and including) the first left parenthesis. Write all the symbols except the left parenthesis to the output (i.e. write the operators to the output).     -- After the last token is read, pop the remainder of the stack and write any symbol (except left parenthesis) to output.  **Example:**  Convert A \* (B + C) \* D to postfix notation.   |  |  |  |  | | --- | --- | --- | --- | | **Move** | **Curren Ttoken** | **Stack** | **Output** | | 1 | A | empty | A | | 2 | \* | \* | A | | 3 | ( | (\* | A | | 4 | B | (\* | A B | | 5 | + | +(\* | A B | | 6 | C | +(\* | A B C | | 7 | ) | \* | A B C + | | 8 | \* | \* | A B C + \* | | 9 | D | \* | A B C + \* D | | 10 |  | empty |  | |
| **Algorithm (Student Work Area):**   1. Scan the infix expression from left to right. 2. If the scanned character is an operand, output it. 3. Else,   If the precedence and associativity of the scanned operator are greater than the precedence and associativity of the operator in the stack(or the stack is empty or the stack contains a ‘(‘ ), then push it.   1. ‘^’ operator is right associative and other operators like ‘+’,’-‘,’\*’ and ‘/’ are left-associative. Check especially for a condition when both, operator at the top of the stack and the scanned operator are ‘^’. In this condition, the precedence of the scanned operator is higher due to its right associativity. So it will be pushed into the operator stack. In all the other cases when the top of the operator stack is the same as the scanned operator, then pop the operator from the stack because of left associativity due to which the scanned operator has less precedence. 2. Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.) 3. If the scanned character is an ‘(‘, push it to the stack. 4. If the scanned character is an ‘)’, pop the stack and output it until a ‘(‘ is encountered, and discard both the parenthesis. 5. Repeat steps 2-6 until the infix expression is scanned. 6. Print the output 7. Pop and output from the stack until it is not empty. |
| **Code (Student Work Area):**  import java.util.\*;  /\*  Problem Statement  Write a program to convert Infix expression into Postfix.  Given a string str consisting of an infix expression, convert it into Postfix  Examples:  Input: str = (a+b)\*c  Output: ab+c\*  \*/  /\*\*   \* problem\_statement\_experiment\_10   \*/  public class problem\_statement\_experiment\_10 {      public static int getPrecedence(char operator) {          switch (operator) {              case '+':              case '-':                  return 1;              case '\*':              case '/':                  return 2;              case '^':                  return 3;              default:                  return -1;          }      }      public static String infixToPostfix(String infixExpression) {          Stack<Character> stack = new Stack<Character>();          String postfixExpression = "";          for (int i = 0; i < infixExpression.length(); i++) {              if (infixExpression.charAt(i) == '(') {                  stack.push(infixExpression.charAt(i));              } else if (infixExpression.charAt(i) == ')') {                  while (!stack.isEmpty() && stack.peek() != '(') {                      postfixExpression += stack.pop();                  }                  stack.pop();              } else if (infixExpression.charAt(i) == '+' || infixExpression.charAt(i) == '-'                      || infixExpression.charAt(i) == '\*' || infixExpression.charAt(i) == '/') {                  while (!stack.isEmpty() && stack.peek() != '('                          && getPrecedence(infixExpression.charAt(i)) <= getPrecedence(stack.peek())) {                      postfixExpression += stack.pop();                  }                  stack.push(infixExpression.charAt(i));              } else {                  postfixExpression += infixExpression.charAt(i);              }          }          while (!stack.isEmpty()) {              postfixExpression += stack.pop();          }          return postfixExpression;      }      public static void main(String[] args) {          String infixExpression = "(a+b)\*c";          System.out.println(infixToPostfix(infixExpression));      }  } |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. Why is conversion required?   Infix expressions are readable and solvable by humans. We can easily distinguish the order of operators, and also can use the parenthesis to solve that part first during solving mathematical expressions. The computer cannot differentiate the operators and parenthesis easily, that’s why postfix conversion is needed.   1. How can we convert infix to prefix and prefix to postfix?   **Infix to Prefix**   1. Reverse the infix expression. Note while reversing each ‘(‘ will become ‘)’ and each ‘)’ becomes ‘(‘. 2. Obtain the “nearly” postfix expression of the modified expression. 3. Reverse the postfix expression.   **Prefix to Postfix**   1. Read the Prefix expression in reverse order (from right to left) 2. If the symbol is an operand, then push it onto the Stack 3. If the symbol is an operator, then pop two operands from the Stack 4. Create a string by concatenating the two operands and the operator after them. 5. string = operand1 + operand2 + operator 6. And push the resultant string back to Stack 7. Repeat the above steps until end of Prefix expression. |

**EXPERIMENT NO. 11**

|  |
| --- |
| **Student Name and Roll Number:** Piyush Gambhir |
| **Semester /Section:** Semester-III – AIML-B |
| **Link to Code:** [Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III (github.com)](https://github.com/Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III) |
| **Date:** |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective(s):**  To familiarize the students with linear data structure Stacks and its application Recursion. |
| **Outcome:**  The students will be able to implement and use Stacks for solving Recursion problems |
| **Problem Statement:**  Write a program to implement Tower of Hanoi. |
| **Background:**  Tower of Hanoi, is a mathematical puzzle which consists of three towers (pegs) and more than one rings is as depicted −  Tower Of Hanoi  These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same. **Rules** The mission is to move all the disks to some another tower without violating the sequence of arrangement. A few rules to be followed for Tower of Hanoi are −   * Only one disk can be moved among the towers at any given time. * Only the "top" disk can be removed. * No large disk can sit over a small disk.  |  |  |  |  | | --- | --- | --- | --- | | Tower of Hanoi puzzle with n disks can be solved in minimum **2n−1** steps. |  |  |  | |
| **Algorithm (Student Work Area):**  **Using Recursion:**   1. Create a function towerOfHanoi where pass the N (current number of disk), from\_rod, to\_rod, aux\_rod. 2. Make a function call for N – 1 th disk. 3. Then print the current the disk along with from\_rod and to\_rod 4. Again make a function call for N – 1 th disk. |
| **Code (Student Work Area):**  /\*  Problem Statement  Write a program to implement Tower of Hanoi  \*/  import java.util.\*;  /\*\*   \* problem\_statement\_experiment\_11   \*/  public class problem\_statement\_experiment\_11 {      // Recursive Implementation of Tower of Hanoi Problem      public static void recursiveTowerOfHanoi(int numberOfDisks, String tower1, String tower2, String tower3) {          if (numberOfDisks == 0) {              System.out.println("No disks to move");              return;          }          if (numberOfDisks == 1) {              System.out.println("Move disk 1 from " + tower1 + " to " + tower3);              return;          }          recursiveTowerOfHanoi(numberOfDisks - 1, tower1, tower3, tower2);          System.out.println("Move disk " + numberOfDisks + " from " + tower1 + " to " + tower3);          recursiveTowerOfHanoi(numberOfDisks - 1, tower2, tower1, tower3);      }      public static void main(String[] args) {          System.out.println("Recursive Implementation of Tower of Hanoi Problem");          System.out.println("-------------------------------------------------");          System.out.println("For 2 disks");          recursiveTowerOfHanoi(2, "A", "B", "C");          System.out.println("--------------------");          System.out.println("For 3 disks");          recursiveTowerOfHanoi(3, "A", "B", "C");          System.out.println("-------------------------------------------------");      }  } |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. What is Recursion?   Recursion is a programming technique using function or algorithm that calls itself one or more times until a specified condition is met at which time the rest of each repetition is processed from the last one called to the first.   1. What is Base condition?   A recursive function calls itself. As you you'd imagine such a process would repeat indefinitely if not stopped by some condition. This condition is known as base condition. A base condition is must in every recursive programs otherwise it will continue to execute forever like an infinite loop.   1. What are the number of steps required to solve n-Disc problem?   n-Disc problem can be solved in a minimum of 2n – 1 step. |

**EXPERIMENT NO. 12**

|  |
| --- |
| **Student Name and Roll Number:** Piyush Gambhir |
| **Semester /Section:** Semester-III – AIML-B |
| **Link to Code:** [Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III (github.com)](https://github.com/Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III) |
| **Date:** |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective(s):**  To familiarize the students with linear data structure Queue and its applications. |
| **Outcome:**  The students will be able to implement and use Queues for solving various problems |
| **Problem Statement:**  Write a program to implement Following operations using Queue:   1. Enqueue() 2. Dequeue() 3. Isfull() 4. Isempty() 5. Peek() 6. Using array implementation 7. Using Linked List Implementation |
| **Background:**  **Queue** is also an abstract data type or a linear data structure, just like [stack data structure](https://www.studytonight.com/data-structures/stack-data-structure), in which the first element is inserted from one end called the **REAR**(also called **tail**), and the removal of existing element takes place from the other end called as **FRONT**(also called **head**).  This makes queue as **FIFO**(First in First Out) data structure, which means that element inserted first will be removed first.  Which is exactly how queue system works in real world. If you go to a ticket counter to buy movie tickets, and are first in the queue, then you will be the first one to get the tickets. Right? Same is the case with Queue data structure. Data inserted first, will leave the queue first.  The process to add an element into queue is called **Enqueue** and the process of removal of an element from queue is called **Dequeue**.  Introduction to Queue **Basic features of Queue**  1. Like stack, queue is also an ordered list of elements of similar data types. 2. Queue is a FIFO( First in First Out ) structure. 3. Once a new element is inserted into the Queue, all the elements inserted before the new element in the queue must be removed, to remove the new element. 4. peek( ) function is oftenly used to return the value of first element without dequeuing it. |
| **Algorithm (Student Work Area):** |
| **Code (Student Work Area):**  /\*  Problem Statement  Write a program to implement Following operations using Queue:  1.Enqueue()  2.Dequeue()  3.Isfull()  4.Isempty()  5.Peek()  a) Using array implementation  b) Using Linked List Implementation  \*/  class queueArray {      int front, rear, size;      int capacity;      int array[];      public queueArray(int capacity) {          this.capacity = capacity;          front = this.size = 0;          rear = capacity - 1;          array = new int[this.capacity];      }      boolean isFull(queueArray queue) {          return (queue.size == queue.capacity);      }      boolean isEmpty(queueArray queue) {          return (queue.size == 0);      }      void enqueue(int item) {          if (isFull(this))              return;          this.rear = (this.rear + 1) % this.capacity;          this.array[this.rear] = item;          this.size = this.size + 1;          System.out.println(item + " enqueued to queue");      }      int dequeue() {          if (isEmpty(this))              return Integer.MIN\_VALUE;          int item = this.array[this.front];          this.front = (this.front + 1) % this.capacity;          this.size = this.size - 1;          return item;      }      int front() {          if (isEmpty(this))              return Integer.MIN\_VALUE;          return this.array[this.front];      }      int rear() {          if (isEmpty(this))              return Integer.MIN\_VALUE;          return this.array[this.rear];      }  }  class queueLinkedList {      class Node {          int key;          Node next;          public Node(int key) {              this.key = key;              this.next = null;          }      }      Node front, rear;      public queueLinkedList() {          this.front = this.rear = null;      }      void isEmpty() {          if (this.front == null)              System.out.println("Queue is empty");      }      void enqueue(int key) {          Node temp = new Node(key);          if (this.rear == null) {              this.front = this.rear = temp;              return;          }          this.rear.next = temp;          this.rear = temp;          System.out.println(key + " enqueued to queue");      }      void dequeue() {          if (this.front == null)              return;          Node temp = this.front;          this.front = this.front.next;          if (this.front == null)              this.rear = null;          System.out.println(temp.key + " dequeued from queue");      }  }  /\*\*   \* problem\_statement\_experiment\_12   \*/  public class problem\_statement\_experiment\_12 {      public static void main(String[] args) {          System.out.println("Using Array Iimplementation\n");          queueArray queue = new queueArray(1000);          queue.enqueue(10);          queue.enqueue(20);          queue.enqueue(30);          queue.enqueue(40);          System.out.println(queue.dequeue() + " dequeued from queue");          System.out.println();          System.out.println("Front item is " + queue.front());          System.out.println("Rear item is " + queue.rear());          System.out.println("\n");          System.out.println("Using Linked List Implementation\n");          queueLinkedList queue1 = new queueLinkedList();          queue1.enqueue(10);          queue1.enqueue(20);          queue1.dequeue();          queue1.dequeue();          queue1.enqueue(30);          queue1.enqueue(40);          queue1.enqueue(50);          queue1.dequeue();          System.out.println();          System.out.println("Queue Front : " + queue1.front.key);          System.out.println("Queue Rear : " + queue1.rear.key);      }  } |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. What are the applications of queues?   Applications of Queues are:   * Disk Scheduling * CPU Scheduling * Mail Queues  1. Queues can be implemented with the help of stack. How?   Queues can be implemented with the help of two stacks in two ways, either by making enqueue operation costly or dequeue operation costly. |

**EXPERIMENT NO. 13**

|  |
| --- |
| **Student Name and Roll Number:** Piyush Gambhir |
| **Semester /Section:** Semester-III – AIML-B |
| **Link to Code:** [Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III (github.com)](https://github.com/Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III) |
| **Date:** |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective(s):**  To familiarize the students with linear data structure Circular Queue and its applications. |
| **Outcome:**  The students will be able to implement and use Circular Queues for solving various problems |
| **Problem Statement:**  Write a program to implement Following operations using Circular Queue:   1. Enqueue() 2. Dequeue()   Using array implementation |
| **Background:**  **Queue** is also an abstract data type or a linear data structure, just like [stack data structure](https://www.studytonight.com/data-structures/stack-data-structure), in which the first element is inserted from one end called the **REAR**(also called **tail**), and the removal of existing element takes place from the other end called as **FRONT**(also called **head**).  This makes queue as **FIFO**(First in First Out) data structure, which means that element inserted first will be removed first.  Which is exactly how queue system works in real world. If you go to a ticket counter to buy movie tickets, and are first in the queue, then you will be the first one to get the tickets. Right? Same is the case with Queue data structure. Data inserted first, will leave the queue first.  The process to add an element into queue is called **Enqueue** and the process of removal of an element from queue is called **Dequeue**.  Circular Queue in C++ **Basic features of Queue**  1. Like stack, queue is also an ordered list of elements of similar data types. 2. Queue is a FIFO( First in First Out ) structure. 3. Once a new element is inserted into the Queue, all the elements inserted before the new element in the queue must be removed, to remove the new element. 4. peek( ) function is oftenly used to return the value of first element without dequeuing it. |
| **Algorithm (Student Work Area):** |
| **Code (Student Work Area):**  /\*  Write a program to implement Following operations using Circular Queue:  1. Enqueue()  2. Dequeue()  Using array implementation  \*/  class circularQueueArray {      int front, rear, size;      int capacity;      int array[];      public circularQueueArray(int capacity) {          this.capacity = capacity;          front = this.size = 0;          rear = capacity - 1;          array = new int[this.capacity];      }      boolean isFull(circularQueueArray queue) {          return (queue.size == queue.capacity);      }      boolean isEmpty(circularQueueArray queue) {          return (queue.size == 0);      }      void enqueue(int item) {          if (isFull(this))              return;          this.rear = (this.rear + 1) % this.capacity;          this.array[this.rear] = item;          this.size = this.size + 1;          System.out.println(item + " enqueued to queue");      }      int dequeue() {          if (isEmpty(this))              return Integer.MIN\_VALUE;          int item = this.array[this.front];          this.front = (this.front + 1) % this.capacity;          this.size = this.size - 1;          return item;      }      int front() {          if (isEmpty(this))              return Integer.MIN\_VALUE;          return this.array[this.front];      }      int rear() {          if (isEmpty(this))              return Integer.MIN\_VALUE;          return this.array[this.rear];      }  }  /\*\*   \* problem\_statement\_experiment\_13   \*/  public class problem\_statement\_experiment\_13 {      public static void main(String[] args) {          circularQueueArray queue = new circularQueueArray(1000);          queue.enqueue(10);          queue.enqueue(20);          queue.enqueue(30);          queue.enqueue(40);          System.out.println();          System.out.println(queue.dequeue() + " dequeued from queue");      }  } |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. What are the applications of Circular queues?   Applications of Circular Queues are:   * Memory Management * Process Scheduling * Traffic Systems  1. What is the complexity of all operations in Circular Queue?   The complexity of all the operations of a circular queue implemented using arrays is O(1). |

**EXPERIMENT NO. 14**

|  |
| --- |
| **Student Name and Roll Number:** Piyush Gambhir |
| **Semester /Section:** Semester-III – AIML-B |
| **Link to Code:** [Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III (github.com)](https://github.com/Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III) |
| **Date:** |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective(s):**  To familiarize the students with linear data structure Doubly Ended Queue and its applications. |
| **Outcome:**  The students will be able to implement and use Doubly ended Queues for solving various problems |
| **Problem Statement:**  Write a program to implement Following operations using Doubly ended Queue:   1. Enqueue() 2. Dequeue() 3. Isfull() 4. Isempty() 5. Peek()   Using array implementation |
| **Background:**  **Queue** is also an abstract data type or a linear data structure, just like [stack data structure](https://www.studytonight.com/data-structures/stack-data-structure), in which the first element is inserted from one end called the **REAR**(also called **tail**), and the removal of existing element takes place from the other end called as **FRONT**(also called **head**).  This makes queue as **FIFO**(First in First Out) data structure, which means that element inserted first will be removed first.  Which is exactly how queue system works in real world. If you go to a ticket counter to buy movie tickets, and are first in the queue, then you will be the first one to get the tickets. Right? Same is the case with Queue data structure. Data inserted first, will leave the queue first.  The process to add an element into queue is called **Enqueue** and the process of removal of an element from queue is called **Dequeue**.   **Basic features of Queue**  1. Like stack, queue is also an ordered list of elements of similar data types. 2. Queue is a FIFO( First in First Out ) structure. 3. Once a new element is inserted into the Queue, all the elements inserted before the new element in the queue must be removed, to remove the new element. 4. peek( ) function is oftenly used to return the value of first element without dequeuing it. |
| **Algorithm (Student Work Area):** |
| **Code (Student Work Area):**  /\*  Problem Statement  Write a program to implement Following operations using Doubly ended Queue:  1. Enqueue()  2. Dequeue()  3. Isfull()  4. Isempty()  5. Peek()  Using array implementation  \*/  class doublyEndedQueueArray {      int front, rear, size;      int capacity;      int array[];      public doublyEndedQueueArray(int capacity) {          this.capacity = capacity;          front = this.size = 0;          rear = capacity - 1;          array = new int[this.capacity];      }      boolean isFull(doublyEndedQueueArray queue) {          return (queue.size == queue.capacity);      }      boolean isEmpty(doublyEndedQueueArray queue) {          return (queue.size == 0);      }      void enqueueFront(int item) {          if (isFull(this))              return;          this.front = (this.front - 1 + this.capacity) % this.capacity;          this.array[this.front] = item;          this.size = this.size + 1;          System.out.println(item + " enqueued to front");      }      void enqueueRear(int item) {          if (isFull(this))              return;          this.rear = (this.rear + 1) % this.capacity;          this.array[this.rear] = item;          this.size = this.size + 1;          System.out.println(item + " enqueued to rear");      }      int dequeueFront() {          if (isEmpty(this))              return Integer.MIN\_VALUE;          int item = this.array[this.front];          this.front = (this.front + 1) % this.capacity;          this.size = this.size - 1;          return item;      }      int dequeueRear() {          if (isEmpty(this))              return Integer.MIN\_VALUE;          int item = this.array[this.rear];          this.rear = (this.rear - 1 + this.capacity) % this.capacity;          this.size = this.size - 1;          return item;      }      int front() {          if (isEmpty(this))              return Integer.MIN\_VALUE;          return this.array[this.front];      }      int rear() {          if (isEmpty(this))              return Integer.MIN\_VALUE;          return this.array[this.rear];      }  }  /\*\*   \* problem\_statement\_experiment\_14   \*/  public class problem\_statement\_experiment\_14 {      public static void main(String[] args) {          doublyEndedQueueArray queue = new doublyEndedQueueArray(1000);          queue.enqueueFront(10);          queue.enqueueRear(20);          queue.enqueueFront(30);          queue.enqueueRear(40);          System.out.println();          System.out.println(queue.dequeueFront() + " dequeued from front");          System.out.println(queue.dequeueRear() + " dequeued from rear");          System.out.println();          System.out.println("Front item is " + queue.front());          System.out.println("Rear item is " + queue.rear());          System.out.println();          System.out.println("Is queue full: " + queue.isFull(queue));          System.out.println("Is queue empty: " + queue.isEmpty(queue));      }  } |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. What are the applications of Doubly ended queues?   Applications of Doubly Ended Queues are:  • Storing a web browser’s history.  • Storing a software application’s list of undo operations.  • Job scheduling algorithm   1. What is the complexity of all operations?   The complexity of all operations in a dequeue implemented using array is O (1). |

**EXPERIMENT NO. 15**

|  |
| --- |
| **Student Name and Roll Number:** Piyush Gambhir |
| **Semester /Section:** Semester-III – AIML-B |
| **Link to Code:** [Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III (github.com)](https://github.com/Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III) |
| **Date:** |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective(s):**  To familiarize the students with Non-linear data structure Binary Search Tree and its operations. |
| **Outcome:**  The students will be able to implement and use Binary Search Tree for solving various problems |
| **Problem Statement:**  Write a program to implement Following operations using Binary Search Tree:   1. Insertion 2. Deletion 3. Traversal [PREORDER, POSTORDER, INORDER] |
| **Background:**  **Binary Search Tree** is a node-based binary tree data structure which has the following properties:   * The left subtree of a node contains only nodes with keys lesser than the node’s key. * The right subtree of a node contains only nodes with keys greater than the node’s key. * The left and right subtree each must also be a binary search tree.     **Insertion In Binary Search Tree:**  1. Start from the root.  2. Compare the inserting element with root, if less than root, then recurse for left, else recurse for right.  3. After reaching the end, just insert that node at left(if less than current) else right.  **Deletion from Binary Search Tree:**  **1)*Node to be deleted is the*** ***leaf:*** Simply remove from the tree.  50 50  / \ delete(20) / \  30 70 ---------> 30 70  / \ / \ \ / \  20 40 60 80 40 60 80  **2) *Node to be deleted has only one child:*** Copy the child to the node and delete the child  50 50  / \ delete(30) / \  30 70 ---------> 40 70  \ / \ / \  40 60 80 60 80  **3) *Node to be deleted has two children:***Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor. Note that inorder predecessor can also be used.  50 60  / \ delete(50) / \  40 70 ---------> 40 70  / \ \  60 80 80 |
| **Algorithm (Student Work Area):** |
| **Code (Student Work Area):**  /\*  Problem Statement  Write a program to implement Following operations using Binary Search Tree:  1.Insertion  2.Deletion  3.Traversal [PREORDER, POSTORDER, INORDER]  \*/  class BinarySearchTree {      static class Node {          int key;          Node left, right;          public Node(int item) {              key = item;              left = right = null;          }      }      Node root;      BinarySearchTree() {          this.root = null;      }      BinarySearchTree(int r) {          this.root = new Node(r);      }      Node insertNode(Node root, int key) {          if (root == null) {              root = new Node(key);              return root;          }          if (key < root.key)              root.left = insertNode(root.left, key);          else if (key > root.key)              root.right = insertNode(root.right, key);          return root;      }      Node deleteNode(Node root, int key) {          if (root == null)              return root;          if (key < root.key)              root.left = deleteNode(root.left, key);          else if (key > root.key)              root.right = deleteNode(root.right, key);          else {              if (root.left == null)                  return root.right;              else if (root.right == null)                  return root.left;              root.key = minValue(root.right);              root.right = deleteNode(root.right, root.key);          }          return root;      }      int minValue(Node root) {          int minv = root.key;          while (root.left != null) {              minv = root.left.key;              root = root.left;          }          return minv;      }      void inorder(Node root) {          if (root != null) {              inorder(root.left);              System.out.print(root.key + " ");              inorder(root.right);          }      }      void preorder(Node root) {          if (root != null) {              System.out.print(root.key + " ");              preorder(root.left);              preorder(root.right);          }      }      void postorder(Node root) {          if (root != null) {              postorder(root.left);              postorder(root.right);              System.out.print(root.key + " ");          }      }  }  /\*\*   \* problem\_statement\_experiment\_15   \*/  public class problem\_statement\_experiment\_15 {      public static void main(String[] args) {          BinarySearchTree tree = new BinarySearchTree(65);          tree.insertNode(tree.root, 30);          tree.insertNode(tree.root, 20);          tree.insertNode(tree.root, 40);          tree.insertNode(tree.root, 70);          tree.insertNode(tree.root, 60);          tree.insertNode(tree.root, 80);          System.out.println("Inorder traversal of the given tree");          tree.inorder(tree.root);          System.out.println();          System.out.println("Preorder traversal of the given tree");          tree.preorder(tree.root);          System.out.println();          System.out.println("Postorder traversal of the given tree");          tree.postorder(tree.root);          System.out.println("\n");          System.out.println("Delete 20");          tree.deleteNode(tree.root, 20);          System.out.println("Inorder traversal of the modified tree");          tree.inorder(tree.root);          System.out.println("\n");          System.out.println("Delete 30");          tree.deleteNode(tree.root, 30);          System.out.println("Inorder traversal of the modified tree");          tree.inorder(tree.root);      }  } |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. What is the difference between Binary Tree and Binary Search Tree?   A Binary Tree is a basic structure with a simple rule that no parent must have more than 2 children whereas the Binary Search Tree is a variant of the binary tree following a particular order with which the nodes should be organized.   1. What is the complexity of all search operations in BST?   For insertion and deletion worst case time complexity is O(n), but in general the time complexity is O(h).  For all types of traversals, the time complexity is O(n). |

**EXPERIMENT NO. 16**

|  |
| --- |
| **Student Name and Roll Number:** Piyush Gambhir |
| **Semester /Section:** Semester-III – AIML-B |
| **Link to Code:** [Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III (github.com)](https://github.com/Piyush-Gambhir/Data-Structures-Lab-Manual-Semester-III) |
| **Date:** |
| **Faculty Signature:** |
| **Marks:** |

|  |
| --- |
| **Objective(s):**  To familiarize the students with different sorting operations. |
| **Outcome:**  The students will be able to implement and use various sorting techniques. |
| **Problem Statement:**  Write a program to implement:   1. Bubble Sort 2. Insertions Sort 3. Selection Sort 4. Quick Sort 5. Merge Sort |
| **Background:**  Sorting is the process of arranging the elements of an array so that they can be placed either in ascending or descending order. For example, consider an array A = {A1, A2, A3, A4, …. An }, the array is called to be in ascending order if element of A are arranged like A1 > A2 > A3 > A4 > A5 > .. > An .  **Consider an array;**  int A[10] = { 5, 4, 10, 2, 30, 45, 34, 14, 18, 9 )  After Sorting array would be:  A[] = { 2, 4, 5, 9, 10, 14, 18, 30, 34, 45 }  There are many techniques by using which, sorting can be performed.   |  |  |  | | --- | --- | --- | | **SN** | **Sorting Algorithms** | **Description** | | 1 | [Bubble Sort](https://www.javatpoint.com/bubble-sort) | It is the simplest sort method which performs sorting by repeatedly moving the largest element to the highest index of the array. It comprises of comparing each element to its adjacent element and replace them accordingly. | | 2 | [Insertion Sort](https://www.javatpoint.com/insertion-sort) | As the name suggests, insertion sort inserts each element of the array to its proper place. It is a very simple sort method which is used to arrange the deck of cards while playing bridge. | | 3 | [Merge Sort](https://www.javatpoint.com/merge-sort) | Merge sort follows divide and conquer approach in which, the list is first divided into the sets of equal elements and then each half of the list is sorted by using merge sort. The sorted list is combined again to form an elementary sorted array. | | 4 | [Quick Sort](https://www.javatpoint.com/quick-sort) | Quick sort is the most optimized sort algorithms which performs sorting in O(n log n) comparisons. Like Merge sort, quick sort also work by using divide and conquer approach. | | 5 | [Selection Sort](https://www.javatpoint.com/selection-sort) | Selection sort finds the smallest element in the array and place it on the first place on the list, then it finds the second smallest element in the array and place it on the second place. This process continues until all the elements are moved to their correct ordering. It carries running time O(n2) which is worst than insertion sort. | |
| **Algorithm (Student Work Area):**  **Bubble Sort**  for (int i = 0; i < n; i++) {    for (int j = 0; j < n - 1; j++) {  // Swap adjacent elements if they are in decreasing order  if (a[j] > a[j + 1]) {  swap(a[j], a[j + 1]);  }  }    } |
| **Code (Student Work Area):**  /\*  Problem Statement  Write a program to implement:  1. Bubble Sort  2. Insertions Sort  3. Selection Sort  4. Quick Sort  5. Merge Sort  \*/  /\*\*   \* problem\_statement\_experiment\_16   \*/  public class problem\_statement\_experiment\_16 {      public static void bubbleSort(int[] arr) {          int n = arr.length;          int temp = 0;          for (int i = 0; i < n; i++) {              for (int j = 1; j < (n - i); j++) {                  if (arr[j - 1] > arr[j]) {                      // swap elements                      temp = arr[j - 1];                      arr[j - 1] = arr[j];                      arr[j] = temp;                  }              }          }      }      public static void insertionSort(int array[]) {          int n = array.length;          for (int j = 1; j < n; j++) {              int key = array[j];              int i = j - 1;              while ((i > -1) && (array[i] > key)) {                  array[i + 1] = array[i];                  i--;              }              array[i + 1] = key;          }      }      public static void selectionSort(int[] arr) {          for (int i = 0; i < arr.length - 1; i++) {              int index = i;              for (int j = i + 1; j < arr.length; j++)                  if (arr[j] < arr[index])                      index = j;              int smallerNumber = arr[index];              arr[index] = arr[i];              arr[i] = smallerNumber;          }      }      public static void quickSort(int[] arr, int start, int end) {          if (start < end) {              int partitionIndex = partition(arr, start, end);              quickSort(arr, start, partitionIndex - 1);              quickSort(arr, partitionIndex + 1, end);          }      }      public static int partition(int[] arr, int start, int end) {          int pivot = arr[end];          int i = (start - 1);          for (int j = start; j < end; j++) {              if (arr[j] <= pivot) {                  i++;                  int temp = arr[i];                  arr[i] = arr[j];                  arr[j] = temp;              }          }          int temp = arr[i + 1];          arr[i + 1] = arr[end];          arr[end] = temp;          return i + 1;      }      public static void mergeSort(int[] arr, int l, int r) {          if (l < r) {              int m = (l + r) / 2;              mergeSort(arr, l, m);              mergeSort(arr, m + 1, r);              merge(arr, l, m, r);          }      }      public static void merge(int[] arr, int l, int m, int r) {          int n1 = m - l + 1;          int n2 = r - m;          int L[] = new int[n1];          int R[] = new int[n2];          for (int i = 0; i < n1; ++i)              L[i] = arr[l + i];          for (int j = 0; j < n2; ++j)              R[j] = arr[m + 1 + j];          int i = 0, j = 0;          int k = l;          while (i < n1 && j < n2) {              if (L[i] <= R[j]) {                  arr[k] = L[i];                  i++;              } else {                  arr[k] = R[j];                  j++;              }              k++;          }          while (i < n1) {              arr[k] = L[i];              i++;              k++;          }          while (j < n2) {              arr[k] = R[j];              j++;              k++;          }      }      public static void printArray(int[] arr) {          int n = arr.length;          for (int i = 0; i < n; ++i)              System.out.print(arr[i] + " ");          System.out.println();      }      public static void main(String[] args) {          int[] arr = { 9, 14, 3, 2, 43, 11, 58, 22 };          System.out.println("Given Array");          printArray(arr);          bubbleSort(arr);          System.out.println("Bubble Sort");          printArray(arr);          System.out.println();          int[] arr1 = { 9, 14, 3, 2, 43, 11, 58, 22 };          System.out.println("Given Array");          printArray(arr);          insertionSort(arr1);          System.out.println("Insertion Sort");          printArray(arr1);          System.out.println();          int[] arr2 = { 9, 14, 3, 2, 43, 11, 58, 22 };          System.out.println("Given Array");          printArray(arr);          selectionSort(arr2);          System.out.println("Selection Sort");          printArray(arr2);          System.out.println();          int[] arr3 = { 9, 14, 3, 2, 43, 11, 58, 22 };          System.out.println("Given Array");          printArray(arr);          quickSort(arr3, 0, arr3.length - 1);          System.out.println("Quick Sort");          printArray(arr3);          System.out.println();          int[] arr4 = { 9, 14, 3, 2, 43, 11, 58, 22 };          System.out.println("Given Array");          printArray(arr);          mergeSort(arr4, 0, arr4.length - 1);          System.out.println("Merge Sort");          printArray(arr4);      }  } |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. **Compare and contrast all Sorting techniques?** |