;

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Regulations 2021)

21CSC201J -DATA STRUCTURES AND ALGORITHMS

LAB MANUAL

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Course Learning Rationale (CLR):

The purpose of learning this course is to:

- CLR-1: Structures, pointers, searching, sorting techniques used to handle a set of data along with time and space complexity
- CLR-2: List structure and its categories
- CLR-3: Linear structures Stack and Queue
- CLR-4: Tree structure with its applications and hashing methods
- CLR-5: Structures graph and implement them

Course Learning Outcomes (CLO):

At the end of the course, learners will be able

- CLO-1:Identify linear and non-linear data structures. Create algorithms for searching and sorting.
- CLO-1: Develop programs using data types like structures, pointers and arrays supported by C programming languages

21CSC201J -DATA STRUCTURES AND ALGORITHMS

devising solution

- CLO-4: Describe and use tree structure while developing programs
- CLO-5: Implement the Graph structure and use it whenever deemed necessary for providing better solution

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Regulations 2021)

21CSC201J -DATA STRUCTURES AND ALGORITHMS

LIST OF EXPERIMENTS

18CSC201J -DATA STRUCTURES AND ALGORITHMSLABORATORY

Lab 1: Implementation of Structures
Lab 2: Implementation of Structures using Pointers
Lab 3: Implementation of Matrix Multiplication – Dynamic Memory allocation
Lab 4: Array Implementation of List
Lab 5: Implementation of Linked List
Lab 6: Implementation of Doubly linked List
Lab 7: Implementation of Stack using array and Linked List
Lab 8: Implementation of Queue using array and Linked list
Lab 9: Applications of Stack, Queue
Lab 10: Implementation of Tree using array
Lab 11: Implementation of BST using linked list
Lab 12: Implementation of B-Trees
Lab 13: Implementation of Graph using Array
Lab 14: Implementation of Shortest path Algorithm
Lab 15: Implementation of Minimal Spanning Tree

TOTAL: 30 HOURS

EX. NO. 1	**Implementation of Structures**
DATE:	

AIM:

To write a C Program for the implementation of structures.

ALGORITHM:

1. Define the structure using the struct keyword, specifying its fields (data members) within curly braces.ure type to hold data.
2. Assign values to the fields of the structure variables.
3. Access and manipulate the fields of the structure variables using the dot (.) operator.
4. Declare variables of the struct

PROGRAM:

```
 #include <stdio.h>
#include <string.h>
struct Person
 {
  char name[50];
  int stu_id;
  float cgpa;
} person1;

int main()
{
  strcpy(person1.name, "George");
  person1.stu_id = 2211;
  person1.cgpa = 9.68;
  printf("Name: %s\n", person1.name);
  printf("student id :%d\n",person1.stu_id );
  printf("student cgpa : %.2f", person1.cgpa);
  return 0;
}
```

OUTPUT:

```
Name: George
student id :2211
student cgpa : 9.68
```

RESULT:

Thus the C Program for the implementation of Structure has been executed successfully.

EX. NO. 2	**Implementation of Structures using Pointers**
DATE:	

AIM:

To write a C Program for Structures using Pointers.

ALGORITHM:

STEP 1: Enter the input for creating a structure of student database.

STEP 2: Create a structure pointer using normal structure variable and pointer variable

STEP 3:Dot (.) operator is used to access the data using normal structure variable and arrow (->) is used to access data using pointer variable.

STEP 4: Student details are printed using normal and pointer structure variable.

PROGRAM:

```
#include <stdio.h>
#include <string.h>
int main()
{
 struct student
 {
  int roll_no;
  char name[30];
  int phone_number;
 };
 struct student p1 = {1,"Brown",123443};
 struct student p2,* p3;
 p3=&p1;
 p2.roll_no = 2;
 strcpy(p2.name,"Sam");
 p2.phone_number = 1234567822;
 p3->roll_no = 3;
 strcpy(p3.name,"Addy");
 p3->phone_number = 1234567844;
 printf("First Student\n");
 printf("roll_no : %d\n", p1.roll_no);
 printf("name : %s\n", p1.name);
 printf("phone_number : %d\n", p1.phone_number);
 printf("Second Student\n");
 printf("roll_no : %d\n", p2.roll_no);
 printf("name : %s\n", p2.name);
 printf("phone_number : %d\n", p2.phone_number);
 printf("Third Student\n");
 printf("roll_no : %d\n", p3->roll_no);
 printf("name : %s\n", p3->name);
 printf("phone_number : %d\n", p3->phone_number);
```

```
  return 0;
}
```

INPUT & OUTPUT

```
First Student
roll_no : 1
name : Brown
phone_number : 123443
Second Student
roll_no : 2
name : Sam
phone_number : 1234567822
Third Student
roll_no : 3
name : Addy
phone_number : 1234567844
```

RESULT:

Thus the Program Structures using Pointers has been executed successfully.

EX. NO. 3	Implementation of matrix Multiplication – Dynamic Memory Allocation
DATE:	

AIM:

To write a C Program to implement matrix Multiplication using Dynamic Memory Allocation

ALGORITHM:

1. Read the dimensions of the two matrices (rows and columns).
2. Allocate memory dynamically for the matrices using the malloc function.
3. Read the elements of the matrices.
4. Perform matrix multiplication using nested loops.
5. Allocate memory for the result matrix.
6. Store the result of multiplication in the result matrix.
7. Display the result matrix.
8. Free the dynamically allocated memory.

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
struct Matrix {
    int rows;
    int columns;
    int **data;
};
struct Matrix* createMatrix(int rows, int columns) {
    struct Matrix* matrix = (struct Matrix*)malloc(sizeof(struct Matrix));
    matrix->rows = rows;
    matrix->columns = columns;
    matrix->data = (int*)malloc(rows * sizeof(int));
    for (int i = 0; i < rows; i++) {
        matrix->data[i] = (int*)malloc(columns * sizeof(int));
    }
    return matrix;
}
void freeMatrix(struct Matrix* matrix) {
    for (int i = 0; i < matrix->rows; i++) {
        free(matrix->data[i]);
    }
    free(matrix->data);
    free(matrix);
}
void readMatrix(struct Matrix* matrix) {
    for (int i = 0; i < matrix->rows; i++) {
        for (int j = 0; j < matrix->columns; j++) {
            scanf("%d", &matrix->data[i][j]);
        }
    }
}
struct Matrix* multiplyMatrices(struct Matrix* matrix1, struct Matrix*
matrix2) {
    if (matrix1->columns != matrix2->rows) {
        printf("Matrix multiplication not possible. Invalid dimensions!\n");
        return NULL;
    }

    struct Matrix* result = createMatrix(matrix1->rows, matrix2->columns);

    for (int i = 0; i < matrix1->rows; i++) {
        for (int j = 0; j < matrix2->columns; j++) {
            int sum = 0;
            for (int k = 0; k < matrix1->columns; k++) {
                sum += matrix1->data[i][k] * matrix2->data[k][j];
            }
            result->data[i][j] = sum;
        }
    }

    return result;
}
void displayMatrix(struct Matrix* matrix) {
    for (int i = 0; i < matrix->rows; i++) {
        for (int j = 0; j < matrix->columns; j++) {
            printf("%d\t", matrix->data[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int rows1, columns1, rows2, columns2;

    printf("Enter the number of rows and columns of the first matrix: ");
    scanf("%d %d", &rows1, &columns1);

    printf("Enter the number of rows and columns of the second matrix: ");
    scanf("%d %d", &rows2, &columns2);
    if (columns1 != rows2) {
        printf("Matrix multiplication is not possible. Invalid dimensions!\n");
        return 1;
    }
    struct Matrix* matrix1 = createMatrix(rows1, columns1);
    struct Matrix* matrix2 = createMatrix(rows2, columns2);
    printf("Enter elements for the first matrix:\n");
    readMatrix(matrix1);

    printf("Enter elements for the second matrix:\n");
    readMatrix(matrix2);
    struct Matrix* resultMatrix = multiplyMatrices(matrix1, matrix2);
    printf("Resultant matrix:\n");
    displayMatrix(resultMatrix);
    freeMatrix(matrix1);
    freeMatrix(matrix2);
    freeMatrix(resultMatrix);
    return 0;
}
```

OUTPUT:

```
Enter the number of rows and columns of the first matrix: 2 2
Enter the number of rows and columns of the second matrix: 2 2
Enter elements for the first matrix:
2 3 3 3
Enter elements for the second matrix:
2 3 4 5
Resultant matrix:
16      21
18      24
```

RESULT:

Thus the C Program to implement matrix Multiplication using Dynamic Memory Allocation has been executed successfully.

EX. NO. 4	Array Implementation of List
DATE:	

AIM:

To create a list using an array in c program.

ALGORITHM:

1. Define the maximum size of the list (array size) and initialize variables for tracking the number of elements in the list.
2. Declare an array to hold the list elements.
3. Implement functions for common list operations (e.g., insertion, deletion, searching, and displaying).
4. In each function, handle edge cases (e.g., checking for empty or full list, handling invalid indices).
5. Test the implemented functions in the main program.

PROGRAM

```
#include <stdio.h>

#define MAX_SIZE 100

struct List {
    int arr[MAX_SIZE];
    int size;
};

// Function to initialize the list
void initList(struct List* list) {
    list->size = 0;
}

// Function to insert an element at the end of the list
void insert(struct List* list, int element) {
    if (list->size < MAX_SIZE) {
        list->arr[list->size] = element;
        list->size++;
    } else {
        printf("List is full. Cannot insert.\n");
    }
}

// Function to delete an element at a given index from the list
void deleteAtIndex(struct List* list, int index) {
    if (index < 0 || index >= list->size) {
        printf("Invalid index.\n");
        return;
    }

    for (int i = index; i < list->size - 1; i++) {
        list->arr[i] = list->arr[i + 1];
    }

    list->size--;
}

// Function to display the list elements
void display(struct List* list) {
    printf("List: ");
    for (int i = 0; i < list->size; i++) {
        printf("%d ", list->arr[i]);
    }
    printf("\n");
}

int main() {
    struct List myList;
    initList(&myList);

    // Insert elements into the list
    insert(&myList, 5);
    insert(&myList, 10);
    insert(&myList, 15);

    // Display the list
    display(&myList);

    // Delete an element at index 1
    deleteAtIndex(&myList, 1);

    // Display the updated list
    display(&myList);

    return 0;
}
```

OUTPUT:

```
        List: 5 10 15
        List: 5 15
```

RESULT:

Thus the C Program for creating a list using an array has been executed successfully.

EX. NO. 5	
DATE:	IMPLEMENTATION OF LINKED LIST

AIM:
To write a c program to implement linked List and its operations.

ALGORITHM:

```
1.   Start
2.   Define single linked list node as self referential structure
3.   Create Head node with label = -1 and next = NULL using
4.   Display menu on list operation
5.   Accept user choice
6.   If choice = 1 then
7.   Locate node after which insertion is to be done
8.   Create a new node and get data part
9.   Insert the new node at appropriate position by manipulating address
10.  Else if choice = 2
11.  Get node's data to be deleted.
12.  Locate the node and delink the node
13.  Rearrange the links
14.  Else
15.  Traverse the list from Head node to node which points to null
16.  Stop
```

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *head;
void beginsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
void main ()
{
int choice =0;
while(choice != 7)
{
```

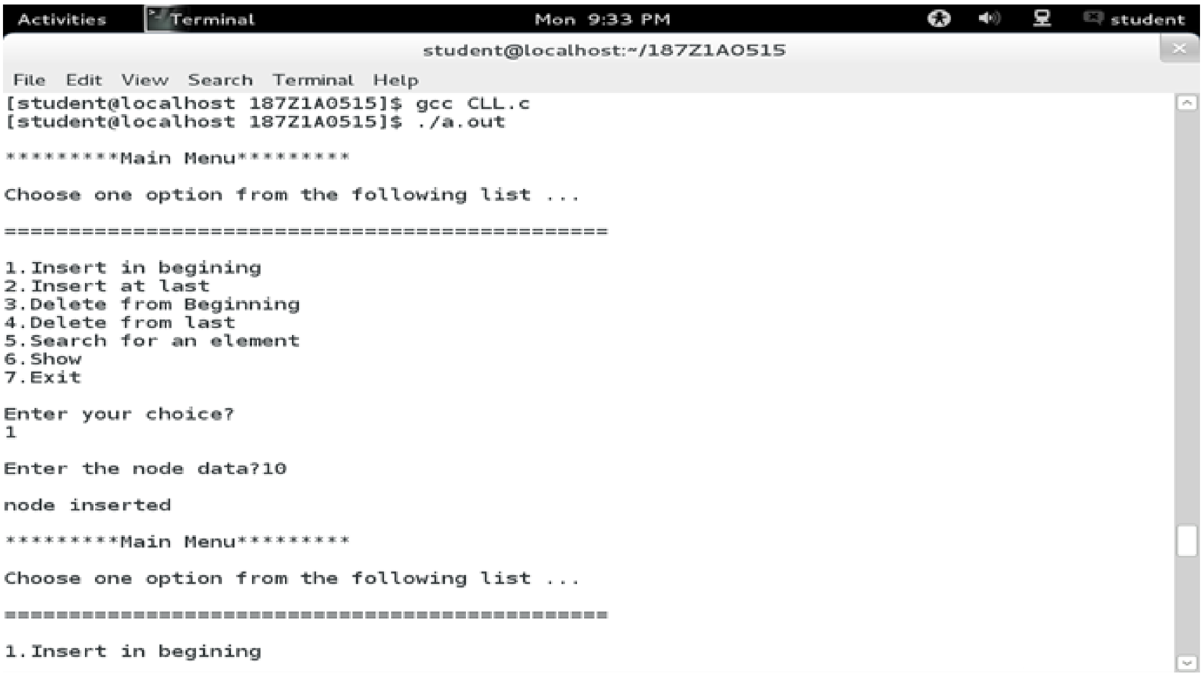
```
printf("\n*********Main Menu*********\n");
printf("\nChoose one option from the following list ...\n");
printf("\n===============================================\n");
printf("\n1.Insert in begining\n2.Insert at last\n3.Delete from Beginning\n4.Delete from last\n5.Search for an element\n6.Show\n7.Exit\n");
printf("\nEnter your choice?\n");
scanf("\n%d",&choice);
switch(choice)
{
case 1:
beginsert();
break;
case 2:
lastinsert();
break;
case 3:
begin_delete();
break;
case 4:
last_delete();
break;
case 5:
search();
break;
case 6:
display();
break;
case 7:
exit(0);
break;
default:
printf("Please enter valid choice..");
}
}
}
void beginsert()
{
struct node *ptr,*temp;
int item;
ptr = (struct node *)malloc(sizeof(struct node));
if(ptr == NULL)
{
printf("\nOVERFLOW");
}
else
{
```

```
printf("\nEnter the node data?");
scanf("%d",&item);
ptr -> data = item;
if(head == NULL)
{
head = ptr;
ptr -> next = head;
}
else
{
temp = head;
while(temp->next != head)
temp = temp->next;
ptr->next = head;
temp -> next = ptr;
head = ptr;
}
printf("\nnode inserted\n");
}
}
void lastinsert()
{
struct node *ptr,*temp;
int item;
ptr = (struct node *)malloc(sizeof(struct node));
if(ptr == NULL)
{
printf("\nOVERFLOW\n");
}
else
{
printf("\nEnter Data?");
scanf("%d",&item);
ptr->data = item;
if(head == NULL)
{
head = ptr;
ptr -> next = head;
}
else
{
temp = head;
while(temp -> next != head)
{
temp = temp -> next;
}
temp -> next = ptr;
ptr -> next = head;
}

printf("\nnode inserted\n");
}
}

void begin_delete()
{
struct node *ptr;
if(head == NULL)
{
printf("\nUNDERFLOW");
}
else if(head->next == head)
{
head = NULL;
free(head);
printf("\nnode deleted\n");
}
else
{   ptr = head;
while(ptr -> next != head)
ptr = ptr -> next;
ptr->next = head->next;
free(head);
head = ptr->next;
printf("\nnode deleted\n");
}
}
void last_delete()
{
struct node *ptr, *preptr;
if(head==NULL)
{
printf("\nUNDERFLOW");
}
else if (head ->next == head)
{
head = NULL;
free(head);
printf("\nnode deleted\n");
}
else
{
ptr = head;
while(ptr ->next != head)
```

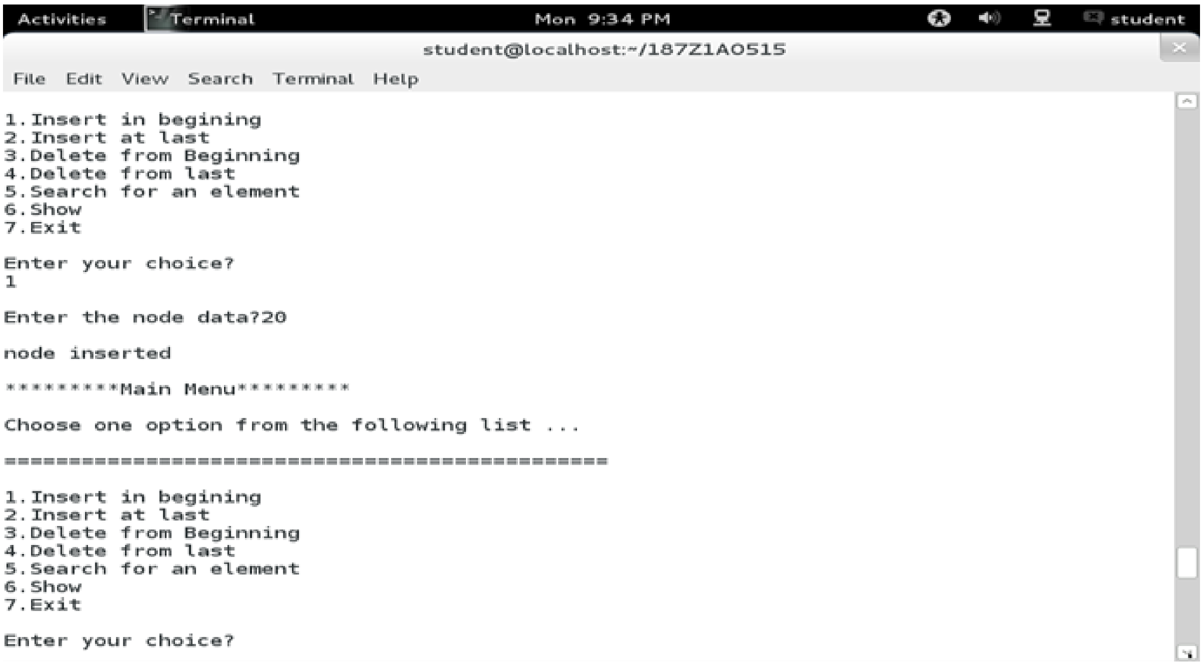
```
{
preptr=ptr;
ptr = ptr->next;
}
preptr->next = ptr -> next;
free(ptr);
printf("\nnode deleted\n");
}
}
void search()
{
struct node *ptr;
int item,i=0,flag=1;
ptr = head;
if(ptr == NULL)
{
printf("\nEmpty List\n");
}
else
{
printf("\nEnter item which you want to search?\n");
scanf("%d",&item);
if(head ->data == item)
{
printf("item found at location %d",i+1);
flag=0;
}
else
{
while (ptr->next != head)
{
if(ptr->data == item)
{
printf("item found at location %d ",i+1);
flag=0;
break;
}
else
{
flag=1;
}
i++;
ptr = ptr -> next;
}
}
if(flag != 0)
{
```

```
printf("Item not found\n");
}}}

void display()
{
struct node *ptr;
ptr=head;
if(head == NULL)
{
printf("\nnothing to print");
}
else
{
printf("\n printing values ... \n");
while(ptr -> next != head)
{
printf("%d\n", ptr -> data);
ptr = ptr -> next;
}
printf("%d\n", ptr -> data);
}
}
```

OUTPUT:


Activities
Terminal
Mon 9:33 PM
student
student@localhost:~/187Z1A0515
File Edit View Search Terminal Help

```
[student@localhost 187Z1A0515]$ gcc CLL.c
[student@localhost 187Z1A0515]$ ./a.out

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice?
1

Enter the node data?10

node inserted

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert in begining
```



Activities
Terminal
Mon 9:34 PM
student
student@localhost:~/187Z1A0515
File Edit View Search Terminal Help

```
1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice?
1

Enter the node data?20

node inserted

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice?
```

RESULT:

Thus the c program to implement linked List and it’s operations has been executed successfully.

EX. NO. 6	
DATE:	IMPLEMENTATION OF DOUBLY LINKED LIST

AIM:

To write a c program to implement doubly linked List

ALGORITHM:

i. Insert at Beginning

```
Start
Input the DATA to be inserted
Create a new node.
NewNode → Data = DATA NewNode →Lpoint =NULL
IF START IS NULL NewNode→ Rpoint = NULL
Else NewNode → Rpoint = START START→Lpoint = NewNode
START =NewNode
Stop
```

ii. Insertion at location:

```
Start
Input the DATA and POS
Initialize TEMP = START; i = 0
Repeat the step 4 if (i less than POS) and (TEMP is not equal to NULL)
TEMP = TEMP → RPoint; i = i +1
If (TEMP not equal to NULL) and (i equal to POS)
        (a) Create a New Node
        (b) NewNode → DATA = DATA
        (c) NewNode → RPoint = TEMP → RPoint
        (d) NewNode → LPoint = TEMP
        (e) (TEMP → RPoint) → LPoint = NewNode
        (f ) TEMP → RPoint = New Node
Else
        (a) Display "Position NOT found"
Stop
```

iii. Insert at End

```
Start
Input DATA to be inserted
Create a NewNode
NewNode → DATA = DATA
NewNode → RPoint = NULL
If (SATRT equal to NULL)
a. START = NewNode
b. NewNode → LPoint=NULL
Else
a. TEMP = START
b. While (TEMP → Next not equal to NULL)
i. TEMP = TEMP → Next
```

```
c. TEMP → RPoint = NewNode

d. NewNode → LPoint = TEMP
Stop
```

iv. Forward Traversal

```
Start
If (START is equal to NULL)
a) Display “The list is Empty”
b) Stop
Initialize TEMP = START
Repeat the step 5 and 6 until (TEMP == NULL )
Display “TEMP → DATA”
TEMP = TEMP → Next
Stop
```

v. Backward Traversal

```
Start
If (START is equal to NULL)
Display “The list is Empty”
Stop
Initialize TEMP = TAIL
Repeat the step 5 and 6 until (TEMP == NULL )
Display “TEMP → DATA”
TEMP = TEMP → Prev
Stop
```

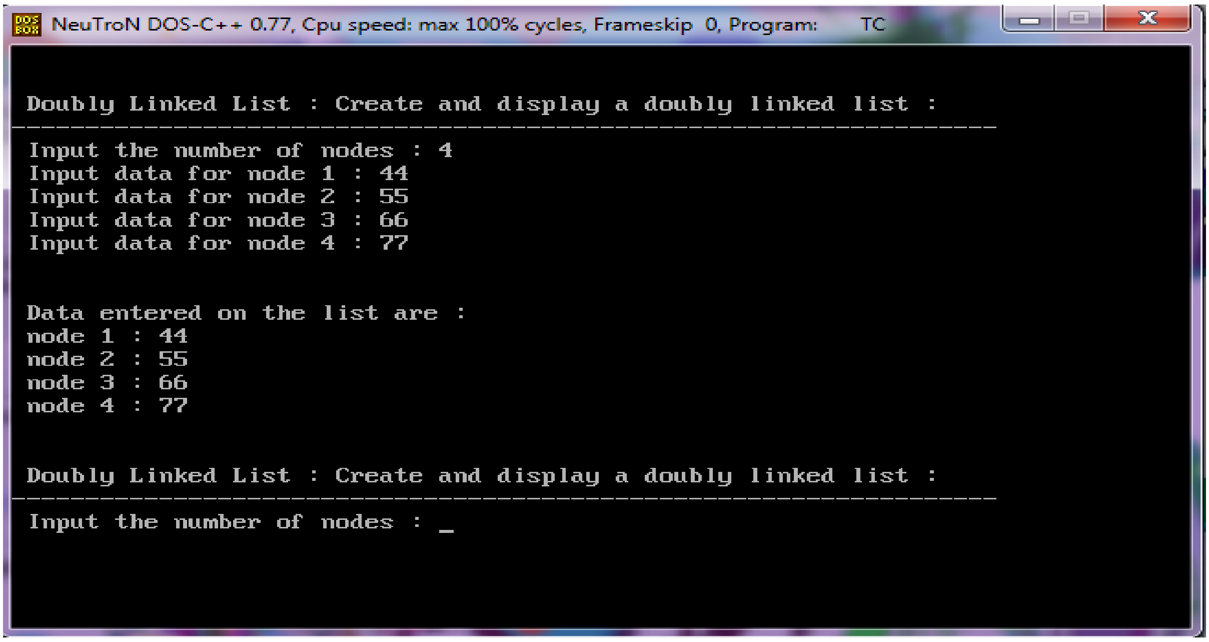
PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
struct node {
int num;
struct node * preptr;
struct node * nextptr;
}*stnode, *ennode;
void DlListcreation(int n);
void displayDlList();
int main()
{
int n;
stnode = NULL;
ennode = NULL;
printf("\n\n Doubly Linked List : Create and display a doubly linked list :\n");
printf("------------------------------------------------------------------\n");
printf(" Input the number of nodes : ");
scanf("%d", &n);
DlListcreation(n);
displayDlList();
```

```
return 0;
}
void DlListcreation(int n)
{
int i, num;
struct node *fnNode;
if(n >= 1)
{
stnode = (struct node *)malloc(sizeof(struct node));
if(stnode != NULL)
{
printf(" Input data for node 1 : "); // assigning data in the first node
scanf("%d", &num);
stnode->num = num;
stnode->preptr = NULL;
stnode->nextptr = NULL;
ennode = stnode;
// putting data for rest of the nodes
 for(i=2; i<=n; i++)
{
 fnNode = (struct node *)malloc(sizeof(struct node));
if(fnNode != NULL)
{
printf(" Input data for node %d : ", i);
scanf("%d", &num);
fnNode->num = num;
fnNode->preptr = ennode;    // new node is linking with the previous node
fnNode->nextptr = NULL;
ennode->nextptr = fnNode;   // previous node is linking with the new node
ennode = fnNode;            // assign new node as last node
}
else
{
printf(" Memory can not be allocated.");
break;
}
}
}
else
{
printf(" Memory can not be allocated.");
}
}
}
void displayDlList()
{
struct node * tmp;
```

```
int n = 1;
if(stnode == NULL)
{
printf(" No data found in the List yet.");
}
else
{
tmp = stnode;
printf("\n\n Data entered on the list are :\n");
while(tmp != NULL)
{
printf(" node %d : %d\n", n, tmp->num);
n++;
tmp = tmp->nextptr; // current pointer moves to the next node
}}}
```

OUTPUT:



NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
 Doubly Linked List : Create and display a doubly linked list :
-------------------------------------------------------------------
 Input the number of nodes : 4
 Input data for node 1 : 44
 Input data for node 2 : 55
 Input data for node 3 : 66
 Input data for node 4 : 77


 Data entered on the list are :
 node 1 : 44
 node 2 : 55
 node 3 : 66
 node 4 : 77


 Doubly Linked List : Create and display a doubly linked list :
-------------------------------------------------------------------
 Input the number of nodes : _
```

RESULT:

Thus the C Program for to insert and delete the elements in Doubly Linked List has been executed successfully.

EX. NO. 7 A	
DATE:	IMPLEMENTATION OF STACK USING ARRAY

AIM:

To write a c program to implement stack using array

ALGORITHM:

Step 1 - Include all the header files which are used in the program and define a constant 'SIZE' with specific value.
Step 2 - Declare all the functions used in stack implementation.
Step 3 - Create a one dimensional array with fixed size (int stack[SIZE])
Step 4 - Define a integer variable 'top' and initialize with '-1'. (int top = -1)
Step 5 - In main method, display menu with list of operations and make suitable function calls to perform operation selected by the user on the stack.

push(value) - Inserting value into the stack

Step 1 - Check whether stack is FULL. (top == SIZE-1)
Step 2 - If it is FULL, then display "Stack is FULL!!! Insertion is not possible!!!" and terminate the function.
Step 3 - If it is NOT FULL, then increment top value by one (top++) and set stack[top] to value (stack[top] = value).

pop() - Delete a value from the Stack

In a stack, pop() is a function used to delete an element from the stack.
Step 1 - Check whether stack is EMPTY. (top == -1)
Step 2 - If it is EMPTY, then display "Stack is EMPTY!!! Deletion is not possible!!!" and terminate the function.
Step 3 - If it is NOT EMPTY, then delete stack[top] and decrement top value by one (top--).

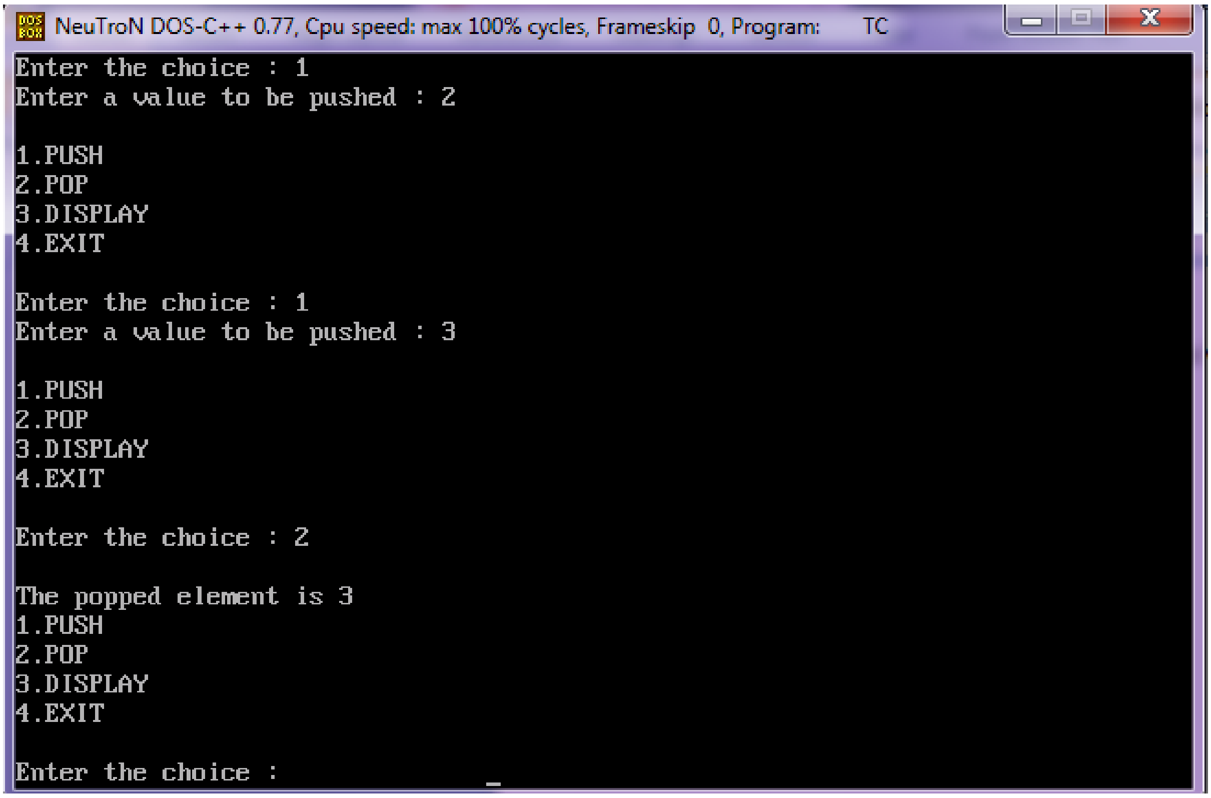
display() - Displays the elements of a Stack

Step 1 - Check whether stack is EMPTY. (top == -1)
Step 2 - If it is EMPTY, then display "Stack is EMPTY!!!" and terminate the function.
Step 3 - If it is NOT EMPTY, then define a variable 'i' and initialize with top. Display stack[i] value and decrement i value by one (i--).
Step 3 - Repeat above step until i value becomes '0'.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int stack[10],choice,n,top,x,i; // Declaration of variables
void push();
void pop();
void display();
int main()
{
 top = -1;     // Initially there is no element in stack
 printf("\n Enter the size of STACK : ");
 scanf("%d",&n);
 printf("\nSTACK IMPLEMENTATION USING ARRAYS\n");
 do
 {
 printf("\n1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n");
 printf("\nEnter the choice : ");
 scanf("%d",&choice);
 switch(choice)
 {
 case 1:
 {
 push();
 break;
 }
 case 2:
 {
 pop();
 break;
 }
 case 3:
 {
 display();
 break;
 }
 case 4:
 {
 break;
 }
 default:
 {
 printf ("\nInvalid Choice\n");
 }}}
 while(choice!=4);
 return 0;
}
void push()
{
 if(top >= n - 1)
 {
 printf("\nSTACK OVERFLOW\n");
 }
 else
 {
 printf("Enter a value to be pushed : ");
 scanf("%d",&x);
 top++;              // TOP is incremented after an element is pushed
 stack[top] = x;   // The pushed element is made as TOP
 }}
void pop()
{
 if(top <= -1)
 {
 printf("\nSTACK UNDERFLOW\n");
 }
 else
 {
 printf("\nThe popped element is %d",stack[top]);
 top--;     //  Decrement TOP after a pop
 }}
void display()
{
 if(top >= 0)
 {
//  Print the stack
 printf("\nELEMENTS IN THE STACK\n\n");
 for(i = top ; i >= 0 ; i--)
 printf("%d\t",stack[i]);
 }
 else
 {
 printf("\nEMPTY STACK\n");
 }}
```

OUTPUT:


NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the choice : 1
Enter a value to be pushed : 2

1.PUSH
2.POP
3.DISPLAY
4.EXIT

Enter the choice : 1
Enter a value to be pushed : 3

1.PUSH
2.POP
3.DISPLAY
4.EXIT

Enter the choice : 2

The popped element is 3
1.PUSH
2.POP
3.DISPLAY
4.EXIT

Enter the choice :
```

RESULT:

Thus the C Program for to Push, Pop and Displaying the elements in stack using array been executed successfully.

EX. NO. 7 B	
DATE:	**IMPLEMENTATION OF STACK USING LINKED LIST**

AIM:
To write a c program to implement stack using Linked List

ALGORITHM:

Step 1 - Include all the header files which are used in the program. And declare all the user defined functions.
Step 2 - Define a 'Node' structure with two members data and next.
Step 3 - Define a Node pointer 'top' and set it to NULL.
Step 4 - Implement the main method by displaying Menu with list of operations and make suitable function calls in the main method.

push(value) - Inserting an element into the Stack
Step 1 - Create a newNode with given value.
Step 2 - Check whether stack is Empty (top == NULL)
Step 3 - If it is Empty, then set newNode → next = NULL.
Step 4 - If it is Not Empty, then set newNode → next = top.
Step 5 - Finally, set top = newNode.

pop() - Deleting an Element from a Stack
Step 1 - Check whether stack is Empty (top == NULL).
Step 2 - If it is Empty, then display "Stack is Empty!!! Deletion is not possible!!!" and terminate the function
Step 3 - If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.
Step 4 - Then set 'top = top → next'.
Step 5 - Finally, delete 'temp'. (free(temp)).

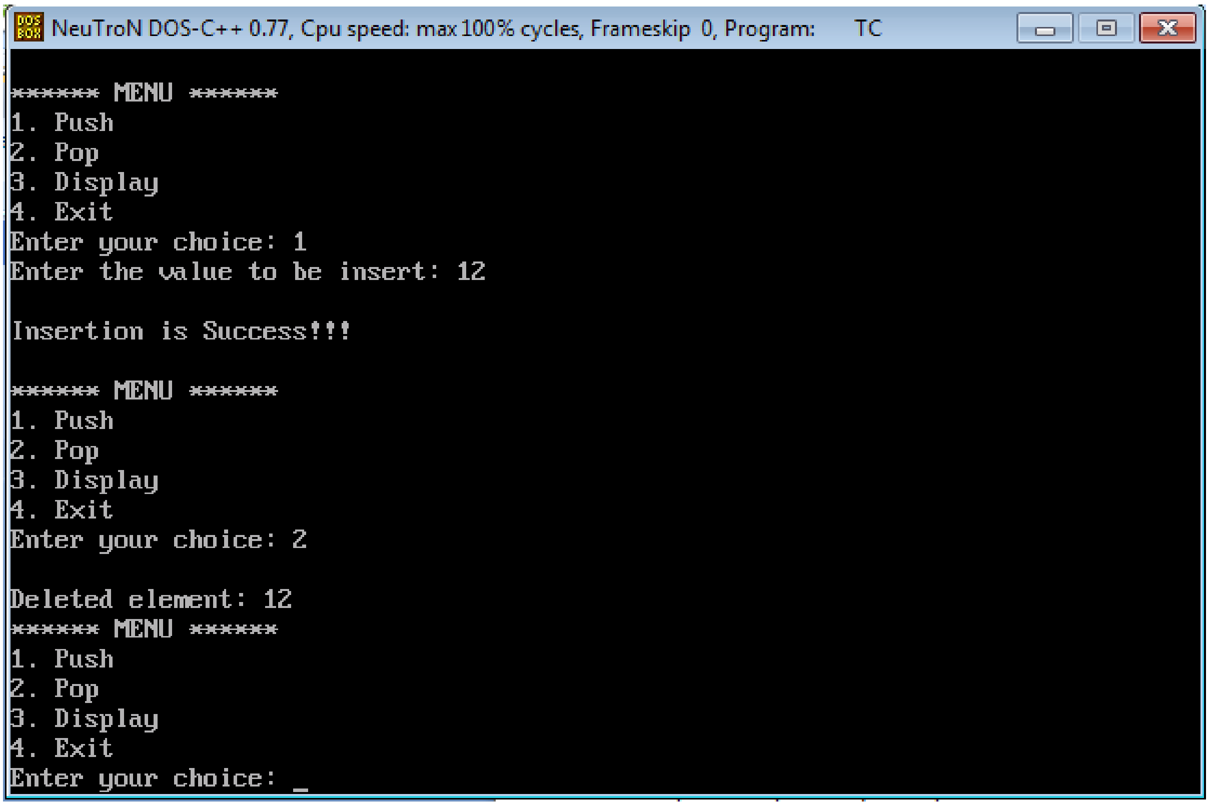
display() - Displaying stack of elements
Step 1 - Check whether stack is Empty (top == NULL).
Step 2 - If it is Empty, then display 'Stack is Empty!!!' and terminate the function.
Step 3 - If it is Not Empty, then define a Node pointer 'temp' and initialize with top.
Step 4 - Display 'temp → data --->' and move it to the next node. Repeat the same until temp reaches to the first node in the stack. (temp → next != NULL).
Step 5 - Finally! Display 'temp → data ---> NULL'.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
struct Node
{
  int data;
  struct Node *next;
}*top = NULL;
void push(int);
void pop();
void display();
void main()
{
 int choice, value;
 clrscr();
 printf("\n:: Stack using Linked List ::\n");
while(1){
printf("\n****** MENU ******\n");
printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
printf("Enter your choice: ");
scanf("%d",&choice);
switch(choice){
case 1: printf("Enter the value to be insert: ");
scanf("%d", &value);
push(value);
break;
case 2: pop(); break;
case 3: display(); break;
case 4: exit(0);
default: printf("\nWrong selection!!! Please try again!!!\n");
}
}
}
void push(int value)
{
struct Node *newNode;
newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;
if(top == NULL)
newNode->next = NULL;
else
newNode->next = top;
top = newNode;
printf("\nInsertion is Success!!!\n");
}
void pop()
{
if(top == NULL)
printf("\nStack is Empty!!!\n");
else{
struct Node *temp = top;
printf("\nDeleted element: %d", temp->data);
top = temp->next;
free(temp);
}
}
```

```
void display()
{
 if(top == NULL)
printf("\nStack is Empty!!!\n");
else{
struct Node *temp = top;
while(temp->next != NULL){
printf("%d--->",temp->data);
temp = temp -> next;
}
printf("%d--->NULL",temp->data);
}
}
```

OUTPUT:


NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
******* MENU *******
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 12

Insertion is Success!!!

******* MENU *******
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2

Deleted element: 12
******* MENU *******
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: _
```

RESULT:

Thus the C Program for to Push, Pop and Displaying the elements in stack using Linked List been executed successfully.

EX. NO. 8 A	
DATE:	**IMPLEMENTATION OF QUEUE USING ARRAY**

AIM:

To write a c program to implement queue using array

ALGORITHM:

Step 1 - Include all the header files which are used in the program and define a constant 'SIZE' with specific value.

Step 2 - Declare all the user defined functions which are used in queue implementation.

Step 3 - Create a one dimensional array with above defined SIZE (int queue[SIZE])

Step 4 - Define two integer variables 'front' and 'rear' and initialize both with '-1'. (int front = -1, rear = -1)

Step 5 - Then implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on queue.

enQueue(value) - Inserting value into the queue

Step 1 - Check whether queue is FULL. (rear == SIZE-1)

Step 2 - If it is FULL, then display "Queue is FULL!!! Insertion is not possible!!!" and terminate the function.

Step 3 - If it is NOT FULL, then increment rear value by one (rear++) and set queue[rear] = value.

deQueue() - Deleting a value from the Queue

Step 1 - Check whether queue is EMPTY. (front == rear)

Step 2 - If it is EMPTY, then display "Queue is EMPTY!!! Deletion is not possible!!!" and terminate the function.

Step 3 - If it is NOT EMPTY, then increment the front value by one (front ++). Then display queue[front] as deleted element. Then check whether both front and rear are equal (front == rear), if it TRUE, then set both front and rear to '-1' (front = rear = -1).

display() - Displays the elements of a Queue

Step 1 - Check whether queue is EMPTY. (front == rear)

Step 2 - If it is EMPTY, then display "Queue is EMPTY!!!" and terminate the function.

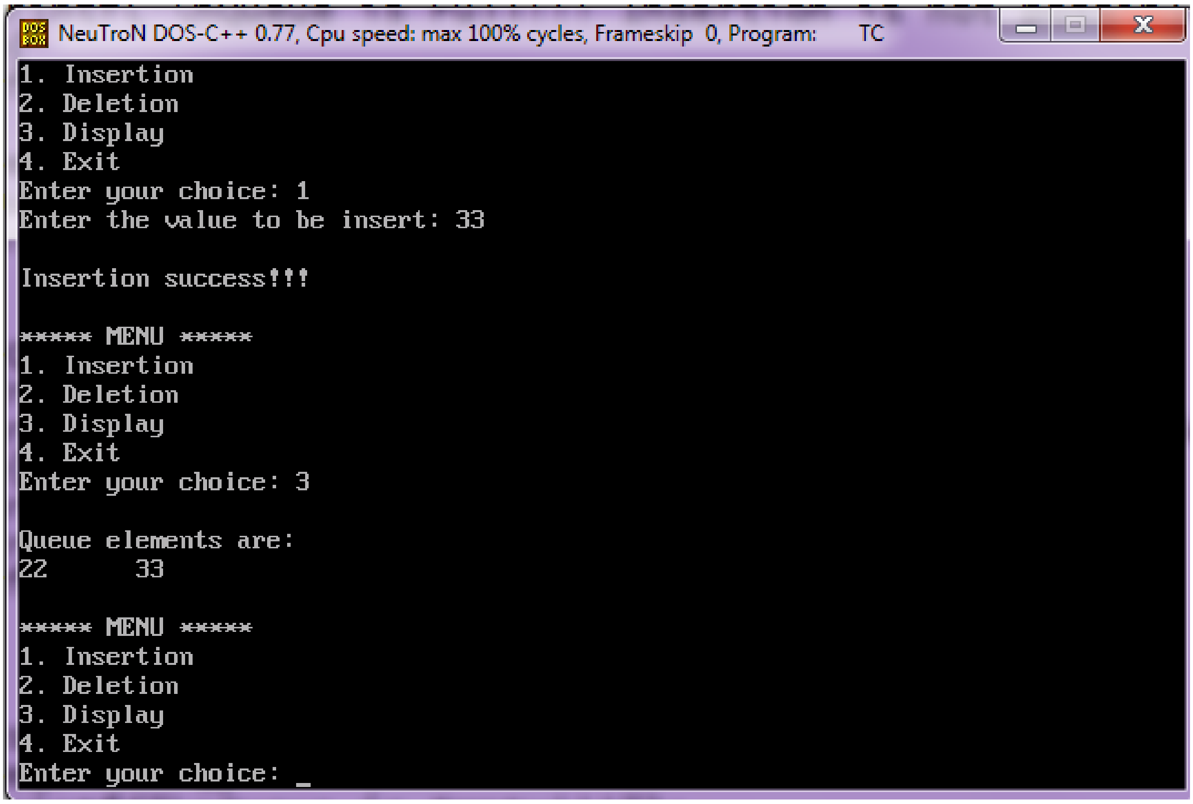
Step 3 - If it is NOT EMPTY, then define an integer variable 'i' and set 'i = front+1'.

Step 4 - Display 'queue[i]' value and increment 'i' value by one (i++). Repeat the same until 'i' value reaches to rear (i <= rear)

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#define SIZE 10
void enQueue(int);
void deQueue();
void display();
int queue[SIZE], front = -1, rear = -1;
void main()
{
int value, choice;
clrscr();
while(1){
printf("\n\n***** MENU *****\n");
printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");
printf("\nEnter your choice: ");
scanf("%d",&choice);
switch(choice){
case 1: printf("Enter the value to be insert: ");
scanf("%d",&value);
enQueue(value);
break;
case 2: deQueue();
break;
case 3: display();
break;
case 4: exit(0);
default: printf("\nWrong selection!!! Try again!!!");
}
}
}
void enQueue(int value){
if(rear == SIZE-1)
printf("\nQueue is Full!!! Insertion is not possible!!!");
else{
if(front == -1)
front = 0;
rear++;
queue[rear] = value;
printf("\nInsertion success!!!");
}
}
void deQueue(){
if(front == rear)
printf("\nQueue is Empty!!! Deletion is not possible!!!");
else{
printf("\nDeleted : %d", queue[front]);
front++;
if(front == rear)
front = rear = -1;
}
}
void display(){
if(rear == -1)
printf("\nQueue is Empty!!!");
else{
int i;
printf("\nQueue elements are:\n");
for(i=front; i<=rear; i++)
printf("%d\t",queue[i]);
}
}
```

OUTPUT:


NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 33

Insertion success!!!

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3

Queue elements are:
22      33

***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: _
```

RESULT:

Thus the C Program for Inserting, Deleting and Displaying the elements in Queue Using Array been executed successfully.

EX. NO. 8 B	**IMPLEMENTATION OF QUEUE USING LINKED LIST**
DATE:	

AIM:

To write a c program for to implement Queue using Linked List

ALGORITHM:

Step 1 - Include all the header files which are used in the program. And declare all the user defined functions.
Step 2 - Define a 'Node' structure with two members data and next.
Step 3 - Define two Node pointers 'front' and 'rear' and set both to NULL.
Step 4 - Implement the main method by displaying Menu of list of operations and make suitable function calls in the main method to perform user selected operation.

enQueue(value) - Inserting an element into the Queue
Step 1 - Create a newNode with given value and set 'newNode → next' to NULL.
Step 2 - Check whether queue is Empty (rear == NULL)
Step 3 - If it is Empty then, set front = newNode and rear = newNode.
Step 4 - If it is Not Empty then, set rear → next = newNode and rear = newNode.

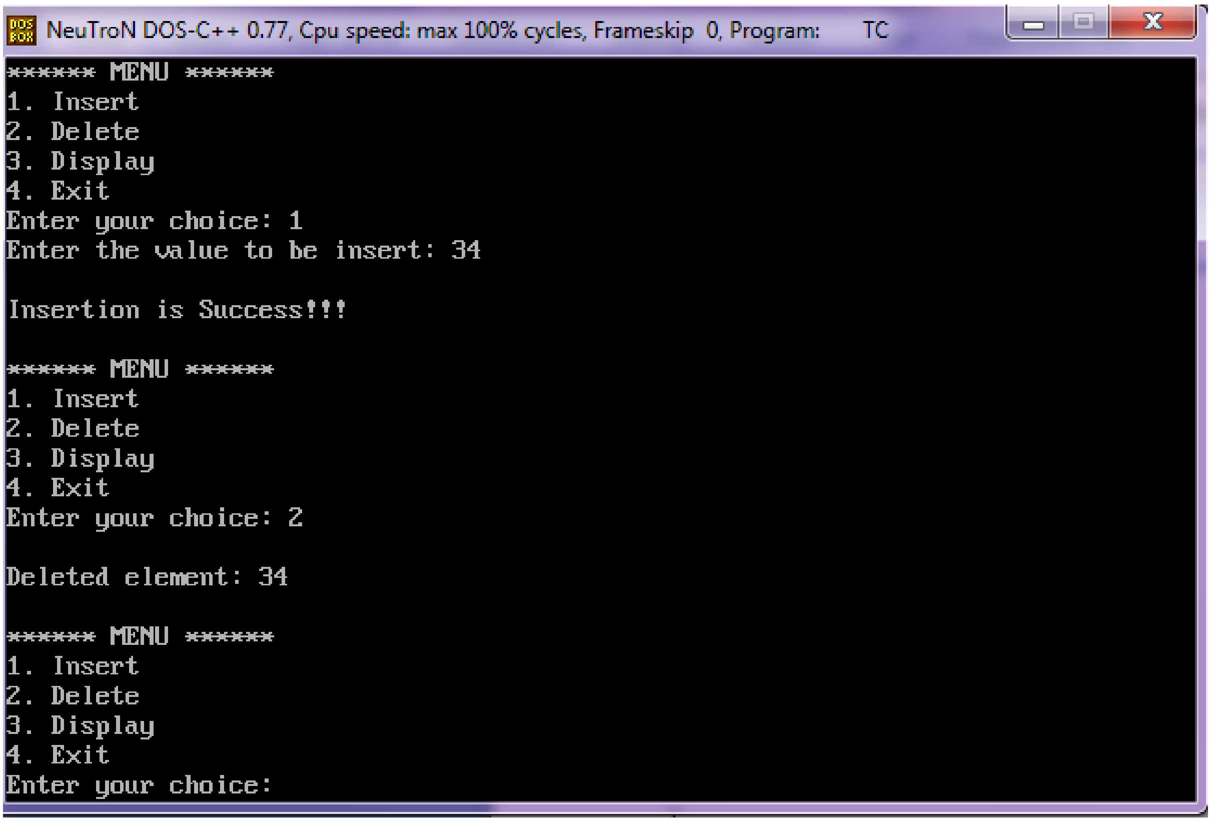
deQueue() - Deleting an Element from Queue
Step 1 - Check whether queue is Empty (front == NULL).
Step 2 - If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate from the function
Step 3 - If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'.
Step 4 - Then set 'front = front → next' and delete 'temp' (free(temp)).

display() - Displaying the elements of Queue
Step 1 - Check whether queue is Empty (front == NULL).
Step 2 - If it is Empty then, display 'Queue is Empty!!!' and terminate the function.
Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with front.
Step 4 - Display 'temp → data --->' and move it to the next node. Repeat the same until 'temp' reaches to 'rear' (temp → next != NULL).
Step 5 - Finally! Display 'temp → data ---> NULL'.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
struct Node
{
int data;
struct Node *next;
}*front = NULL,*rear = NULL;
void insert(int);
void delete();
void display();
void main()
{
int choice, value;
clrscr();
printf("\n:: Queue Implementation using Linked List ::\n");
while(1){
printf("\n****** MENU ******\n");
printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
printf("Enter your choice: ");
scanf("%d",&choice);
switch(choice){
case 1: printf("Enter the value to be insert: ");
scanf("%d", &value);
insert(value);
break;
case 2: delete(); break;
case 3: display(); break;
case 4: exit(0);
default: printf("\nWrong selection!!! Please try again!!!\n");
}
}
}
void insert(int value)
{
struct Node *newNode;
newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;
newNode -> next = NULL;
if(front == NULL)
front = rear = newNode;
else{
rear -> next = newNode;
rear = newNode;
}
printf("\nInsertion is Success!!!\n");
}
void delete()
{
if(front == NULL)
printf("\nQueue is Empty!!!\n");
else{
struct Node *temp = front;
front = front -> next;
printf("\nDeleted element: %d\n", temp->data);
free(temp);
}
}
void display()
{
if(front == NULL)
printf("\nQueue is Empty!!!\n");
else{
struct Node *temp = front;
while(temp->next != NULL){
printf("%d--->",temp->data);
temp = temp -> next;
}
printf("%d--->NULL\n",temp->data);
}
}
```

OUTPUT:



NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 34

Insertion is Success!!!

****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2

Deleted element: 34

****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
```

RESULT:

Thus the C Program for Inserting, Deleting and Displaying the elements in Queue Using Linked List been executed successfully.

EX. NO. 9 A	
DATE:	IMPLEMENTATION OF STACK APPLICATION'S- INFIX TO POSTFIX

AIM:
To write a c program to implement Application of Stack (Infix to Post Fix)

ALGORITHM:
To convert Infix Expression into Postfix Expression using a stack data structure, We can use the following steps...

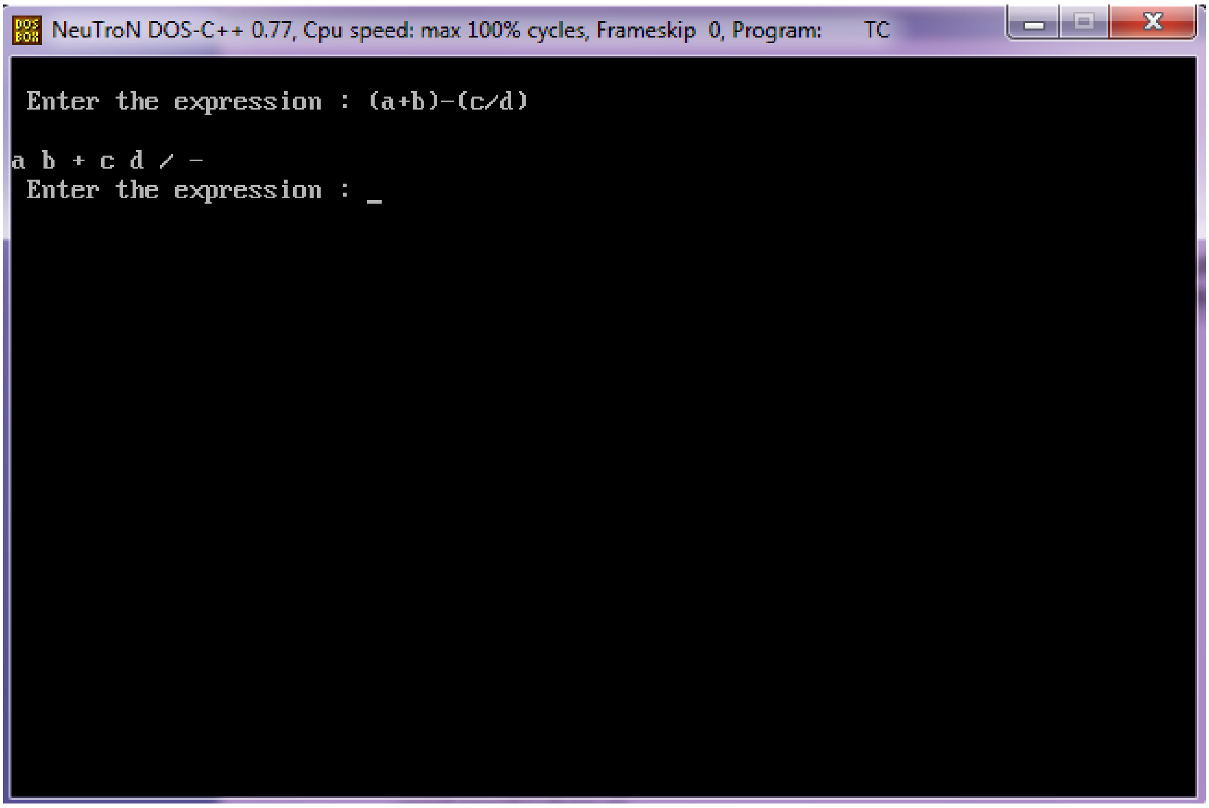
- Read all the symbols one by one from left to right in the given Infix Expression.
- If the reading symbol is operand, then directly print it to the result (Output).
- If the reading symbol is left parenthesis '(', then Push it on to the Stack.
- If the reading symbol is right parenthesis ')', then Pop all the contents of stack until respective left parenthesis is poped and print each poped symbol to the result.
- If the reading symbol is operator (+ , - , * , / etc.,), then Push it on to the Stack.first pop the operators which are already on the stack that have higher or equal precedence than current operator and print them to the result.

PROGRAM

```
#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 == '-')
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);
}
else
{
while(priority(stack[top]) >= priority(*e))
printf("%c ",pop());
push(*e);
}
e++;
}

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

OUTPUT:


NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
 Enter the expression : (a+b)-(c/d)

a b + c d / -
 Enter the expression : _
```

RESULT:

Thus the C Program for converting Infix to postfix has been executed successfully.

EX. NO. 9 B	**IMPLEMENTATION OF STACK APPLICATION'S- Tower of Hanoi**
DATE:	

AIM:

To write a c program to implement Application of Stack (Tower of Hanoi)

ALGORITHM:

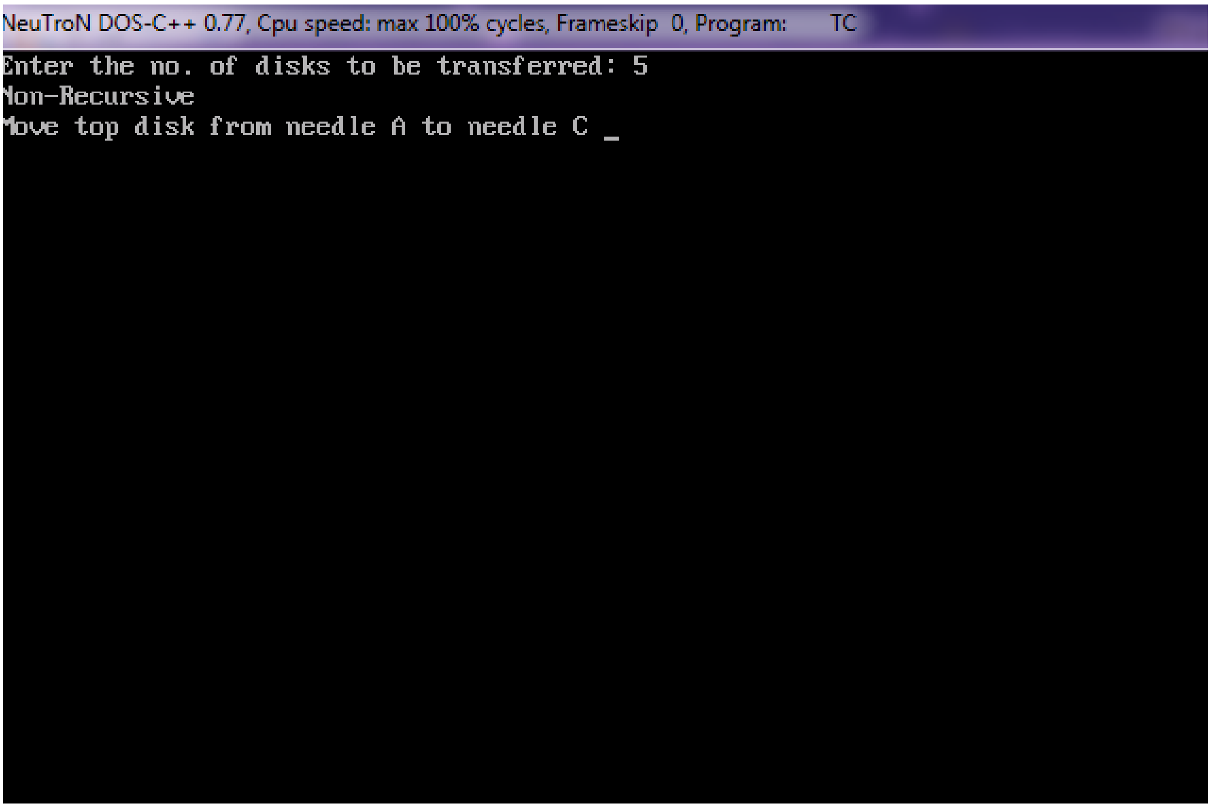
```
START
Procedure Hanoi (disk, source, dest, aux)

   IF disk == 1, THEN
      move disk from source to dest
   ELSE
      Hanoi(disk - 1, source, aux, dest)     // Step 1
      move disk from source to dest          // Step 2
      Hanoi(disk - 1, aux, dest, source)     // Step 3
   END IF

END Procedure
STOP
```

PROGRAM

```
#include<conio.h>
#include<stdio.h>
/* Non-Recursive Function*/
void hanoiNonRecursion(int num,char sndl,char indl,char dndl)
{
char stkn[50],stksndl[50],stkindl[50],stkdndl[50],stkadd[50],temp;
int top,add;
top=NULL;
one:
if(num==1)
{
printf("\nMove top disk from needle %c to needle %c ",sndl,dndl);
}
two:
    top=top+1;
    stkn[top]=num;
    stksndl[top]=sndl;
    stkindl[top]=indl;
    stkdndl[top]=dndl;
    stkadd[top]=3;
    num=num-1;
    sndl=sndl;
    temp=indl;
    indl=dndl;
    dndl=temp;
    goto one;
three:
    printf("\nMove top disk from needle %c to needle %c ",sndl,dndl);
    top=top+1;
    stkn[top]=num;
    stksndl[top]=sndl;
    stkindl[top]=indl;
    stkdndl[top]=dndl;
    stkadd[top]=5;
    num=num-1;
    temp=sndl;
    sndl=indl;
    indl=temp;
    dndl=dndl;
    goto one;
four:
    if(top==NULL)
      return;
    num=stkn[top];
    sndl=stksndl[top];
    indl=stkindl[top];
    dndl=stkdndl[top];
    add=stkadd[top];
    top=top-1;
    if(add==3)
      goto three;
    else if(add==5)
      goto four;
}
/* Recursive Function*/
void  hanoiRecursion( int num,char ndl1, char ndl2, char ndl3)
{
if ( num == 1 ) {
printf( "\nMove top disk from needle %c to needle %c.", ndl1, ndl2 );
return;
}
hanoiRecursion( num - 1,ndl1, ndl3, ndl2 );
printf( "\nMove top disk from needle %c to needle %c.", ndl1, ndl2 );
hanoiRecursion( num - 1,ndl3, ndl2, ndl1 );
}
int main()
{
int no;
//clrscr();
printf("Enter the no. of disks to be transferred: ");
scanf("%d",&no);
```

```
if(no<1)
printf("\nThere's nothing to move.");
else
printf("Non-Recursive");
hanoiNonRecursion(no,'A','B','C');
printf("\nRecursive");
hanoiRecursion(no,'A','B','C');
return 0;
}
```

OUTPUT:



NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the no. of disks to be transferred: 5
Non-Recursive
Move top disk from needle A to needle C _
```

Enter the no. of disks to be transferred: 3

Non-Recursive

Move top disk from needle A to needle C
Move top disk from needle A to needle B
Move top disk from needle C to needle B
Move top disk from needle A to needle C
Move top disk from needle B to needle A
Move top disk from needle B to needle C
Move top disk from needle A to needle C

Recursive

Move top disk from needle A to needle B.
Move top disk from needle A to needle C.
Move top disk from needle B to needle C.
Move top disk from needle A to needle B.

```
Move top disk from needle C to needle A.
Move top disk from needle C to needle B.
Move top disk from needle A to needle B.
```

RESULT:

Thus the C Program for converting Tower of Hanoi has been executed successfully.

EX. NO. 9 C	
DATE:	IMPLEMENTATION OF QUEUE APPLICATION'S- CPU SCHEDULING

AIM:

To write a c program to implement Application of Queue (CPU Scheduling)

ALGORITHM:

First Come First Serve, is just like FIFO (First in First out) Queue data structure, where the data element which is added to the queue first, is the one who leaves the queue first

STEP 1: Enter all the inputs needed for CPU-FCFS scheduling

(a)Total number of processes

(b)Process Order and Burst Time

STEP 2: Store the process in a queue based on First Come First Serve basis at the rear end of the queue and calculate the waiting time

Waiting time=Burst time of all the process stored in queue

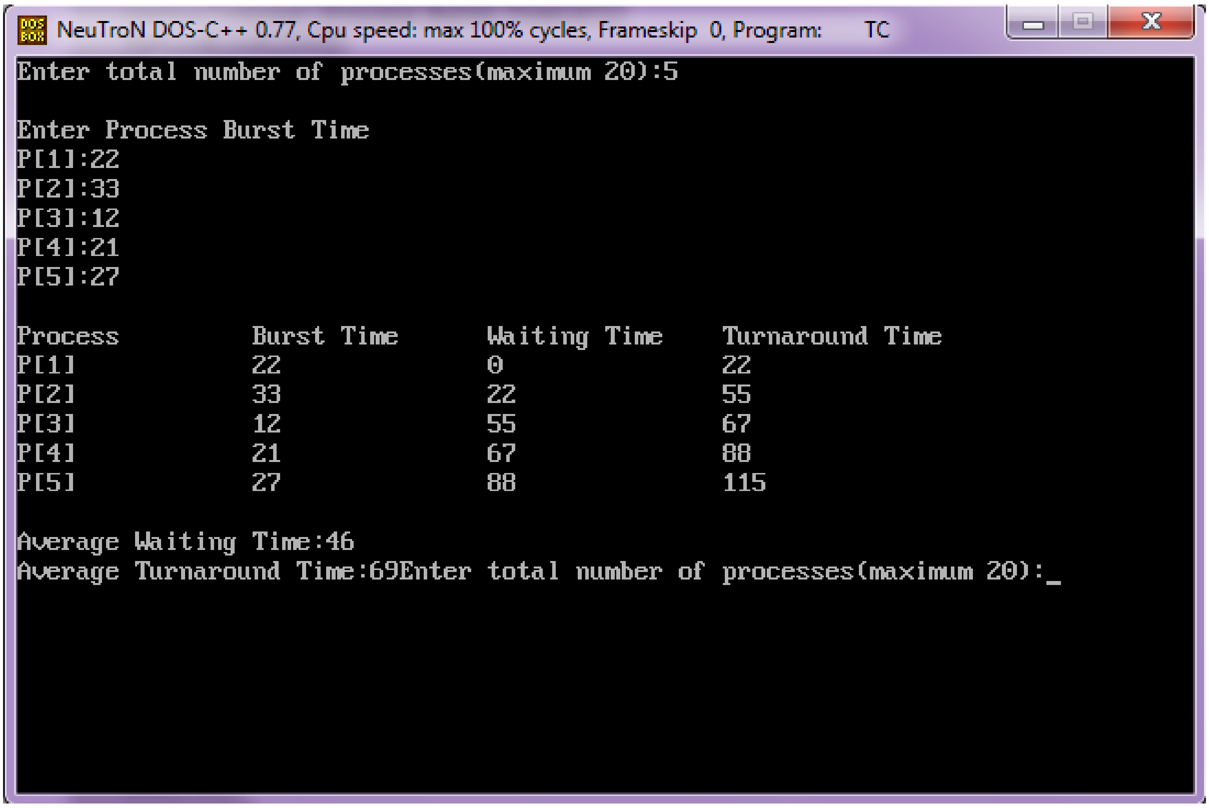
STEP 3: Pick a process one by one from the queue from the front end and do the following

Turnaround time=Waiting time Burst time

STEP 4: Calculate the average waiting time and average turnaround time.

PROGRAM

```
#include<stdio.h>
 int main()
{
int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
printf("Enter total number of processes(maximum 20):");
scanf("%d",&n);
printf("\nEnter Process Burst Time\n");
for(i=0;i<n;i++)
{
printf("P[%d]:",i+1);
scanf("%d",&bt[i]);
}
wt[0]=0;    //waiting time for first process is 0
//calculating waiting time
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
}
printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");
//calculating turnaround time
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i];
avwt+=wt[i];
avtat+=tat[i];
printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
}
avwt/=i;
avtat/=i;
printf("\n\nAverage Waiting Time:%d",avwt);
printf("\nAverage Turnaround Time:%d",avtat);
return 0;
}
```



NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter total number of processes(maximum 20):5

Enter Process Burst Time
P[1]:22
P[2]:33
P[3]:12
P[4]:21
P[5]:27

Process         Burst Time      Waiting Time    Turnaround Time
P[1]            22              0               22
P[2]            33              22              55
P[3]            12              55              67
P[4]            21              67              88
P[5]            27              88              115

Average Waiting Time:46
Average Turnaround Time:69Enter total number of processes(maximum 20):_
```

RESULT:

Thus the C Program for CPU Scheduling First Come First Serve has been executed successfully.

EX. NO. 10 A	
DATE:	**IMPLEMENTATION OF TREE USING ARRAY**

AIM:

To write a c program to implement of Tree using array

ALGORITHM:

(A)CREATION AND INSERTION

```
STEP 1:If root is NULL
  then create root node
return

STEP 2: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
```

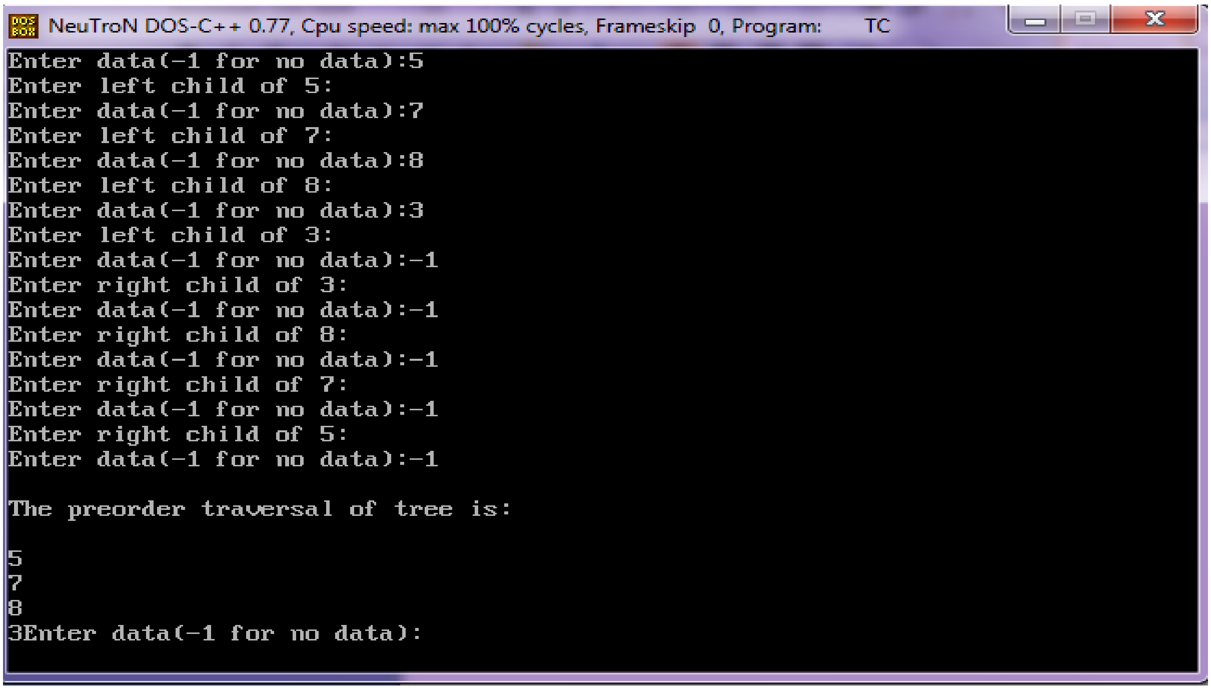
(B)SEARCH

```
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
```

PROGRAM

```
#include<stdio.h>
typedef struct node
{
int data;
struct node *left;
struct node *right;
} node;
node *create()
{
node *p;
int x;
printf("Enter data(-1 for no data):");
scanf("%d",&x);
if(x==-1)
return NULL;
p=(node*)malloc(sizeof(node));
p->data=x;
printf("Enter left child of %d:\n",x);
p->left=create();
printf("Enter right child of %d:\n",x);
p->right=create();
return p;
}
void preorder(node *t)                  //address of root node is passed in t
{
if(t!=NULL)
{
printf("\n%d",t->data);                 //visit the root
preorder(t->left);              //preorder traversal on left subtree
preorder(t->right);             //preorder traversal om right subtree
}
}
int main()
{
node *root;
root=create();
printf("\nThe preorder traversal of tree is:\n");
preorder(root);
return 0;
}
```

Output


NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter data(-1 for no data):5
Enter left child of 5:
Enter data(-1 for no data):7
Enter left child of 7:
Enter data(-1 for no data):8
Enter left child of 8:
Enter data(-1 for no data):3
Enter left child of 3:
Enter data(-1 for no data):-1
Enter right child of 3:
Enter data(-1 for no data):-1
Enter right child of 8:
Enter data(-1 for no data):-1
Enter right child of 7:
Enter data(-1 for no data):-1
Enter right child of 5:
Enter data(-1 for no data):-1

The preorder traversal of tree is:

5
7
8
3Enter data(-1 for no data):
```

Enter data(-1 for no data):5

Enter left child of 5:

Enter data(-1 for no data):7

Enter left child of 7:

Enter data(-1 for no data):8

Enter left child of 8:

Enter data(-1 for no data):3

Enter left child of 3:

Enter data(-1 for no data):-1

Enter right child of 3:

Enter data(-1 for no data):-1

```
Enter right child of 8:

Enter data(-1 for no data):-1

Enter right child of 7:

Enter data(-1 for no data):-1

Enter right child of 5:

Enter data(-1 for no data):-1
```

The preorder traversal of tree is:

```
5
7
8
3
```

RESULT:
Thus the C Program for Implementation of Tree has been executed successfully.

EX. NO. 10 B	
DATE:	**IMPLEMENTATION OF BINARY TREE USING ARRAY**

AIM:

To write a c program to implement ofBinary Tree using array

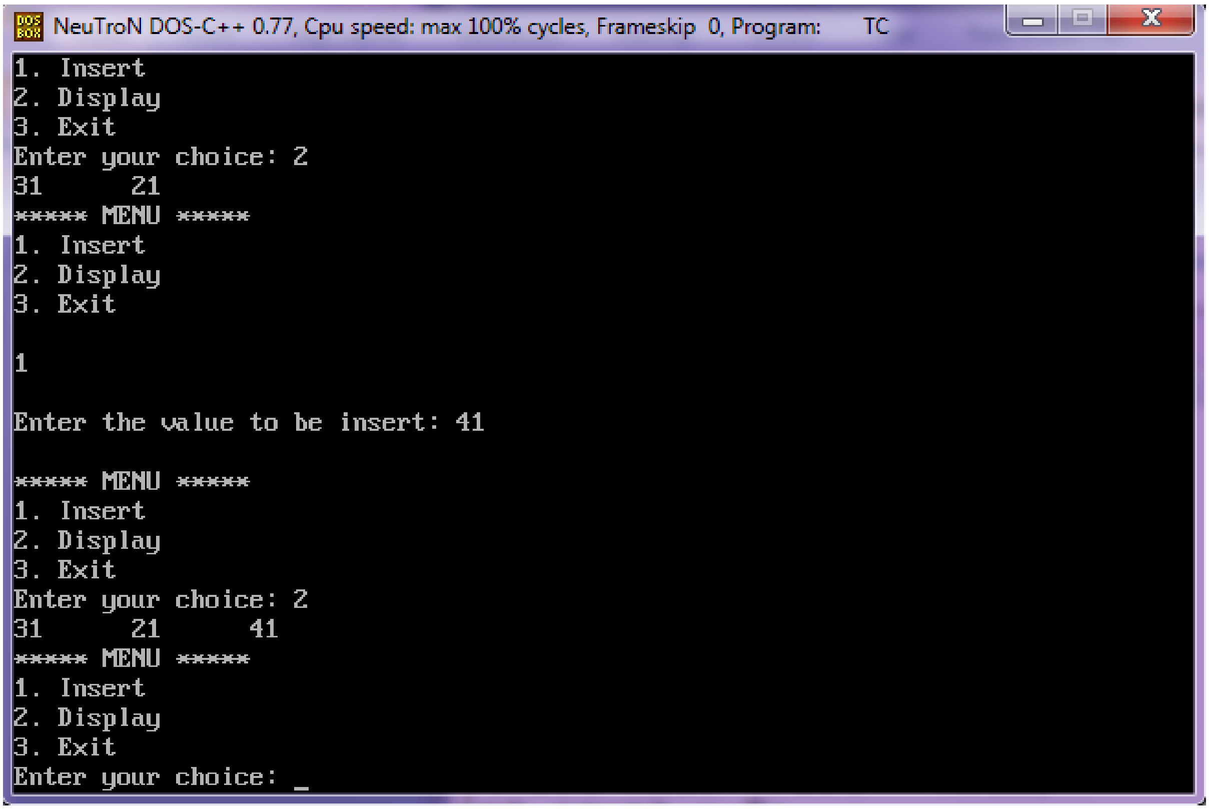
ALGORITHM:

```
structure BTREE
declare CREATE( ) --> btree
    ISMTBT(btree,item,btree) --> boolean
    MAKEBT(btree,item,btree) --> btree
    LCHILD(btree) --> btree
    DATA(btree) --> item
    RCHILD(btree) --> btree
 for all p,r in btree, d in item let
  ISMTBT(CREATE)::=true
  ISMTBT(MAKEBT(p,d,r))::=false
  LCHILD(MAKEBT(p,d,r))::=p; LCHILD(CREATE)::=error
  DATA(MAKEBT(p,d,r))::d; DATA(CREATE)::=error
  RCHILD(MAKEBT(p,d,r))::=r; RCHILD(CREATE)::=error
 end
end BTREE
```

PROGRAM

```
#include<stdio.h>
#include<conio.h>
struct Node{
int data;
struct Node *left;
struct Node *right;
};
struct Node *root = NULL;
int count = 0;
struct Node* insert(struct Node*, int);
void display(struct Node*);
void main(){
int choice, value;
clrscr();
printf("\n----- Binary Tree -----\n");
while(1){
printf("\n***** MENU *****\n");
printf("1. Insert\n2. Display\n3. Exit");
printf("\nEnter your choice: ");
scanf("%d",&choice);
switch(choice){
case 1: printf("\nEnter the value to be insert: ");
scanf("%d", &value);
root = insert(root,value);
break;
case 2: display(root); break;
case 3: exit(0);
default: printf("\nPlease select correct operations!!!\n");
}
}
}
struct Node* insert(struct Node *root,int value){
struct Node *newNode;
newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;
if(root == NULL){
newNode->left = newNode->right = NULL;
root = newNode;
count++;
}
else{
if(count%2 != 0)
root->left = insert(root->left,value);
else
root->right = insert(root->right,value);
}
return root;
}
// display is performed by using Inorder Traversal
void display(struct Node *root)
{
if(root != NULL){
display(root->left);
printf("%d\t",root->data);
display(root->right);
}
}
```

OUTPUT:


NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
1.  Insert
2.  Display
3.  Exit
Enter your choice: 2
31       21
***** MENU *****
1.  Insert
2.  Display
3.  Exit

1

Enter the value to be insert: 41

***** MENU *****
1.  Insert
2.  Display
3.  Exit
Enter your choice: 2
31       21       41
***** MENU *****
1.  Insert
2.  Display
3.  Exit
Enter your choice: _
```

RESULT:

Thus the C Program for Implementation of Binary Tree has been executed successfully.

EX. NO. 11	**IMPLEMENTATION OF BINARY SEARCH TREE USING LINKED LIST**
DATE:	

AIM:

To write a c program to implement BST using linked list

ALGORITHM:

```
struct node* search(int data){
   struct node *current = root;
   printf("Visiting elements: ");

   while(current->data != data){

      if(current != NULL) {
         printf("%d ",current->data);

         //go to left tree
         if(current->data > data){
            current = current->leftChild;
         }  //else go to right tree
         else {
            current = current->rightChild;
         }

         //not found
         if(current == NULL){
            return NULL;
         }
      }
   }

   return current;
}
```

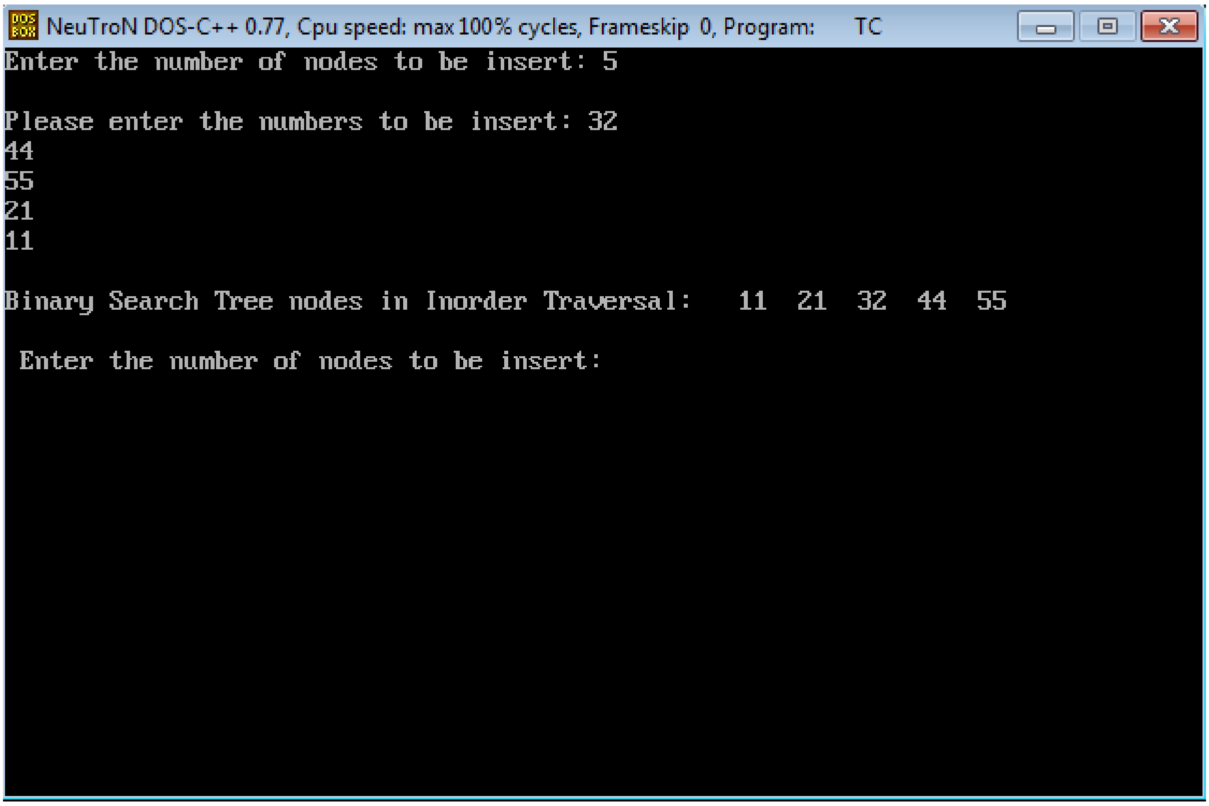
PROGRAM

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

struct node
{
int data;
struct node *left;
struct node *right;
};
```

```
void inorder(struct node *root)
{
if(root)
{`
inorder(root->left);
printf("  %d",root->data);
inorder(root->right);
}
}
int main()
{
int n , i;
struct node *p , *q , *root;
printf("Enter the number of nodes to be insert: ");
scanf("%d",&n);
printf("\nPlease enter the numbers to be insert: ");
for(i=0;i<i++)
{
p = (struct node*)malloc(sizeof(struct node));
scanf("%d",&p->data);
p->left = NULL;
p->right = NULL;
if(i == 0)
{
root = p; // root always point to the root node
}
else
{
q = root;   // q is used to traverse the tree
while(1)
{
if(p->data > q->data)
{
if(q->right == NULL)
{
q->right = p;
break;
}
else
q = q->right;
}
else
{
if(q->left == NULL)
{
```

```
q->left = p;
break;
}
else
q = q->left;
}
}
}
}
printf("\nBinary Search Tree nodes in Inorder Traversal: ");
inorder(root);
printf("\n");
return 0;
}
```

OUTPUT:



NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the number of nodes to be insert: 5

Please enter the numbers to be insert: 32
44
55
21
11

Binary Search Tree nodes in Inorder Traversal:   11  21  32  44  55

 Enter the number of nodes to be insert:
```

RESULT:

Thus the C Program for Implementation of Binary SearchTree using Linked List has been executed successfully.

EX. NO. 12	
DATE:	IMPLEMENTATION OF B-TREES

AIM:

To write a c program to implement B - Tree

ALGORITHM

B-TREE-CREATE(T)

```
 x  ALLOCATE-NODE()
 leaf[x]  TRUE
 n[x]  0
 DISK-WRITE(x)
 root[T]  x
```

B-TREE-INSERT(T,k)

```
 r  root[T]
 if n[r] = 2t - 1
         then s  ALLOCATE-NODE()
             root[T]  s
             leaf[s]  FALSE
           n[s]  0
           c1[s]  r
           B-TREE-SPLIT-CHILD(s,1,r)
           B-TREE-INSERT-NONFULL(s,k)
 else B-TREE-INSERT-NONFULL(r,k)
```

B-TREE-SEARCH(x, k)

```
  i  1
  while i  n[x] and k  keyi[x]
          do i  i + 1
  if i  n[x] and k = keyi[x]
         then return (x, i)
  if leaf [x]
         then return NIL
  else DISK-READ(ci[x])
        return B-TREE-SEARCH(ci[x], k)
```

B-TREE -DELETION

(i)If the key k is in node x and x is a leaf, delete the key k from x.

(ii)If the key k is in node x and x is an internal node, do the following.

a.If the child y that precedes k in node x has at least t keys, then find the predecessor k' of k in the sub tree rooted at y. Recursively delete k', and replace k by k' in x. (Finding k' and deleting it can be performed in a single downward pass.)

b.Symmetrically, if the child z that follows k in node x has at least t keys, then find the successor k' of k in the sub tree rooted at z. Recursively delete k', and replace k by k' in x. (Finding k' and deleting it can be performed in a single downward pass.)

c. Otherwise, if both y and z have only t- 1 keys, merge k and all of z into y, so that x loses both k and the pointer to z, and y now contains 2t - 1 keys. Then, free z and recursively delete k from y.

(iii)If the key k is not present in internal node x, determine the root ci[x] of the appropriate sub tree that must contain k, if k is in the tree at all. If ci[x] has only t - 1 keys, execute step 3a or 3b as necessary to guarantee that we descend to a node containing at least t keys. Then, finish by recursing on the appropriate child of x.

a.If ci[x] has only t - 1 keys but has a sibling with t keys, give ci[x] an extra key by moving a key from x down into ci[x], moving a key from ci[x]'s immediate left or right sibling up into x, and moving the appropriate child from the sibling into ci[x].

b.If ci[x] and all of ci[x]'s siblings have t - 1 keys, merge ci with one sibling, which involves moving a key from x down into the new merged node to become the median key for that node.

PROGRAM

```
#include <stdio.h>
#include<conio.h>
#include <stdlib.h>
#define M 3
typedef struct _node {
int n; /*n < M No. of keys in node will always less than order of B tree*/
int keys[M - 1];
struct _node *p[M]; /* (n+1 pointers will be in use) */
} node;
node *root = NULL;
typedef enum KeyStatus { Duplicate, SearchFailure, Success, InsertIt, LessKeys,
} KeyStatus;
void insert(int key);
void display(node *root, int);
void DelNode(int x);
void search(int x);
KeyStatus ins(node *r, int x, int* y, node** u);
int searchPos(int x, int *key_arr, int n);
KeyStatus del(node *r, int x);
void eatline(void);
void inorder(node *ptr);
int main() {
clrscr();
int key;
int choice;
printf("Creation of B tree for M=%d\n", M);
```

```
while (1) {
printf("1.Insert\n2.Delete\n3.Search\n4.Display\n5.Quit\nEnter your choice: ");
scanf("%d", &choice); eatline();
switch (choice) {
case 1:
printf("Enter the key: ");
scanf("%d", &key); eatline();
insert(key);
break;
case 2:
printf("Enter the key : ");
scanf("%d", &key); eatline();
DelNode(key);
break;
case 3:
printf("Enter the key : ");
scanf("%d", &key); eatline();
search(key);
break;
case 4:
printf("Btree is :\n");
display(root, 0);
break;
case 5:
exit(1);
default:
printf("Wrong choice\n");
break;
}}
return 0;
}
void insert(int key) {
node *newnode;
int upKey;
KeyStatus value;
value = ins(root, key, &upKey, &newnode);
if (value == Duplicate)
printf("Key already available\n");
if (value == InsertIt) {
node *uproot = root;
root = (node*)malloc(sizeof(node));
root->n = 1;
root->keys[0] = upKey;
root->p[0] = uproot;
root->p[1] = newnode;
}}
KeyStatus ins(node *ptr, int key, int *upKey, node **newnode) {
```

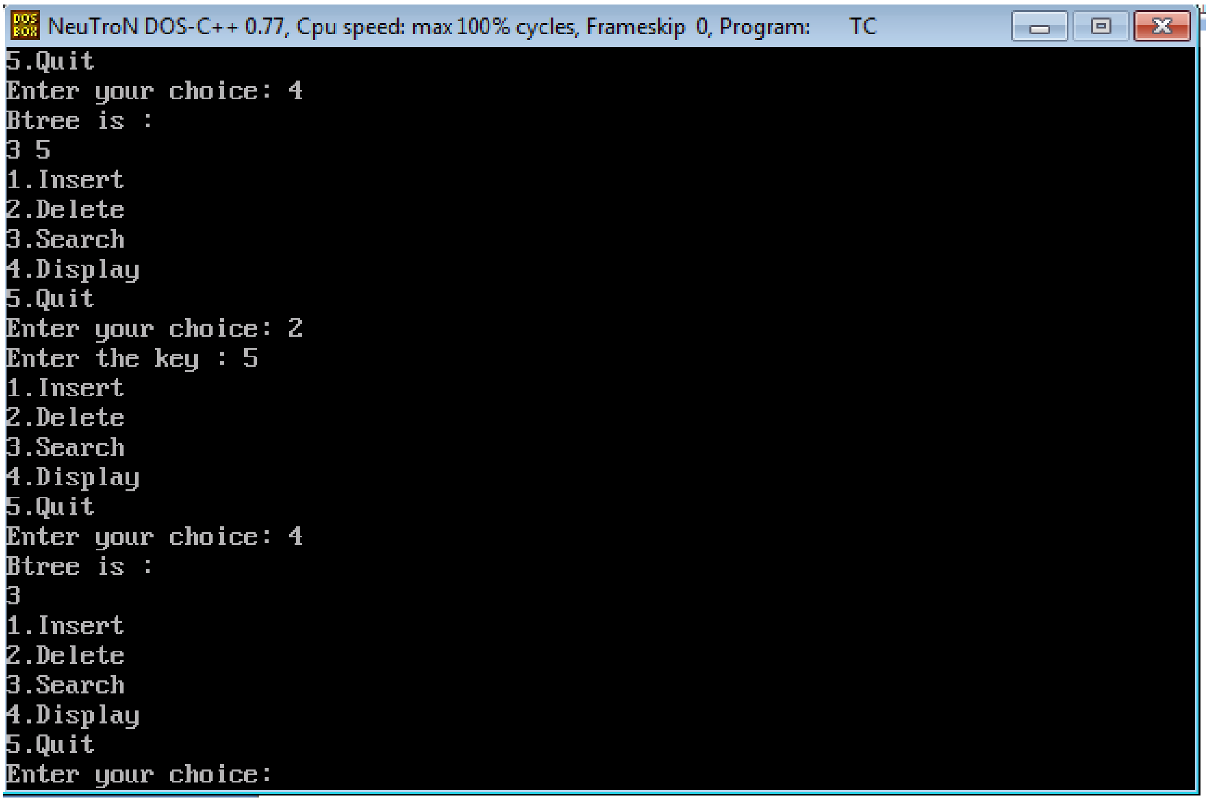
```
node *newPtr, *lastPtr;
int pos, i, n, splitPos;
int newKey, lastKey;
KeyStatus value;
if (ptr == NULL) {
*newnode = NULL;
*upKey = key;
return InsertIt;
}
n = ptr->n;
pos = searchPos(key, ptr->keys, n);
if (pos < n && key == ptr->keys[pos])
return Duplicate;
value = ins(ptr->p[pos], key, &newKey, &newPtr);
if (value != InsertIt)
return value;
/*If keys in node is less than M-1 where M is order of B tree*/
if (n < M - 1) {
pos = searchPos(newKey, ptr->keys, n);
/*Shifting the key and pointer right for inserting the new key*/
for (i = n; i>pos; i--) {
ptr->keys[i] = ptr->keys[i - 1];
ptr->p[i + 1] = ptr->p[i];
}
/*Key is inserted at exact location*/
ptr->keys[pos] = newKey;
ptr->p[pos + 1] = newPtr;
++ptr->n; /*incrementing the number of keys in node*/
return Success;
}
/*If keys in nodes are maximum and position of node to be inserted is last*/
if (pos == M - 1) {
lastKey = newKey;
lastPtr = newPtr;
}
else { /*If keys in node are maximum and position of node to be inserted is not
last*/
lastKey = ptr->keys[M - 2];
lastPtr = ptr->p[M - 1];
for (i = M - 2; i>pos; i--) {
ptr->keys[i] = ptr->keys[i - 1];
ptr->p[i + 1] = ptr->p[i];
}
ptr->keys[pos] = newKey;
ptr->p[pos + 1] = newPtr;
}
splitPos = (M - 1) / 2;
```

```
(*upKey) = ptr->keys[splitPos];
(*newnode) = (node*)malloc(sizeof(node));/*Right node after split*/
ptr->n = splitPos; /*No. of keys for left splitted node*/
(*newnode)->n = M - 1 - splitPos;/*No. of keys for right splitted node*/
for (i = 0; i < (*newnode)->n; i++) {
(*newnode)->p[i] = ptr->p[i + splitPos + 1];
if (i < (*newnode)->n - 1)
(*newnode)->keys[i] = ptr->keys[i + splitPos + 1];
else
(*newnode)->keys[i] = lastKey;
}
(*newnode)->p[(*newnode)->n] = lastPtr;
return InsertIt;
}
void display(node *ptr, int blanks) {
if (ptr) {
int i;
for (i = 1; i <= blanks; i++)
printf(" ");
for (i = 0; i < ptr->n; i++)
printf("%d ", ptr->keys[i]);
printf("\n");
for (i = 0; i <= ptr->n; i++)
display(ptr->p[i], blanks + 10);
}}
void search(int key) {
int pos, i, n;
node *ptr = root;
printf("Search path:\n");
while (ptr) {
n = ptr->n;
for (i = 0; i < ptr->n; i++)
printf(" %d", ptr->keys[i]);
printf("\n");
pos = searchPos(key, ptr->keys, n);
if (pos < n && key == ptr->keys[pos]) {
printf("Key %d found in position %d of last displayed node\n", key, i);
return;
}
ptr = ptr->p[pos];
}
printf("Key %d is not available\n", key);
}
int searchPos(int key, int *key_arr, int n) {
int pos = 0;
while (pos < n && key > key_arr[pos])
pos++;
```

```
return pos;
}
void DelNode(int key) {
node *uproot;
KeyStatus value;
value = del(root, key);
switch (value) {
case SearchFailure:
printf("Key %d is not available\n", key);
break;
case LessKeys:
uproot = root;
root = root->p[0];
free(uproot);
break;
default:
return;
}}
KeyStatus del(node *ptr, int key) {
int pos, i, pivot, n, min;
int *key_arr;
KeyStatus value;
node **p, *lptr, *rptr;
if (ptr == NULL)
return SearchFailure;
n = ptr->n;
key_arr = ptr->keys;
p = ptr->p;
min = (M - 1) / 2;
//Search for key to delete
pos = searchPos(key, key_arr, n);
if (p[0] == NULL) {
if (pos == n || key < key_arr[pos])
return SearchFailure;
/*Shift keys and pointers left*/
for (i = pos + 1; i < n; i++)
{
key_arr[i - 1] = key_arr[i];
p[i] = p[i + 1];
}
return --ptr->n >= (ptr == root ? 1 : min) ? Success : LessKeys;
}
if (pos < n && key == key_arr[pos]) {
node *qp = p[pos], *qp1;
int nkey;
while (1) {
nkey = qp->n;
qp1 = qp->p[nkey];
if (qp1 == NULL)
break;
qp = qp1;
}
key_arr[pos] = qp->keys[nkey - 1];
qp->keys[nkey - 1] = key;
}
value = del(p[pos], key);
if (value != LessKeys)
return value;
if (pos > 0 && p[pos - 1]->n > min) {
pivot = pos - 1; /*pivot for left and right node*/
lptr = p[pivot];
rptr = p[pos];
/*Assigns values for right node*/
rptr->p[rptr->n + 1] = rptr->p[rptr->n];
for (i = rptr->n; i>0; i--) {
rptr->keys[i] = rptr->keys[i - 1];
rptr->p[i] = rptr->p[i - 1];
}
rptr->n++;
rptr->keys[0] = key_arr[pivot];
rptr->p[0] = lptr->p[lptr->n];
key_arr[pivot] = lptr->keys[--lptr->n];
return Success;
}
if (pos < n && p[pos + 1]->n > min) {
pivot = pos; /*pivot for left and right node*/
lptr = p[pivot];
rptr = p[pivot + 1];
/*Assigns values for left node*/
lptr->keys[lptr->n] = key_arr[pivot];
lptr->p[lptr->n + 1] = rptr->p[0];
key_arr[pivot] = rptr->keys[0];
lptr->n++;
rptr->n--;
for (i = 0; i < rptr->n; i++) {
rptr->keys[i] = rptr->keys[i + 1];
rptr->p[i] = rptr->p[i + 1];
}/*End of for*/
rptr->p[rptr->n] = rptr->p[rptr->n + 1];
return Success;
}
if (pos == n)
pivot = pos - 1;
else
```

```
pivot = pos;
lptr = p[pivot];
rptr = p[pivot + 1];
lptr->keys[lptr->n] = key_arr[pivot];
lptr->p[lptr->n + 1] = rptr->p[0];
for (i = 0; i < rptr->n; i++) {
lptr->keys[lptr->n + 1 + i] = rptr->keys[i];
lptr->p[lptr->n + 2 + i] = rptr->p[i + 1];
}
lptr->n = lptr->n + rptr->n + 1;
free(rptr); /*Remove right node*/
for (i = pos + 1; i < n; i++) {
key_arr[i - 1] = key_arr[i];
p[i] = p[i + 1];
}
return --ptr->n >= (ptr == root ? 1 : min) ? Success : LessKeys;
}
void eatline(void) {
char c;
while ((c = getchar()) != '\n');
}
void inorder(node *ptr) {
if (ptr) {
if (ptr->n >= 1) {
inorder(ptr->p[0]);
printf("%d ", ptr->keys[0]);
inorder(ptr->p[1]);
if (ptr->n == 2) {
printf("%d ", ptr->keys[1]);
inorder(ptr->p[2]);
}
}
}
}
```

OUTPUT:



NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
5.Quit
Enter your choice: 4
Btree is :
3 5
1.Insert
2.Delete
3.Search
4.Display
5.Quit
Enter your choice: 2
Enter the key : 5
1.Insert
2.Delete
3.Search
4.Display
5.Quit
Enter your choice: 4
Btree is :
3
1.Insert
2.Delete
3.Search
4.Display
5.Quit
Enter your choice:
```

RESULT:

Thus the C Program for Implementation of B- Tree using Linked List has been executed successfully.

EX. NO. 13	
DATE:	**IMPLEMENTATION OF GRAPH USING ARRAY**

AIM:

To write a c program to implementGraph using Array to print the adjacent Matrix.

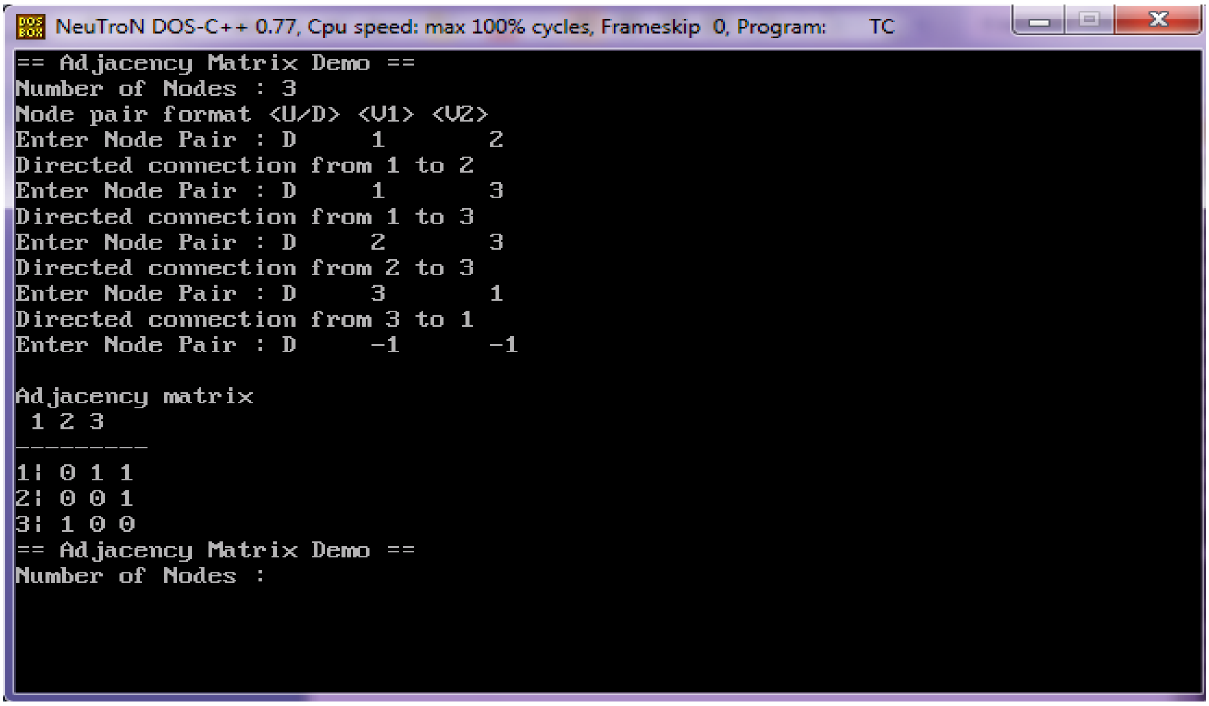
ALGORITHM

Creation of a graph
STEP 1:Get the number of nodes to be inserted for undirected graph
STEP 2:Assign max_edges=n*(n-1)
STEP 3:Read the origin,destination value and weight value from the user
STEP 4:If(origin==0)&&(dest==0) then terminate the process
STEP 5:Otherwise read the weight value of the edge from the user
STEP 6:If(origin>n||origin<=0) then print “Invalid edge”
STEP 7:Otherwise assign adj[origin][destin]=wt;
STEP 8: adi[destin][origin]=wt;

PROGRAM

```
#include <stdio.h>
#include <conio.h>
int main(int argc, char* argv[])
{
int **adj_matrix;
char d;
int r, c, nodes;
printf ("== Adjacency Matrix Demo ==\n");
printf ("Number of Nodes : ");
scanf ("%d", &nodes);
/* Dynamic allocation of matrix array */
adj_matrix = (int **) malloc (sizeof(int **)*nodes);
if(!adj_matrix) {
printf ("Fatal error in memory allocation!");
return -1;
}
for (r = 0; r < nodes; r++) {
adj_matrix[r] = (int *) malloc(sizeof(int)*nodes);
if(!adj_matrix[r]) {
printf ("Fatal error in memory allocation!");
return -1;
}
}
for (r = 0; r < nodes; r++) {
for (c = 0; c < nodes; c++) {
adj_matrix[r][c] = 0;
}
}
r = 0;
c = 0;
printf ("Node pair format <U/D><V1><V2>\n");
do {
printf ("Enter Node Pair : ");
fflush(stdin);
scanf ("%c %d %d", &d, &r, &c);
if (r > 0 && r <= nodes && c > 0 && c <= nodes){
adj_matrix[r - 1][c - 1] = 1;
if(d == 'U' || d == 'u'){
adj_matrix[c - 1][r - 1] = 1;
printf ("Undirected connection between %d to %d\n", r, c);
} else {
printf ("Directed connection from %d to %d\n", r, c);
}
}
}while(r > 0 && c > 0);
printf("\nAdjacency matrix\n");
printf(" ");
for (c = 0; c < nodes; c++) {
printf("%.1d ", c + 1);
}
printf("\n");
for (c = 0; c < nodes; c++) {
printf("---");
}
printf("\n");
for (r = 0; r < nodes; r++) {
printf("%.1d| ", r+1);
for (c = 0; c < nodes; c++) {
printf("%.1d ", adj_matrix[r][c]);
}
printf("\n");
}
return 0;
}
```

OUTPUT:



NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
== Adjacency Matrix Demo ==
Number of Nodes : 3
Node pair format <U/D> <V1> <V2>
Enter Node Pair : D     1       2
Directed connection from 1 to 2
Enter Node Pair : D     1       3
Directed connection from 1 to 3
Enter Node Pair : D     2       3
Directed connection from 2 to 3
Enter Node Pair : D     3       1
Directed connection from 3 to 1
Enter Node Pair : D     -1      -1

Adjacency matrix
 1 2 3
---------
1| 0 1 1
2| 0 0 1
3| 1 0 0
== Adjacency Matrix Demo ==
Number of Nodes :
```

RESULT:

Thus the C Program for Implementation of Graph has been executed successfully.

EX. NO. 14	**IMPLEMENTATION OF SHORTEST PATH ALGORITHM**
DATE:	

AIM:

To write a c program to implementshortest path Algorithm

ALGORITHM

STEP 1:Assign to every node a distance value. Set it to zero for our initial node and to infinity for all other nodes.

STEP 2:Mark all nodes as unvisited. Set initial node as current.

STEP 3:For current node, consider all its unvisited neighbors and calculate their distance (from the initial node). For example, if current node (A) has distance of 6, and an edge connecting it with another node (B) is 2, the distance to B through A will be 6+2=8. If this distance is less than the previously recorded distance (infinity in the beginning, zero for the initial node), overwrite the distance.

STEP 4:When we are done considering all neighbors of the current node, mark it as visited. A visited node will not be checked ever again; its distance recorded now is final and minimal.

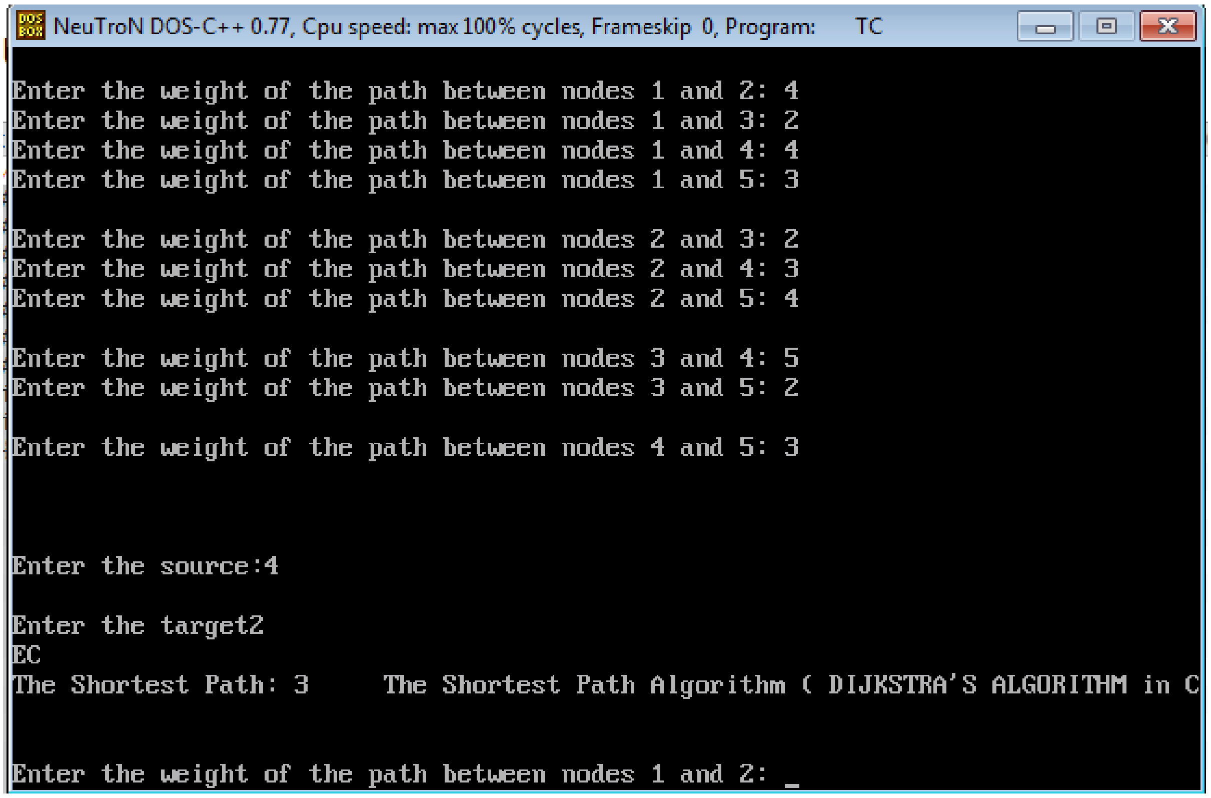
STEP 5:Set the unvisited node with the smallest distance (from the initial node) as the next "current node" and continue from step 3 .

```
        (i)function Dijkstra(Graph, source):
        (ii)for each vertex v in Graph:// Initializations
 (iii)dist[v] := infinity              // Unknown distance function from source to v
        (iv)previous[v] := undefined   // Previous node in optimal path from source
        (v)dist[source] := 0           // Distance from source to source
        (vi)Q := the set of all nodes in Graph
        // All nodes in the graph are unoptimized - thus are in Q
        (vii)while Q is not empty:     // The main loop
        (viii)u := vertex in Q with smallest dist[]
        (ix)if dist[u] = infinity:
        (x) break                      // all remaining vertices are inaccessible
        (xi) remove u from Q
        (xii)for each neighbor v of u:  // where v has not yet been removed from
                alt := dist[u] + dist_between(u, v)
(xiii) if alt < dist[v]:            // Relax (u,v,a)
                dist[v] := alt
        (xiv) previous[v] := u
        (xv)return previous []
```

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#include<string.h>
#include<math.h>
#define IN 99
#define N 6
int dijkstra(int cost[][N], int source, int target);
int main()
{
int cost[N][N],i,j,w,ch,co;
int source, target,x,y;
printf("\t The Shortest Path Algorithm ( DIJKSTRA'S ALGORITHM in C \n\n");
for(i=1;i< N;i++)
for(j=1;j< N;j++)
cost[i][j] = IN;
for(x=1;x< N;x++)
{
for(y=x+1;y< N;y++)
{
printf("Enter the weight of the path between nodes %d and %d: ",x,y);
scanf("%d",&w);
cost [x][y] = cost[y][x] = w;
}
printf("\n");
}
printf("\nEnter the source:");
scanf("%d", &source);
printf("\nEnter the target");
scanf("%d", &target);
co = dijsktra(cost,source,target);
printf("\nThe Shortest Path: %d",co);
}
int dijsktra(int cost[][N],int source,int target)
{
int dist[N],prev[N],selected[N]={0},i,m,min,start,d,j;
char path[N];
for(i=1;i< N;i++)
{
dist[i] = IN;
prev[i] = -1;
}
start = source;
selected[start]=1;
dist[start] = 0;
while(selected[target] ==0)
{
min = IN;
m = 0;
for(i=1;i< N;i++)
{
d = dist[start] +cost[start][i];
if(d< dist[i]&&selected[i]==0)
{
dist[i] = d;
prev[i] = start;
}
if(min>dist[i] && selected[i]==0)
{
min = dist[i];
m = i;
}
}
start = m;
selected[start] = 1;
}
start = target;
j = 0;
while(start != -1)
{
path[j++] = start+65;
start = prev[start];
}
path[j]='\0';
strrev(path);
printf("%s", path);
return dist[target];
}
```

OUTPUT:



NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the weight of the path between nodes 1 and 2: 4
Enter the weight of the path between nodes 1 and 3: 2
Enter the weight of the path between nodes 1 and 4: 4
Enter the weight of the path between nodes 1 and 5: 3

Enter the weight of the path between nodes 2 and 3: 2
Enter the weight of the path between nodes 2 and 4: 3
Enter the weight of the path between nodes 2 and 5: 4

Enter the weight of the path between nodes 3 and 4: 5
Enter the weight of the path between nodes 3 and 5: 2

Enter the weight of the path between nodes 4 and 5: 3



Enter the source:4

Enter the target2
EC
The Shortest Path: 3     The Shortest Path Algorithm ( DIJKSTRA'S ALGORITHM in C


Enter the weight of the path between nodes 1 and 2: _
```

RESULT:

Thus the C Program for Implementation of Shortest Path Algorithm Graph has been executed successfully.

EX. NO. 15 A	IMPLEMENTATION OF MINIMAL SPANNING TREE PRIMS ALGORITHM
DATE:	

AIM:

To write a c program to implementminimal spanning tree using Prims Algorithm

ALGORITHM

```
Step1: Start the program
    Step2: Basic operation performed is
            a) Creation of graph
            b) Creation of a minimum spanning tree
            c) Display the minimum spanning tree
    Step3: Creation of a graph
            a) Get the number of nodes to be inserted for undirected graph
            b) Assign max_edges=n*(n-1)
            c) Read the origin,destination value and weight value from the user
            d) If(origin==0)&&(dest==0) then terminate the process
            e) Otherwise read the weight value of the edge from the user
            f) If(origin>n||origin<=0) then print “Invalid edge”
            g) Otherwise assign adj[origin][destin]=wt;
                                adi[destin][origin]=wt;
    Step4: Creation of a minimum spanning tree
            a) The maketree() is used to create a minimum spanning tree of  graph
            b) Assign initial wt=0
            c) Initially assign state[i].predecesor=0
                          state[i].dist=infinity;
                              state[i].status=TEMP;
            d) Change the predecessor,dist,status value frequently based on
               prim’s procedure
    Step5: Display the minimum spanning tree
    Step6: Stop the program
 U = U ∪ {v}
```

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#define infinity 9999
#define MAX 20
int G[MAX][MAX],spanning[MAX][MAX],n;
int prims();
int main()
{
int i,j,total_cost;
```

```
printf("Enter no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
total_cost=prims();
printf("\nspanning tree matrix:\n");
for(i=0;i<n;i++)
{
printf("\n");
for(j=0;j<n;j++)
printf("%d\t",spanning[i][j]);
}
printf("\n\nTotal cost of spanning tree=%d",total_cost);
return 0;
}
int prims()
{
int cost[MAX][MAX];
int u,v,min_distance,distance[MAX],from[MAX];
int visited[MAX],no_of_edges,i,min_cost,j;
//create cost[][] matrix,spanning[][]
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
if(G[i][j]==0)
cost[i][j]=infinity;
else
cost[i][j]=G[i][j];
spanning[i][j]=0;
}
//initialise visited[],distance[] and from[]
distance[0]=0;
visited[0]=1;
for(i=1;i<n;i++)
{
distance[i]=cost[0][i];
from[i]=0;
visited[i]=0;
}
min_cost=0;             //cost of spanning tree
no_of_edges=n-1;              //no. of edges to be added
while(no_of_edges>0)
{
//find the vertex at minimum distance from the tree
min_distance=infinity;
for(i=1;i<n;i++)
if(visited[i]==0&&distance[i]<min_distance)
{
v=i;
min_distance=distance[i];
}
u=from[v];
//insert the edge in spanning tree
spanning[u][v]=distance[v];
spanning[v][u]=distance[v];
no_of_edges--;
visited[v]=1;
//updated the distance[] array
for(i=1;i<n;i++)
if(visited[i]==0&&cost[i][v]<distance[i])
{
distance[i]=cost[i][v];
from[i]=v;
}
min_cost=min_cost+cost[u][v];
}
return(min_cost);
}
```

OUTPUT:

```
Enter no. of vertices: 6
Enter the adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0
Spanning tree matrix:
0 3 1 0 0 0
3 0 0 0 3 0
1 0 0 0 0 4
0 0 0 0 0 2
0 3 0 0 0 0
0 0 4 2 0 0
Total cost of spanning tree=13
```

RESULT:

Thus the c program to implement minimal spanning tree using Prims Algorithm has been executed successfully

EX. NO. 15 B	**IMPLEMENTATION OF MINIMAL SPANNING TREE KRUSKALS ALGORITHM**
DATE:	

AIM:

To write a c program to implement minimal spanning tree

ALGORITHM

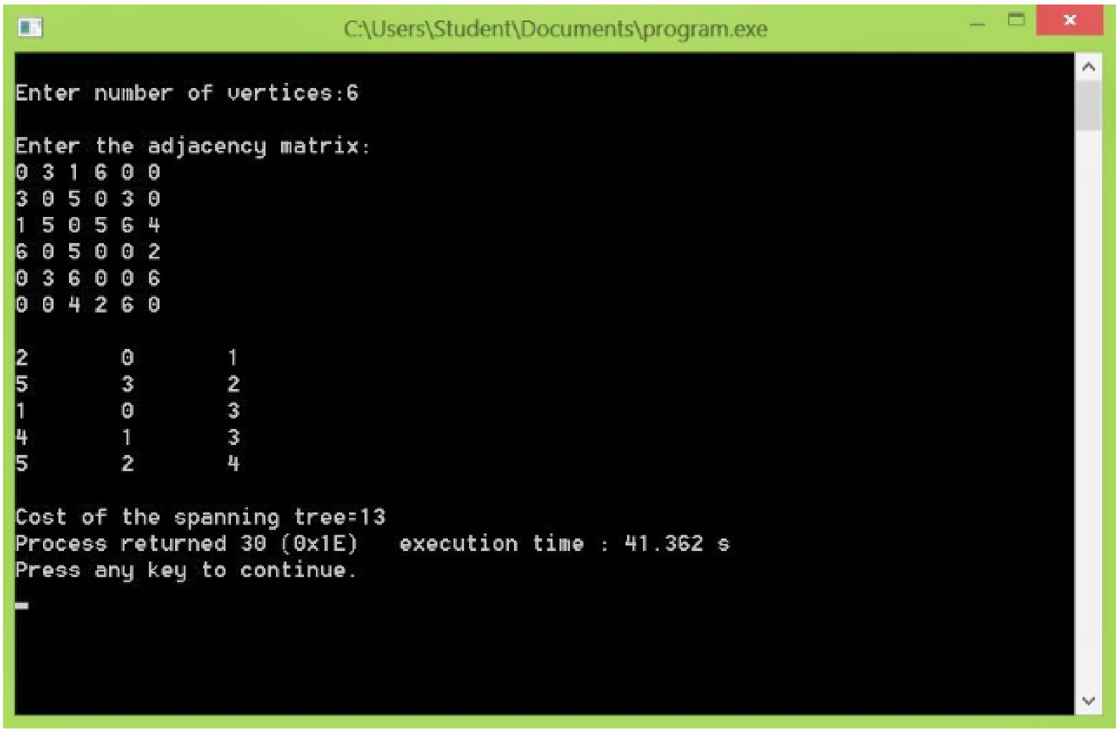
```
KRUSKAL(G):
A = ∅
For each vertex v ∈ G.V:
    MAKE-SET(v)
For each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v):
    if FIND-SET(u) ≠ FIND-SET(v):
    A = A ∪ {(u, v)}
    UNION(u, v)
return A
```

PROGRAM

```
#include<stdio.h>
 #define MAX 30
 typedef struct edge
{
int u,v,w;
}edge;
 typedef struct edgelist
{
edge data[MAX];
int n;
}edgelist;
edgelist elist;
int G[MAX][MAX],n;
edgelist spanlist;
void kruskal();
int find(int belongs[],int vertexno);
void union1(int belongs[],int c1,int c2);
void sort();
void print();
void main()
{
int i,j,total_cost;
printf("\nEnter number of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
kruskal();
print();
}
void kruskal()
{
int belongs[MAX],i,j,cno1,cno2;
elist.n=0;
for(i=1;i<n;i++)
for(j=0;j<i;j++)
{
if(G[i][j]!=0)
{
elist.data[elist.n].u=i;
elist.data[elist.n].v=j;
elist.data[elist.n].w=G[i][j];
elist.n++;
}
}
sort();
for(i=0;i<n;i++)
belongs[i]=i;
spanlist.n=0;
for(i=0;i<elist.n;i++)
{
cno1=find(belongs,elist.data[i].u);
cno2=find(belongs,elist.data[i].v);
if(cno1!=cno2)
{
spanlist.data[spanlist.n]=elist.data[i];
spanlist.n=spanlist.n+1;
union1(belongs,cno1,cno2);
}
}
}
int find(int belongs[],int vertexno)
{
return(belongs[vertexno]);
}
void union1(int belongs[],int c1,int c2)
{
int i;
for(i=0;i<n;i++)
if(belongs[i]==c2)
belongs[i]=c1;
}
void sort()
```

```
{
int i,j;
edge temp;
for(i=1;i<elist.n;i++)
for(j=0;j<elist.n-1;j++)
if(elist.data[j].w>elist.data[j+1].w)
{
temp=elist.data[j];
elist.data[j]=elist.data[j+1];
elist.data[j+1]=temp;
}
}
void print()
{
int i,cost=0;
for(i=0;i<spanlist.n;i++)
{
printf("\n%d\t%d\t%d",spanlist.data[i].u,spanlist.data[i].v,spanlist.data[i].w);
cost=cost+spanlist.data[i].w;
}
printf("\n\nCost of the spanning tree=%d",cost);
}
```

OUTPUT:



```
Enter number of vertices:6

Enter the adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0

2       0       1
5       3       2
1       0       3
4       1       3
5       2       4

Cost of the spanning tree=13
Process returned 30 (0x1E)   execution time : 41.362 s
Press any key to continue.
```

RESULT:

Thus the c program to implement minimal spanning tree using Kruskals Algorithm has been executed successfully.