**Data Structures**

**(CSL 209)**

**Lab Workbook**



Faculty name: Student name:

Roll No.:

Semester:

Group:

**Department of Computer Science and Engineering**

**The NorthCap University, Gurugram- 122017, India**

**Session 2024-25**

**INDEX**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S.No** | **Experiment** | **Date of Experiment** | **Date of Submission** | **Student Sign** | **CO Covered** | **Faculty**  **Sign** |
| 1 | Create an array of integer with size n. Return the difference between the largest and the smallest value inside that array. |  |  |  | CO1 |  |
| 2 | 1. 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 element |  |  |  | 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 |  |  |  | CO2 |  |
| 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 |  |  |  | CO2 |  |
| 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   At a specific location |  |  |  | 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 |  |  |  | CO2 |  |
| 7 | Write a program to implement stack using arrays and linked lists. |  |  |  | CO3 |  |
| 8 | Write a program to reverse a sentence using stack. |  |  |  | CO3 |  |
| 9 | Write a program to check for balanced parenthesis in a given expression. |  |  |  | CO3 |  |
| 10 | Write a program to convert infix expression to prefix and postfix expression. |  |  |  | CO3 |  |
| 11 | Write a program to implement Tower of Hanoi using stacks |  |  |  | CO3 |  |
| 12 | Write a program to implement Linear Queue using Array and Linked Lists. |  |  |  | CO3 |  |
| 13 | Write a program to implement Circular Queue using Array and Linked Lists. |  |  |  | CO3 |  |
| 14 | Write a program to implement Doubly Ended Queue using Array and Linked Lists. |  |  |  | CO3 |  |
| 15 | Write a Program to implement Binary Search Tree operations. |  |  |  | CO4 |  |
| 16 | Write a program to implement Bubble Sort, Selection Sort, Quick Sort, Merge Sort and Insertion Sort algorithm. |  |  |  | CO5 |  |

**EXPERIMENT NO. 1**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **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. |
|  |
| **Code (Student Work Area):**  **Write a program in java to 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;  public class ArrayDifference {  // Method to find the maximum value in an array  public static int findMax(int[] arr) {  int max = arr[0];  for (int i = 1; i < arr.length; i++) {  if (arr[i] > max) {  max = arr[i];  }  }  return max;  }  // Method to find the minimum value in an array  public static int findMin(int[] arr) {  int min = arr[0];  for (int i = 1; i < arr.length; i++) {  if (arr[i] < min) {  min = arr[i];  }  }  return min;  }  // Main method  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);    // Input size of the array  System.out.print("Enter the size of the array: ");  int n = scanner.nextInt();    // Declare and initialize the array  int[] arr = new int[n];    // Input elements of the array  System.out.println("Enter " + n + " elements:");  for (int i = 0; i < n; i++) {  arr[i] = scanner.nextInt();  }    // Find maximum and minimum values  int max = findMax(arr);  int min = findMin(arr);    // Calculate and print the difference  int difference = max - min;  System.out.println("The difference between the largest and smallest values is: " + difference);  }  }    Q2) to enter an array  package Lakshya;  import java.util.Scanner;  public class ArrayInput  {  public static void main(String[] args)  {  Scanner sc = new Scanner(System.*in*);  System.*out*.print("Enter the length of the array: ");  int length = sc.nextInt();  int[] array = new int[length];  for (int i = 0; i < length; i++)  {  System.*out*.print("Enter element " + (i + 1) + ": ");  array[i] = sc.nextInt();  }  System.*out*.print("The array you entered is: ");  for (int i : array)  {  System.*out*.print(i + " ");  }  }  }    Q3) to find element from array and check whether there is duplicate or not ?  package Lakshya;  import java.util.HashSet;  import java.util.Scanner;  import java.util.Set;  public class demo2 {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  System.out.print("Enter the length of the array: ");  int length = scanner.nextInt();  int[] array = new int[length];  for (int i = 0; i < length; i++) {  System.out.print("Enter element " + (i + 1) + ": ");  array[i] = scanner.nextInt();  }  System.out.print("The array you entered is: ");  for (int i : array) {  System.out.print(i + " ");  }  System.out.println();  System.out.print("Enter the element to find: ");  int elementToFind = scanner.nextInt();    boolean found = false;  for (int i = 0; i < length; i++) {  if (array[i] == elementToFind) {  System.out.println("Element " + elementToFind + " found at index " + i);  found = true;  break;  }  }  if (!found) {  System.out.println("Element " + elementToFind + " not found in the array.");  }  Set<Integer> set = new HashSet<>();  boolean duplicatesExist = false;  System.out.print("Duplicate values in the array: ");  for (int i = 0; i < length; i++) {  if (!set.add(array[i])) {  System.out.print(array[i] + " ");  duplicatesExist = true;  }  }  if (!duplicatesExist) {  System.out.print("None");  }  System.out.println();  scanner.close();  }  } |
| **Question Bank:**   1. What is Data Structure?   Structure and organisation of data in memory with related operation.   1. Why Array is called as Linear Data Structure?   Because they are allocated contiguous memory locations.   1. What type of Indexing is used in Java?   Multikey indexing. First element of array is indexed as 0 and last as ‘size-1’.   1. How to find the missing number in integer array of 1 to 100?   Traverse through the array from 0 to size-1 and collect missing numbers.   1. How to find the second-highest value in a numeric array.   Find the maximum value of array. Eliminate it and find the maxima of remaining elements as ‘second highest’ value.   1. How to swap the first and last elements of an array.   temp= array[0];  array[0]=array[size-1];  arrary[size-1]=temp;   1. Write a Java Program to check if see if Array contains a specific value. (Linear Search)   int elem=4;  int array[10]={1,3,5,6,7,3,5,4,9,10};  for (int i=0; i<10;i++){  if(array[i]==elem)  {  System.out.println(“Element found at position=”+(i+1));  break;  }  } |

**EXPERIMENT NO. 2**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **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:  * Every element at an even index * Every odd element * All elements in reverse order * 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):** |
| **Code (Student Work Area):**   1. Problem Statement:   Write a java 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 element  package Lakshya;  import java.util.Random;  public class demo4 {  public static void main(String[] args) {  // Initialize the array with 10 random integers  int[] arr = new int[10];  Random rand = new Random();  for (int i = 0; i < arr.length; i++) {  arr[i] = rand.nextInt(100); // Random integers between 0 and 99  }  // Print the original array  System.out.println("Original array:");  for (int num : arr) {  System.out.print(num + " ");  }  System.out.println("\n");  // Print every element at an even index  System.out.println("Elements at even indices:");  for (int i = 0; i < arr.length; i += 2) {  System.out.print(arr[i] + " ");  }  System.out.println();  // Print every odd element  System.out.println("Odd elements:");  for (int num : arr) {  if (num % 2 != 0) {  System.out.print(num + " ");  }  }  System.out.println();  // Print all elements in reverse order  System.out.println("Elements in reverse order:");  for (int i = arr.length - 1; i >= 0; i--) {  System.out.print(arr[i] + " ");  }  System.out.println();  // Print only the first and last element  System.out.println("First and last elements:");  if (arr.length > 0) {  System.out.print(arr[0] + " ");  if (arr.length > 1) {  System.out.print(arr[arr.length - 1]);  }  }  System.out.println();  }  } |
| **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. How to reverse the array elements. 3. How to find the index of an array element 4. How to remove a specific element from an array. 5. How to insert an element (specific position) into an array. |

**EXPERIMENT NO. 3**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **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):** |
| **Code (Student Work Area):**   1. Problem Statement:   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  import java.util.Scanner;  public class ArrayOperations {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  int[] arr = new int[5];    // Read 5 numbers into the array  System.out.println("Enter 5 integers:");  for (int i = 0; i < arr.length; i++) {  arr[i] = scanner.nextInt();  }  // Sum of all elements  int totalSum = 0;  for (int num : arr) {  totalSum += num;  }  System.out.println("Sum of all elements: " + totalSum);  // Sum of alternate elements (considering the first element as alternate)  int alternateSum = 0;  for (int i = 0; i < arr.length; i += 2) {  alternateSum += arr[i];  }  System.out.println("Sum of alternate elements: " + alternateSum);  // Finding the second highest element in the array  if (arr.length < 2) {  System.out.println("Array must have at least two elements to find the second highest element.");  return;  }  int highest = Integer.MIN\_VALUE;  int secondHighest = Integer.MIN\_VALUE;  for (int num : arr) {  if (num > highest) {  secondHighest = highest;  highest = num;  } else if (num > secondHighest && num != highest) {  secondHighest = num;  }  }  // Check if secondHighest was updated  if (secondHighest == Integer.MIN\_VALUE) {  System.out.println("No second highest element found.");  } else {  System.out.println("Second highest element: " + secondHighest);  }  }  } |
| **Question Bank:**   1. How we can count occurrence of a given number in the array and its frequency.   2. How we can print the following in 2-D integer array with each element of maximum 2 digits  a) Elements of the entered array.  b) Elements of the array after each element is multiplied by 2 if it is an odd number.  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. 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.  5. 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. |

**EXPERIMENT NO. 4**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **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):** |
| **Code (Student Work Area):** |
| **Output – Screenshots (Student Work Area):** |
| **Question Bank:**  How Linked List id different from Arrays?  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 * Pairwise swap elements of a given linked list * Get the location of first and last occurrence of an element in a single LinkedList * Remove duplicates from an unsorted linked list * Delete alternate nodes of a Linked List. |

**EXPERIMENT NO. 5**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **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):** |
| **Code (Student Work Area):** |
| **Output – Screenshots (Student Work Area):** |
| **Question Bank:**   1. What is Doubly Linked List? 2. What is the complexity of Traversal, Insertion and deletion operations in doubly linked list? |

**EXPERIMENT NO. 6**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **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):** |
| **Output – Screenshots (Student Work Area):** |
| **Question Bank:**  How Circular Linked List id different from Singly Linked List?  Analyze the complexity of Traversal, insertion and Deletion operations in Linked List? |

**EXPERIMENT NO. 7**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **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:   * POP * PUSH * PEEK * ISEMPTY * ISFULL  1. Use Arrays for Implementation 2. 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):** |
| **Code (Student Work Area):** |
| **Output – Screenshots (Student Work Area):** |
| **Question Bank:**  What are Stacks?  **What are the applications of stacks?** |

**EXPERIMENT NO. 8**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **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):** |
| **Code (Student Work Area):** |
| **Output – Screenshots (Student Work Area):** |
| **Question Bank:**  What are Stacks?  **How we can split a sentence and push it into the stack?** |

**EXPERIMENT NO. 9**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **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):** |
| **Code (Student Work Area):** |
| **Output – Screenshots (Student Work Area):** |
| **Q: How a stack helps in syntax analysis or compilation of a program?** |

**EXPERIMENT NO. 10**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **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 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):** |
| **Code (Student Work Area):** |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. **Why conversion is required?** 2. **How we can convert infix to prefix and prefix to postfix?** |

**EXPERIMENT NO. 11**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **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):** |
| **Code (Student Work Area):** |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. **What is Recursion?** 2. **What is Base condition?** 3. **What are the number of steps required to solve n-Disc problem?** |

**EXPERIMENT NO. 12**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **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):** |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. **What are the applications of queues?** 2. **Queues can be implemented with the help of stack. How?** |

**EXPERIMENT NO. 13**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **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):** |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. **What are the applications of Circular queues?** 2. **What is the complexity of all operations in Circular Queue?** |

**EXPERIMENT NO. 14**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **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):** |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. **What are the applications of Doubly ended queues?** 2. **What is the complexity of all operations?** |

**EXPERIMENT NO. 15**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **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):** |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. **What is the difference between Binary Tree and Binary Search Tree?** 2. **What is the complexity of all search operations in BST?** |

**EXPERIMENT NO. 16**

|  |
| --- |
| **Student Name and Roll Number:** |
| **Semester /Section:** |
| **Link to Code:** |
| **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):** |
| **Code (Student Work Area):** |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**  **Compare and contrast all Sorting techniques?** |

**17.**