DATA STRUCTURES USING C

LAB MANUAL

DATA STRUCTURES LAB MANUAL

Subject Code: **22BCA2C05L**

Class: I Year II Semester (BCA)

Prepared By

Dr. Sanjeev Kumar Mandal

School of CS and IT,

Jain University

DATA STRUCTURES LABORATORY OBJECTIVE

The objective of this laboratory exercise is to teach students various data structures and their implementation. This explains the various operations possible on these data structures. This lab exercise complements the data structures course taught in the class. Students will gain practical knowledge by writing and executing programs in C using various data structures such as arrays, linked lists, stacks, queues and 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 a given problem.

3. **Understand** the applications of data structures.

4. **Choose** the appropriate data structure and algorithm method for a specific application.

5. **Understand** and apply fundamental algorithmic problems including Tree traversals, Graph traversals.

6. **Compare** different implementations of data structures and to recognize the advantages and disadvantages of them.

7. **Write** complex applications using structured programming methods.

Program List

Session No.	Programs	Page No.
1	1. To find the length of a string 2. To concatenate two strings 3. To copy a string to another string 4. To reverse a string	
2	5. To generate "n" terms of the Fibonacci series 6. To find the GCD of two numbers	
3	7. To insert an element at a given position in an array 8. To delete an element from a given position in an array 9. To perform Bubble sort	
4	10. To perform Insertion sort 11. To perform Selection sort	
5	12. To perform Linear search 13. To perform Binary search	
6	14. To perform stack operations	
7	15. To perform Queue operations	
8	16. To perform circular Queue operations	
9	17. To convert Infix expression to Postfix expression	
10	18. Insertion of elements to a linked list	
11	19. Deletion of elements from a linked list	
12	20. To create a binary tree and perform tree traversal	

1. A) To find the length of a string

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

int main()
{
    char Str[1000];
    int i;

    printf("Enter the String: ");
    scanf("%s", Str);

    for (i = 0; Str[i] != '\0'; ++i);

    printf("Length of Str is %d", i);

    return 0;
}
```

Output

_____

_____

_____

_____

_____

_____

_____

__

b) To concatenate two strings

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

int main()
{
    char a[100], b[100];

    printf("Enter the first string\n");
    gets(a);

    printf("Enter the second string\n");
    gets(b);

    strcat(a,b);

    printf("String obtained on concatenation is %s\n",a);

    return 0;
}
```

Output

_____

_____

_____

_____

c) To copy a string to another string

```c
#include <stdio.h>
int main() {
    char s1[100], s2[100], i;
    printf("Enter string s1: ");
    fgets(s1, sizeof(s1), stdin);
    for (i = 0; s1[i] != '\0'; ++i) {
        s2[i] = s1[i];
    }
    s2[i] = '\0';
    printf("String s2: %s", s2);
    return 0;
}
```

Output

_____
_____
_____
_____
_____
_____
_____
___

d) To reverse a string

```c
#include<stdio.h>
#include<string.h>
int main(void)
{
    char mystrg[60];
    int leng, g;
    // Printing the program name and what the program will do
    printf("Program in C for reversing a given string \n ");
    printf("Please insert the string you want to reverse: ");
// fetch the input string from the user
    scanf( "%s", mystrg );
    // This will find the length of your string with the help of strlen() function of string.h header file
    leng = strlen(mystrg);
    // iterate through each and every character of the string for printing it backwards or reverse direction
    for(g = leng - 1; g >= 0; g--) {
        printf("%c", mystrg[g]);
    }
    return 0;
}
```

Output

_____

_____

2. A) To generate "n" terms of the Fibonacci series

```c
#include <stdio.h>

int fibo(int);

void main()
{
        int     num, result,i;
        printf("Enter the number of terms to be generated \n");
        scanf("%d", &num);

        if (num < 0)
                printf("Fibonacci of negative number is not possible.\n");
        else
        {
                for ( i=1;i<=num;i++)
                {
                        result = fibo(i);
                        printf("The %d term in fibonacci series is %d\n", i,
result);
                }
        }
}
```

// Function to find the nth term in the Fibonacci series

```c
int fibo(int term)
{
        if (term == 1)
```

```
            return 0;
        else if (term == 2)
            return 1;
        else
            return(fibo(term - 1) + fibo(term - 2));
    }
```

Output

_____
_____
_____
_____
_____
_____
_____
___

b) To find the GCD of two numbers

```c
#include <stdio.h>
int hcf(int n1, int n2);
int main() {
    int n1, n2;
    printf("Enter two positive integers: ");
    scanf("%d %d", &n1, &n2);
    printf("G.C.D of %d and %d is %d.", n1, n2, hcf(n1, n2));
    return 0;
}
```

Output

_____

_____

_____

_____

_____

_____

_____

—

3. A) To insert an element at a given position in an array

```c
#include <stdio.h>
int main()
{
   int array[50], position, c, n, value;
      printf("Enter number of elements in the array\n");
   scanf("%d", &n);
   printf("Enter %d elements\n", n);
   for (c = 0; c < n; c++)
      scanf("%d", &array[c]);
      printf("Please enter the location where you want to insert an new element\n");
   scanf("%d", &position);
   printf("Please enter the value\n");
   scanf("%d", &value);
   for (c = n - 1; c >= position - 1; c--)
      array[c+1] = array[c];
   array[position-1] = value;
   printf("Resultant array is\n");
   for (c = 0; c <= n; c++)
```

```
    printf("%d\n", array[c]);
  return 0;
}
```

Output

_____
_____
_____
_____
_____
_____
_____
__

b) To delete an element from a given position in an array

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

int main ()
{
  // declaration of the int type variable
  int arr[50];
  int pos, i, num; // declare int type variable
  printf (" \n Enter the number of elements in an array: \n ");
  scanf (" %d", &num);


  printf (" \n Enter %d elements in array: \n ", num);
```

```c
// use for loop to insert elements one by one in array
for (i = 0; i < num; i++ )
{   printf (" arr[%d] = ", i);
    scanf (" %d", &arr[i]);
}


// enter the position of the element to be deleted
printf( " Define the position of the array element where you want to delete: \n ");
scanf (" %d", &pos);


// check whether the deletion is possible or not
if (pos >= num+1)
{
    printf (" \n Deletion is not possible in the array.");
}
else
{
    // use for loop to delete the element and update the index
    for (i = pos - 1; i < num -1; i++)
    {
        arr[i] = arr[i+1]; // assign arr[i+1] to arr[i]
    }

    printf (" \n The resultant array is: \n");
```

```c
    // display the final array
    for (i = 0; i< num - 1; i++)
    {
        printf (" arr[%d] = ", i);
        printf (" %d \n", arr[i]);
    }
    }
    return 0;
}
```

4.

a) Program to Sort 'N' numbers in ascending order using Bubble sort

```c
#include <stdio.h>
#define MAXSIZE 10

void main()
{
  int array[MAXSIZE];
  int i, j, num, temp;

  printf("Enter the value of num \n");
  scanf("%d", &num);
  printf("Enter the elements one by one \n");
  for (i = 0; i < num; i++)
  {
    scanf("%d", &array[i]);
  }
  printf("Input array is \n");
  for (i = 0; i < num; i++)
  {
    printf("%d\n", array[i]);
  }
  /*   Bubble sorting begins */
  for (i = 0; i < num; i++)
  {
```
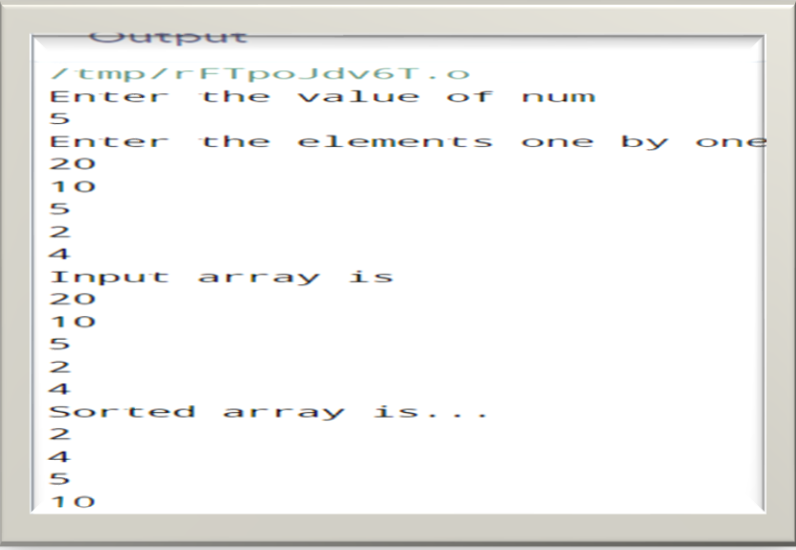
```c
for (j = 0; j < (num - i - 1); j++)
{
    if (array[j] > array[j + 1])
    {
        temp = array[j];
        array[j] = array[j + 1];
        array[j + 1] = temp;
    }
}
}
printf("Sorted array is...\n");
for (i = 0; i < num; i++)
{
    printf("%d\n", array[i]);
}
}
```

```
Output
/tmp/rFTpoJdv6T.o
Enter  the  value  of  num
5
Enter  the  elements  one  by  one
20
10
5
2
4
Input  array  is
20
10
5
2
4
Sorted  array  is...
2
4
5
10
```
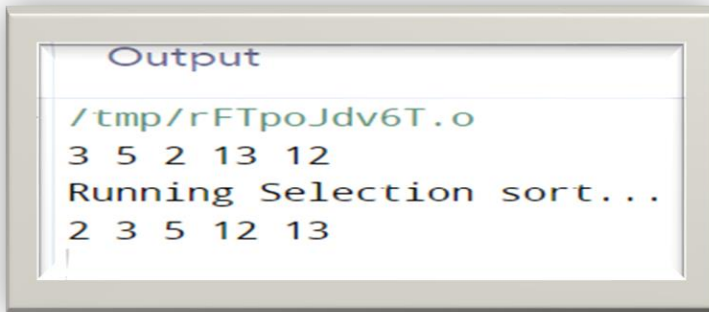
b) Program to Sort 'N' numbers in ascending order using Selection sort

```c
#include<stdio.h>

void printArray(int* A, int n){
    for (int i = 0; i < n; i++)
    {
        printf("%d ", A[i]);
    }
    printf("\n");
}

void selectionSort(int *A, int n){
    int indexOfMin, temp;
    printf("Running Selection sort...\n");
    for (int i = 0; i < n-1; i++)
    {
        indexOfMin = i;
        for (int j = i+1; j < n; j++)
        {
            if(A[j] < A[indexOfMin]){
                indexOfMin = j;
            }
        }
```

```
    // Swap A[i] and A[indexOfMin]

    temp = A[i];

    A[i] = A[indexOfMin];

    A[indexOfMin] = temp;

  }

}

int main(){

  int A[] = {3, 5, 2, 13, 12};

  int n = 5;

  printArray(A, n);

  selectionSort(A, n);

  printArray(A, n);

  return 0;

}
```

```
 Output

/tmp/rFTpoJdv6T.o
3 5 2 13 12
Running Selection sort...
2 3 5 12 13
```

Lab Session 5 (Searching)

a. **Program for the implementation of Linear search.**

```c
#include <stdio.h>

int search(int arr[], int N, int x)
{
    int i;
    for (i = 0; i < N; i++)
        if (arr[i] == x)
            return i;
    return -1;
}

// Driver's code
int main(void)
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;
    int N = sizeof(arr) / sizeof(arr[0]);

    // Function call
    int result = search(arr, N, x);
    (result == -1)
        ? printf("Element is not present in array")
        : printf("Element is present at index %d", result);
    return 0;
}
```

b. **Program for the implementation of Binary search.**

// C program to implement recursive Binary Search

#include <stdio.h>

// A recursive binary search function. It returns

// location of x in given array arr[l..r] is present,

// otherwise -1

int binarySearch(int arr[], int l, int r, int x)

{

    if (r >= l) {

        int mid = l + (r - l) / 2;

        // If the element is present at the middle

        // itself

        if (arr[mid] == x)

            return mid;

        // If element is smaller than mid, then

        // it can only be present in left subarray

        if (arr[mid] > x)

```c
            return binarySearch(arr, l, mid - 1, x);

        // Else the element can only be present
        // in right subarray
        return binarySearch(arr, mid + 1, r, x);
    }

    // We reach here when element is not
    // present in array
    return -1;
}

int main(void)
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 10;
    int result = binarySearch(arr, 0, n - 1, x);
    (result == -1)
            ? printf("Element is not present in array")
            : printf("Element is present at index %d", result);
    return 0;
}
```

6. Menu driven program for array implementation of stack

```c
#include<stdio.h>
int stack[100],choice,n,top,x,i;
void push(void);
void pop(void);
void display(void);
int main()
{
    top=-1;
    printf("\n Enter the size of STACK[MAX=100]:");
    scanf("%d",&n);
    printf("\n\t STACK OPERATIONS USING ARRAY");
    printf("\n\t-----------------------------");
    printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
    do
    {
        printf("\n Enter the Choice:");
        scanf("%d",&choice);
        switch(choice)
```

```c
{
    case 1:
    {
        push();
        break;
    }
    case 2:
    {
        pop();
        break;
    }
    case 3:
    {
        display();
        break;
    }
    case 4:
    {
        printf("\n\t EXIT POINT ");
        break;
    }
    default:
    {
        printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
    }
```

```c
        }
    }
    while(choice!=4);
    return 0;
}
void push()
{
    if(top>=n-1)
    {
        printf("\n\tSTACK is over flow");

    }
    else
    {
        printf(" Enter a value to be pushed:");
        scanf("%d",&x);
        top++;
        stack[top]=x;
    }
}
void pop()
{
    if(top<=-1)
    {
```

```c
        printf("\n\t Stack is under flow");
    }
    else
    {
        printf("\n\t The popped elements is %d",stack[top]);
        top--;
    }
}
void display()
{
    if(top>=0)
    {
        printf("\n The elements in STACK \n");
        for(i=top; i>=0; i--)
            printf("\n%d",stack[i]);
        printf("\n Press Next Choice");
    }
    else
    {
        printf("\n The STACK is empty");
    }

}
```

```
Output

/tmp/s91tZ6ib84.o
Enter the size of STACK[MAX=100]:5
STACK OPERATIONS USING ARRAY

    ---------------------------------
      1.PUSH
      2.POP
      3.DISPLAY
4.EXIT
 Enter the Choice:1
 Enter a value to be pushed:51
 Enter the Choice:1
 Enter a value to be pushed:52
 Enter the Choice:1
 Enter a value to be pushed:53
 Enter the Choice:1
 Enter a value to be pushed:57
 Enter the Choice:1
 Enter a value to be pushed:55
 Enter the Choice:1
 STACK is over flow
```

7. Menu driven program for array implementation of queues.

```c
#include <stdio.h>
#define MAX 50
void insert();
void delete();
void display();
```

```c
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
    int choice;
    while (1)
    {
        printf("1.Insert element to queue \n");
        printf("2.Delete element from queue \n");
        printf("3.Display all elements of queue \n");
        printf("4.Quit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
            insert();
            break;
            case 2:
            delete();
            break;
            case 3:
            display();
            break;
```

```c
            case 4:
            exit(1);
            default:
            printf("Wrong choice \n");
        } /* End of switch */
    } /* End of while */
} /* End of main() */


void insert()
{
    int add_item;
    if (rear == MAX - 1)
    printf("Queue Overflow \n");
    else
    {
        if (front == - 1)
        /*If queue is initially empty */
        front = 0;
        printf("Inset the element in queue : ");
        scanf("%d", &add_item);
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
} /* End of insert() */
```

```c
void delete()
{
    if (front == - 1 || front > rear)
    {
        printf("Queue Underflow \n");
        return ;
    }
    else
    {
        printf("Element deleted from queue is : %d\n", queue_array[front]);
        front = front + 1;
    }
} /* End of delete() */

void display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
```

```
    }
} /* End of display() */
```

```
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 10
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 15
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 20
```

```
Enter your choice : 1
Inset the element in queue : 30
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 2
Element deleted from queue is : 10
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 3
Queue is :
15 20 30
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 4
```

8. To perform circular Queue operations

```c
#include <stdio.h>
#define SIZE 5
int items[SIZE];
int front = -1, rear = -1;


// Check if the queue is full
int isFull() {
  if ((front == rear + 1) || (front == 0 && rear == SIZE - 1)) return 1;
  return 0;
}


// Check if the queue is empty
int isEmpty() {
  if (front == -1) return 1;
  return 0;
}


// Adding an element
void enQueue(int element) {
  if (isFull())
    printf("\n Queue is full!! \n");
  else {
    if (front == -1) front = 0;
```

```c
    rear = (rear + 1) % SIZE;

    items[rear] = element;

    printf("\n Inserted -> %d", element);

  }

}


// Removing an element

int deQueue() {

  int element;

  if (isEmpty()) {

    printf("\n Queue is empty !! \n");

    return (-1);

  } else {

    element = items[front];

    if (front == rear) {

      front = -1;

      rear = -1;

    }

    // Q has only one element, so we reset the

    // queue after dequeing it. ?

    else {

      front = (front + 1) % SIZE;

    }

    printf("\n Deleted element -> %d \n", element);

    return (element);
```

```c
  }
}

// Display the queue
void display() {
  int i;
  if (isEmpty())
    printf(" \n Empty Queue\n");
  else {
    printf("\n Front -> %d ", front);
    printf("\n Items -> ");
    for (i = front; i != rear; i = (i + 1) % SIZE) {
      printf("%d ", items[i]);
    }
    printf("%d ", items[i]);
    printf("\n Rear -> %d \n", rear);
  }
}

int main() {
  // Fails because front = -1
  deQueue();

  enQueue(1);
  enQueue(2);
```

```
    enQueue(3);
    enQueue(4);
    enQueue(5);


    // Fails to enqueue because front == 0 && rear == SIZE - 1
    enQueue(6);


    display();
    deQueue();
    display();
    enQueue(7);
    display();
    // Fails to enqueue because front == rear + 1
    enQueue(8);
    return 0;
}
```

```
  Output

/tmp/dWnjVM7E2N.o
Queue is empty !!

 Inserted -> 1
 Inserted -> 2
 Inserted -> 3
 Inserted -> 4
 Inserted -> 5
 Queue is full!!

 Front -> 0
 Items -> 1 2 3 4 5
 Rear -> 4

 Deleted element -> 1

 Front -> 1
 Items -> 2 3 4 5
Rear -> 4

 Inserted -> 7
 Front -> 1
 Items -> 2 3 4 5 7
 Rear -> 0

 Queue is full!!
```

9. To convert Infix expression to Postfix expression

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

char stack[100];
int top = -1;

void push(char x)
{
    stack[++top] = x;
}

char pop()
{
    if(top == -1)
        return -1;
    else
        return stack[top--];
}

int priority(char x)
{
    if(x == '(')
        return 0;
    if(x == '+' || x == '-')
```

```c
        return 1;
    if(x == '*' || x == '/')
        return 2;
    return 0;
}


int main()
{
    char exp[100];
    char *e, x;
    printf("Enter the expression : ");
    scanf("%s",exp);
    printf("\n");
    e = exp;

    while(*e != '\0')
    {
        if(isalnum(*e))
            printf("%c ",*e);
        else if(*e == '(')
            push(*e);
        else if(*e == ')')
        {
            while((x = pop()) != '(')
                printf("%c ", x);
```
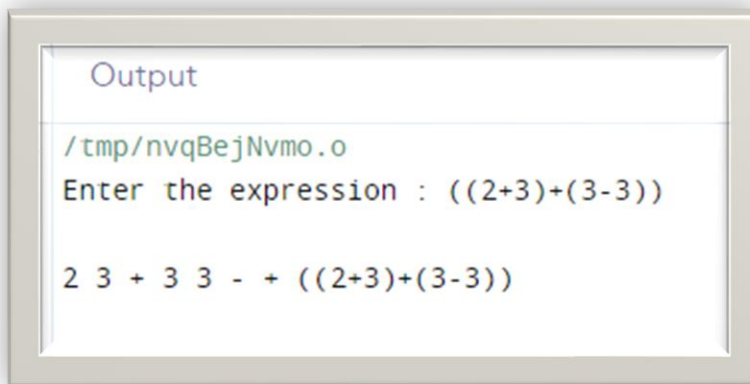
```c
    }
    else
    {
        while(priority(stack[top]) >= priority(*e))
            printf("%c ",pop());
        push(*e);
    }
    e++;
}


while(top != -1)
{
    printf("%c ",pop());
}return 0;
}
```

```
  Output

/tmp/nvqBejNvmo.o
Enter the expression : ((2+3)+(3-3))

2 3 + 3 3 - + ((2+3)+(3-3))
```

10. Insertion of elements to a linked list

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

int main()
{
  //node structure
  struct node
  {
     int data;
     struct node *next;
  };

  //declaring nodes
  struct node *head,*middle,*last;

  //allocating memory for each node
  head   = malloc(sizeof(struct node));
  middle = malloc(sizeof(struct node));
  last   = malloc(sizeof(struct node));

  //assigning values to each node
  head->data   = 10;
  middle->data = 20;
  last->data   = 30;

  //connecting each nodes. head->middle->last
  head->next   = middle;
  middle->next = last;
  last->next   = NULL;

  //temp is a reference for head pointer.
  struct node *temp = head;

  //till the node becomes null, printing each nodes data
  while(temp != NULL)
  {
```

```c
        printf("%d->",temp->data);
        temp = temp->next;
    }
    printf("NULL");

    return 0;
}
```

Output

/tmp/nvqBejNvmo.o
10->20->30->NULL

11. Deletion of elements from a linked list

```c
#include<stdio.h>
#include<stdlib.h>
void create(int);
void delete_specified();
struct node
{
    int data;
    struct node *next;
};
struct node *head;
void main ()
{
    int choice,item;
    do
    {
        printf("\n1.Append List\n2.Delete node\n3.Exit\n4.Enter your choice?");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            printf("\nEnter the item\n");
            scanf("%d",&item);
            create(item);
            break;
            case 2:
            delete_specified();
            break;
            case 3:
            exit(0);
            break;
            default:
            printf("\nPlease enter valid choice\n");
        }
```

```c
    }while(choice != 3);
}
void create(int item)
    {
        struct node *ptr = (struct node *)malloc(sizeof(struct node *));
        if(ptr == NULL)
        {
            printf("\nOVERFLOW\n");
        }
        else
        {
            ptr->data = item;
            ptr->next = head;
            head = ptr;
            printf("\nNode inserted\n");
        }

    }
void delete_specified()
    {
        struct node *ptr, *ptr1;
        int loc,i;
        scanf("%d",&loc);
        ptr=head;
        for(i=0;i<loc;i++)
        {
            ptr1 = ptr;
            ptr = ptr->next;

            if(ptr == NULL)
            {
                printf("\nThere are less than %d elements in the list..\n",loc);
                return;
            }
        }
        ptr1 ->next = ptr ->next;
        free(ptr);
        printf("\nDeleted %d node ",loc);
    }
```

12. To create a binary tree and perform tree traversal

```c
#include <stdio.h>
#include <stdlib.h>
struct node {
  int item;
  struct node* left;
  struct node* right;
};


// Inorder traversal
void inorderTraversal(struct node* root) {
  if (root == NULL) return;
  inorderTraversal(root->left);
  printf("%d ->", root->item);
  inorderTraversal(root->right);
}


// preorderTraversal traversal
void preorderTraversal(struct node* root) {
  if (root == NULL) return;
  printf("%d ->", root->item);
  preorderTraversal(root->left);
  preorderTraversal(root->right);
}
```

```c
// postorderTraversal traversal
void postorderTraversal(struct node* root) {
  if (root == NULL) return;
  postorderTraversal(root->left);
  postorderTraversal(root->right);
  printf("%d ->", root->item);
}


// Create a new Node
struct node* createNode(value) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->item = value;
  newNode->left = NULL;
  newNode->right = NULL;

  return newNode;
}


// Insert on the left of the node
struct node* insertLeft(struct node* root, int value) {
  root->left = createNode(value);
  return root->left;
}


// Insert on the right of the node
```

```c
struct node* insertRight(struct node* root, int value) {
  root->right = createNode(value);
  return root->right;
}

int main() {
  struct node* root = createNode(1);
  insertLeft(root, 12);
  insertRight(root, 9);

  insertLeft(root->left, 5);
  insertRight(root->left, 6);

  printf("Inorder traversal \n");
  inorderTraversal(root);

  printf("\nPreorder traversal \n");
  preorderTraversal(root);

  printf("\nPostorder traversal \n");
  postorderTraversal(root);
}
```

```
Output

/tmp/nvqBejNvmo.o
Inorder traversal
5 ->12 ->6 ->1 ->9 ->
Preorder traversal
1 ->12 ->5 ->6 ->9 ->
Postorder traversal
5 ->6 ->12 ->9 ->1 ->
```