Laboratory Manual

of

DATA STRUCTURE USING C

(Subject Code - 054003)

Information Technology

Semester IV

Government Polytechnic Nainital

Department: Information Technology

Uttarakhand Board of Technical Education, Roorkee

Practical: Course Outcome Matrix

1. Students will be able to design and analyse the time/space efficiency of the data structure.
2. Students will be able to understand basic data structures such as arrays, linked lists, stacks, queues and trees.
3. Students will be capable to identity the appropriate data structure for given problem.
4. Student will have practical knowledge on the applications of data structures.
5. Student will able to analyze the various sorting and searching algorithms.

S.No.	Practical	CO1	CO2	CO3	CO4	CO5
1.	Write a program in 'C' to perform PUSH and POP operations in stack by using array.	✓	✓	–	–	–
2.	Write a program in 'C' to display the reverse of a string using a stack.	–	–	✓	–	✓
3.	Write a program in 'C' to evaluate a post fix expression.	–	–	✓	–	✓
4.	Write a program in 'C' to create a queue containing ten elements and perform delete and insert operations using array.	✓	✓	–	–	–
5.	Write a program in 'C' to implement recursive function.	–	–	✓		✓
6.	Write a program in 'C' to create a singly linked list containing at least five elements. Make necessary assumptions.	✓	✓	–	–	–
7.	Write a program in 'C' to delete the first node that contains an integer data item of a single linked list.	✓	✓	–	–	–
8.	Write a program in 'C' to create a binary tree.	✓	✓	✓	–	–
9.	Write a program in 'C' for pre-order traversal of a binary tree.	–	–	✓	–	–
10.	Write a program in 'C' for binary searching.	–	–	✓	✓	✓
11.	Write a program in 'C' to sort 'N' Numbers using Insertion sort.	–	–	✓	✓	✓
12.	Write a program in 'C' to sort 'N' Numbers using bubble sort.	–	–	✓	✓	✓
13.	Write a program in 'C' to sort 'N' Numbers using selection sort.	–	–	✓	✓	✓
14.	Write a program in 'C' to sort 'N' Numbers using s Quick Sort	–	–	✓	✓	✓

General Laboratory Instructions

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:
    a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
    b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
    c. Proper Dress code and Identity card.
4. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
5. Student ought to refer the reference books, lab manual etc.
6. Student should not hesitate to ask any difficulties they face during the conduct of practical.
7. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviours with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Index

नैनीताल
श्रम विना न किमपि साध्यम्

Practical No. 1

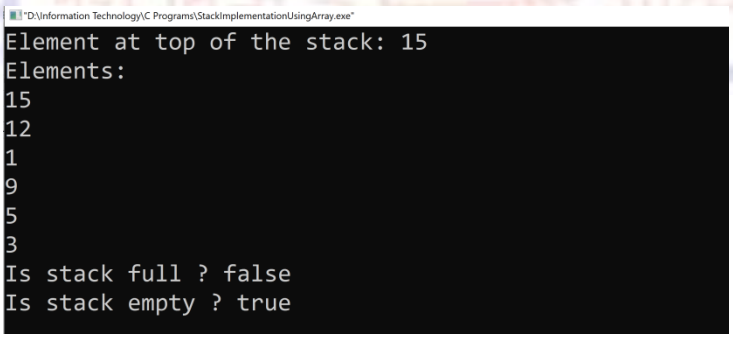
Objective:
Write a program in 'C' to perform PUSH and POP operations in stack by using array.

Program:

```
#include <stdio.h>

int MAXSIZE = 8;
int stack[8];
int top = -1;

int peek() {
  return stack[top];
}

int isEmpty() {
  if(top == -1)
    return 1;
  else
    return 0;
}

int isFull() {
  if(top == MAXSIZE)
    return 1;
  else
    return 0;
}

int pop() {
  int data;
  if(isEmpty()) {
    printf("Could not retrieve data, Stack is empty.\n");
  } else {
    data = stack[top];
    top = top - 1;
    return data;
  }
}
```

```
void push(int data) {
  if(isFull()) {
        printf("Could not insert data, Stack is full.\n");
  } else {
    top = top + 1;
    stack[top] = data;
  }
}

int main() {
  // push items on to the stack
  push(3);
  push(5);
  push(9);
  push(1);
  push(12);
  push(15);

  printf("Element at top of the stack: %d\n" ,peek());
  printf("Elements: \n");

  // print stack data
  while(!isEmpty()) {
    int data = pop();
    printf("%d\n",data);
  }

  printf("Is stack full ? %s\n" , isFull()?"true":"false");
  printf("Is stack empty ? %s\n" , isEmpty()?"true":"false");

  return 0;
}
```

Output:



Practical No. 2

Objective:
Write a program in 'C' to display the reverse of a string using a stack.

Program:

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

#define max 100
int top,stack[max];

void push(char x){
    // Push(Inserting Element in stack) operation
    if(top == max-1){
       printf("stack overflow");
    }  else {
       stack[++top]=x;
    }
}

void pop(){
   // Pop (Removing element from stack)
    printf("%c",stack[top--]);
}


void main()
{
  char str[]="Government Polytechnic Nainital";
  int len = strlen(str);
  int i;

  for(i=0;i<len;i++)
      push(str[i]);

  for(i=0;i<len;i++)
    pop();
}
```

Output:


"D:\Information Technology\C Programs\ReverseStringUsingStack.exe"
latiniaN cinhcetyloP tnemnrevoG

Practical No. 3

Objective:
Write a program in 'C' to evaluate a post fix expression.

Program:

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

int MAXSIZE = 50;
int stack[50];
int top = -1;

int peek() {
  return stack[top];
}

int isEmpty() {
  if(top == -1)
    return 1;
  else
    return 0;
}

int isFull() {
  if(top == MAXSIZE)
    return 1;
  else
    return 0;
}

int pop() {
  int data;
  if(isEmpty()) {
    printf("Could not retrieve data, Stack is empty.\n");
  } else {
    data = stack[top];
    top = top - 1;
    return data;
  }
```

```
}

void push(int data) {
  if(isFull()) {
        printf("Could not insert data, Stack is full.\n");
  } else {
    top = top + 1;
    stack[top] = data;
  }
}

int precedence(char symbol)
{/* Function for precedence */
 /* exponent operator, highest precedence*/
        if(symbol == '^')
        {
                return(3);
        }
        else if(symbol == '*' || symbol == '/')
        {
                return(2);
        }
        else if(symbol == '+' || symbol == '-')
        /* lowest precedence */
        {
                return(1);
        }
        else
        {
                return(0);
        }
}

long int eval_post(char postfix[])
{
      long int a,b,temp,result;
      unsigned int i;

      for(i=0;i<strlen(postfix);i++)
      {
           if(postfix[i]<='9' && postfix[i]>='0')
```

```
                push(postfix[i]-'0');
            else
            {
                a=pop();
                b=pop();
                switch(postfix[i])
                {
                case '+':
                    temp=b+a;
                    break;
                case '-':
                    temp=b-a;
                    break;
                case '*':
                    temp=b*a;
                    break;
                case '/':
                    temp=b/a;
                    break;
                case '%':
                    temp=b%a;
                    break;
                case '^':
                    temp=pow(b,a);
                }
                push(temp);
            }
        }
        result=pop();
        return result;
    }/*End of eval_post */


    void main()
    {
       char infix[50],postfix[50],ch,elem;
       int i=0,k=0;
       long int postfixEvaluation;
       printf("Enter Infix Expression : ");
       scanf("%s",infix);
       while( (ch=infix[i++]) != '\0')
```

```
    {
       if( ch == '(')
          push(ch);
       else
          if(isalnum(ch))
             postfix[k++]=ch;
          else
             if( ch == ')')
             {
                while( stack[top] != '(')
                   postfix[k++]=pop();
                elem=pop(); /* Remove ( */
             }
             else
             {     /* Operator */
                while( precedence(stack[top]) >= precedence(ch) )
                   postfix[k++]=pop();
                push(ch);
             }
    }
    while( top != -1)    /* Pop from stack till empty */
       postfix[k++]=pop();
    postfix[k]='\0';        /* Make postfix as valid string */
    printf("\nPostfix Expression =  %s\n",postfix);


    /* Evaluate postfix expression */
    postfixEvaluation=eval_post(postfix);
    printf("Value of expression : %ld\n",postfixEvaluation);
}
```

Output:

```
C:\Users\lohan\OneDrive\Desktop\Learning\DSC\CPrograms\PostfixEvaluation1.exe
Enter Infix Expression : (1+2)*7-6+3

Postfix Expression =  12+7*6-3+
Value of expression : 18
```

Practical No. 4

Objective:
Write a program in 'C' to create a queue containing ten elements and perform delete and insert operations using array.

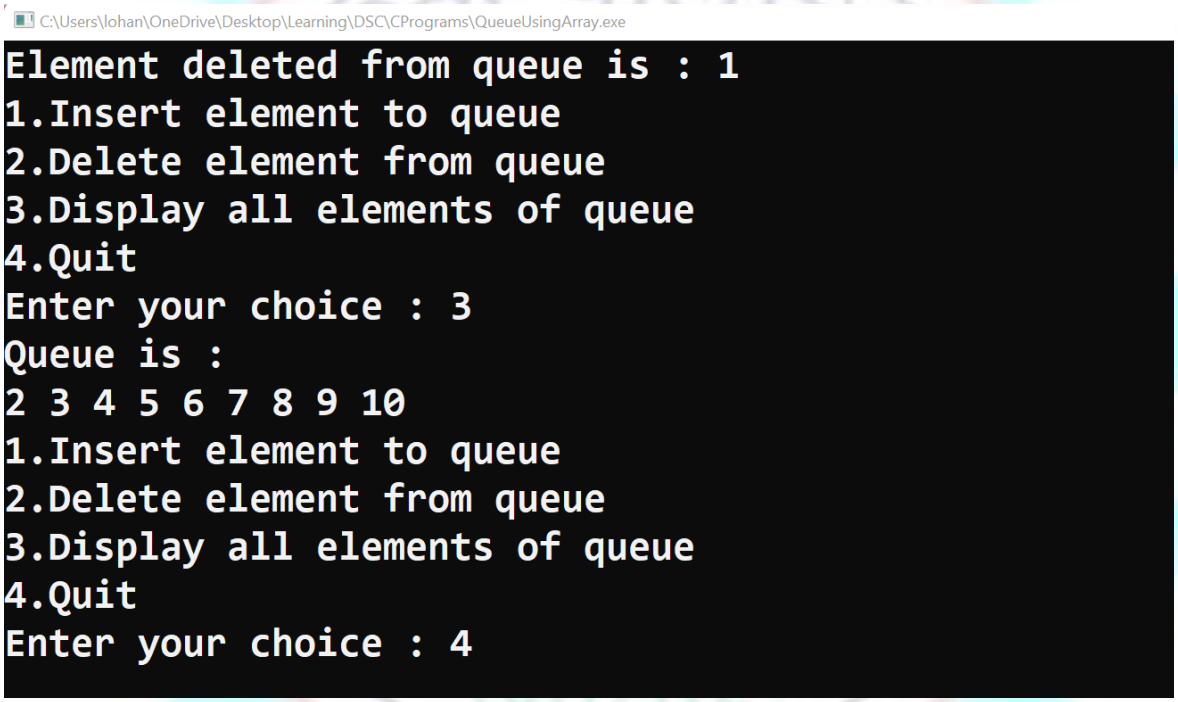
Program:

```
#include <stdio.h>
#define MAX 10
void insert();
void delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
void 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;
         case 4:
         exit(1);
         default:
         printf("Wrong choice \n");
```

```
        }
    }
}

void insert()
{
    int add_item;
    if (rear == MAX - 1){
        printf("Queue Overflow \n");
        printf("Value of Rear %d and Front %d is \n",rear,front);
    }
    else
    {
        if (front == - 1)
            front = 0;
        printf("Insert the element in queue : ");
        scanf("%d", &add_item);
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
}

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;
    }
}

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");
    }
}
```

Output:



Practical No. 5

Objective:
Write a program in 'C' to implement recursive function.

Program 5.1 :

```
// Fibonacci Series
#include <stdio.h>
int fibonacci(int i) {
  if(i == 0) {
    return 0;
  }
  if(i == 1) {
    return 1;
  }
  return fibonacci(i-1) + fibonacci(i-2);
}

int  main() {
  int i, n;
  printf("\nFIBONACCI SERIES");
  printf("\nPlease enter how many elements of the series you want to print : ");
  scanf("%d",&n);
  for (i = 0; i < n; i++) {
    printf("%d\t  ", fibonacci(i));
  }
  //printf("Total fibonacci calls are %d : ",count);
  return 0;
}
```

Output:

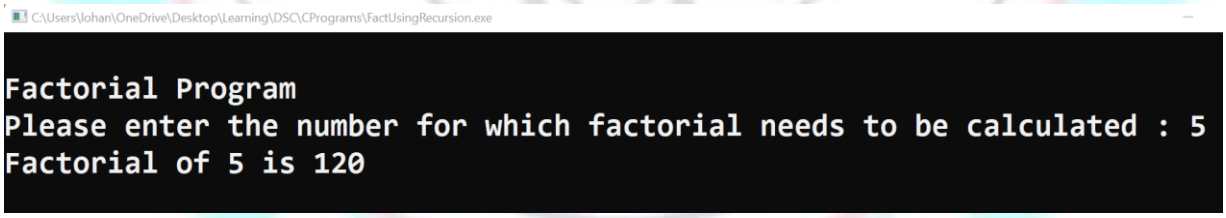
```
C:\Users\lohan\OneDrive\Desktop\Learning\DSC\CPrograms\FibonacciUsingRecursion.exe

FIBONACCI SERIES
Please enter how many elements of the series you want to print : 5
0         1       1       2       3
Process returned 0 (0x0)   execution time : 2.424 s
Press any key to continue.
```

Program 5.2 :

// Program to calculate factorial of a number

```
#include <stdio.h>
int factorial(int i) {
  if(i <= 1) {
    return 1;
  }
  return i * factorial(i - 1);
}

int  main() {
  int n;
  printf("\nFactorial Program");
  printf("\nPlease enter the number for which factorial needs to be calculated : ");
  scanf("%d",&n);
  printf("Factorial of %d is %d\n", n, factorial(n));
  return 0;
}
```

Output:



Program 5.3 :

// C program for Tower of Hanoi using Recursion

```
#include <stdio.h>
void towerOfHanoi(int, char, char, char);
int main()
{
   int numberOfDisks;
   printf("Enter the number of disks : ");
   scanf("%d", &numberOfDisks);
   printf("The sequence of moves involved in the Tower of Hanoi are :\n");
   towerOfHanoi(numberOfDisks, 'A', 'C', 'B');
   return 0;
}
void towerOfHanoi(int numberOfDisks, char inputTower, char outputTower,
                  char intermediateTower)
{
   if (numberOfDisks == 1)
   {
      printf("\n Move disk 1 from tower %c to tower %c", inputTower, outputTower);
      return;
   }
   towerOfHanoi(numberOfDisks - 1, inputTower, intermediateTower, outputTower);
   printf("\n Move disk %d from tower %c to tower %c", numberOfDisks, inputTower,
   outputTower);
   towerOfHanoi(numberOfDisks - 1, intermediateTower, outputTower, inputTower);
}
```

Output:

```
C:\Users\lohan\OneDrive\Desktop\Learning\DSC\CPrograms\TowerOfHanoi.exe
Enter the number of disks : 3
The sequence of moves involved in the Tower of Hanoi are :

 Move disk 1 from tower A to tower C
 Move disk 2 from tower A to tower B
 Move disk 1 from tower C to tower B
 Move disk 3 from tower A to tower C
 Move disk 1 from tower B to tower A
 Move disk 2 from tower B to tower C
 Move disk 1 from tower A to tower C
Process returned 0 (0x0)   execution time : 3.314 s
Press any key to continue.
```

Practical No. 6

Objective:
Write a program in 'C' to create a singly linked list containing at least five elements. Make necessary assumptions.

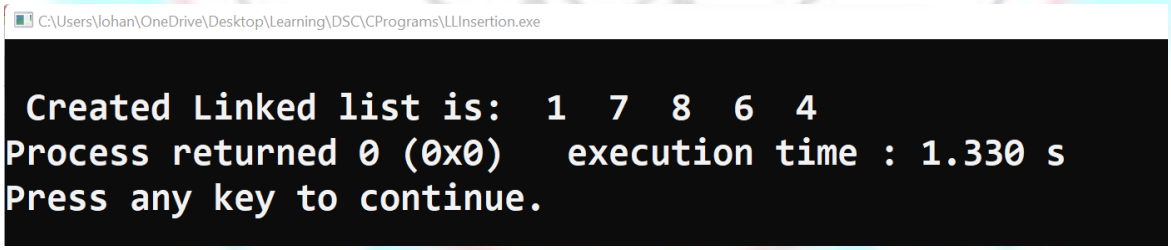
Program :

```
/* A complete working C program to demonstrate all insertion methods on Linked List */
#include <stdio.h>
#include <stdlib.h>
// A linked list node
struct Node
{
   int data;
   struct Node *next;
};
/* Given a reference (pointer to pointer) to the head of a list and
an int, inserts a new node on the front of the list. */
void push(struct Node** head_ref, int new_data)
{
   /* 1. allocate node */
   struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
   /* 2. put in the data */
   new_node->data = new_data;
   /* 3. Make next of new node as head */
   new_node->next = (*head_ref);
   /* 4. move the head to point to the new node */
   (*head_ref) = new_node;
}
/* Given a node prev_node, insert a new node after the given prev_node */
void insertAfter(struct Node* prev_node, int new_data)
{
/*1. check if the given prev_node is NULL */
   if (prev_node == NULL)
   {
      printf("the given previous node cannot be NULL");
      return;
   }
   /* 2. allocate new node */
```

```
        struct Node* new_node =(struct Node*) malloc(sizeof(struct Node));
        /* 3. put in the data */
        new_node->data = new_data;
        /* 4. Make next of new node as next of prev_node */
        new_node->next = prev_node->next;
        /* 5. move the next of prev_node as new_node */
        prev_node->next = new_node;
    }
    /* Given a reference (pointer to pointer) to the head
    of a list and an int, appends a new node at the end */
    void append(struct Node** head_ref, int new_data)
    {
        /* 1. allocate node */
        struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
        struct Node *last = *head_ref; /* used in step 5*/
        /* 2. put in the data */
        new_node->data = new_data;
        /* 3. This new node is going to be the last node, so make next of
        it as NULL*/
        new_node->next = NULL;/* 4. If the Linked List is empty, then make the new node
as head */
        if (*head_ref == NULL)
        {
            *head_ref = new_node;
            return;
        }
        /* 5. Else traverse till the last node */
        while (last->next != NULL)
            last = last->next;
        /* 6. Change the next of last node */
        last->next = new_node;
        return;
    }
    // This function prints contents of linked list starting from head
    void printList(struct Node *node)
    {
        while (node != NULL)
        {
            printf(" %d ", node->data);
            node = node->next;
        }
```

```
}
/* Driver program to test above functions*/
int main()
{
   /* Start with the empty list */
   struct Node* head = NULL;
   // Insert 6. So linked list becomes 6->NULL
   append(&head, 6);
   // Insert 7 at the beginning. So linked list becomes 7->6->NULL
   push(&head, 7);
   // Insert 1 at the beginning. So linked list becomes 1->7->6->NULL
   push(&head, 1);
   // Insert 4 at the end. So linked list becomes 1->7->6->4->NULL
   append(&head, 4);
   // Insert 8, after 7. So linked list becomes 1->7->8->6->4->NULL
   insertAfter(head->next, 8);
   printf("\n Created Linked list is: ");
   printList(head);
   return 0;
}
```

Output:



Practical No. 7

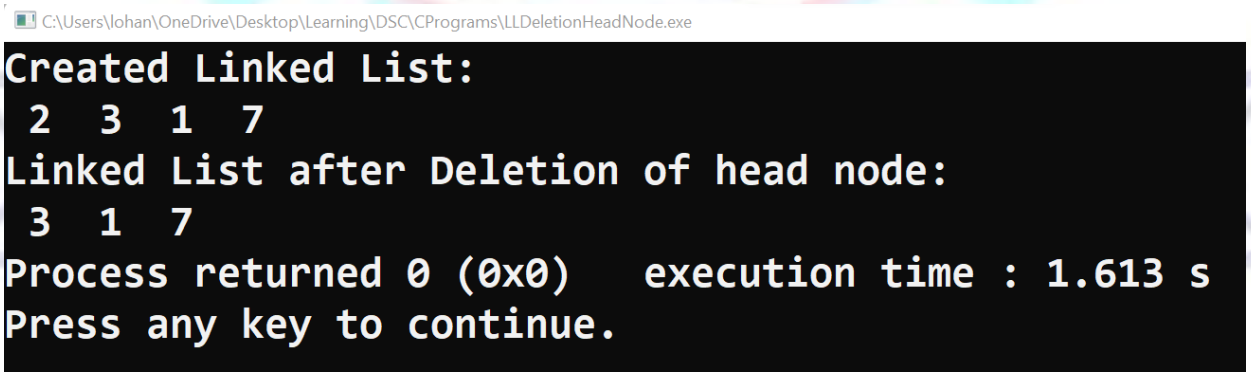
Objective:
Write a program in 'C' to delete the first node that contains an integer data item of a single linked list.

Program :

```
/* A complete working C program to demonstrate deletion of head node in singly linked list */
#include <stdio.h>
#include <stdlib.h>
// A linked list node
struct Node
{
   int data;
   struct Node *next;
};
/* Given a reference (pointer to pointer) to the head of a list and an int, inserts a new node on the front of the list. */
void push(struct Node** head_ref, int new_data)
{
   struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
   new_node->data = new_data;
   new_node->next = (*head_ref);
   (*head_ref) = new_node;
}
/* Given a reference (pointer to pointer) to the head of a list and a key, deletes the first occurrence of key in linked list */
void deleteNode(struct Node **head_ref)
{
   // Store head node
   struct Node* temp = *head_ref;
   if (temp != NULL)
   {
      *head_ref = temp->next; // Changed head
      free(temp); // free old head
      return;
   }

}
```

```
// This function prints contents of linked list starting from the given node
void printList(struct Node *node)
{
   while (node != NULL)
   {
      printf(" %d ", node->data);
      node = node->next;
   }
}
/* Driver program to test above functions*/
int main()
{
   /* Start with the empty list */
   struct Node* head = NULL;
   push(&head, 7);
   push(&head, 1);
   push(&head, 3);
   push(&head, 2);
   puts("Created Linked List: ");
   printList(head);
   deleteNode(&head);
   puts("\nLinked List after Deletion of head node: ");
   printList(head);
   return 0;
}
```

Output:



Program 7.2 :

// A complete working C program to demonstrate deletion (any node based on searched key) in singly linked list

```
#include <stdio.h>
#include <stdlib.h>
// A linked list node
struct Node
{
   int data;
   struct Node *next;
};
/* Given a reference (pointer to pointer) to the head of a list and an int, inserts a new node on the front of the list. */
void push(struct Node** head_ref, int new_data)
{
   struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
   new_node->data = new_data;
   new_node->next = (*head_ref);
   (*head_ref) = new_node;
}
/* Given a reference (pointer to pointer) to the head of a list and a key, deletes the first occurrence of key in linked list */
void deleteNode(struct Node **head_ref, int key)
{
   // Store head node
   struct Node* temp = *head_ref, *prev;// If head node itself holds the key to be deleted
   if (temp != NULL && temp->data == key)
   {
      *head_ref = temp->next; // Changed head
      free(temp); // free old head
      return;
   }
   // Search for the key to be deleted, keep track of the previous node as we need to change 'prev->next'
   while (temp != NULL && temp->data != key)
   {
      prev = temp;
      temp = temp->next;
   }
```

```
    // If key was not present in linked list
    if (temp == NULL)
        return;
    // Unlink the node from linked list
    prev->next = temp->next;
    free(temp); // Free memory
}
// This function prints contents of linked list starting from the given node
void printList(struct Node *node)
{
    while (node != NULL)
    {
        printf(" %d ", node->data);
        node = node->next;
    }
}
/* Driver program to test above functions*/
int main()
{
    /* Start with the empty list */
    struct Node* head = NULL;
    push(&head, 7);
    push(&head, 1);
    push(&head, 3);
    push(&head, 2);
    puts("Created Linked List: ");
    printList(head);
    deleteNode(&head, 1);
    puts("\nLinked List after Deletion of 1: ");
    printList(head);
    return 0;
}
```

Output:

```
C:\Users\lohan\OneDrive\Desktop\Learning\DSC\CPrograms\LLDeletion.exe
Created Linked List:
 2  3  1  7
Linked List after Deletion of 1:
 2  3  7
Process returned 0 (0x0)   execution time : 1.829 s
Press any key to continue.
```

Practical No. 8

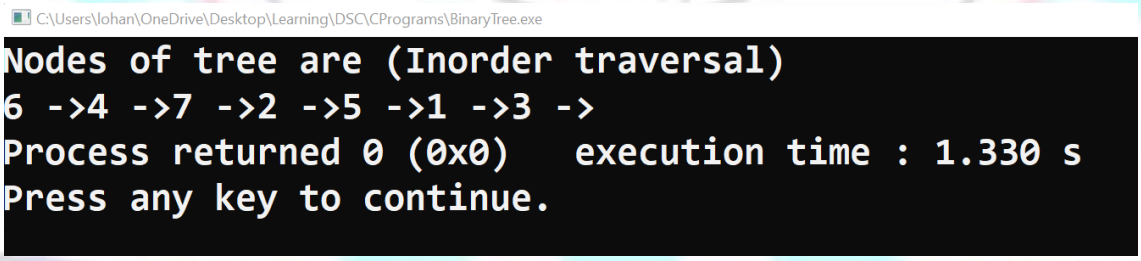
Objective:
Write a program in 'C' to create a binary tree.

Program :

```
// Creation of binary tree in C and print the node using inorder traversal

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

struct node {
  int item;
  struct node* left;
  struct node* right;
};

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

// Inorder traversal : LMR
```

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

int main() {
  struct node* root = createNode(1);
  insertLeft(root, 2);
  insertRight(root, 3);

  insertLeft(root->left, 4);
  insertRight(root->left, 5);

  insertLeft(root->left->left, 6);
  insertRight(root->left->left, 7);

  printf("Nodes of tree are (Inorder traversal) \n");
  inorderTraversal(root);
}
```

Output:



Practical No. 9

Objective:
Write a program in 'C' for pre-order traversal of a binary tree.

Program :

```
// Tree traversal in C

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

struct node {
  int item;
  struct node* left;
  struct node* right;
};

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

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

// postorderTraversal traversal : LRM
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
struct node* insertRight(struct node* root, int value) {
  root->right = createNode(value);
  return root->right;
}

int main() {
  struct node* root = createNode(1);
  insertLeft(root, 2);
  insertRight(root, 3);

  insertLeft(root->left, 4);
  insertRight(root->left, 5);

  insertLeft(root->left->left, 6);
  insertRight(root->left->left, 7);

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

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

  printf("\nPostorder traversal \n");
```

```
  postorderTraversal(root);
}
```

Output:

Practical No. 10

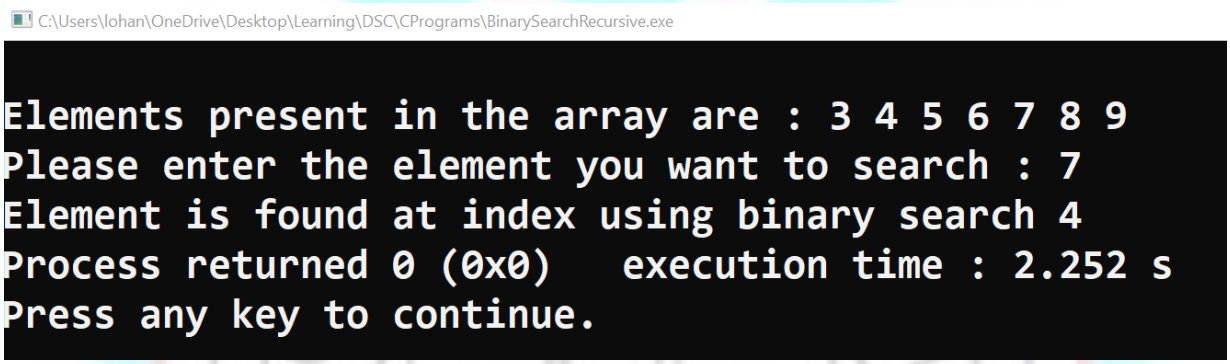
Objective:
Write a program in 'C' for binary searching.

Program :

```
// Binary Search in C with recursion

#include <stdio.h>
int binarySearch(int array[], int x, int low, int high) {
  if (high >= low) {
    int mid = low + (high - low) / 2;

    // If found at mid, then return it
    if (array[mid] == x)
      return mid;

    // Search the left half
    if (array[mid] > x)
      return binarySearch(array, x, low, mid - 1);

    // Search the right half
    return binarySearch(array, x, mid + 1, high);
  }

  return -1;
}

int main(void) {
  int array[] = {3, 4, 5, 6, 7, 8, 9},x;
  int n = sizeof(array) / sizeof(array[0]);
  printf("\nElements present in the array are : ");
  for(int i=0;i<=6;i++){
    printf("%d ",array[i]);
  }
  printf("\nPlease enter the element you want to search : ");
  scanf("%d",&x);
  int result = binarySearch(array, x, 0, n - 1);
  if (result == -1)
    printf("Not found");
```

```
  else
    printf("Element is found at index using binary search %d", result);
}
```

Output:



Practical No. 11

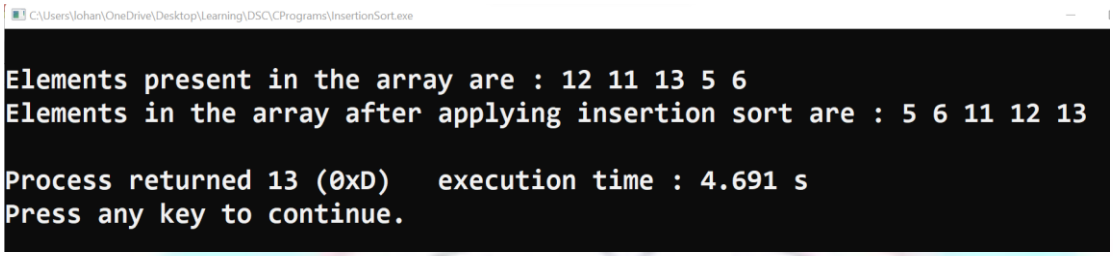
Objective:
Write a program in 'C' to sort 'N' Numbers using Insertion sort.

Program :

```
// C program for insertion sort
#include <math.h>
#include <stdio.h>

/* Main function */
void main()
{
   int arr[5] = {12, 11, 13, 5, 6};
   int n = 5;
   printf("\nElements present in the array are : ");
   for(int i=0;i<n;i++){
      printf("%d ",arr[i]);
   }

    int key, j;
    for (int i = 1; i < n; i++) {
      key = arr[i];
      j = i - 1;

      /* Move elements of arr[0..i-1], that are greater than key, to one position ahead
of their current position */
      while (j >= 0 && arr[j] > key) {
         arr[j + 1] = arr[j];
         j = j - 1;
      }
      arr[j + 1] = key;
    }

    printf("\nElements in the array after applying insertion sort are : ");
    for (int i = 0; i < n; i++)
      printf("%d ", arr[i]);
    printf("\n");

    getch();
```

```
}
```

Output:


C:\Users\lohan\OneDrive\Desktop\Learning\DSC\CPrograms\InsertionSort.exe
Elements present in the array are : 12 11 13 5 6
Elements in the array after applying insertion sort are : 5 6 11 12 13
Process returned 13 (0xD)   execution time : 4.691 s
Press any key to continue.

Practical No. 12

Objective:
Write a program in 'C' to sort 'N' Numbers using bubble sort.

Program :

```
// C program for implementation of Bubble sort
#include <stdio.h>
void main()
{
   int arr[7] = {64, 34, 25, 12, 22, 11, 90};
   int n = 7;
   printf("\nElements present in the array are : ");
   for(int i=0;i<n;i++){
      printf("%d ",arr[i]);
   }
   int temp;
   for (int i = 0; i < n-1; i++){
     // Last i elements are already in place
     for (int j = 0; j < n-i-1; j++){
        if (arr[j] > arr[j+1]){
          temp = arr[j];
          arr[j] = arr[j+1];
          arr[j+1] = temp;
        }
     }
   }
   printf("\nElements in the array after applying bubble sort are : ");
   for (int i=0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");

   getch();
}
```

Output:

```
C:\Users\lohan\OneDrive\Desktop\Learning\DSC\CPrograms\BubbleSort.exe

Elements present in the array are : 64 34 25 12 22 11 90
Elements in the array after applying bubble sort are : 11 12 22 25 34 64 9
0
```

Practical No. 13

Objective:
Write a program in 'C' to sort 'N' Numbers using selection sort.

Program :

```
// C program for implementation of selection sort
#include <stdio.h>
void main()
{
   int arr[7] = {64, 34, 25, 12, 22, 11, 90};
   int n = 7;
   printf("\nElements present in the array are : ");
   for(int i=0;i<n;i++){
      printf("%d ",arr[i]);
   }
   int temp, min;
   // One by one move boundary of unsorted subarray
   for (int i = 0; i < n-1; i++){
     // Find the minimum element in unsorted array
     min = i;
     for (int j = i+1 ; j < n; j++){
        if (arr[j] < arr[min])
            min = j;
     }
     // Swap the found minimum element with the first element
     temp = arr[min];
     arr[min] = arr[i];
     arr[i] = temp;
   }
   printf("\nElements in the array after applying selection sort are : ");
   for (int i=0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");

   getch();
}
```

Output :

```
C:\Users\lohan\OneDrive\Desktop\Learning\DSC\CPrograms\SelectionSort.exe

Elements present in the array are : 64 34 25 12 22 11 90
Elements in the array after applying selection sort are : 11 12 22 25 34 64 90
```

Practical No. 14

Objective:
Write a program in 'C' to sort 'N' Numbers using s Quick Sort.

Program :

```
// Quick sort in C
#include <stdio.h>

// function to swap elements
void swap(int *a, int *b) {
  int t = *a;
  *a = *b;
  *b = t;
}

// function to find the partition position
int partition(int array[], int low, int high) {

  // select the rightmost element as pivot
  int pivot = array[high];

  // pointer for greater element
  int i = (low - 1);

  // traverse each element of the array
  // compare them with the pivot
  for (int j = low; j < high; j++) {
    if (array[j] <= pivot) {

      // if element smaller than pivot is found
      // swap it with the greater element pointed by i
      i++;

      // swap element at i with element at j
      swap(&array[i], &array[j]);
    }
  }

  // swap the pivot element with the greater element at i
```

```
  swap(&array[i + 1], &array[high]);

  // return the partition point
  return (i + 1);
}

void quickSort(int array[], int low, int high) {
  if (low < high) {

    // find the pivot element such that
    // elements smaller than pivot are on left of pivot
    // elements greater than pivot are on right of pivot
    int pi = partition(array, low, high);

    // recursive call on the left of pivot
    quickSort(array, low, pi - 1);

    // recursive call on the right of pivot
    quickSort(array, pi + 1, high);
  }
}

// function to print array elements
void printArray(int array[], int size) {
  for (int i = 0; i < size; ++i) {
    printf("%d  ", array[i]);
  }
  printf("\n");
}

// main function
int main() {
  int data[] = {8, 7, 2, 1, 0, 9, 6};

  int n = sizeof(data) / sizeof(data[0]);

  printf("Unsorted Array\n");
  printArray(data, n);

  // perform quicksort on data
  quickSort(data, 0, n - 1);
```

```
  printf("Sorted array in ascending order: \n");
  printArray(data, n);
}
```

Output: