Data Structure and Algorithm Laboratory

**Assignment No. 1**

**Basic Algorithm and C++ Program**

| **Aim** |
| --- |
| Write an algorithm and C++ program to implement the following scenario:  Consider the students’ records containing students’ names and their examination score. Design an algorithm and write a C++ program that does the following:  i) calculates the average of examination marks.  ii) finds out and displays the names of all the students whose examination scores are below the average examination score.  iii) calculates the highest examination score.  iv) prints the names of all the students whose examination scores are the same as the highest examination score.  (Consider each part i), ii), iii) and iv) as an individual sub problem. The algorithm should depict the solutions of the sub problems.) |

| **Objective(s)** | |
| --- | --- |
| **1** | To study basic building blocks of C++ |
| **2** | To understand the concept of algorithm |
| **3** | To learn about variable declaration, input statements, output statements. |

| **Theory** |
| --- |
| 1. What is programming?  2. What is algorithm?  3. State and explain broad categories of software with suitable example.  4. What are the basic building blocks of C++ program?  5. Describe in details the steps involved in problem solving process.  6. Describe the analysis phase of programming.  (Write the detailed answers to the theory questions and upload the handwritten scanned copy along with the code in the classroom.) |

| **Conclusion** |
| --- |
|  |

**Assignment No. 2**

**Building Blocks, Input and Output of C++**

| **Aim** |
| --- |
| Implement the following statements:  1) Write a C++ program and algorithm that estimates and prints monthly wages for a worker. User will input the gross payment amount. The total payment is estimated and finalized subsequent to the following deductions:  Income Tax: 10%  State Tax: 3%  Social Security Tax: 2%  Medicare Tax: 2.78%  Pension Plan: 6%  2) Write a program that generates the following output:  CCCCCCCCC ++ ++  CC ++ ++  CC ++++++++++++++ +++++++++++++++  CC ++++++++++++++ +++++++++++++++  CC ++ ++  CCCCCCCCC ++ ++  (Do not write algorithm.)  3) Write a a C++ program and algorithm that asks the user to enter the temperature in Fahrenheit and outputs the equivalent temperature in Celsius. The formula to convert the temperature is: Celsius = 5 / 9 \* (Fahrenheit – 32). |

| **Objective(s)** | |
| --- | --- |
| **1** | To study the concept of a stream and examine input and output streams |
| **2** | To learn how to read data from the standard input device |
| **3** | To study input failures |

| **Theory** |
| --- |
| 1. Write an algorithm for swapping two values.  2. Write an algorithm to find the larger of two numbers.  3. Write an algorithm to find whether a number is even or odd.  4. Write an algorithm to find the sum of first N natural numbers.  5. A C++ program is required to calculate the value of distance, in miles, given this relationship: distance = average - speed × time  a. For this programming problem, how many outputs are required?  b. How many inputs does this problem have?  c. Write an algorithm for converting the input items into output items.  (Write the detailed answers to the theory questions and upload the handwritten scanned copy along with the code in the classroom.) |

| **Conclusion** |
| --- |
|  |

Data Structure and Algorithm Laboratory

**Assignment No. 3**

**Arrays and Strings**

| **Aim** |
| --- |
| Write a menu-driven program to perform various string operations such as copy, length, reversing, palindrome, concatenation and, to find occurrence of a sub-string using and without using library functions. |

| **Objective(s)** | |
| --- | --- |
| **1** | To study basics of linear data structures |
| **2** | To learn the concept of array and types of array |
| **3** | To understand use of array |
| **4** | To understand string operations without using library functions |

| **Theory** |
| --- |
| 1. What is array? How are arrays declared? How are the elements of an array stored in memory?  2. Define the terms: a) ADT b) Persistent data structures.  3. In what respect linear data structures differ from non-linear data structures? 4. What are advantages of array data structure?  5. What are disadvantages of array data structure?  6. State and explain applications of arrays.  7. What are the properties of abstract data types?  8. How are arrays declared? |

**Algorithms:**

**1. Algorithm for finding the string length**

**Algorithm mystrlen (char str[] )**

This algorithm reads a Source String character by character and counts characters till end of the string and returns length of the String.

**Pre-condition : String should be accepted**

**Post-condition : Length of the String should be calculated.**

**Return : Length of the String.**

**1** Initialize index = 0

**2** while ( src[index] != ‘\0’ ) repeat step 3 & 4

**3** len = len +1

**4** index = index + 1

**5** end

**6** return len

Page **1** of **5**

Data Structure and Algorithm Laboratory

**2. Algorithm for String Copy operation**

**Algorithm mystrcpy (char des[] , char src[] )**

This algorithm reads a Source String character by character and copies it to the destination string. **Pre-condition : Source string should be accepted**

**Post-condition : copy of source should be present in destination string.**

**Return : reference to Copied String**

**1** index = 0

**2** while (src[index] != ‘\0’ ) repeat step 3 & 4

**3** dest[index] = src[index]

**4** index = index + 1

**5** end

**6** dest[index] =‘\0’

**7** return reference to the Copied String

**3. Algorithm for Sting concatenation Operation**

**Algorithm mystrcat (char des[] , char src[] )**

This algorithm reads the source string character by character and appends each character at the end of the destination string and returns the reference to the destination string.

**Pre-condition : Both the Strings should be accepted**

**Post-condition : Source String should be appended at the end of destination string Return : reference to destination String**

**1** index1 = 0

**2** while( not end of the des)

**3** index1 = index1 + 1

**4** end

**5** index2 = 0

**6** while ( not end of src )

**7** des[index1] = src[index2]

**8** index1 = index1 +1

**9** index2 = index2 + 1

**10** end

**11** terminate destination string (des) by ‘\0’ character

**12** return reference to destination string (des)

Page **2** of **5**

Data Structure and Algorithm Laboratory

**4. Algorithm for String Reverse Operation**

**Algorithm mystrrev (char src[] )**

This algorithm reads the source string character by character and adds each character to the source string in reverse order and returns the reference to the destination string.

**Pre-condition : The source string should be accepted**

**Post-condition : Reverse of the Source string should be present in source string. Return : reference to source string**

**1** start\_index = 0

**2** end\_index = 0

**3** while ( src [ end\_index ] != ‘\0’ )

**4** end\_index = end\_index + 1

**5** end

**6** end\_index = end\_index – 1

**7** while ( start\_index < end\_index )

**8** Swap src[start\_index] & src[end\_index]

**9** start\_index = start\_index + 1

**10** end\_index = end\_index – 1

**11** end

**10** return reference to source string

**5. Algorithm for Palindrome check**

**Algorithm mypalindrome (char src[] )**

This algorithm reads the String character by character and compares first character to last character, second to second last and continues till middle character (checks if string is palindrome). **Pre-condition : Source String should be accepted**

**Post-condition : Result as palindrome or not a palindrome.**

**Return : Boolean value as a result (1/0)**

**1** start\_index = 0

**2** end\_index = strlen(src)-1

**3** while ( start\_index < end\_index)

**4** if src[start\_index] == src[end\_index]

**5** start\_index = start\_index + 1

**6** end\_index = end\_index - 1

**7** else

**8** goto step 10

**9** end

**10** if (start\_index < end\_index )

**11** return 0

**12** else

**13** return 1

Page **3** of **5**

Data Structure and Algorithm Laboratory

**6. Algorithm for String compare operations**

**Algorithm mystrcmp( char str1[],char str2[])**

This algorithm reads both the Strings character by character and compares characters till end of the string and returns length of the String.

**Pre-condition : Both the Strings should be accepted**

**Post-condition : Equality of the Strings should be checked.**

**Return : Result of comparison (+ve if str1 > str2, -ve if str1 < str2 , Zero if equal) (ascii difference)**

**1** index= 0;

**2** while( str1[index] != ‘\0’ || str2[index ]!= ‘\0’)

**3** if (str1[index] == str2[index])

**4** index = index + 1

**5** else

**6** break;

**7** end

**8** end

**9** diff = str1[index] - str2[index]

**10** return diff

**7. Algorithm for substring operations**

**Algorithm SubString (char str1[],char str2[])**

This algorithm reads both the Strings character by character and compares characters to check occurrence of str2 in str1 till end of the string and returns boolean result (true/false).

**Pre-condition : Both the Strings should be accepted**

**Post-condition : Check for occurrence of string2 in string1.**

**Return : No of occurrence of the substring in the main string**

**1** L1 = string length of str1

**2** L2 = string length of str2

**3** if L2 > L1

**4** Return 0 // not a substring i.e. 0 occurrences

**5** end

**6** Count = 0 // occurrence count

**7** for i = 0 to L1 – L2

**8** for j = 0 to L2-1

**9** if str1[i+j] is not equal to str2[j]

**10** break;

**11** end if

**12** end for

**13** if j == L2

**14** Increment count // Increment substring count

**15** end if

**16** end for

**17** return Count

Page **4** of **5**

**Test Conditions:**

1. Input two strings with same length. 2. Input two strings without same length.

Data Structure and Algorithm Laboratory

3. Input such string whose reverse will be equal as original.

4. Input empty strings.

**Sample Input Output**

**\*\*\* MENU FOR STRING OPERATIONS \*\*\***

1. String Length

2. String Copy

3. String Concatenation

4. String Reverse

5. String Palindrome

6. String Compare

7. Substring

**Input :**

Let String 1 = “ Fundamentals”

String 2 = “ment”

Choose one of the operations to be performed.

**Output :**

1. StringLength (string1 )

Length of string1 i.e. 12 will be returned

2. StringCopy (string1,string2)

String 2 will also consist “ Fundamentals”

3. StringConcat (string1,string2)

String2 will get appended to string1

String2 will have “ FundamentalsFundamentals”

4. StringReverse (string1,string2)

String2 will consist reverse of string1 i.e. “slatnemadnuF”

5. palindrome(string1)

if string1 consists “ Fundamentals”

6. StringCompare (string1,string2)

String 1 is greater than string 2

7. SubString (string1,string2)

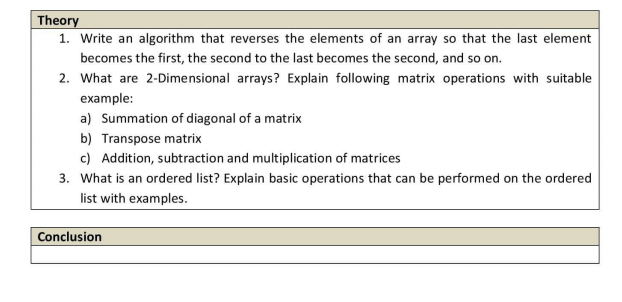
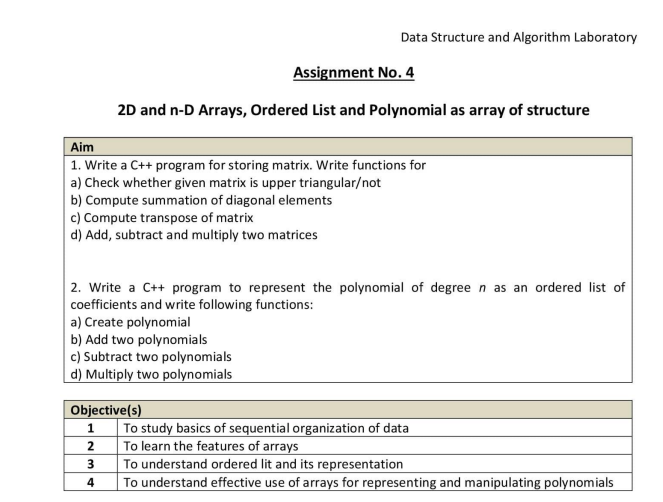
If string1 = “ Fundamentals” and string2 =”ment”

String 2 is the substring of string1.

The string is not a pallindrome.

| **Conclusion** |
| --- |
|  |

Page **5** of **5**

****

Data Structure and Algorithm Laboratory

**Assignment No. 5**

**Linked Lists**

| **Aim** |
| --- |
| 1. Write a C++ program to maintain employee’s information using singly linked list. Store Employee ID and Employee Name. Write functions for  a) Computing total no. of employees in the industry  b) Displaying all employee’s details (Employee ID and Employee Name)  c) Inserting new employee  d) Deleting existing employee  e) Displaying list in reverse order using recursion  f) If there are two linked lists for two departments, then concatenate two lists.  2. Write a C++ program to sort binary numbers with the help of doubly linked list. Write functions for  a) Addition of two binary numbers  b) Calculation of 1’s complement of a given binary number  c) Calculation of 2’s complement of a given binary number |

| **Objective(s)** | |
| --- | --- |
| **1** | To study basics of linked list |
| **2** | To learn the features of linked list over array |
| **3** | To understand the concept of linked list and its representation |
| **4** | To study types of linked lists: Singly linked list, Circular linked list, Doubly linked list, Doubly circular linked list |

| **Theory** |
| --- |
| 1. Define: Linked list. Comment on: The features of linked list over array.  2. How many pointers are required for implementing a singly linked list?  3. State and explain different types of linked list. How a linked list node can be represented?  4. Compare: Singly linked list and Doubly linked list.  5. Give applications of linked list.  6. Write down the detailed algorithms for the given problem statement. |

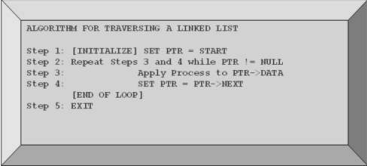
**Algorithms:**

**a) Singly Linked List Operations:**

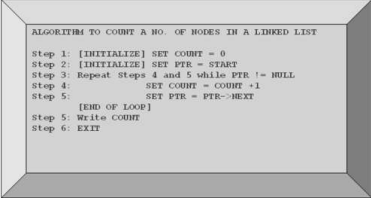
**1) Algorithm for Traversing a Linked List**

Page **1** of **12**

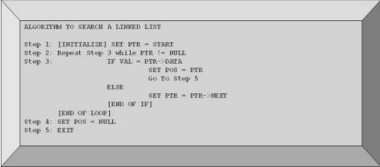
Data Structure and Algorithm Laboratory



**2) Algorithm to count a no. of nodes in a liked list**

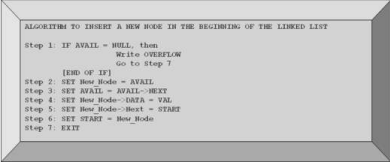
****

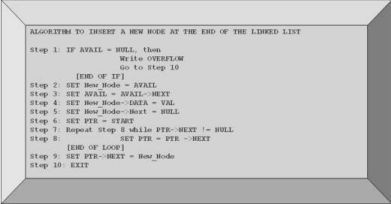
**3) Algorithm for Searching a Linked List**

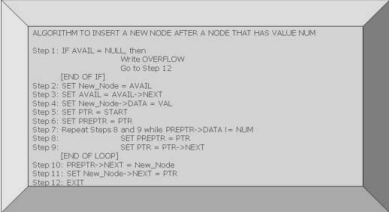
****Page **2** of **12**

**4) Inserting a Node at the Beginning**

Data Structure and Algorithm Laboratory

**5) Inserting a Node at the End**

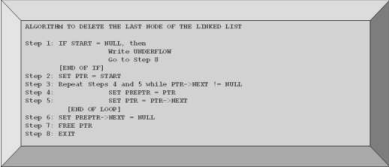
**6) Inserting a Node after Node that has Value NUM**

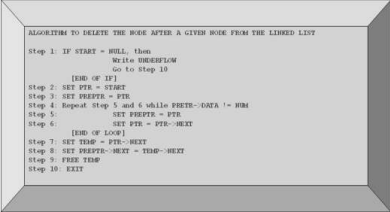
****Page **3** of **12**

**7) Deleting the First Node**

Data Structure and Algorithm Laboratory

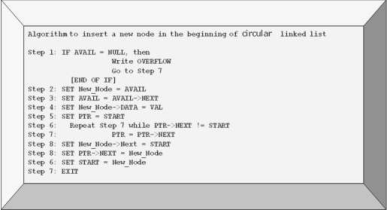
**8) Deleting the Last Node**

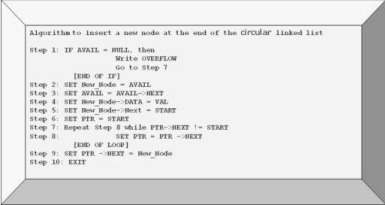
**9) Deleting the Node After a Given Node**

****Page **4** of **12**

**b) Circular Singly Linked List Operations: 1) Inserting a Node at the Beginning**

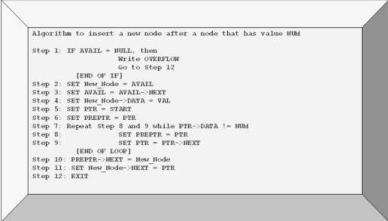
Data Structure and Algorithm Laboratory

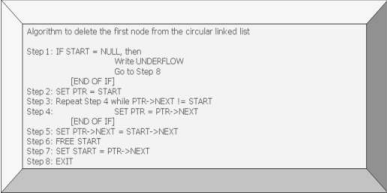
**2) Inserting a Node at the End**

****Page **5** of **12**

Data Structure and Algorithm Laboratory

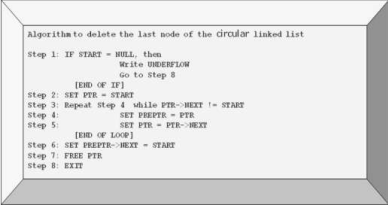
**3) Inserting a Node after Node that has Value NUM**

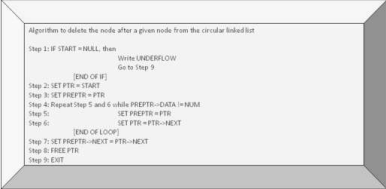
**4) Deleting the First Node**

****Page **6** of **12**

**5) Deleting the Last Node**

Data Structure and Algorithm Laboratory

**6) Deleting the Node After a Given Node**

****Page **7** of **12**

**c) Doubly Linked List Operations: 1) Inserting a Node at the Beginning**

Data Structure and Algorithm Laboratory

**2) Inserting a Node at the End**

****Page **8** of **12**

Data Structure and Algorithm Laboratory

**3) Inserting a Node after Node that has Value NUM**

**4) Deleting the First Node**

**5) Deleting the Last Node**

****Page **9** of **12**

**6) Deleting the Node After a Given Node**

Data Structure and Algorithm Laboratory



**d) Circular Doubly Linked List Operations:**

**1) Inserting a Node at the Beginning**

****Page **10** of **12**

**2) Inserting a Node at the End**

Data Structure and Algorithm Laboratory

**3) Inserting a Node after Node that has Value NUM**

**4) Deleting the First Node**

****Page **11** of **12**

**5) Deleting the Last Node**

Data Structure and Algorithm Laboratory

**6) Deleting the Node After a Given Node**

****

| **Conclusion** |
| --- |
|  |

Page **12** of **12**

Data Structure and Algorithm Laboratory

**Assignment No. 6**

**Stack ADT as an Array**

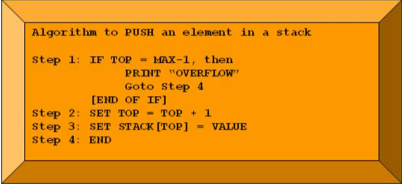
| **Aim** |
| --- |
| In any language program mostly syntax error occurs due to unbalancing delimiter such as ( ),{ },[ ]. Write C++ program using stack to check whether given expression is well parenthesized or not. Implement Stack ADT as an Array. |

| **Objective(s)** | |
| --- | --- |
| **1** | To study basics of stack |
| **2** | To learn the stack operations |
| **3** | To understand the concept of Stack ADT as an Array |

| **Theory** |
| --- |
| 1. Define: Stack.  2. What are different operations that can be performed on a stack?  3. What are the applications of stack?  4. Comment on: Stack ADT as an Array.  5. Write algorithms for PUSH, POP and PEEK/TOP operations of the stack. |

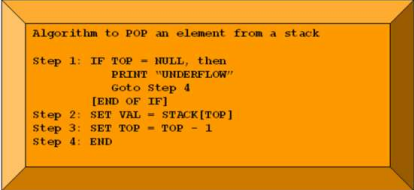
**Algorithms:**

**a) PUSH Operation:**

****Page **1** of **2**

Data Structure and Algorithm Laboratory

**b) POP Operation:**

**c) PEEK/TOP Operation:**

****

| **Conclusion** |
| --- |
|  |

Page **2** of **2**

Data Structure and Algorithm Laboratory

**Assignment No. 7**

**Stack ADT as a Linked List**

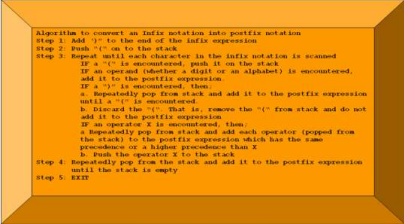
| **Aim** |
| --- |
| Write a program to implement stack as an abstract data type using linked list and use this ADT for conversion of infix expression to postfix, prefix and evaluation of postfix/prefix expression. |

| **Objective(s)** | |
| --- | --- |
| **1** | To study basics of stack |
| **2** | To learn the stack operations |
| **3** | To understand the concept of stack ADT as a linked list |

| **Theory** |
| --- |
| 1. What is stack overflow and underflow?  2. Differentiate between: Array and Stack.  3. How a stack implemented using a linked list differs from a stack implemented using an array?  4. How stacks are used in a non-recursive program?  5. Explain: Infix, Prefix and Postfix Expressions.  6. Convert the following infix expressions to their equivalent postfix expressions: a) A + B \* C / (E - F)  b) (A ^ B \* (C + (D \* E) - F ) ) / G  c) (A + (B \* C - (D / E ^ F) \* G) \* H) |

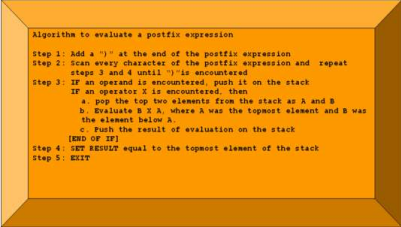
**Algorithms:**

**a) Infix to Postfix Conversion:**

****Page **1** of **2**

**b) Evaluation of Postfix Expression:**

Data Structure and Algorithm Laboratory



| **Conclusion** |
| --- |
|  |

Page **2** of **2**

Data Structure and Algorithm Laboratory

**Assignment No. 8**

**Implementation of Circular Queue**

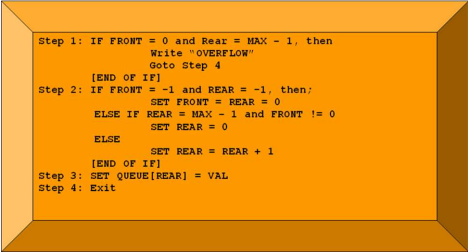
| **Aim** |
| --- |
| Implementation of circular queue using array and perform following operations on it. i) Add a record ii) Delete a record iii) Checking Empty iv) Checking Underflow v) Checking overflow |

| **Objective(s)** | |
| --- | --- |
| **1** | To understand the concept of Queue. |
| **2** | To understand the concept of Circular Queue. |
| **3** | To study how Queue is represented as an ADT. |

| **Theory** |
| --- |
| 1. What is Queue? Explain Queue operations with neat diagrams.  2. Explain how Queue can be implemented as an ADT.  3. What is Circular Queue? Explain with example. |

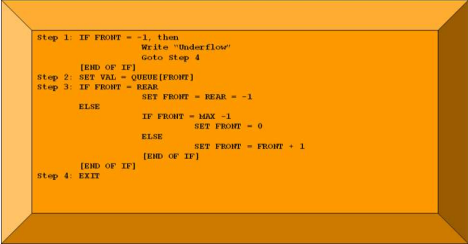
**Algorithms:**

**a) Algorithm to Insert an Element in a Circular Queue:**

****Page **1** of **2**

Data Structure and Algorithm Laboratory

**b) Algorithm to Delete an Element from a Circular Queue:**

****

| **Conclusion** |
| --- |
|  |

Page **2** of **2**

Data Structure and Algorithm Laboratory

**Assignment No. 9**

**Searching and Sorting**

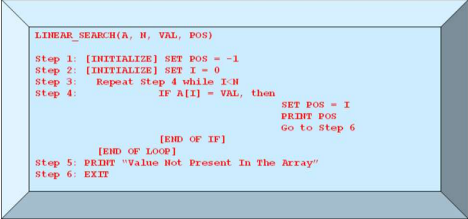
| **Aim** |
| --- |
| To implement the following Searching and Sorting methods:  Searching: Sequential/Linear Search and Binary Search.  Sorting: Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Heap Sort and Quick Sort. |

| **Objective(s)** | |
| --- | --- |
| **1** | To study searching strategies. |
| **2** | To study sorting techniques. |
| **3** | To implement searching and sorting methods. |

| **Theory** |
| --- |
| 1. State and explain: sequential search and binary search. Write detailed algorithm for the same.  2. State and explain: Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Heap Sort and Quick Sort. Write the detailed algorithm for all sorting methods. |

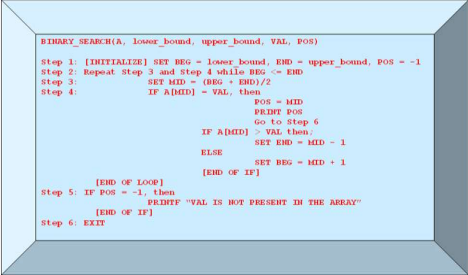
**Algorithms:**

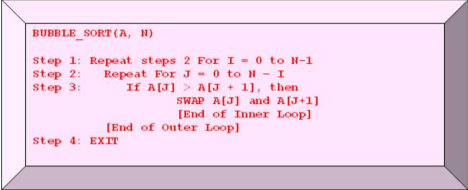
**Linear Search:**

****Page **1** of **5**

Data Structure and Algorithm Laboratory

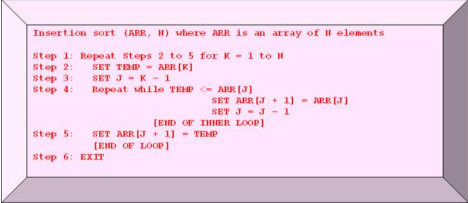
**Binary Search:**

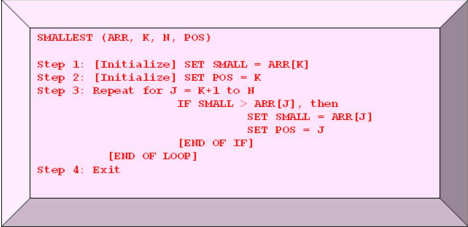
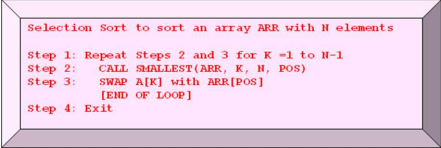
**Bubble Sort:**

****Page **2** of **5**

Data Structure and Algorithm Laboratory

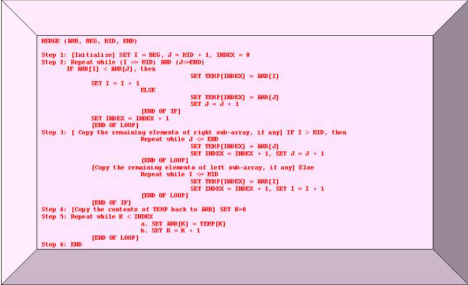
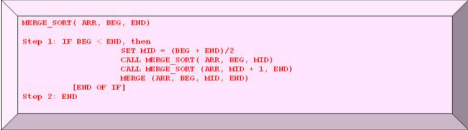
**Insertion Sort:**

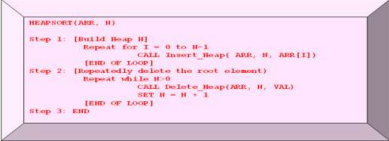
**Selection Sort:**

****Page **3** of **5**

Data Structure and Algorithm Laboratory

**Merge Sort:**

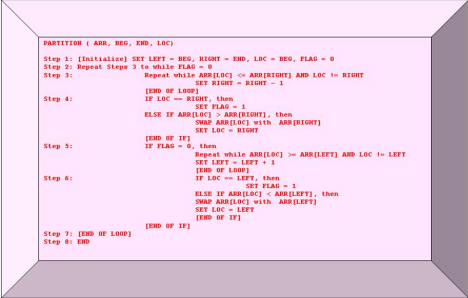
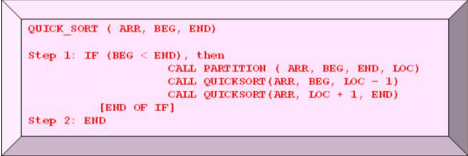
**Heap Sort:**

****(Students should write Insert\_Heap and Delete\_Heap functions)

Page **4** of **5**

Data Structure and Algorithm Laboratory

**Quick Sort:**

****

| **Conclusion** |
| --- |
|  |

Page **5** of **5**

Data Structure and Algorithm Laboratory

**Assignment No. 10**

**Operations on Binary Search Tree**

| **Aim** |
| --- |
| Implement binary search tree as an ADT. Perform following operations: a) Insert, b) Delete, c) Depth of the tree, d) Search a node, e) Find its mirror image f) Print original g) Mirror image level wise. |

| **Objective(s)** | |
| --- | --- |
| **1** | To understand the concept of binary search tree as a data structure. |
| **2** | To study applications of BST. |

| **Theory** |
| --- |
| 1. What is Binary search tree?  2. What are the members of structure of tree & what is the size of structure?  3. What are rules to construct binary search tree?  4. How general tree is converted into binary tree?  5. What is binary threaded tree?  6. What is use of thread in traversal? |

**Definition of binary search tree**

A binary tree in which each internal node x stores an element such that the element stored in the left subtree of x are less than or equal to x and elements stored in the right subtree of x are greater than or equal to x. This is called binary-search-tree property.

**Searching**

Searching a binary tree for a specific value can be a recursive or iterative process. This explanation covers a recursive method. We begin by examining the root node. If the tree is null, the value we are searching for does not exist in the tree. Otherwise, if the value equals the root, the search is successful. If the value is less than the root, search the left subtree. Similarly, if it is greater than the root, search the right subtree. This process is repeated until the value is found or the indicated subtree is null. If the searched value is not found before a null subtree is reached, then the item must not be present in the tree.

**Insertion**

Insertion begins as a search would begin; if the root is not equal to the value, we search the left or right subtrees as before. Eventually, we will reach an external node and add the value as its right or left child, depending on the node's value. In other words, we examine the root and recursively insert the new node to the left subtree if the new value is less than the root, or the right subtree if the new value is greater than or equal to the root.

Page **1** of **4**

Data Structure and Algorithm Laboratory

**Deletion**

There are three possible cases to consider:

∙ **Deleting a leaf (node with no children):** Deleting a leaf is easy, as we can simply remove it from the tree.

∙ **Deleting a node with one child:** Remove the node and replace it with its child. ∙ **Deleting a node with two children:** Call the node to be deleted N. Do not delete N. Instead, choose either its in-order successor node or its in-order predecessor node, R. Replace the value of N with the value of R, then delete R.

As with all binary trees, a node's in-order successor is the left-most child of its right subtree, and a node's in-order predecessor is the right-most child of its left subtree. In either case, this node will have zero or one children. Delete it according to one of the two simpler cases above.

**ALGORITHM:**

Define structure for Binary Tree (Mnemonics, Left Pointer, Right Pointer)

**Insert Node:**

Insert(Root, Node)

Root is a variable of type structure ,represent Root of the Tree. Node is a variable of type structure ,represent new Node to insert in a tree.

Step 1: Repeat Steps 2,3 & 4 Until Node do not insert at appropriate position. Step 2: If Node Data is Less that Root Data & Root Left Tree is NULL

Then insert Node to Left.

Else Move Root to Left

Step 3 : Else If Node Data is Greater that Equal that Root Data & Root Right Tree is NULL Then insert Node to Right.

Else Move Root to Right.

Step 4: Stop.

**Search Node:**

Search (Root, Mnemonics)

Root is a variable of type structure ,represent Root of the Tree. Mnemonics is array of character. This function search Mnemonics in a Tree.

Step 1: Repeat Steps 2,3 & 4 Until Mnemonics Not find && Root != NULL Step 2: If Mnemonics Equal to Root Data

Then print message Mnemonics present.

Step 3 : Else If Mnemonics Greater that Equal that Root Data

Then Move Root to Right.

Step 4 : Else Move Root to Left.

Step 5: Stop.

**Delete Node:**

Dsearch(Root, Mnemonics)

Root is a variable of type structure ,represent Root of the Tree. Mnemonics is array of character. Stack is an pointer array of type structure. PTree(Parent of Searched Node),Tree(Node to be deleted), RTree(Pointg to Right Tree),Temp are pointer variable of type structure; Step 1: Search Mnemonics in a Binary Tree

Step 2: If Root = = NULL Then Tree is NULL

Page **2** of **4**

Data Structure and Algorithm Laboratory

Step 3: Else //Delete Leaf Node

If Tree->Left = = NULL && Tree->Right = = NULL

Then a) If Root = = Tree Then Root = NULL;

b) If Tree is a Right Child PTree->Right=NULL;

Else PTree->Left=NULL;

Step 4: Else // delete Node with Left and Right children

If Tree->Left != NULL && Tree->Right != NULL

Then a) RTree=Temp=Tree->Right;

b)Do steps i && ii while Temp->Left !=NULL

i) RTree=Temp;

ii) Temp=Temp->Left;

c)RTree->Left=Temp->Right;

d)If Root == Tree//Delete Root Node

Root=Temp;

e)If Tree is a Right Child PTree->Right=Temp;

Else PTree->Left=Temp;

f)Temp->Left=Tree->Left;

g)If RTree!=Temp

Then Temp->Right = Tree->Right;

Step 5: Else //with Right child

If Tree->Right!= NULL

Then a) If Root==Tree Root = Root->Right;

b) If Tree is a Right Child PTree->Right=Tree->Right;

Else PTree->Left=Tree->Left;

Step 6: Else //with Left child

If Tree->Left != NULL

Then a) If Root==Tree Root = Root->Left;

b) If Tree is a Right Child PTree->Right=Tree->Left;

Else PTree->Left=Tree->Left;

Step 7: Stop.

**Depth First Search:**

Root is a variable of type structure ,represent Root of the Tree. Stack is an pointer array of type structure. Top variable of type integer. DFS(Root)

Step 1: Repeat Steps 2,3,4,5,6 Until Stack is Empty

Step 2: print Root Data

Step 3: If Root->Right != NULL//Root Has a Right SubTree

Then Stack[Top++] = Tree->Right;//Push Right Tree into Stack Step 4: Root = Root ->Left;//Move to Left

Step 5: If Root == NULL

Step 6: Root = Stack[--Top];//Pop Node from Staacl

Step 7: Stop.

**Breath First Search (Level wise Display):**

Root is a variable of type structure ,represent Root of the Tree. Queue is an pointer array of type structure. Front & Rear variable of type integer. BFS(Root)

Step 1:If Root == NULL Then Empty Tree;

Step 2: Else Queue[0] = Root;// insert Root of the Tree in a Queue

Page **3** of **4**

Step 3: Repeat Steps 4,5,6 & 7 Until Queue is Empty

Data Structure and Algorithm Laboratory

Step 4: Tree=Queue[Front++]; //Remove Node From Queue

Step 5: print Root Data

Step 6: If Root->Left != NULL

Then Queue[++Rear] = Tree->Left;//insert Left Subtree in a Queue

Step 7: If Root->Right != NULL

Then Queue[++Rear] = Root->Right; //insert Left Subtree in a Queue

Else if Root->Right == NULL And Root->Left == NULL

Leaf++;//Number of Leaf Nodes

Step 8: Stop.

**Mirror Image:**

Root is a variable of type structure ,represent Root of the Tree.

Queue is an pointer array of type structure. Front & Rear variable of type integer. Mirror(Root)

Step 1: Queue[0]=Root;//Insert Root Node in a Queue

Step 2: Repeat Steps 3,4,5,6,7 & 8 Until Queue is Empty

Step 3: Root = Queue[Front++];

Step 4: Temp1 = Root->Left;

Step 5: Root->Left = Root->Right;

Step 6: Root->Right = Temp1;

Step 7: If Root->Left != NULL

Then Queue[Rear++] = Tree->Left;//insert Left SubTree

Step 8: If Root->Right != NULL

Then Queue[Rear++] = Root->Right;//insert Right SubTree

Step 9: Stop.

| **Conclusion** |
| --- |
|  |

Page **4** of **4**

Data Structure and Algorithm Laboratory

**Assignment No. 11**

**Minimum Spanning Tree (MST)**

**Implementation of Prim’s & Kruskal’s algorithms**

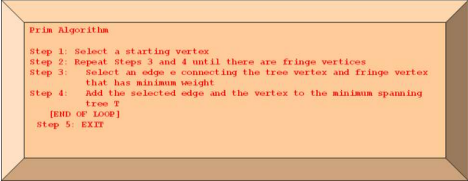
| **Aim** |
| --- |
| Represent any real world graph using adjacency list/adjacency matrix find minimum spanning tree using Prim’s or Kruskal’s algorithm. |

| **Objective(s)** | |
| --- | --- |
| **1** | Learn the concepts of graph as a data structure and their applications in everyday life. |
| **2** | Understand graph representation (adjacency matrix, adjacency list, adjacency multi list) |

| **Theory** |
| --- |
| 1. What is a graph? Explain in brief the basic terminologies used in graph.  2. State and explain different representations of graph.  3. Explain Prim’s & Kruskal’s Algorithm with suitable example. |

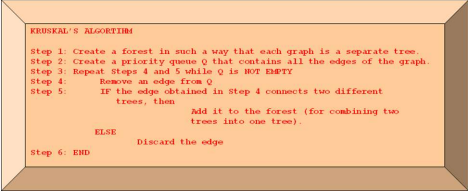
Algorithm:

**Prim’s Algorithm**

****Page **1** of **2**

Data Structure and Algorithm Laboratory

**Kruskal’s Algorithm**

****

| **Conclusion** |
| --- |
|  |

Page **2** of **2**