**Data Structures**

**(CSL 209)**

**Lab Practical Workbook**



Faculty name: Dr. Ruchika Lalit Student name: Arjun Bhardwaj

Roll No.: 21CSU211

Semester: 3rd

Group: FS-A (Group-1

**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** |
| **1** | **Create an array of integer with**  **size n. Return the difference**  **between the largest and the**  **smallest value inside that array.** | **9-15** | **04/08/22** | **23/11/22** |  | **CO1** |  |
| **2** | **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** | **16-20** | **18/08/22** | **23/11/22** |  | **CO2** |  |
| **3** | **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** | **21-26** | **01/09/22** | **23/11/22** |  | CO1 |  |
| **4** | **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** | **27-38** | **01/09/22** | **23/11/22** |  | **CO1** |  |
| **5** | **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**  ** At a specific location** | **39-54** | **08/09/22** | **23/11/22** |  | **CO2** |  |
| **6** | **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**  **At a specific location** | **55-68** | **08/09/22** | **23/11/22** |  | **CO2** |  |
| **7** | **Write a program to implement**  **stack using arrays and linked**  **lists.** | **69-84** | **15/09/22** | **23/11/22** |  | **CO3** |  |
| **8** | **Write a program to reverse a**  **sentence using stack.** | **85-91** | **15/09/22** | **23/11/22** |  | **CO3** |  |
| **9** | **Write a program to check for**  **balanced parenthesis in a given**  **expression.** | **92-97** | **22/09/22** | **23/11/22** |  | **CO3** |  |
| **10** | **Write a program to convert infix**  **expression to prefix and postfix**  **expression.** | **98-105** | **22/09/22** | **23/11/22** |  | **CO3** |  |
| **11** | **Write a program to implement**  **Tower of Hanoi using stacks** | **106-109** | **06/10/22** | **23/11/22** |  | **CO3** |  |
| 12 | **Write a program to implement**  **Linear Queue using Array and**  **Linked Lists.** | **110-117** | **19/10/22** | **23/11/22** |  | **CO3** |  |
| **13** | **Write a program to implement**  **Circular Queue using Array and**  **Linked Lists.** | **118-128** | **27/10/22** | **23/11/22** |  | **CO4** |  |
| **14** | **Write a program to implement**  **Doubly Ended Queue using**  **Array and Linked Lists.** | **129-139** | **3/11/22** | **23/11/22** |  | **CO4** |  |
| **15** | **Write a Program to implement**  **Binary Search Tree operations.** | **140-151** | **10/11/22** | **23/11/22** |  | **CO4** |  |
| **16** | **Write a program to implement**  **Bubble Sort, Selection Sort,**  **Quick Sort, Merge Sort and**  **Insertion Sort algorithm.** | **152-176** | **17/11/22** | **23/11/22** |  | **CO6** |  |

**EXPERIMENT NO. 1**

|  |
| --- |
| **Student Name and Roll Number:** Arjun Bhardwaj & 21CSU211 |
| **Semester /Section:** 3rd /FS-A |
| **Link to Code:** |
| **Date: 04/08/22** |
| **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. START       2. Initialize arr[], len,i,max,min as variables       3. Input length of array(len).       4. Run a loop till i<len for input of array element.       5. Assign max and min as arr[0] element.       6. Run a loop till i<len for max and min value Within the loop Apply if statement for checking max and min value   {Max value: if arr[i]>max assign max=arr[i]}  {min value :if arr[i]<min assign min=arr[i]}   * + - 1. Print the desired outputs       2. END |
| **Code (Student Work Area):**  import java.util.\*;  import java.util.Scanner;  public class Array  {  public static void main(String[] args)  {  System.out.println("Size of the array is: ");  Scanner sc = new Scanner(System.in);  int size;  size = sc.nextInt();    System.out.println("Enter the array elements: ");  int arr[] = new int[size];  for(int i =0; i<size; i++)  {  arr[i] = sc.nextInt();  }  int min = arr[0];  int max = arr[0];  for(int i=0; i<size; i++)  {  if (arr[i]<min)  {  min = arr[i];  }  if (arr[i]>max)  {  max = arr[i];  }  }  System.out.println("The min array element is: "+min);  System.out.println("The max array element is: "+max);  System.out.println("The difference is: ");  System.out.println(max-min);  }  }  **Output (Student Work Area):** |
| **Question Bank:**   1. What is Data Structure?   Ans- A data structure is **a storage that is used to store and organize data**   1. Why Array is called as Linear Data Structure?   Ans- A linear array, is a list of finite numbers of elements stored in the memory. In a linear array, **we can store only homogeneous data elements**.   1. What type of Indexing is used in Java?   Ans-sequential and linked   1. How to find the missing number in integer array of 1 to 100?   Ans-  import java.util.\*;  class MissingNumber  {  public static void main(String[] args)  {  System.out.println("enter the size of the Array:");  Scanner sc=new Scanner(System.in);  int size=sc.nextInt();  int arr[]=new int[size];  for( int i=0;i<size-1;i++)  {  System.out.println("enter the elements of array:");  arr[i]=sc.nextInt();  }  int arrSum=0;  int sum=(size\*(size+1))/2;    for(int i=0;i<size-2;i++)  {  arrSum=arrSum+arr[i];  }  System.out.println("the missing number is:"+(sum-arrSum ));  }  }   1. How to find the second-highest value in a numeric array.   Ans-   * + - 1. START       2. Initialize maxone=0, maxtwo=0,len,dif,i,j       3. Find the length of the array       4. Run a loop i=0 till i<len with i=i+1   Check if(maxone<array[i])  And set maxtwo=maxone;maxone=array[i]  Check Else-if(maxtwo<Array[i])  And set maxtwo=array[i]   * + - 1. Dif=maxone-maxtwo       2. Print maxone,maxtwo and dif       3. END  1. How to swap the first and last elements of an array.    * + 1. START        2. Initialize array num[]={20,30,40} and x,len as variable        3. Find the length of the array (len)        4. Run loop till i=0;i<len;i=i+1   Print num[i]   * + - 1. Print (“Before swap”)       2. Set x= num[0] And num[0]=num[len-1] And num[len-1]=x       3. Run loop till i=0;i<len;i=i+1   Print num[i]   * + - 1. Print (“After Swap”)       2. END  1. Write a Java Program to check if see if Array contains a specific value. (Linear Search)   Ans-   1. Start 2. run loop from 0 to n   2. if the arr[i] is equal to the desired value then return it  3. Return the i index  4. if the desired element isn’t present then return -1. |

**EXPERIMENT NO. 2**

|  |
| --- |
| **Student Name and Roll Number:** Arjun Bhardwaj & 21CSU211 |
| **Semester /Section:** 3rd / FS-A |
| **Link to Code:** |
| **Date: 18/08/22** |
| **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):**   * + - 1. START       2. Initialize arr[] with 10 elements and i,m,j as variables       3. Run a loop i=0 till i<10 with i=i+2   Print elements at even index of the array   * + - 1. Run a loop i=0 till i<10 with i=i+1   Check if((arr[i]%2)!=0);  Print every odd elements of the array   * + - 1. Run a loop j=9 till j>=0 with j=j-1   Print elements of the array in reversed order   * + - 1. Print arr[0] and arr[9] as first and last element respectively       2. END |
| Code (Student Work Area):  import java.util.\*;  public class Arr  {  public static void main(String[] args)  {  int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  System.out.print(" elements at even index: ");  for (int i = 0; i < arr.length; i = i + 2)  {  System.out.print(arr[i] + (","));  }  System.out.print("\n");  System.out.print(" elements at odd index: ");  for (int i = 0; i < arr.length; i++)  {  int m = arr[i];  if ((m % 2) != 0)  {  System.out.print(arr[i] + ",");  }  }  System.out.println("\n\nDisplaying elements of Array in Reverse Order");    for (int i = arr.length - 1; i >= 0; i--)  {  System.out.print(arr[i] + " ");  }    System.out.println("\nFirst Element: " + arr[0] + "\nLast Element: " + arr[arr.length - 1]);  }  }  Output: |
| **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. Initialize 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   If (arr[i[==0)  {x=x+1}   1. Print zeros to the front x times 2. The remaining number of 1s will be 1- (x) 3. Print the remaining elements 4. END 5. How to reverse the array elements?    * + 1. START        2. Initialize arr[] with 10 elements and i,m,j as variables        3. Run a loop j=9 till j>=0 with j=j-1   Print elements of the array in reversed order  4. End   1. How to find the index of an array element?    * + 1. START        2. Initialize arr[]={1,2,3,4,5} and len,x as a variable        3. Find the length of the array(len)        4. Input the character who’s index is needed and store it in x        5. Run loop till i=0;i<len;i=i+1   Check if (arr[i]==x)  Print (“Index of character is:”+i)   * + - 1. End  1. How to remove a specific element from an array?    * + 1. START        2. Initialize arr[]={1,2,3,4,5} and len,x as a variable        3. Find the length of the array(len)        4. Input which element need to be deleted and store it in x        5. Input the character who’s index is needed and store it in x        6. 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  1. How to insert an element (specific position) into an array?   1. START  2. Initialize arr[] = {1, 2, 3, 4, 5} and, and len x,n,pos as a variable  3. Get the the variables n and pos as inputs for element and position respectively  4. Initialise a new array arr\_new[] with size one greater than arr[]  5. Copy all the elements from previous array into the new array till the position pos  6.  Insert the element n at position pos  7.  Insert the rest of the elements from the previous array into the new array after the pos  8. END |

**EXPERIMENT NO. 3**

|  |
| --- |
| **Student Name and Roll Number:** Arjun Bhardwaj & 21CSU211 |
| **Semester /Section:** 3rd / FS-A |
| **Link to Code:** |
| **Date: 01/09/22** |
| **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):**   * + - 1. START       2. Initialize arr[]={1,2,3,4,5} and sum =0,sum2=0 as variables       3. Run loop till i=0 till i<=4 withi=i+1   sum = sum + arr[i]   * + - 1. Run loop till i=0 till i<=4 withi=i+2   sum2 = sum2 + arr[i]   * + - 1. Initialize maxone=0, maxtwo=0,len,dif,i,j       2. Find the length of the array       3. Run a loop i=0 till i<len with i=i+1   Check if(maxone<array[i])  And set maxtwo=maxone;maxone=array[i]  Check Else-if(maxtwo<Array[i])  And set maxtwo=array[i]   * + - 1. Print maxtwo,sum,sum2   End |
| **Code (Student Work Area):**  import java.util.\*;  public class Arra {  public static void main(String[] args) {  int sum=0;  int altsum=0;  int arr[]=new int[5];  Scanner sc=new Scanner(System.in);  for (int i = 0; i < arr.length; i++) {  arr[i]=sc.nextInt();  }  int max=arr[0];  int t=0;  for (int i = 0; i < arr.length; i++) {  sum=sum+arr[i];  }  for (int i = 0; i < arr.length; i=i+2) {  altsum=altsum+arr[i];  }  for (int i = 0; i < arr.length; i++) {  if (arr[i]>max) {  t=max;  max=arr[i];  }  }  System.out.println("Sum of array:"+sum+"\n"+"Sum of alternate elements:"+altsum+"\n"+"Second Highest No:"+t);  }  **}**  **OUTPUT :-** |
| **Question Bank:**   1. How we can count occurrence of a given number in the array and its frequency. 2. Ans: Start 3. Initialize arr[]={1,2,3,4,5} and oc=0 as a variable 4. Enter the element whose frequency you want to know and store in x 5. 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   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.  1.START  2.input of no of rows and column as r and m respectively  3.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]}}  4.END  b) Elements of the array after each element is multiplied by 2 if it is an odd number.  1.START  2.input of no of rows and column as r and m respectively  3. 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  print a[i][j] }}  4.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.  Ans- 1. START  2. 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  3. 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  4. Print count6,count7  5. END  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.  1. START  2. Initialize arr[] = {10, 20, 30, 40, 50}, variable n, and variable x as length of arr  3. run i = 0 till i &lt; x with i=i+1)  If(i+1 != x)  n = arr[i]  arr[i] = arr[i+1]  arr[i+1] = n  4. Print arr  5. END   1. 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.   1.Start  2. Initialize minimum value min to  3. Traverse the array to find the minimum element in the array  4. While traversing if any element smaller than min\_idx is found then swap both the  values.  5. Then, increment min\_idx to point to next element  6. Repeat until array is sorted  7. Then, check if the number of elements present in the array is even or odd  8. If odd, then simply return the mid value of the array  9. Else, the median is the average of the two middle values  10. END |

**EXPERIMENT NO. 4**

|  |
| --- |
| **Student Name and Roll Number:** Arjun Bhardwaj & 21CSU211 |
| **Semester /Section:** 3rd / FS-A |
| **Link to Code:** CSL209 |
| **Date: 01/09/22** |
| **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):**   1. Create a class Node which has two attributes: data and next. Next is a pointer to the next node in the list. 2. Create another class which has two attributes: head and tail. 3. addNode() will add a new node to the list:    1. Create a new node.    2. It first checks, whether the head is equal to null which means the list is empty.    3. If the list is empty, both head and tail will point to the newly added node.    4. If the list is not empty, the new node will be added to end of the list such that tail's next will point to a newly added node. This new node will become the new tail of the list. 4. countNodes() will count the nodes present in the list:    1. Define a node current which will initially point to the head of the list.    2. Declare and initialize a variable count to 0.    3. Traverse through the list till current point to null.    4. Increment the value of count by 1 for each node encountered in the list. 5. display() will display the nodes present in the list:    1. Define a node current which will initially point to the head of the list.   Traverse through the list till current points to null |
| Code (Student Work Area):  import java.util.\*;  public class SinglyLL  {  static class Node  {  int data;  Node next;  Node(int data) {  this.data = data;  this.next = null;  }  }  Node head = null;  Scanner sc = new Scanner(System.in);  public void Creation()  {  int data, m, x;  Node temp = head;  do  {  System.out.println("Enter data: ");  data = sc.nextInt();  Node new\_node = new Node(data);    if (head == null)  {  head = new\_node;  }  else  {  System.out.println("Enter the number where you wish to insert: 1-beginning, 2-end, 3-a specefic location");  m = sc.nextInt();  switch (m)  {  case 1:  new\_node.next = head;  head = new\_node;  break;  case 2:  while (temp.next != null) {  temp = temp.next;  }  temp.next = new\_node.next;  break;  case 3:  System.out.println("Enter position where you want to insert the node: ");  int p = sc.nextInt();  Node temp1 = head;  for (int i = 0; i < (p - 1); i++) {  temp1 = temp1.next;  }  new\_node.next = temp1.next;  temp1.next = new\_node;  break;  }  }  System.out.println("Do you wish to continue? Press 1.");  x = sc.nextInt();  }  while (x == 1);  }  public void Deletion()  {  int data, n, m, p;  do  {  if (head == null)  {  System.out.println("LL is empty!");  }  else  {  System.out.println("Enter position from where you wish to delete the node: 1-beginning, 2-end, 3-specefic position");  m = sc.nextInt();  Node temp = head;  Node ptr = temp.next;  switch (m)  {  case 1:  temp = temp.next;  head = temp;  break;  case 2:  while (ptr.next != null)  {  temp = ptr;  ptr = ptr.next;  }  temp.next = null;  break;  case 3:  System.out.println("Enter position of node to be deleted: ");  p = sc.nextInt();  for (int i = 0; i < (p - 1); i++)  {  temp = ptr;  ptr = ptr.next;  }  temp.next = ptr.next;  break;  }  }  System.out.println("Do you want to delete more data? If yes, press 1.");  n = sc.nextInt();  }  while (n == 1);  }  public void Traverser()  {  Node temp = head;  if (head == null)  {  System.out.println("LL doesn't exist!");  }  else  {  while (temp != null)  {  System.out.println(temp.data);  temp = temp.next;  }  }  }  public static void main(String[] args)  {  SinglyLL ll = new SinglyLL();  ll.Creation();  ll.Deletion();  ll.Traverser();  }  } |
| **Output – Screenshots (Student Work Area):** |
| **Question Bank:**  How Linked List id different from Arrays?  Ans. An array is a grouping of data elements of equivalent data type. A linked list is a group of entities called a node. The node includes two segments: data and address.  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   Ans.   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   Ans.   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 first and last occurrence of an element in a single LinkedList   Ans.   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   Ans.  1. Create a function ‘getResult()’ that will accept one parameter, i.e., one head pointer of the linked list.  2. 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.  3. Assign the value of ‘head’ to ‘temp1’ and assign the null value to ‘temp2’.  4. 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.  5. Store value of ‘temp1’ in ‘temp2’.  6. Make one nested iteration using the ‘while’ loop, which will terminate if the value of the ‘next’ pointer of ‘temp2’ is equal to null.  7. 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.  8. Increment the ‘next’ pointer of the ‘temp1’ node to its next node.   * Delete alternate nodes of a Linked List.   Ans.   1. Start 2. Create a struct variable node that has data and a next pointer. 3. Initialize the linked list by setting the head variable to NULL, indicating an empty list. 4. Write an insert method to insert elements or nodes in the list. ‘ 5. Write a delete function to remove alternate elements from the list. This function will iterate through the list and skips one item, and delete every other alternating item from the list. 6. Write a function to display the elements of the list at any time to print output and verify results. 7. End |

**EXPERIMENT NO. 5**

|  |
| --- |
| **Student Name and Roll Number:** Arjun Bhardwaj & 21CSU211 |
| **Semester /Section:** 3rd / FS-A |
| **Link to Code:** CSL209 |
| **Date: 08/09/22** |
| **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):**   1. Define a Node class which represents a node in the list. It will have three properties: data, previous which will point to the previous node and next which will point to the next node. 2. Define another class for creating a doubly linked list, and it has two nodes: head and tail. Initially, head and tail will point to null. 3. addNode() will add node to the list:    1. It first checks whether the head is null, then it will insert the node as the head.    2. Both head and tail will point to a newly added node.    3. Head's previous pointer will point to null and tail's next pointer will point to null.    4. If the head is not null, the new node will be inserted at the end of the list such that new node's previous pointer will point to tail.    5. The new node will become the new tail. Tail's next pointer will point to null. 4. display() will show all the nodes present in the list.  * Define a new node 'current' that will point to the head. * Print current.data till current points to null.   Current will point to the next node in the list in each iteration. |
|  |
| **Code (Student Work Area):**  import java.util.\*;  public class DLL {  static class Node {  int data;  Node next;  Node prev;  Node(int data) {  this.data = data;  this.next = null;  this.prev = null;  }  }  Node head = null;  Node tail = null;  Scanner sc = new Scanner(System.in);  public void Creation() {  int data, n, m, p;  do {  System.out.println("Enter data: ");  data = sc.nextInt();  Node new\_node = new Node(data);  if (head == null)  {  head = new\_node;  tail = new\_node;    } else  {  System.out.println("Where do you wish to insert data: 1-beginning, 2-end, 3-specefic position");  m = sc.nextInt();  Node temp = head;  Node ptr = temp.next;  switch (m) {  case 1:  new\_node.next = head;  head.prev = new\_node;  head = new\_node;  break;  case 2:  tail.next = new\_node;  new\_node.prev = tail;  tail = new\_node;  break;  case 3:  System.out.println("Enter position where node is to be inserted: ");  p = sc.nextInt();  for (int i = 0; i < (p - 1); i++) {  temp = ptr;  temp = temp.next;  ptr = ptr.next;  }  new\_node.next = temp.next;  temp.next = new\_node;  break;  }  }  System.out.println("Do you want to enter more data? If yes, press 1.");  n = sc.nextInt();  }  while (n == 1);  }  public void Deletion()  {  int data;  int n, m, p;  do  {  if (head == null)  {  System.out.println("LL is empty!");  }  else  {  System.out.println("Enter position from where you wish to delete the node: 1-beginning, 2-end, 3-specefic position");  m = sc.nextInt();  Node temp = head;  Node ptr = temp.next;  switch (m) {  case 1:  temp = temp.next;  head = temp;  head.prev = null;  break;  case 2:  Node temp1 = tail;  temp1 = temp1.prev;  temp1.next = null;  break;  case 3:  System.out.println("Enter position of node to be deleted: ");  p = sc.nextInt();  Node temp2 = head;  Node ptr1 = temp2.next;  for (int i = 0; i < (p - 1); i++) {  temp2 = ptr1;  ptr1 = ptr1.next;  }  temp2.next = ptr1.next;  ptr.next.prev = temp;  break;  }  }  System.out.println("Do you want to delete more data? If yes, press 1.");  n = sc.nextInt();  }  while (n == 1);  }  public void Display()  {  Node temp = head;  if (temp == null)  {  System.out.println("LL does not exist!");  } else  {  while (temp != null)  {  System.out.println(temp.data + " ");  temp = temp.next;  }  }  }  public static void main(String[] args)  {  DoublyLL dList = new DoublyLL();  dList.Creation();  dList.Display();  dList.Deletion();  dList.Display();  }  }  **Output – Screenshots (Student Work Area):** |

**EXPERIMENT NO. 6**

|  |
| --- |
| **Student Name and Roll Number** Arjun Bhardwaj & 21CSU211 |
| **Semester /Section:** 3rd / FS-A |
| **Link to Code:** CSL209 |
| **Date: 08/09/22** |
| **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  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  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  **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  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 |
| **Algorithm (Student Work Area):**  1.if ptr = null  writeoverflow   gotostep11   [end of if]  2.set new\_node = ptr  3. set ptr = ptr -> next  4. set new\_node -> data = val  5. set temp = head  6. repeat step 8 while temp -> next != head  7. set temp = temp -> next  [end of loop]  8. set new\_node -> next = head  9. set temp → next = new\_node  10. set head = new\_node  11. exit |
|  |
| **Code (Student Work Area):**  import java.util.\*;  public class CircularLL  {  static class Node  {  int data;  Node next;  Node(int data)  {  this.data = data;  this.next = null;  }  }  Node head = null;  Node tail = null;  Scanner sc = new Scanner(System.in);  public void Create()  {  int data, n, m, p;  do  {  System.out.println("Enter data");  data = sc.nextInt();  Node new\_node = new Node(data);  if (head == null)  {  head = new\_node;  tail = new\_node;  new\_node.next = head;  }  else  {  tail = new\_node;  tail.next = head;  new\_node.next = head;  System.out.println("Enter the position where you wish to insert new node: 1-beginning, 2-end, 3-specefic position: ");  m = sc.nextInt();  Node temp = head;  switch (m)  {  case 1:  new\_node.next = head;  head = new\_node;  tail.next = head;  break;  case 2:  tail.next = new\_node;  tail = new\_node;  new\_node.next = head;  break;  case 3:  System.out.println("Enter position of node to be inserted: ");  p = sc.nextInt();  for (int i = 0; i < (p - 1); i++) {  temp = temp.next;  }  new\_node.next = temp.next;  temp.next = new\_node;  break;  }  }  System.out.println("Do you wish to enter more nodes? If yes, press 1.");  n = sc.nextInt();  }  while (n == 1);  }  public void Delete()  {  int data, m, p;  int n;  do  {  if (head == null)  {  System.out.println("Where do you want to delete the nodes from? 1-beginning, 2-end, 3-specefic position: ");  m = sc.nextInt();  switch (m)  {  case 1:  Node temp = head;  temp = temp.next;  head = temp;  tail.next = head;  break;  case 2:  Node temp1 = head;  Node ptr = temp1.next;  while (ptr.next != null)  {  temp1 = ptr;  ptr = ptr.next;  }  temp1.next = head;  tail = temp1;  break;  case 3:  System.out.println("Enter position of node to be deleted: ");  p = sc.nextInt();  Node temp2 = head;  Node ptr1 = temp2.next;  for (int i = 0; i < (p - 1); i++)  {  temp2 = ptr1;  ptr1 = ptr1.next;  }  temp2.next = ptr1.next;  break;  }  }  System.out.println("Do you wish to delete more data? Press 1.");  n = sc.nextInt();  }  while (n == 1);  }  public void Traverse()  {  Node temp = head;  if (head == null)  System.out.println("LL doesn't exist");  else  {  while (temp != null);  {  System.out.println(temp.data +" ");  temp = temp.next;  }  }  }  public static void main(String[] args)  {  CircularLL cl = new CircularLL();  cl.Create();  cl.Traverse();  cl.Delete();  cl.Traverse();  }  }  **Output – Screenshots (Student Work Area):** |
| **Question Bank:**  How Circular Linked List id different from Singly Linked List?  Ans. Circular 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.  Analyze the complexity of Traversal, insertion and Deletion operations in Linked List?  Ans. **•Traversal - access each element of the linked list**  **•Insertion - adds a new element to the linked list**  **•Deletion - removes the existing elements**  **Traverse a Linked List**  **Displaying the contents of a linked list is very simple. We keep moving the temp node to the next one and display its contents.**  **When temp is NULL, we know that we have reached the end of the linked list so we get out of the while loop.** |

**EXPERIMENT NO. 7**

|  |
| --- |
| **Student Name and Roll Number:** Arjun Bhardwaj & 21CSU211 |
| **Semester /Section:** 3rd / FS-A |
| **Link to Code:** CSL209 |
| **Date: 15/09/22** |
| **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):**  1. Checks if the stack is full.  2. If the stack is full, produces an error and exit.  3. If the stack is not full, increments top to point next empty space.  4. Adds data element to the stack location, where top is pointing.  5. Returns success. |
| Code (Student Work Area):  1. Linked List  import java.util.\*;  class StackL  {  Scanner sc = new Scanner(System.in);    static class Node  {  int data;  Node next;  Node(int data)  {  this.data = data;  this.next = null;  }  }  Node top = null;  void Push(Scanner sc)  {  System.out.println("Enter data: ");  int data = sc.nextInt();  Node new\_node = new Node(data);  if(top == null)  {  top = new\_node;  }  else  {  new\_node.next = top;  top = new\_node;  }  }  void Pop()  {  if(top == null)  {  System.out.println("Stack is empty!");  }  else  {  top = top.next;  }  }  void Display()  {  Node temp = top;  while(temp != null)  {  System.out.println(temp.data);  temp = temp.next;  }  }  }  public class Stack\_LL  {  public static void main(String[] args)  {  StackL l = new StackL();  Scanner sc = new Scanner(System.in);  int m;  do  {  System.out.println("Enter your choice: 1-Push, 2-Pop, 3-Display");  int c = sc.nextInt();  switch (c)  {  case 1:  {  l.Push(sc);  break;  }  case 2:  {  l.Pop();  break;  }  case 3:  {  l.Display();  break;  }  }  System.out.println("Enter 9 to go back to the Main Menu: ");  System.out.println("Or Enter any key to EXIT!");  m = sc.nextInt();  }  while (m == 9);  {  System.out.println("EXITED SUCCESSFULLY!");  }  l.Push(sc);  l.Display();  l.Pop();  l.Display();  }  }  2. Array  import java.util.\*;  class Stack  {  Scanner scc = new Scanner(System.in);  int top = -1;  int n = 10;  int arr[] = new int[n];    void Push(Scanner sc)  {  if(top == (n-1))  {  System.out.println("Underflow Condition!");  }  else  {  System.out.println("Enter data: ");  int i = scc.nextInt();  top = top + 1;  arr[top] = i;  System.out.println("Item inserted!");  }  }  void Pop()  {  if(top == -1)  {  System.out.println("Underflow Condition!");  }  else  {  top = top -1;  System.out.println("Item deleted!");  }  }  void Display()  {  System.out.println("Items are: ");  for(int j=top; j>-1; j--)  {  System.out.println(arr[j]);  }  }  }  public class Stack\_Arr  {  public static void main(String[] args)  {  Stack s = new Stack();  Scanner sc = new Scanner(System.in);  int l;  do  {  System.out.println("Enter your choice: 1-Push, 2-Pop, 3-Display");  int c = sc.nextInt();  switch (c)  {  case 1:  {  s.Push(sc);  break;  }  case 2:  {  s.Pop();  break;  }  case 3:  {  s.Display();  break;  }  }  System.out.println("Enter 9 to go back to the Main Menu: ");  System.out.println("Or Enter any key to EXIT!");  l = sc.nextInt();  }  while(l==9);  {  System.out.println("EXITED SUCCESSFULLY!");  }  s.Push(sc);  s.Pop();  s.Display();  }  } |
| **Output – Screenshots (Student Work Area):**   * + 1. **LL** |
| **Question Bank:**  1)What are Stacks?  Ans) In computer science, a stack is an abstract data type that serves as a collection of elements, with two main operations: Push, which adds an element to the collection, and Pop, which removes the most recently added element that was not yet removed  2)What are the applications of stacks?  Ans) A Stack can be used for evaluating expressions consisting of operands and operators. Stacks can be used for Backtracking, i.e., to check parenthesis matching in an expression. It can also be used to convert one form of expression to another form. It can be used for systematic Memory Management. |

**EXPERIMENT NO. 8**

|  |
| --- |
| **Student Name and Roll Number:** Arjun Bhardwaj & 21CSU211 |
| **Semester /Section:** 3rd / FS-A |
| **Link to Code:** CSL209 |
| **Date: 15/09/22** |
| **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 |
| Code (Student Work Area):  Code (Student Work Area):  import java.util.\*;  class Stack\_Str {  int size;  int top;  char[] a;  boolean isEmpty() { return (top < 0); }  Stack\_Str(int n)  {  top = -1;  size = n;  a = new char[size];  }  boolean push(char x)  {  if (top >= size) {  System.out.println("Stack Overflow");  return false;  }  else {  a[++top] = x;  return true;  }  }  char pop()  {  if (top < 0) {  System.out.println("Stack Underflow");  return 0;  }  else {  char x = a[top--];  return x;  }  }  }  class Main {  public static void reverse(StringBuffer str)  {  int n = str.length();  Stack\_Str obj = new Stack\_Str(n);  int i;  for (i = 0; i < n; i++)  obj.push(str.charAt(i));  for (i = 0; i < n; i++) {  char ch = obj.pop();  str.setCharAt(i, ch);  }  }  public static void main(String args[])  {  Scanner input = new Scanner(System.in);  System.out.println("Enter the string: ");  String inp = input.next();  StringBuffer s = new StringBuffer(inp);  reverse(s);  System.out.println("Reversed string is " + s);  }  } |
| **Output – Screenshots (Student Work Area):** |
| **Question Bank:**  What are Stacks?  In computer science, a stack is an abstract data type that serves as a collection of elements, with two main operations: Push, which adds an element to the collection, and Pop, which removes the most recently added element that was not yet removed.  How we can split a sentence and push it into the stack?  Ans) Concat returns a new array that contains the additional values, but you would still have to do something with the array afterwards |

**EXPERIMENT NO. 9**

|  |
| --- |
| **Student Name and Roll Number:** Arjun Bhardwaj & 21CSU211 |
| **Semester /Section:** 3rd / FS-A |
| **Link to Code:** CSL209 |
| **Date: 22/09/22** |
| **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):**   * Declare a character stack (say **temp**). * Now traverse the string exp.   + If the current character is a starting bracket ( **‘(‘ or ‘{‘  or ‘[‘**) then push it to stack.   + If the current character is a closing bracket ( **‘)’ or ‘}’ or ‘]’**) then pop from stack and if the popped character is the matching starting bracket then fine.   + Else brackets are**Not Balanced**. * After complete traversal, if there is some starting bracket left in stack then **Not balanced**, else **Balanced**. |
| Code (Student Work Area):  import java.util.\*;  public class BalancedBrackets  {  static boolean areBracketsBalanced(String expr)  {  Deque<Character> stack = new ArrayDeque<Character>();    for (int i = 0; i < expr.length(); i++) {  char x = expr.charAt(i);  if (x == '(' || x == '[' || x == '{')  {  stack.push(x);  continue;  }  if(stack.isEmpty())  return false;  char check;  switch (x)  {  case ')':  check = stack.pop();  if (check == '{' || check == '[')  return false;  break;  case '}':  check = stack.pop();  if (check == '(' || check == '[')  return false;  break;  case ']':  check = stack.pop();  if (check == '(' || check == '{')  return false;  break;  }  }    return (stack.isEmpty());  }    public static void main(String[] args)  {  String expr = "([{}])";    if (areBracketsBalanced(expr))  System.out.println("Balanced ");    else  System.out.println("Not Balanced ");  }  } |
| **Q: How a stack helps in syntax analysis or compilation of a program?**  **Ans. A Stack helps to manage the data in the 'Last in First out' method. When the variable is not used outside the function in any program, the Stack can be used. It allows you to control and handle memory allocation and deallocation. It helps to automatically clean up the objects.** |

**EXPERIMENT NO. 10**

|  |
| --- |
| **Student Name and Roll Number:** Arjun Bhardwaj & 21CSU211 |
| **Semester /Section:** 3rd / FS-A |
| **Link to Code:** CSL209 |
| **Date: 22/09/22** |
| **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):**   1. Scan the infix notation from left to right one character at a time. 2. If the next symbol scanned as an operand, append it to the postfix string. 3. If the next symbol scanned as an operator, the:    1. Pop and append to the postfix string every operator on the stack that:       1. Is above the most recently scanned left parenthesis, and       2. Has precedence higher than or is a right-associative operator of equal precedence to that of the new operator symbol.    2. Push the new operator onto the stack 4. If a left parenthesis is scanned, push it into the stack. 5. If a right parenthesis is scanned, all operators down to the most recently scanned left parenthesis must be popped and appended to the postfix string. Furthermore, the pair of parentheses must be discarded. 6. When the infix string is fully scanned, the stack may still contain some operators. All the remaining operators should be popped and appended to the postfix string. |
| Code (Student Work Area):  import java.io.\*;  class Stack\_ItP  {  char a[] = new char[100];  int top = -1;  void push(char c)  {  try  {  a[++top] = c;  }  catch (StringIndexOutOfBoundsException e)  {  System.out.println("Stack full, no room to push, size=100");  System.exit(0);  }  }  char pop()  {  return a[top--];  }  boolean isEmpty()  {  return (top == -1) ? true : false;  }  char peek()  {  return a[top];  }  }  public class InfixToPostfix  {  static Stack\_ItP operators = new Stack\_ItP();  public static void main(String argv[]) throws IOException  {  String infix;    BufferedReader keyboard = new BufferedReader(new InputStreamReader(System.in));    System.out.print("\nEnter the infix expression you want to convert: ");  infix = keyboard.readLine();    System.out.println("Postfix expression for the given infix expression is:" + toPostfix(infix));  }  private static String toPostfix(String infix)  {  char symbol;  String postfix = "";  for (int i = 0; i < infix.length(); ++i)    {  symbol = infix.charAt(i);    if (Character.isLetter(symbol))  postfix = postfix + symbol;  else if (symbol == '(')  {  operators.push(symbol);  }  else if (symbol == ')')  {  while (operators.peek() != '(')  {  postfix = postfix + operators.pop();  }  operators.pop();  }  else  {  while (!operators.isEmpty() && !(operators.peek() == '(') && prec(symbol) <= prec(operators.peek()))  postfix = postfix + operators.pop();  operators.push(symbol);  }  }  while (!operators.isEmpty())  postfix = postfix + operators.pop();  return postfix;  }  static int prec(char x)  {  if (x == '+' || x == '-')  return 1;  if (x == '\*' || x == '/' || x == '%')  return 2;    return 0;  } |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. **Why conversion is required?**   **Ans)** 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 we can convert infix to prefix and prefix to postfix?**   **Ans)**  Step 1: Reverse the infix expression i.e A+B\*C will become C\*B+A. Note while reversing each ‘(‘ will become ‘)’ and each ‘)’ becomes ‘(‘.  Step 2: Obtain the “nearly” postfix expression of the modified expression i.e CB\*A+.  Step 3: Reverse the postfix expression. Hence in our example prefix is +A\*BC. |

**EXPERIMENT NO. 11**

|  |
| --- |
| **Student Name and Roll Number:** Arjun Bhardwaj & 21CSU211 |
| **Semester /Section:** 3rd / FS-A |
| **Link to Code:** CSL209 |
| **Date: 06/10/22** |
| **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):**  START  Procedure Hanoi(disk, source, dest, aux)  IF disk == 1, THEN  move disk from source to dest  ELSE  Hanoi(disk - 1, source, aux, dest) // Step 1  move disk from source to dest // Step 2  Hanoi(disk - 1, aux, dest, source) // Step 3  END IF    END Procedure  STOP |
| Code (Student Work Area):  import java.io.\*;  import java.math.\*;  import java.util.\*;  class GFG  {  static void towerOfHanoi(int n, char from\_rod, char to\_rod, char aux\_rod)  {  if (n == 0)  {  return;  }  towerOfHanoi(n - 1, from\_rod, aux\_rod, to\_rod);  System.out.println("Move disk " + n + " from rod " + from\_rod + " to rod "  + to\_rod);  towerOfHanoi(n - 1, aux\_rod, to\_rod, from\_rod);  }    public static void main(String args[])  {  int N = 3;    towerOfHanoi(N, 'A', 'C', 'B');  }  } |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. **What is Recursion?**   Ans)Recursion is a process in which the function calls itself indirectly or directly in order to solve the problem. The function that performs the process of recursion is called a recursive function. There are certain problems that can be solved pretty easily with the help of a recursive algorithm.   1. **What is Base condition?**   Ans)  In the recursive program, the solution to the base case is provided and the solution to the bigger problem is expressed in terms of smaller problems.   1. **What are the number of steps required to solve n-Disc problem?**   Ans) Tower of Hanoi puzzle with n disks can be solved in minimum 2n−1 steps. |

**EXPERIMENT NO. 12**

|  |
| --- |
| **Student Name and Roll Number:** Arjun Bhardwaj & 21CSU211 |
| **Semester /Section:** 3rd / FS-A |
| **Link to Code:** CSL209 |
| **Date: 19/10/22** |
| **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):  begin procedure isfull  if rear equals to MAXSIZE  return true  else  return false  endif    end procedure  begin procedure isempty  if front is less than MIN OR front is greater than rear  return true  else  return false  endif    end procedure  procedure enqueue(data)    if queue is full  return overflow  endif    rear ← rear + 1  queue[rear] ← data  return true    end procedure  procedure dequeue    if queue is empty  return underflow  end if  data = queue[front]  front ← front + 1  return true  end procedure |
| **Code (Student Work Area):**   * + 1. Array   import java.util.\*;  class Queue  {  Scanner sc = new Scanner(System.in);  int f = -1, r = -1;  int n = 10;  int que[] = new int[n];  int data;    void Enqueue(Scanner sc)  {  if(r == (n-1))  {  System.out.println("Overflow Condition!");  }    else  {  System.out.println("Enter data: ");  data = sc.nextInt();  if(f == -1 && r == -1)  {  f=0;  r=0;  que[r] = data;  }  else  {  r = r+1;  que[r] = data;  }  }  }  void Dequeue()  {  if(f == -1 && r == -1)  {  System.out.println("Underflow Condition");  }  else  {  f = f + 1;  }  }  void Display()  {  System.out.println("Items are: ");  for(int i = f; i<=r; i++)  {  System.out.println(que[i]);  }  }  }  public class Queue\_Arr  {  public static void main(String[] args)  {  Queue q = new Queue();  Scanner sc = new Scanner(System.in);  int l;  do  {  System.out.println("Enter your choice: 1-Enqueue, 2-Dequeue, 3-Display");  int c = sc.nextInt();  switch (c)  {  case 1:  {  q.Enqueue(sc);  break;  }  case 2:  {  q.Dequeue();  break;  }  case 3:  {  q.Display();  break;  }  }  System.out.println("Enter 0 to go back to the Main Menu: ");  System.out.println("Or Enter any key to EXIT!");  l = sc.nextInt();  }  while(l==0);  {  System.out.println("EXITED SUCCESSFULLY!");  }  q.Enqueue(sc);  q.Dequeue();  q.Display();  }  } |
| **Output – Screenshots (Student Work Area):**   * + - 1. **Array** |
| **QUESTION BANK:**   1. **What are the applications of queues?**    1. When a resource is shared among multiple consumers. Examples include[CPU scheduling](https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/), [Disk Scheduling](https://www.geeksforgeeks.org/disk-scheduling-algorithms/).    2. Spooling in printers    3. Buffer for devices like keyboard 2. **Queues can be implemented with the help of stack. How?**   i. Follow the below steps to implement the push(s, x) operation:   * 1. Enqueue x to q2.   2. One by one dequeue everything from q1 and enqueue to q2.   3. Swap the queues of q1 and q2.   ii. Follow the below steps to implement the pop(s) operation:  a. Dequeue an item from q1 and return it. |

**EXPERIMENT NO. 13**

|  |
| --- |
| **Student Name and Roll Number:** Arjun Bhardwaj & 21CSU211 |
| **Semester /Section:** 3rd / FS-A |
| **Link to Code:** CSL209 |
| **Date: 27/10/22** |
| **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):  The circular queue work as follows:   * Two pointers FRONT and REAR * FRONT track the first element of the queue * REAR track the last elements of the queue * initially, set value of FRONT and REAR to -1  1. Enqueue Operation  1. check if the queue is full 2. for the first element, set value of FRONT to 0 3. circularly increase the REAR index by 1 (i.e. if the rear reaches the end, next it would be at the start of the queue) 4. add the new element in the position pointed to by REAR  2. Dequeue Operation  1. check if the queue is empty 2. return the value pointed by FRONT 3. circularly increase the FRONT index by 1 4. for the last element, reset the values of FRONT and REAR to -1   However, the check for full queue has a new additional case:   * Case 1: FRONT = 0 && REAR == SIZE - 1 * Case 2: FRONT = REAR + 1   The second case happens when REAR starts from 0 due to circular increment and when its value is just 1 less than FRONT, the queue is full. |
| Code (Student Work Area):  import java.util.\*;  public class CQueue  {  int SIZE = 5;  int front, rear;  int items[] = new int[SIZE];  CQueue()  {  front = -1;  rear = -1;  }    boolean isFull()  {  if (front == 0 && rear == SIZE - 1)  {  return true;  }  if (front == rear + 1)  {  return true;  }  return false;  }    boolean isEmpty()  {  if (front == -1)  return true;  else  return false;  }    void enQueue(int element)  {  if (isFull()) {  System.out.println("Queue is full");  }  else  {  if (front == -1)  front = 0;  rear = (rear + 1) % SIZE;  items[rear] = element;  System.out.println("Inserted " + element);  }  }    int deQueue()  {  int element;  if (isEmpty())  {  System.out.println("Queue is empty");  return (-1);  }  else  {  element = items[front];  if (front == rear)  {  front = -1;  rear = -1;  }  else  {  front = (front + 1) % SIZE;  }  return (element);  }  }  void display()  {    int i;  if (isEmpty())  {  System.out.println("Empty Queue");  }    else  {  System.out.println("Front -> " + front);  System.out.println("Items -> ");  for (i = front; i != rear; i = (i + 1) % SIZE)  System.out.print(items[i] + " ");  System.out.println(items[i]);  System.out.println("Rear -> " + rear);  }  }  public static void main(String[] args)  {  CQueue q = new CQueue();    q.deQueue();  q.enQueue(1);  q.enQueue(2);  q.enQueue(3);  q.enQueue(4);  q.enQueue(5);  q.enQueue(6);  q.display();  int elem = q.deQueue();  if (elem != -1)  {  System.out.println("Deleted Element is " + elem);  }  q.display();  q.enQueue(7);  q.display();  q.enQueue(8);  }  } |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. **What are the applications of Circular queues?**   Ans) Memory management: circular queue is used in memory management. Process Scheduling: A CPU uses a queue to schedule processes. Traffic Systems: Queues are also used in traffic systems.   1. **What is the complexity of all operations in Circular Queue?**   Ans) The complexity of the enqueue and dequeue operations of a circular queue is O(1) for (array implementations |

**EXPERIMENT NO. 14**

|  |
| --- |
| **Student Name and Roll Number:** Arjun Bhardwaj & 21CSU211 |
| **Semester /Section:** 3rd / FS-A |
| **Link to Code:** CSL209 |
| **Date: 3/11/22** |
| **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):**  Take an array (deque) of size n.  Set two pointers at the first position and set front = -1 and rear = 0.  1. Insert at the Front   1. This operation adds an element at the front. 2. Check the position of front. 3. If front < 1, reinitialize front = n-1 (last index). 4. Else, decrease front by 1. 5. Add the new key 5 into array[front].   2. Insert at the Rear   1. This operation adds an element to the rear. 2. Check if the array is full. 3. If the deque is full, reinitialize rear = 0. 4. Else, increase rear by 1. 5. Add the new key 5 into array[rear].   3. Delete from the Front   * + 1. The operation deletes an element from the front.     2. Check if the deque is empty.     3. If the deque is empty (i.e. front = -1), deletion cannot be performed (underflow condition).     4. If the deque has only one element (i.e. front = rear), set front = -1 and rear = -1.     5. Else if front is at the end (i.e. front = n - 1), set go to the front front = 0.     6. Else, front = front + 1.  4. Delete from the Rear This operation deletes an element from the rear.   1. Check if the deque is empty. 2. If the deque is empty (i.e. front = -1), deletion cannot be performed (underflow condition). 3. If the deque has only one element (i.e. front = rear), set front = -1 and rear = -1, else follow the steps below. 4. If rear is at the front (i.e. rear = 0), set go to the front rear = n - 1. 5. Else, rear = rear - 1.  5. Check Empty This operation checks if the deque is empty. If front = -1, the deque is empty. 6. Check Full This operation checks if the deque is full. If front = 0 and rear = n - 1 OR front = rear + 1, the deque is full. |
| Code (Student Work Area):  Code (Student Work Area):  class Deque  {  static final int MAX = 100;  int arr[];  int front;  int rear;  int size;  public Deque(int size)  {  arr = new int[MAX];  front = -1;  rear = 0;  this.size = size;  }  boolean isFull()  {  return ((front == 0 && rear == size - 1) || front == rear + 1);  }  boolean isEmpty()  {  return (front == -1);  }  void insertfront(int key)  {  if (isFull())  {  System.out.println("Overflow");  return;  }  if (front == -1)  {  front = 0;  rear = 0;  }  else if (front == 0)  front = size - 1;  else  front = front - 1;  arr[front] = key;  }  void insertrear(int key)  {  if (isFull()) {  System.out.println(" Overflow ");  return;  }  if (front == -1)  {  front = 0;  rear = 0;  }  else if (rear == size - 1)  rear = 0;  else  rear = rear + 1;  arr[rear] = key;  }  void deletefront()  {  if (isEmpty())  {  System.out.println("Queue Underflow\n");  return;  }  if (front == rear) {  front = -1;  rear = -1;  } else if (front == size - 1)  front = 0;  else  front = front + 1;  }  void deleterear()  {  if (isEmpty()) {  System.out.println(" Underflow");  return;  }  if (front == rear) {  front = -1;  rear = -1;  } else if (rear == 0)  rear = size - 1;  else  rear = rear - 1;  }  int getFront()  {  if (isEmpty())  {  System.out.println(" Underflow");  return -1;  }  return arr[front];  }  int getRear()  {  if (isEmpty() || rear < 0)  {  System.out.println(" Underflow\n");  return -1;  }  return arr[rear];  }  public static void main(String[] args)  {  Deque dq = new Deque(4);  System.out.println("Insert element at rear end : 12 ");  dq.insertrear(12);  System.out.println("insert element at rear end : 14 ");  dq.insertrear(14);  System.out.println("get rear element : " + dq.getRear());  dq.deleterear();  System.out.println("After delete rear element new rear become : " + dq.getRear());  System.out.println("inserting element at front end");  dq.insertfront(13);  System.out.println("get front element: " + dq.getFront());  dq.deletefront();  System.out.println("After delete front element new front become : " + +dq.getFront());  }  } |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. **What are the applications of Doubly ended queues?**  * Applied as both stack and queue, as it supports both operations. * 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?**   Ans) In a growing array, the amortized time complexity of all deque operations is O(1). Additionally, the time complexity of random access by index is O(1); but the time complexity of insertion or deletion in the middle is O(n). |

**EXPERIMENT NO. 15**

|  |
| --- |
| **Student Name and Roll Number:** Arjun Bhardwaj & 21CSU211 |
| **Semester /Section:** 3rd / FS-A |
| **Link to Code:** CSL209 |
| **Date: 10/11/22** |
| **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):**  *Algorithm Inorder(tree)*   1. *Traverse the left subtree, i.e., call Inorder(left->subtree)* 2. *Visit the root.* 3. *Traverse the right subtree, i.e., call Inorder(right->subtree)*   *Algorithm Preorder(tree)*   1. *Visit the root.* 2. *Traverse the left subtree, i.e., call Preorder(left->subtree)* 3. *Traverse the right subtree, i.e., call Preorder(right->subtree)*   *Algorithm Postorder(tree)*   1. *Traverse the left subtree, i.e., call Postorder(left->subtree)* 2. *Traverse the right subtree, i.e., call Postorder(right->subtree)* 3. *Visit the root* |
| Code (Student Work Area):  class Node  {  int key;  Node left, right;  public Node(int item)  {  key = item;  left = right = null;  }  }  class BinaryTree  {  Node root;  BinaryTree() { root = null; }    void printInorder(Node node)  {  if (node == null)  return;    printInorder(node.left);    System.out.print(node.key + " ");    printInorder(node.right);  }    void printInorder() { printInorder(root); }    public static void main(String[] args)  {  BinaryTree tree = new BinaryTree();  tree.root = new Node(1);  tree.root.left = new Node(2);  tree.root.right = new Node(3);  tree.root.left.left = new Node(4);  tree.root.left.right = new Node(5);    System.out.println("\nInorder traversal of binary tree is ");  tree.printInorder();  }  }  class Node1  {  int key;  Node1 left, right;  public Node1(int item)  {  key = item;  left = right = null;  }  }  class BinaryTree1  {  Node1 root;  BinaryTree1() { root = null; }  void printPreorder1(Node1 node)  {  if (node == null)  return;    System.out.print(node.key + " ");    printPreorder1(node.left);    printPreorder1(node.right);  }    void printPreorder1() { printPreorder1(root); }    public static void main(String[] args)  {  BinaryTree1 tree = new BinaryTree1();  tree.root = new Node1(1);  tree.root.left = new Node1(2);  tree.root.right = new Node1(3);  tree.root.left.left = new Node1(4);  tree.root.left.right = new Node1(5);    System.out.println("Preorder traversal of binary tree is ");  tree.printPreorder1();  }  }  class Node2  {  int key;  Node2 left, right;  public Node2(int item)  {  key = item;  left = right = null;  }  }  class BinaryTree2  {    Node2 root;  BinaryTree2() { root = null; }    void printPostorder2(Node2 node2)  {  if (node2 == null)  return;    printPostorder2(node2.left);    printPostorder2(node2.right);    System.out.print(node2.key + " ");  }  void printPostorder2() { printPostorder2(root); }    public static void main(String[] args)  {  BinaryTree2 tree = new BinaryTree2();  tree.root = new Node2(1);  tree.root.left = new Node2(2);  tree.root.right = new Node2(3);  tree.root.left.left = new Node2(4);  tree.root.left.right = new Node2(5);    System.out.println( "\nPostorder traversal of binary tree is ");  tree.printPostorder2();  }  } |
| **Output – Screenshots (Student Work Area):** |
| **QUESTION BANK:**   1. **What is the difference between Binary Tree and Binary Search Tree?**      1. **What is the complexity of all search operations in BST?**   **Binary Search Tree (BST):** BST is a special type of binary tree in which left child of a node has value less than the parent and right child has value greater than parent. Consider the left skewed BST shown in Figure 2.  https://media.geeksforgeeks.org/wp-content/uploads/skewl2.png   * **Searching:** For searching element 1, we have to traverse all elements (in order 3, 2, 1). Therefore, searching in binary search tree has worst case complexity of O(n). In general, time complexity is O(h) where h is height of BST. * **Insertion**: For inserting element 0, it must be inserted as left child of 1. Therefore, we need to traverse all elements (in order 3, 2, 1) to insert 0 which has worst case complexity of O(n). In general, time complexity is O(h). * **Deletion:** For deletion of element 1, we have to traverse all elements to find 1 (in order 3, 2, 1). Therefore, deletion in binary tree has worst case complexity of O(n). In general, time complexity is O(h). |

**EXPERIMENT NO. 16**

|  |
| --- |
| **Student Name and Roll Number:** Arjun Bhardwaj & 21CSU211 |
| **Semester /Section:** 3rd / FS-A |
| **Link to Code:** CSL209 |
| **Date: 17/11/22** |
| **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. | |
| Code (Student Work Area):  BubbleSort  public class BubbleSort {  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]){    temp = arr[j-1];  arr[j-1] = arr[j];  arr[j] = temp;  }    }  }    }  public static void main(String[] args) {  int arr[] ={3,60,35,2,45,320,5};    System.out.println("Array Before Bubble Sort");  for(int i=0; i < arr.length; i++){  System.out.print(arr[i] + " ");  }  System.out.println();    bubbleSort(arr);    System.out.println("Array After Bubble Sort");  for(int i=0; i < arr.length; i++){  System.out.print(arr[i] + " ");  }    }  }  Insertion Sort  public class InsertionSort {  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 main(String a[]){  int[] arr1 = {9,14,3,2,43,11,58,22};  System.out.println("Before Insertion Sort");  for(int i:arr1){  System.out.print(i+" ");  }  System.out.println();    insertionSort(arr1);    System.out.println("After Insertion Sort");  for(int i:arr1){  System.out.print(i+" ");  }  }  }  Merge Sort  class MergeSort {  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++;  }  }  void sort(int arr[], int l, int r)  {  if (l < r) {    int m = l + (r - l) / 2;    sort(arr, l, m);  sort(arr, m + 1, r);    merge(arr, l, m, r);  }  }  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[] = { 12, 11, 13, 5, 6, 7 };  System.out.println("Given Array");  printArray(arr);  MergeSort ob = new MergeSort();  ob.sort(arr, 0, arr.length - 1);  System.out.println("\nSorted array");  printArray(arr);  }  }  Quick Sort  import java.io.\*;  class QuickSort {  static void swap(int[] arr, int i, int j)  {  int temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  }  static int partition(int[] arr, int low, int high)  {  // pivot  int pivot = arr[high];  int i = (low - 1);  for (int j = low; j <= high - 1; j++) {  if (arr[j] < pivot) {  i++;  swap(arr, i, j);  }  }  swap(arr, i + 1, high);  return (i + 1);  }  static void quickSort(int[] arr, int low, int high)  {  if (low < high) {  int pi = partition(arr, low, high);  quickSort(arr, low, pi - 1);  quickSort(arr, pi + 1, high);  }  }  static void printArray(int[] arr, int size)  {  for (int i = 0; i < size; i++)  System.out.print(arr[i] + " ");  System.out.println();  }  public static void main(String[] args)  {  int[] arr = { 10, 7, 8, 9, 1, 5 };  int n = arr.length;  quickSort(arr, 0, n - 1);  System.out.println("Sorted array: ");  printArray(arr, n);  }  }  // This code is contributed by Ayush Choudhary  Selection Sort  public class SelectionSort {  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 main(String a[]){  int[] arr1 = {9,14,3,2,43,11,58,22};  System.out.println("Before Selection Sort");  for(int i:arr1){  System.out.print(i+" ");  }  System.out.println();    selectionSort(arr1);    System.out.println("After Selection Sort");  for(int i:arr1){  System.out.print(i+" ");  }  }  } |
| **Output – Screenshots (Student Work Area):**  **Shape  Description automatically generated with medium confidenceShape  Description automatically generated with medium confidenceGraphical user interface, application  Description automatically generated** |
| **QUESTION BANK:**  **Compare and contrast all Sorting techniques?**  **Ans. Bubble sort and Insertion sort –**  **Average and worst case time complexity: n^2**  **Best case time complexity: n when array is already sorted.**  **Worst case: when the array is reverse sorted.**    **Selection sort –**  **Best, average and worst case time complexity: n^2 which is independent of distribution of data.**    **Merge sort –**  **Best, average and worst case time complexity: nlogn which is independent of distribution of data.**  **Insertion sort –**  **Best, average and worst case time complexity: nlogn which is independent of distribution of data.**    **Quick sort –**  **It is a divide and conquer approach with recurrence relation:**    **T(n) = T(k) + T(n-k-1) + cn**  **Worst case: when the array is sorted or reverse sorted, the partition algorithm divides the array in two subarrays with 0 and n-1 elements. Therefore,**    **T(n) = T(0) + T(n-1) + cn**  **Solving this we get, T(n) = O(n^2)**  **Best case and Average case: On an average, the partition algorithm divides the array in two subarrays with equal size. Therefore,** |