Lab Manual

Data Structures (P)

IT-201

L T P - - 2	PRS:15

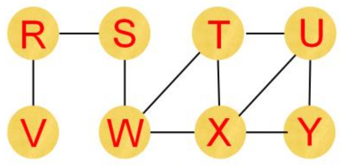
	Experiment 1 (Programs on Strings and Passing Pointers to Functions & Array)
I.	Write a function that accepts as input a string and determines the frequency of occurences of each of the distinct characters in string. Test your function using suitable data.
II.	Write a function, strndel, that accepts a string and two integers, start and length. Return a new string that is equivalent to the original string, except that length characters beginning at start have been removed.
III.	Write a function, strdel, that accepts a string and a character. The function returns string with the first occurence of character removed.
IV.	WAP with function to swap two numbers using call by reference.
V.	WAP with function to swap two integer arrays of same size using call by reference.
VI.	WAP to reverse an array by swapping (without using additional memory).
VII.	WAP with function to swap two strings using pointers.
VIII.	WAP to find the number of Non Repeated elements in an Array.
IX.	WAP to identify the missing numbers in a given Array within the range [1...N].
	Experiment 2 (Dynamic Memory Allocation and File Operations)
I.	WAP to find the Median of the elements after merging the two sorted Arrays of same size.
II.	WAP to store an information in array of structure, dynamically, and also give a function to display the current information. The program should give user a choice for inserting a data or to display the current data. Implement the program for *Student structures* (contains student_name, student_roll_no, total_marks).
III.	Implement the above program for *Employee structures* (contains employee_name, emp_no, emp_salary).
IV.	Implement the above program for *Faculty structures* (contains faculty_name, faculty_ID, subject_codes, class_names).
V.	WAP to Create a new file, Open an existing file, read the file to search a given word, write into the file and close the file.

Experiment 3 (Searching and Sorting in Non-decreasing order)	
	Test the following programs on various cases such as already sorted, reverse sorted and average sorted cases. Also verify whether the sorting algorithm is stable. Count the number of swaps and number of comparisons taken place.
I.	WAP to implement Linear search and Binary search on 1D array of Integers.
II.	WAP to implement Insertion and Selection sort on 1D array of strings.
III.	WAP to implement Quick sort on 1D array of characters.
IV.	WAP to implement Merge Sort on 1D array of *Student structures* (contains student_name, student_roll_no, total_marks) with key as student_roll_no.
V.	WAP to implement Bubble and Radix sort on 1D array of *Faculty structures* (contains faculty_name, faculty_ID, subject_codes, class_names) with key as faculty_ID.
Experiment 4 (Sparse Arrays and Matrix Operations)	
I.	WAP to store and display a Lower-Right triangular matrix in RMO and CMO fashion.
II.	WAP to store and display an Upper-Right triangular matrix in RMO and CMO fashion.
III.	WAP to store and display a Tri-Diagonal matirx in RMO and CMO fashion.
IV.	WAP to store and display a Lower-Left triangular matrix in RMO and CMO fashion.
V.	WAP to store and display an Upper-Left triangular matrix in RMO and CMO fashion.
VI.	WAP to store and display a C matrix in RMO and CMO fashion. (C matrix contains non-zero elements in first row, last row and first column only)
VII.	WAP to store and display a Z matrix in RMO and CMO fashion. (Z matrix contains non-zero elements in first row, last row and right diagonal only.)
VIII.	WAP using switch statement to perform the following functions: Transpose of a matrix, computing determinant of a matirx, addition of two matrices and multiplication of two matrices. The program should handle a wrong input exceptions such as size mis-match, etc.
IX.	WAP for addition and multiplication of two sparse matrices. The matrices can be any of the following type,
	a. Lower-Right triangularmatrix
	b. Upper-Right triangular matrix
	c. Tri-Diagonal matrix
	d. Lower-Left triangular matrix
	e. Upper-Left triangular matrix
	Firstly the program should ask the user about the type(s) of input matrices, store the given matrices either in RMO or CMO fashion in 1D Array and then perform the respective operation and display the result.

Experiment 5 (Stacks and Queues)	
I. (a)	WAP to implement Stack ADT using Arrays which has basic operations as Create(), IsEmpty(), Push(), Pop(), IsFull() with appropriate prototype to a functions.
(b)	WAP to evaluate a given postfix expression using stack ADT.
II. (a)	WAP to Cyclically Permute the elements of an Array.
(b)	WAP to check the given string is palindrome using stack.
(c)	WAP to check the given expression is correctly parenthesized.
III.	WAP to Implement two overlapping Stacks, facing in opposite directions, using an Array and Check for Overflow & Underflow conditions.
IV.	WAP to implement a 3-stacks of size *'m'* in an array of size *'n'* with all the basic operations such as IsEmpty(i), Push(i), Pop(i), IsFull(i) where *'i'* denotes the stack number (1,2,3), $m \cong n/3$. Stacks are not overlapping each other. Leftmost stack facing the left direction and other two stacks are facing in the right direction.
V. (a)	WAP to transform infix expression into equivalent postfix expression using stack. Also use the user defined operators, $,#, etc, with appropiate priorities. Eg. A+(B*C-D/E$F)*G)*H, {*,/} > $ > {+,-}
(b)	WAP to tranform infix expression into equivalent prefix expression, similar as above.
VI.(a)	WAP wihch gives the solution to the Tower of Hanoi problem for n disks. Test the program using: a) N=3, b) N=4.
(b)	WAP to solve Maze problem of size mxn.
VII.(a)	WAP to implement Queue ADT using Arrays with the basic functions of Create(), IsEmpty(), Insert(), Delete() and IsFull() with suitable prototype to a functions.
(b)	WAP to implement Queue using Stacks.
VIII.(a)	WAP to implement 2 overlapping queues in an Array of size 'N'. There are facing in opposite direction to eachother. Give IsEmpty(i), Insert(i), Delete(i) and IsFull(i) routines for i^{th} queue.
(b)	WAP to implement Dequeue using Arrays with all the basic operations.

Experiment 6 (Singly Linked List)	
I.	WAP to construct simple linear linked list using dynamic memory allocation for the given elements with the following functions,
	(a) Inserting a new node,
	(b) Accessing a node (finding the position wrt header),
	(c) Removing a node with particular key value,
	(d) Complete deletion of a linked list, and
	(e) Displaying the current list.
	(f) Copy the linked list and return the pointer of the new list.
	Implement the above program for the elements as Strings.
II. (a)	WAP to reverse a singly linked list using one auxillary pointer. And try without using any auxillary pointer.
(b)	WAP to implement Stack ADT using Linked list with the basic operations as Create(), IsEmpty(), Push(), Pop(), IsFull() with appropriate prototype to a functions.
III. (a)	WAP to implement Queue ADT using Linked list with the basic functions of Create(), IsEmpty(), Insert(), Delete() and IsFull() with suitable prototype to a functions.
(b)	WAP to perform BFS on the given tree (represented in Linked list form) using Queue ADT.
IV. (a)	WAP for polynomial addition and multiplication. Represent polynomial in linked list form with suitable data structure.
(b)	WAP to compare two polynomials. Represent polynomial in linked list form with suitable data structure.
V.	WAP to swap elements in pairs in the given linked list. (e.g.: 1->2->3->4->5->null, then result should be 2->1->4->3->5->null.)
VI. (a)	WAP to find second last node of the given linked list.
(b)	WAP to concatenate two singly linked list in sorted order either ascending or descending.
(c)	WAP to sort singly linked list.
VII. (a)	WAP to print alternate nodes from the list.
(b)	WAP to concatenate the even elements from the two linked list.
(c)	WAP to find first common element between two linked list.
(d)	WAP to find the number of occurrence of all the elements in a linked list.

Experiment 7 (Circular Singly Linked List and Doubly Linked List)	
I.	WAP to create a Circular Singly Linked List for the given elements with the following functions,
	(a) Inserting a node, before the node with key *givenkey,*
	(b) Inserting a node, after the node with key *givenkey,*
	(c) Accessing a node (finding the position wrt header),
	(d) Removing a node with particular key value,
	(e) Complete deletion of a list,
	(f) Displaying the current list, and
	(g) Sorting the list.
	Implement the above program for the elements as Strings.
II.	WAP to implement doubly linked list with the following operations;
	(a) Inserting a node, before the node with key *givenkey,*
	(b) Inserting a node, after the node with key *givenkey,*
	(c) Inserting a node at i^{ith} location wrt header.(assuming header at 1^{st} location)
	(d) Accessing a node (finding the position wrt header),
	(e) Removing a node with particular key value,
	(f) Complete deletion of a list,
	(g) Displaying the current list in clockwise fashion,
	(h) Displaying the current list in anti-clockwise fashion, and
	(i) Sorting the list.
	Implement the above program for the elements as *Student structures* (contains student_name, student_roll_no, total_marks) with key as student_roll_no.
III.	WAP to represent various sparse matrices using linked list and give addition function to add two sparse matrices.
IV.	Implement the above program (1) with the elements as *Employee structures* (contains employee_name, emp_no, emp_salary) with key as emp_no.
V.	Implement the above program (2) with the elements as *Faculty structures* (contains faculty_name, faculty_ID, subject_codes, class_names) with key as faculty_ID.
Experiment 8 (Binary Trees, BST, Heaps, Heap sort and Threaded Binary Trees)	
I. (a)	WAP to build a binary tree for the given elements (integers) and also give traversal functions : inorder, preorder, postorder.
(b)	WAP to traverse a given binary tree in inorder, preorder, postorder, converse inorder, converse preorder, converse postorder fashion. (converse - traverse in a reverse direction)
(c)	WAP to transform given tree into a binary tree.
(d)	WAP to represent an arithematic expression in binary tree format.

II.(a)	WAP to implement BST with insertion, search and deletion operation for the elements as strings.
(b)	WAP using function to find inorder-predecessor and inorder-successor for any node in a BST.
(c)	WAP using functions to find the height, number of levels, number of leave nodes, number of internal nodes and total number of nodes in the given BST.
III.(a)	WAP to implement priority queues using heaps (min heap or max heap) with add and delete functions.
(b)	WAP to perform heap sort on the given list of elements.
Experiment 9 (Graphs)	
I. (a)	WAP to represent the given graph in the form of adjacency list.
	 Fig:- A Graph.
(b)	WAP to find cut-set of the given graph.
II. (a)	WAP to represent the given graph in the form of adjacency matrix.
(b)	WAP to find cut-vertex of the given graph.
III.	WAP to perform BFS for any given graph.
IV.	WAP to perform DFS for any given graph.
Advanced Problems (Optional)	
I.	WAP to implement Shell and Bucket sort on 1D array of *Employee structures* (contains employee_name, emp_no, emp_salary) with key as emp_no.
II.	WAP to find the middle element of the given linked list in one pass. (Hint: take 2 pointers p and q. Initially p will point to 1st node and q will point to 2nd node. Advance p always by 1 and q by 2. Proceed this way until q reached last node, so now p will point to the middle element.)
III.	Write an efficient program to remove duplicates from a singly linked list.
IV.	WAP to convert BST into inorder Thereaded binary tree.
V.	WAP to convert BST into preorder Thereaded binary tree.
VI.	WAP to convert BST into postorder Thereaded binary tree.
VII.	WAP to find minimum cost spanning tree for the given tree using kruskal's method.
VIII.	WAP to find minimum cost spanning tree for the given tree using prims's method.