Institute of Engineering & Management
Department of Computer Science & Engineering
Data Structure Laboratory for 2nd year 3rd semester
Code: CS 392

Data Structure &
Algorithms Laboratory
Manual
(CS 392)

Instructor:
Prof. Nilanjana Dutta Roy
OBJECTIVE:

The objective of this lab is to teach students various data structures and to explain them algorithms for performing various operations on these data structures. This lab complements the data structures course. Students will gain practical knowledge by writing and executing programs in C using various data structures such as arrays, linked lists, stacks, queues, trees, graphs, hash tables and search trees.

OUTCOMES:

Upon the completion of Data Structures practical course, the student will be able to:

1. Design and analyze the time and space efficiency of the data structure.
2. Identity the appropriate data structure for given problem.
3. Understand the applications of data structures.
4. Choose the appropriate data structure and algorithm design method for a specified application.
5. Understand which algorithm or data structure to use in different scenarios.
6. Understand and apply fundamental algorithmic problems including Tree traversals, Graph traversals.
7. Compare different implementations of data structures and to recognize the advantages and disadvantages of them.
8. Write complex applications using structured programming methods.
Department of Computer Science & Engineering
Data Structure Laboratory for 2nd year 3rd semester
Code: CS 392

General Instructions for the students:

1. Sit on your assigned computer. Do not change your place.
2. Complete your assignments within the same day itself.
3. Based on the performance (class performance & behavior), each student will be awarded with 5 marks on the same day. At the end of the semester, these marks will be considered as final internal assessment marks.
4. Save your programs in a folder with your Name, Section, Roll no & Group no. After finishing the work, send the folder to your mail by that day itself & after sending, delete it from the computer.
5. Bring your assignments on every next class.
6. Do not use pen drive.
7. If you are facing any problem with Monitor, Keyboard, Mouse, Power cable etc., register you complain in the particular log register.
8. Keep your Laboratory clean.
Institute of Engineering & Management
Department of Computer Science & Engineering
Data Structure Laboratory for 2\textsuperscript{nd} year 3\textsuperscript{rd} semester
Code: CS 392

Guidelines to write the assignments:

Problem Statement: Write the Problem here. Ex: Assignment 1. No1. (Font: Calibri 12)

Algorithm: Write/Type the algorithm for the particular problem here (Font: Calibri 11).

Source code: Paste/write your program’s source code here (Font: Calibri 11).

Output: Paste/write/type the output of the executed program here (Font: Calibri 11).

Name: , Sec , Roll , Group
Institute of Engineering & Management  
Department of Computer Science & Engineering  
Data Structure Laboratory for 2\textsuperscript{nd} year 3\textsuperscript{rd} semester  
Code: CS 392

\textbf{Student Data sheet}

Name:  
Class Roll no:  
Mobile no:  
Section & Group no:  
email\_id:  
Father’s Name with mobile no:  

1. Regular Lab marks (out of 5):

<table>
<thead>
<tr>
<th>Day1</th>
<th>Day2</th>
<th>Day3</th>
<th>Day4</th>
<th>Day5</th>
<th>Day6</th>
<th>Day7</th>
<th>Day8</th>
<th>Day9</th>
<th>Day10</th>
<th>Day11</th>
<th>Day12</th>
<th>Day13</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Remarks:  

(NILANJANA DUTTA ROY)  
Signature: (Student)  
Signature: (Instructor)
Institute of Engineering & Management  
Department of Computer Science & Engineering  
Data Structure Laboratory Assignments for 2\textsuperscript{nd} year 3\textsuperscript{rd} semester  
Code: CS 392

\textbf{Syllabus MAKAUT:}

Data Structure Lab  
Code: CS 392  
Contact: 3P  
Credit: 2

Experiments should include but not limited to:

- **Implementation of array operations**: Stacks and Queues: adding, deleting elements  
  Circular Queue: Adding & deleting elements  
  Merging Problem: Evaluation of expressions operations on multiple stacks & queues.
- **Implementation of linked lists**: inserting, deleting, and inverting a linked list.  
  Implementation of stacks & queues using linked lists, Polynomial addition, Polynomial multiplication, Sparse Matrices: Multipli-
  cation, addition.
- **Trees**: Recursive and No recursive traversal of Trees, Threaded binary tree traversal, AVL tree implementation
- **Application of Trees**: Application of sorting and searching algorithms
- **Graph**: BFS, DFS, Spanning tree by Prim’s & Krushkal’s
- **Hash tables implementation**: searching, inserting and deleting, searching & sorting techniques.
ASSIGNMENT 1 [1Day]:
(Revision of C)

1. Take any random elements of same kind and arrange them in a sequential fashion (array).
   a. Program to Find the Number of Elements in an Array (Don’t ask user to enter the size)
   b. Print the Alternate Elements in an Array
   c. Increment every Element of the Array by one & Print Incremented Array (pass the whole array through a function and make necessary changes within the function body itself.
   d. Find the Number of Non Repeated Elements in an Array
   e. Segregate 0s on Left Side & 1s on right side of the Array (Traverse Array only once)
   f. Pass the middle value of the array to a function to modify it by adding 10 with it.
   g. Sort the above array using Bubble sort logic and print it.

2. Write a program to read, display, add & subtract two distances. Distance must be defined using kms & meters. (Use a structure, two functions)

3. Write a program to enter a character & determine whether it is a Vowel or not using switch case statement.

4. Implement Binary search.

5. Write a program to read a text, delete all the semi-colons it has & finally replace all ‘,’ with a ‘.’

6. Write a program to copy the last n characters of a character array in another character array. Also, convert the lower case letters into upper case while copying.
ASSIGNMENT 2 [2 days]:
(STACK)
a) Implement Stack using Array.
b) Convert any infix expression into its postfix form using Stack.
c) Convert any infix expression into its prefix form using Stack.
d) Evaluate any prefix expression using stack.
e) Implement Tower of Hanoi problem.

ASSIGNMENT 3 [1 day]:
(QUEUE)
a) Implement Queue using array.
b) Implement circular queue.
c) Implement de-queue.

ASSIGNMENT 4 [2 Days]:
(Linked list)
a) Program to implement single Linked list. (Insert from Beginning, at end, anywhere in the list. Delete from front, end & from anywhere)
b) Count the number of nodes.
c) Search any element from a single Linked List.
d) Print the Single linked list in its reverse order.

ASSIGNMENT 4 [2 Days] continued..
(Linked list)
a) Implement Circular queue.
b) Implement Josephus problem using circular linked list.
c) Add two polynomial expressions using Linked List.
d) Implement Stack & Queue using Linked List.
e) Implement Sparse Matrix using Linked List.
ASSIGNMENT 5 [3 Days]  
(TREE)  
a) Program to implement Binary tree using Double Linked List. (Insert & delete elements)  
b) Implement BST using double Linked list.  
c) Implement AVL Tree (Insertion, deletion).  
d) Implement B Tree & B+ tree.

ASSIGNMENT 6 [1 Days]  
(GRAPH)  
a) Implement DFS & BFS algorithms.  
b) Find out minimal spanning tree using Prim’s & Krushkal’s algorithm.

ASSIGNMENT 7 [2 Days]  
(SORTING & SEARCHING)  
a) Merge sort.  
b) Quick sort  
c) Heap sort  
d) Insertion sort  
e) Selection sort.
Institute of Engineering & Management  
Department of Computer Science & Engineering  
Data Structure Laboratory Assignments for 2nd year 3rd semester  
Code: CS 392

Course Plan:

<table>
<thead>
<tr>
<th>Sl no</th>
<th>Day</th>
<th>DETAILS</th>
</tr>
</thead>
</table>
| 1     | Day 1 | 1. Take any random elements of same kind and arrange them in a sequential fashion (array).  
|       |       | a. Program to Find the Number of Elements in an Array (Don’t ask user to enter the size)  
|       |       | b. Print the Alternate Elements in an Array  
|       |       | c. Increment every Element of the Array by one & Print Incremented Array (pass the whole array through a function and make necessary changes within the function body itself).  
|       |       | d. Find the Number of Non Repeated Elements in an Array  
|       |       | e. Segregate 0s on Left Side & 1s on right side of the Array (Traverse Array only once)  
|       |       | f. Pass the middle value of the array to a function to modify it by adding 10 with it.  
|       |       | g. Sort the above array using Bubble sort logic and print it. |
| 2     | Day2-Day3 | STACK:  
|       |       | 1) Implement Stack using Array.  
|       |       | 2) Convert any infix expression into its postfix form using Stack.  
|       |       | 3) Convert any infix expression into its prefix form using Stack  
|       |       | 4) Evaluate any prefix expression using stack.  
|       |       | 5) Evaluation of the postfix expression using stack.  
<p>|       |       | 6) Implement Tower of Hanoi problem. |</p>
<table>
<thead>
<tr>
<th>Day</th>
<th>3</th>
<th>QUEUE:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1) Implement Queue using array.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2) Implement circular queue.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3) Implement de-queue.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Day</th>
<th>4</th>
<th>LINKED LIST (Day 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1) Program to implement single Linked list. (Insert from Beginning, at end, anywhere in the list. Delete from front, end &amp; from anywhere)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2) Count the number of nodes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3) Search any element from a single Linked List.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4) Print the Single linked list in its reverse order.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5) Divide the linked list in equal number of nodes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6) Sort two linked lists &amp; merge them together to get a final sorted list.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Day</th>
<th>5</th>
<th>LINKED LIST (Day 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1) Implement Circular Linked list.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2) Implement Josephus problem using circular linked list.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3) Add two polynomial expressions using Linked List.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4) Implement Stack &amp; Queue using Linked List.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5) Implement Sparse Matrix using Linked List.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Day</th>
<th>6</th>
<th>TREE (Day 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1) Program to implement Binary tree using Double Linked List. (Insert &amp; delete elements)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2) Binary tree traversal</td>
</tr>
<tr>
<td>Day</td>
<td>Task</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td></td>
</tr>
</tbody>
</table>
| 7   | TREE (Day 2)  
3) Implement BST using double Linked list.  
4) Implement AVL Tree (Insertion, deletion). |
| 8   | TREE (Day 3)  
4) Implement B Tree & B+ tree. |
| 9   | GRAPH  
1) Implement DFS & BFS algorithms.  
2) Find out minimal spanning tree using Prim’s & Krushkal’s algorithm. |
| 10  | SORTING (Day 1)  
1) Merge sort.  
2) Quick sort  
3) Heap sort |
| 11  | Hashing  
1) Hash table implementation  
2) Inserting, Searching & deletion from Hash table |
Assignment 1

- **OBJECTIVE**: To brush up previous knowledge of C.
- **RESOURCE**: GCC compiler
- **PROCEDURE**: Go to debug -> run or press CTRL + F9 to run the program.

**PROGRAM LOGIC:**

1. a. C Program to Find the Number of Elements in an Array

```c
#include <stdio.h>
#include<conio.h>
#include <stdlib.h>

int main()
{
    int array[] = {15, 50, 34, 20, 10, 79, 100, 500};
    int n;
    n = sizeof(array);
    printf("Size of the given array is %d\n", n/sizeof(int));
    getch();
    return 0;
}
```

b. C Program to Print the Alternate Elements in an Array

```c
#include <stdio.h>
#include<conio.h>
#include <stdio.h>

int main()
{
    int array[10];
    int i, j, temp;
    printf("enter the element of an array \n");
    for (i = 0; i < 10; i++)
    {
        scanf("%d", &array[i]);
        printf("Alternate elements of a given array \n");
        for (i = 0; i < 10; i += 2)
```
c. Program to Increment every Element of the Array by one & Print Incremented Array

```c
#include <stdio.h>
#include<conio.h>

void incrementArray(int arr[]);

int main()
{
    int i;
    int array[4] = {10, 20, 30, 40};

    incrementArray(array);
    for (i = 0; i < 4; i++)
        printf("%d\t", array[i]);  // Prints 2, 3, 4, 5
    getch();
    return 0;
}

void incrementArray(int arr[])
{
    int i;

    for (i = 0; i < 4; i++)
        arr[i]++;     // this alters values in array in main()
}
```

d. Program to Find the Number of Non Repeated Elements in an Array

```c
#include <stdio.h>
#include<conio.h>

int main()
{
    int array[50];
    int *ptr;
    int i, j, k, size, n;

    printf("Enter size of the array: ");
    scanf("%d", &n);
    printf("Enter %d elements of an array: ", n);
    for (i = 0; i < n; i++)
        scanf("%d", &array[i]);
    size = n;
```
ptr = array;
for (i = 0; i < size; i++)
{
    for (j = 0; j < size; j++)
    {
        if (i == j)
        {
            continue;
        }
        else if (*(ptr + i) == *(ptr + j))
        {
            k = j;
            size--;
            while (k < size)
            {
                *(ptr + k) = *(ptr + k + 1);
                k++;
            }
            j = 0;
        }
    }
}
printf("\n The array after removing duplicates is: ");
for (i = 0; i < size; i++)
{
    printf(" %d", array[i]);
}
getch();
return 0;

e. Program to Segregate 0s on Left Side & 1s on right side of the Array (Traverse Array only once)

#include <stdio.h>
#include<conio.h>

/*Function to segregate all 0s on left and all 1s on right*/
void segregate0and1(int array[], int size)
{
    int left = 0, right = size-1;

    while (left < right)
    {
        /* Increment left index while we see 0 at left */
        while (array[left] == 0 && left < right)
            left++;
        /* Decrement right index while we see 1 at right */
        while (array[right] == 1 && left < right)
            right--;
/ * If left is smaller than right then there is a 1 at left and a 0 at right. Exchange it */
    if (left < right)
    {
        array[left] = 0;
        array[right] = 1;
        left++;
        right--;
    }
}

int main()
{
    int arr[] = {0, 1, 0, 1, 1, 0};
    int array_size = 6, i = 0;

    segregate0and1(arr, array_size);
    printf("segregated array is ");
    for (i = 0; i < 6; i++)
        printf("%d ", arr[i]);
    getchar();
    return 0;
}

f. Bubble sort
#include <stdio.h>
#include<conio.h>

int main()
{
    int array[100], n, c, d, swap;

    printf("Enter number of elements\n");
    scanf("%d", &n);

    printf("Enter %d integers\n", n);
    for (c = 0; c < n; c++)
        scanf("%d", &array[c]);

    for (c = 0; c < ( n - 1 ); c++)
    {
        for (d = c+1 ; d < n ; d++)
        {
            if (array[c] > array[d]) /* For decreasing order use < */
            {
                swap       = array[c];
                array[c]   = array[d];
                array[d] = swap;
            }
        }
    }
}
Assignment2, No. 1

- **OBJECTIVE**: Implement stack using array.
- **RESOURCE**: GCC compiler
- **PROCEDURE**: Go to debug -> run or press CTRL + F9 to run the program.
- **PROGRAM LOGIC / ALGORITHM:**

Step 1: Start

Step 2: Declare global variable top, s[100], n

Step 3: Input size of stack from user and assign it to n.

Step 4: Assign top=-1

Step 5: n--

Step 6: Input choice from user and store in ch

Step 7: If ch=1 go to step 8, if ch=2, go to step 12, if ch = 3 go to step 15, if ch=4 go to step 18, else go to step 6.

Step 8: go to the push() function. Enter a value of item Step
9: If top+1<=n then top ++ and store item in s[top] Step
10: Else Display OVERFLOW

Step 11: Go to Step 6

Step 12: if top !=-1, then top-- and display element at s[op] before decrement.

Step 13: Else Display UNDERFLOW

Step 14: Go to step 6

Step 15: run a loop from 0 to top
Step 16: Display element at s[loop variable]
Step 17: Go to step 6
Step 18: Stop

**Assignment 2, No 2**

**Convert any infix expression to its postfix form using stack.**

**Algorithm:**

Step 1: Start

Step 2: Initialize global variables top, exp[100], s[100], e[100]

Step 3: Assign the priority operators ladder as a[]

Step 4: assign top=-1 and char b[100].

Step 5: Assign p=0 and r=0, integers

Step 6: Assign the expression in infix to exp and add an opening and closing bracket at the beginning and end

Step 7: Run a loop tile exp[p]=='0'

Step 8: initialize char c as exp[p] and then increment p

Step 9: Initialize int i=0 as loop variable and f=0 as flag variable

Step 10: Run a loop to check if c is in the operator ladder

Step 11: if present , f=1

Step 12: if i==3 or i==5 then priority hs to be checked till next operator therefore i++

Step 13: if f==0, then input c into the final expression e[r++]

Step 14: if f==1, then assign second flag f1 and check in present stack from priority ladder if any operator of higher priority is present, if so, go to step 15 else go to step 17

Step 15: Go to the pop function, where the value at s[top] is assigned to e[r++] and top-- is done Step 16: Repeat step 14 till f1=0

Step 17: if c is not a closing bracket then push c into the stack s[top++]
Step 18: If c is a closing bracket then pop out the elements till the opening bracket is found. top -
Step 19: go to display() function and print e[loop variable] till r

Step 20: STOP

Assignment 2, No 3

Convert any infix expression into its prefix form using stack.

Algorithm:

Step 1. Push “)” onto STACK, and add “(“ to end of the A
Step 2. Scan A from right to left and repeat step 3 to 6 for each element of A until the STACK is empty
Step 3. If an operand is encountered add it to B
Step 4. If a right parenthesis is encountered push it onto STACK
Step 5. If an operator is encountered then:
   a. Repeatedly pop from STACK and add to B each operator (on the top of STACK) which has
      same
      or higher precedence than the operator. b.
      Add operator to STACK
Step 6. If left parenthesis is encountered then
   a. Repeatedly pop from the STACK and add to B (each operator on top of stack until a left parenthesis is encountered)
   b. Remove the left parenthesis
Step 7. Exit

Assignment 2, No. 4

Evaluate any prefix expression using stack.

Algorithm:
Step 1: Start

Step 2: Initialize global variables int stack[100], char post[100], top=-1

Step 3: Accept the prefix expression and store in post[] and then reverse the string
Step 4: Run a loop to check for numbers and if present the insert it in stack
Step 5: if it is an operator then go to calc() function
Step 6: Take out the last two elements in the stack and assign ans= a operator b. Where a is the second last element and b is the last element
Step 7: Repeat step 4 till the entire length is traversed
Step 8: display the result
Step 9: Stop

Assignment 2, No. 5

Evaluate any postfix expression using stack.

Algorithm:

Step 1: Start
Step 2: Initialize global variables int stack[100], char post[100], top=-1
Step 3: Accept the postfix expression and store in post[]

Step 4: Run a loop to check for numbers and if present the insert it in stack
Step 5: if it is an operator then go to calc() function
Step 6: Take out the last two elements in the stack and assign ans= a operator b.
Step 7: Repeat step 4 till the entire length is traversed
Step 8: display the result
Step 9: Stop
Assignment 2, No. 6

Implement Tower of Hanoi problem.

Algorithm:

Step 1: Start
Step 2: Initialize variable so='s', te='t', de='d
Step 3: Initialize n with the number of discs from user
Step 4: send the variables to tower()
Step 5: if n==1. Display message
Step 6: send parameters to recursion using tower( n-1, source, dest, temp) and tower(n-1, temp, dest, source)
Step 7: Display appropriate message
Step 8: Stop

Assignment 3

Assignment 3, No 1

Implement queue using array.

Algorithm:

Step 1: Start
Step 2: Initialize global variables \( r=-1, f=-1, n, q[100] \)

Step 3: Assign the size of the queue to \( n \) and make it maximum size by \( n-- \)
Step 4: Initialize int ch with choice from user to insert, delete or display

Step 5: if user enter 1. Then check if \( r==n \). If not then enter element and insert in \( q[] \)

Step 6: if user enter 2. Then check if \( f==-1 \). If not then \( f++ \) and display \( q[f] \) before increment.
Step 7: if user enter 3 then display with a loop variable from \( f \) to \( r \)

Step 8: if user enter 4 then break and move out of loop
Step 9: Stop

Assignment 3, No 2

Implement circular queue.

Algorithm:

Step 1: Start

Step 2: Initialize global variables \( r=-1, f=-1, s[100] \) and \( ms \)

Step 3: Assign the size of the queue to \( ms \) by inputting from user

Step 4: Run an infinite while loop and inside it initialize \( x \) with choice from user
Step 5: if user enter 1. Then go to step 9
Step 6: if user enter 2. Then go to step 11.

Step 7: if user enter 3 then display with a loop variable from \( f \) to \( r \)
Step 8: if user enter any other input then go to step 18

Step 9: if \( f===(r+1)\%ms \) then display queue is full. else if \( f==-1 \) and \( r==-1 \) then \( f=0, \) and \( r=0 \). Step 10: Insert element at \( s[(r+1)\%ms ++] \). Go to step 4.

Step 11: if \( f==-1 \). Then display queue is empty. Step 12: if \( f==r \). Then \( f=-1, r=-1 \)

Step 13: else \( (f+1)\%ms=f \). Go to Step 4
Step 14: if f<r. Display from f to r of s[] using loop variable
Step 15: if f>r. The with loop variable display f to ms-1 of s[] followed by 0 to r of s[]
Step 17: Stop

Assignment 3, No 3

Implement de-queue.

Algorithm:

Step 1: Start

Step 2: Initialize global variables r=-1,f=-1,s[100] and ms

Step 3: Assign the size of the queue to ms by inputting from user

Step 4: Run a while loop till x=5 and inside it initialize x with choice from user Step 5: if user enter 1. Then go to step 9. If 2 then go to step 11

Step 6: if user enter 2. Then go to step 12. if 4 then go to Step 14 Step 7: if user enter 3 then display with a loop variable from f to r Step 8: if user enter 5 , go to Step 15

Step 9: if r=ms-1 then display queue is full. else if f=-1 and r=-1 then f=0, and r=0 Step 10: Insert element at s[r ++]. Go to step 4.

Step 11: if f==0. Then display queue is full.

Step 12: if r==‐1 . Display empty. if f==r, then f=r=‐1 Step 13: r--. Go to Step 4

Step 14: if f=r=-1. Display empty. else f++. Go to step 4 Step 15: display with loop variable from s[f] to s[r]. Step 17: Stop
Assignment 4

Assignment 4, No 1

Write a program to implement a linked list, which can be used to insert from the beginning, to insert from the end, to insert from anywhere, to delete from beginning, to delete from end, to delete from anywhere and to count the number of nodes in the linked list.

Algorithm:

Step 1: Start

Step 2: Declare the structure node with variables int data and struct node *next
Step 3: create the first node in the main() function and insert data to it

Step 4: Display the choices and input user's desired choice from keyboard

Step 5: if user enter 1. Then go to insB(). where the node is assigned in the beginning
Step 6: if user enter 2. Then to insE(). where the node is assigned in the end

Step 7: if user enter 3 then to insA(). where the node is assigned in the position entered by the user
Step 8: if user enter 4. Then go to delB(). where the node is deleted from the beginning

Step 9: if user enter 5. Then to delE(). where the node is deleted from the end

Step 10: if user enter 6 then to delA(). where the node is deleted from the position entered by the user
Step 11: if user enter 7 then break and move out of loop

Step 12: Display the linked list

Step 13: Count the number of nodes and display the number
Step 14: Stop
Assignment 4, No 2

Write a program to find out the maximum and minimum value in a linked list.

Algorithm:

Step 1: Start

Step 2: Declare the structure node with variables int data and struct node *next Step 3: create the first node in the main() function and insert data to it

Step 4: Enter nodes from the end by going to insE() till the user enters other choice than 1. temp=head Step 5: Run a while loop till the end of linked list

Step 6: initialise first value of head as the max and min

Step 7: if max is less than temp->data . Then replace max with it Step 8: if min is greater than temp->data. Then replace min with it Step 9: temp=temp->next

Step 10: Outside the loop display the values of max and min Step 11: Stop

Assignment 4, No 3

Write a program to reverse a linked list

Algorithm:

Step 1: Start

Step 2: Declare the structure node with variables int data and struct node *next Step 3: create the first node in the main() function and insert data to it

Step 4: Give option to user and go to insE() to insert new nodes at the end to create the linked list Step 5: Run a while loop to go to the end of the linked list and save the last node as l
Step 6: Initialise and increment the last node to t2

Step 7: Take the node before l and attach it to the end of t2 Step 8: increment t2 to the end.

Step 9: Make the third last node t1 as before l Step 10: : Repeat steps 6 to 9 till the head node is l Step 11: head=l

Step 12: l=t2

Step 13: Run a loop to display from head to l Step 14: Stop

Assignment 4, No 4

Write a program to split a linked list in half and make the second half and first half exchange places.

Algorithm:

Step 1: Start

Step 2: Declare the structure node with variables int data and struct node *next
Step 3: create the first node in the main() function and insert data to it
Step 4: Take option from user and enter the nodes to the end using the function insE()
Step 5: : initialize counter c=0.

Step 6: Initialize Node *temp, *m Step 7: temp=m=head
Step 8: Run a while loop till temp->next='\0' Step 9: if(c%2==0) then m=m->next
Step 10: temp=temp->next
Step 11: Outside the loop, make temp->next=head Step 12: head=m->next->next
Step 13: m->next='\0' Step 14: Stop
Assignment 4, No 5

Write a program to sort a linked list

Algorithm:

Step 1: Start

Step 2: Declare the structure node with variables int data and struct node *next

Step 3: create the nodes by inserting elements from the array arr[] into the forming linked list

Step 4: Display the linked list

Step 5: go to the function bubblesort() Step

Step 6: make swapped =0

Step 7: Run a do while loop as long as swapped is not 0

Step 8: run a while loop and if a greater value is encountered then swap that with the greater value we are sorting with using the swap() function

Step 9: if swapping takes place then make swapped=1 Step

Step 10: again make swapped =0 and repeat steps 7 to 9 Step 11: go back to main() function

Step 12: Display the linked list Step

Step 13: Stop
Assignment 5

Assignment 5, No 1

Write a program to implement BST.

Algorithm:

Step 1: Start

Step 2: Declare the structure sbst with variables int data and struct stack *right and *left Step 3: Name struct sbst as bst

Step 4: Display the choices of the user, if user enters 1 then go to insert() function and enter, repeat step 4.

Step 5: if user enter 2. Then go to delete() function and remove element entered, go to step 4. Step 6: if user enter 3. Then display the elements in inorder form, go to step 4

Step 7: if user enter 4 then go to search() function to locate element to be searched, go to step 4 Step 8: if user enters 5 then go to step 9

Step 9: Stop.

Pre order
Until all nodes are traversed –
Step 1 – Recursively traverse left subtree.
Step 2 – Visit root node.
Step 3 – Recursively traverse right subtree.

In order
Until all nodes are traversed –
Step 1 – Visit root node.
Step 2 – Recursively traverse left subtree.
Step 3 – Recursively traverse right subtree.

Post Order
Assignment 5, No 2

Write a program to implement max heap sorting.

Algorithm:

Step 1: Start

Step 2: Declare an array a[] and input values from user.

Step 3: Call the heapsort() function and pass variables suitable.

Step 4: From the heapsort() function call the buildheap() function, which creates the max heap by calling heapify() function at each root, thus, traversing through each root and making it contain the largest value in its sub tree.

Step 5: To make a Descending order priority queue, the root element is pushed into an array and replaced with the minimum element or last element of tree. The heap size is reduced.

Step 6: Repeat step 5 till all elements are pushed into the array.
Step 7: Display the array.

Step 8: Stop
INSERT
If root is NULL
    then create root node
return

If root exists then
    compare the data with node.data

while until insertion position is located

    If data is greater than node.data
        goto right subtree
    else
        goto left subtree
endwhile

insert data

end If
Search
If root.data is equal to search.data
    return root
else
while data not found

If data is greater than node.data
    goto right subtree
else
    goto left subtree

If data found
    return node

endwhile

return data not found

end if

Assignment 7
Assignment 7, No 1

To implement merge sort.

Algorithm:

Step 1: Start

Step 2: Declare an array a[], enter elements from user into the array.

Step 3: Send the array with its first and last element's position to the mergesort() function.
Step 4: Divide the array into single elements, then compare between to array elements and join them to form array.

Step 5: Repeat step 4 till a complete ordered array is given.
Step 6: Display the result.

Step 7: Stop

Assignment 7, No 2

Write a program to implement quick sorting.

Algorithm:

Step 1: Start

Step 2: Declare an array a[], enter elements from user into the array.

Step 3: Send the array with its first and last element's position to the quicksort() function.

Step 4: Set the first element as pivot and exchange the minimum element than pivot from the last and maximum from pivot and exchange them.

Step 5: When they cross each other then exchange maximum from last with pivot and divide the array into two part.

Step 6: Repeat steps 4 and 5 till they become a sorted list of single elements. Display array. Step 7: Stop