DATA STRUCTURES

Computer Science & Engineering

B.Tech – 2nd Year 1st Semester

- Study Materials
- Lecture PPTs
- Video Lectures
- Lab Manuals

- Seminar Topics
- Seminar PPTs
- Projects
- Documentation

LAB MANUALS

Author : Rajinikanth B | Regulation: R13 | Year 2016

List of Programs

Week1:

Write a C program that uses functions to perform the following:

a) Create a singly linked list of integers.
b) Delete a given integer from the above linked list.
c) Display the contents of the above list after deletion.

Week2:

Write a C program that uses functions to perform the following:

a) Create a doubly linked list of integers.
b) Delete a given integer from the above doubly linked list.
c) Display the contents of the above list after deletion.

Week3:

Write a C program that uses stack operations to convert a given infix expression into its postfix Equivalent, Implement the stack using an array.

Week 4:

Write C programs to implement a double ended queue ADT using i)array and ii)doubly linked list respectively.

Week 5:

Write a C program that uses functions to perform the following:

a) Create a binary search tree of characters

b) Traverse the above Binary search tree recursively in Postorder.

Week 6:

Write a C program that uses functions to perform the following:

a) Create a binary search tree of integers.
b) Traverse the above Binary search tree non recursively in inorder.

Week 7:

Write C programs for implementing the following sorting methods to arrange a list of integers in Ascending order :

a) Insertion sort
b) Merge sort

Week 8:

Write C programs for implementing the following sorting methods to arrange a list of integers in ascending order:

a) Quick sort
b) Selection sort

Week 9:

i) Write a C program to perform the following operation:

a)Insertion into a B-tree.

ii) Write a C program for implementing Heap sort algorithm for sorting a given list of integers in ascending order.

Week 10:

Write a C program to implement all the functions of a dictionary (ADT) using hashing.

Week 11:

Write a C program for implementing Knuth-Morris- Pratt pattern matching algorithm.

Week 12:

Write C programs for implementing the following graph traversal algorithms:

a)Depth first traversal
b)Breadth first traversal

Week1:

Write a C program that uses functions to perform the following:

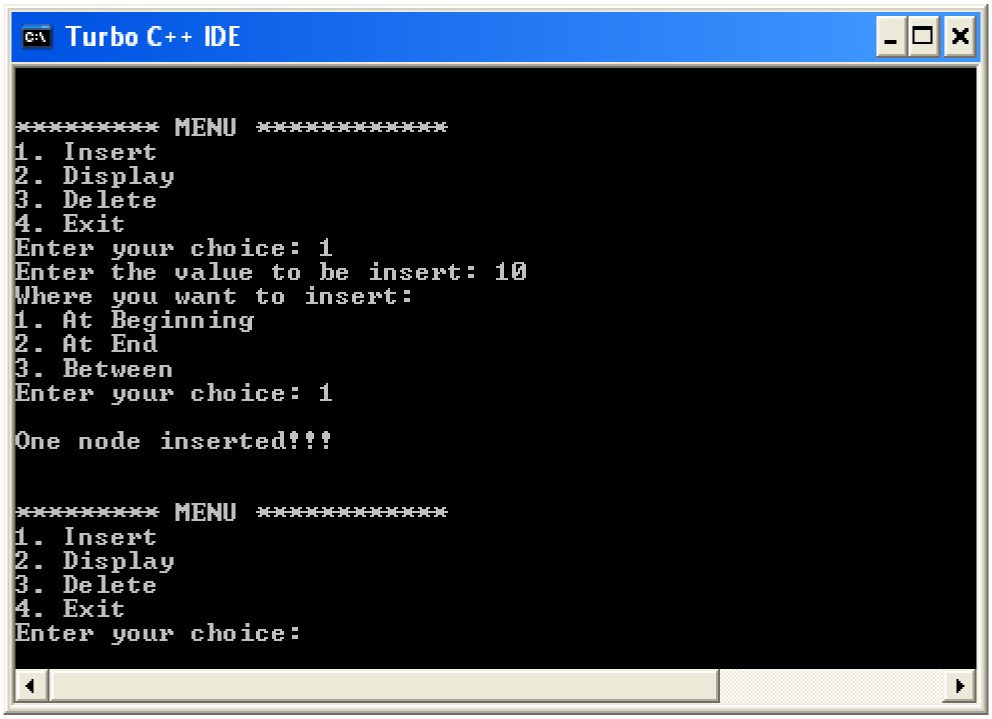
a) Create a singly linked list of integers.
b) Delete a given integer from the above linked list.
c) Display the contents of the above list after deletion.

Solution:

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

void insertAtBeginning(int);
void insertAtEnd(int);
void insertBetween(int,int,int);
void display();
void removeBeginning();
void removeEnd();
void removeSpecific(int);

struct Node{
   int data;
   struct Node *next;
}*head = NULL;

void main(){
   int choice,value,choice1,loc1,loc2;
   clrscr();
   while(1){
   mainMenu: printf("\n\n********* MENU ************\n1. Insert\n2. Display\n3. Delete\n4. Exit\nEnter your choice: ");
   scanf("%d",&choice);
   switch(choice)    {
      case 1:    printf("Enter the value to be insert: ");
                 scanf("%d",&value);
                 while(1){
                 printf("Where you want to insert: \n1. At Beginning\n2. At End\n3. Between\nEnter your choice: ");
                 scanf("%d",&choice1);
                 switch(choice1)   {
                    case 1:       insertAtBeginning(value);
                                  break;
                    case 2:       insertAtEnd(value);
                                  break;
                    case 3:      printf("Enter the two values where you wanto insert: ");
                                  scanf("%d%d",&loc1,&loc2);
                                  insertBetween(value,loc1,loc2);
                                  break;
                    default:      printf("\nWrong Input!! Try again!!!\n\n");
                                  goto mainMenu;
                 }
                 goto subMenuEnd;
                 }
                 subMenuEnd:
                 break;


      case 2:    display();
                 break;
      case 3:    printf("How do you want to Delete: \n1. From Beginning\n2. From End\n3. Spesific\nEnter your choice: ");
                 scanf("%d",&choice1);
                 switch(choice1)   {
                    case 1:      removeBeginning();
                                 break;
                    case 2:      removeEnd(value);
                                 break;
                    case 3:     printf("Enter the value which you wanto delete: ");
                                 scanf("%d",&loc2);
                                 removeSpecific(loc2);
                                 break;
                    default:     printf("\nWrong Input!! Try again!!!\n\n");
                                 goto mainMenu;
                 }
                 break;
      case 4:    exit(0);
      default: printf("\nWrong input!!! Try again!!\n\n");
   }
   }
}

void insertAtBeginning(int value){
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   if(head == NULL)    {
      newNode->next = NULL;
      head = newNode;
   }
   else    {
      newNode->next = head;
      head = newNode;
   }
   printf("\nOne node inserted!!!\n");
}
void insertAtEnd(int value){
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = NULL;
   if(head == NULL)
        head = newNode;
   else    {
      struct Node *temp = head;
      while(temp->next != NULL)
        temp = temp->next;
      temp->next = newNode;
   }
   printf("\nOne node inserted!!!\n");
}
void insertBetween(int value, int loc1, int loc2){
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   if(head == NULL)    {
      newNode->next = NULL;
      head = newNode;
   }
   else   {
      struct Node *temp = head;
      while(temp->data != loc1 && temp->data != loc2)
        temp = temp->next;
      newNode->next = temp->next;
      temp->next = newNode;
   }
   printf("\nOne node inserted!!!\n");
}

void removeBeginning(){
   if(head == NULL)
        printf("\n\nList is Empty!!!");
   else   {
      struct Node *temp = head;
      if(head->next == NULL)      {
         head = NULL;
         free(temp);
      }
      else  {
        head = temp->next;
        free(temp);
        printf("\nOne node deleted!!!\n\n");
      }
   }
}
void removeEnd(){
   if(head == NULL) {
      printf("\nList is Empty!!!\n");
   }
   else {
      struct Node *temp1 = head,*temp2;
      if(head->next == NULL)
        head = NULL;
      else  {
         while(temp1->next != NULL){
            temp2 = temp1;
            temp1 = temp1->next;
         }
         temp2->next = NULL;
      }
      free(temp1);
      printf("\nOne node deleted!!!\n\n");
   }
}
void removeSpecific(int delValue){
   struct Node *temp1 = head, *temp2;
   while(temp1->data != delValue) {
     if(temp1 -> next == NULL){
        printf("\nGiven node not found in the list!!!");
        goto functionEnd;
     }
     temp2 = temp1;
     temp1 = temp1 -> next;
   }
   temp2 -> next = temp1 -> next;
   free(temp1);
   printf("\nOne node deleted!!!\n\n");
   functionEnd:
}
void display(){
   if(head == NULL){
      printf("\nList is Empty\n");
   }
   else {
      struct Node *temp = head;
      printf("\n\nList elements are - \n");
```

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

Output:



```
********** MENU *************
1. Insert
2. Display
3. Delete
4. Exit
Enter your choice: 1
Enter the value to be insert: 10
Where you want to insert:
1. At Beginning
2. At End
3. Between
Enter your choice: 1

One node inserted!!!


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

Week2:

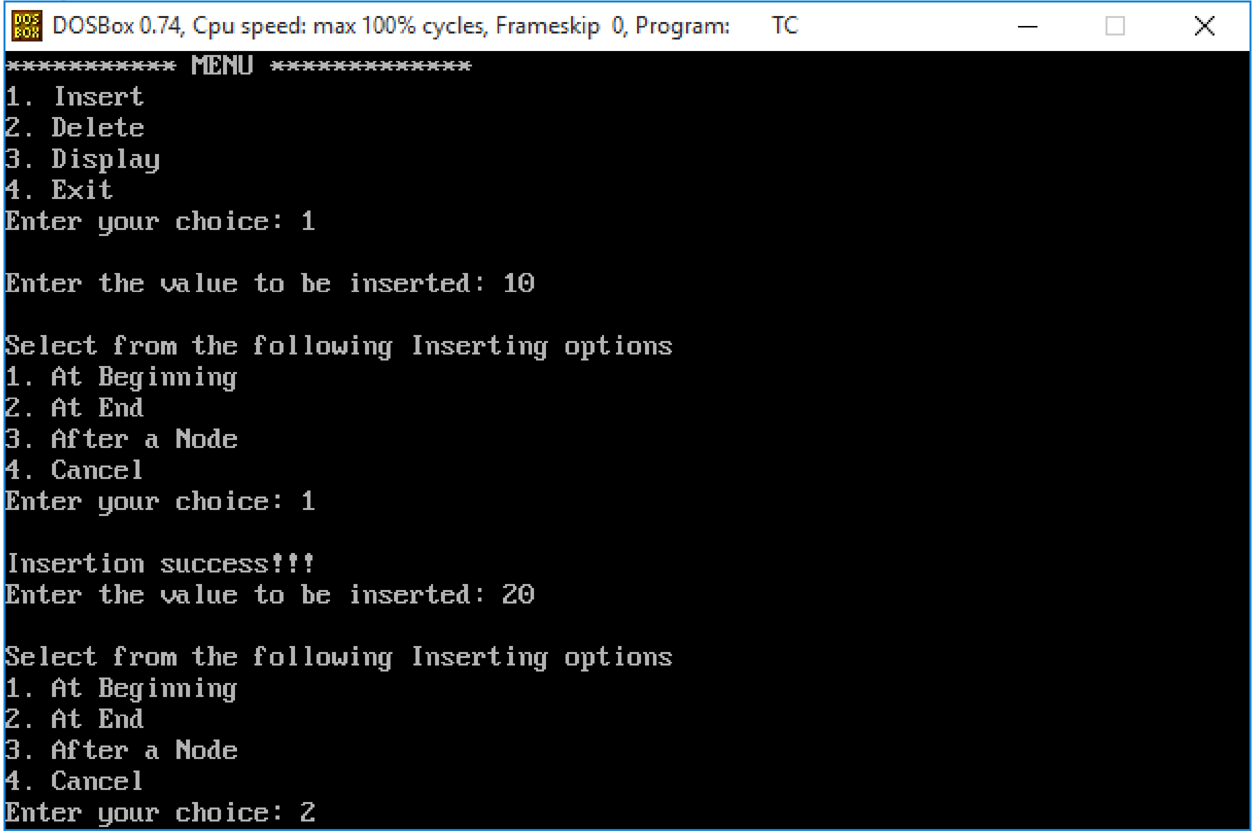
Write a C program that uses functions to perform the following:

a) Create a doubly linked list of integers.
b) Delete a given integer from the above doubly linked list.
c) Display the contents of the above list after deletion.

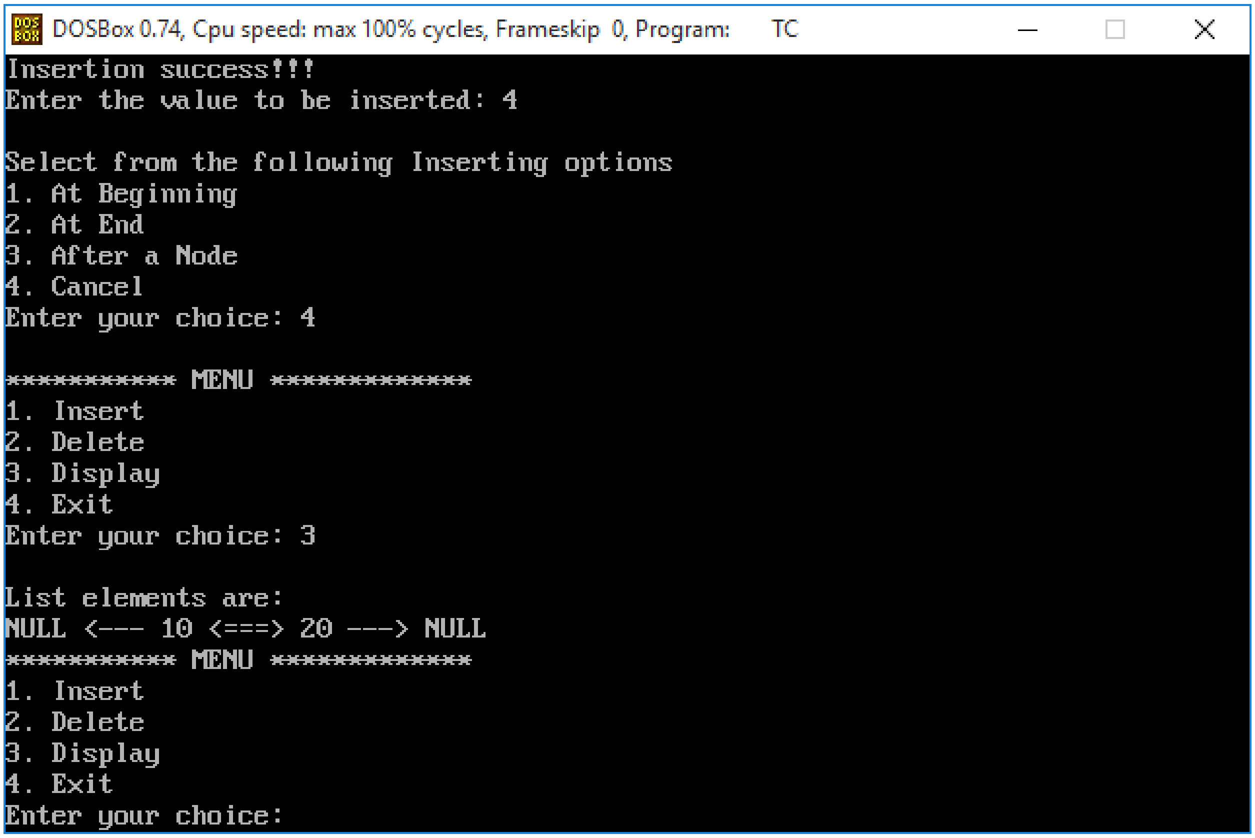
Solution:

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

void insertAtBeginning(int);
void insertAtEnd(int);
void insertAfter(int,int);
void deleteBeginning();
void deleteEnd();
void deleteSpecific(int);
void display();

struct Node
{
   int data;
   struct Node *previous, *next;
}*head = NULL;

void main()
{
   int choice1, choice2, value, location;
   clrscr();
   while(1)
   {
      printf("\n*********** MENU *************\n");
      printf("1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter your choice: ");
      scanf("%d",&choice1);
      switch(choice1)
      {
         case 1: printf("Enter the value to be inserted: ");
                     scanf("%d",&value);
                 while(1)
                 {
                    printf("\nSelect from the following Inserting options\n");
                    printf("1. At Beginning\n2. At End\n3. After a Node\n4.
Cancel\nEnter your choice: ");
                    scanf("%d",&choice2);
                    switch(choice2)
                    {
                       case 1:   insertAtBeginning(value);
                                 break;
                       case 2:   insertAtEnd(value);
                                 break;
                       case 3:   printf("Enter the location after which you want to
insert: ");
                                 scanf("%d",&location);
                                 insertAfter(value,location);
                                 break;
                       case 4:   goto EndSwitch;
                       default: printf("\nPlease select correct Inserting
option!!!\n");
                    }
                 }
         case 2: while(1)
                 {
                    printf("\nSelect from the following Deleting options\n");
                    printf("1. At Beginning\n2. At End\n3. Specific Node\n4.
Cancel\nEnter your choice: ");
                    scanf("%d",&choice2);
                    switch(choice2)
                    {
                       case 1:    deleteBeginning();
                                  break;
                       case 2:    deleteEnd();
                                  break;
                       case 3:    printf("Enter the Node value to be deleted: ");
                                  scanf("%d",&location);
                           deleteSpecific(location);
                                  break;
                       case 4:    goto EndSwitch;
                       default: printf("\nPlease select correct Deleting
option!!!\n");
                    }
                 }
                 EndSwitch: break;
         case 3: display();
              break;
         case 4: exit(0);
         default: printf("\nPlease select correct option!!!");
      }
   }
}

void insertAtBeginning(int value)
{
    struct Node *newNode;
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode -> data = value;
    newNode -> previous = NULL;
    if(head == NULL)
    {
       newNode -> next = NULL;
       head = newNode;
    }
    else
    {
       newNode -> next = head;
       head = newNode;
    }
    printf("\nInsertion success!!!");
}
void insertAtEnd(int value)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode -> data = value;
   newNode -> next = NULL;
   if(head == NULL)
   {
      newNode -> previous = NULL;
      head = newNode;
   }
   else
   {
      struct Node *temp = head;
      while(temp -> next != NULL)
         temp = temp -> next;
      temp -> next = newNode;
      newNode -> previous = temp;
   }
   printf("\nInsertion success!!!");
}
void insertAfter(int value, int location)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode -> data = value;
   if(head == NULL)
   {
      newNode -> previous = newNode -> next = NULL;
      head = newNode;
   }
   else
   {
      struct Node *temp1 = head, *temp2;
      while(temp1 -> data != location)
      {
         if(temp1 -> next == NULL)
         {
            printf("Given node is not found in the list!!!");
            goto EndFunction;
         }
         else
         {
            temp1 = temp1 -> next;
         }
      }
      temp2 = temp1 -> next;
      temp1 -> next = newNode;
      newNode -> previous = temp1;
      newNode -> next = temp2;
      temp2 -> previous = newNode;
      printf("\nInsertion success!!!");
   }
   EndFunction:
}
void deleteBeginning()
{
   if(head == NULL)
      printf("List is Empty!!! Deletion not possible!!!");
   else
   {
      struct Node *temp = head;
      if(temp -> previous == temp -> next)
      {
         head = NULL;
         free(temp);
      }
      else{
         head = temp -> next;
         head -> previous = NULL;
         free(temp);
      }
      printf("\nDeletion success!!!");
   }
}
void deleteEnd()
{
   if(head == NULL)
      printf("List is Empty!!! Deletion not possible!!!");
   else
   {
      struct Node *temp = head;
      if(temp -> previous == temp -> next)
      {
         head = NULL;
         free(temp);
      }
      else{
         while(temp -> next != NULL)
            temp = temp -> next;
         temp -> previous -> next = NULL;
         free(temp);
      }
      printf("\nDeletion success!!!");
   }
}
void deleteSpecific(int delValue)
{
   if(head == NULL)
      printf("List is Empty!!! Deletion not possible!!!");
   else
   {
      struct Node *temp = head;
      while(temp -> data != delValue)
      {
         if(temp -> next == NULL)
         {
            printf("\nGiven node is not found in the list!!!");
            goto FuctionEnd;
         }
         else
         {
            temp = temp -> next;
         }
      }
      if(temp == head)      {
         head = NULL;
         free(temp);
      }
      else
      {
         temp -> previous -> next = temp -> next;
         free(temp);
      }
      printf("\nDeletion success!!!");
   }
   FuctionEnd:
}
void display(){
   if(head == NULL)
      printf("\nList is Empty!!!");
   else
   {
      struct Node *temp = head;
      printf("\nList elements are: \n");
      printf("NULL <--- ");
      while(temp -> next != NULL)
      {
         printf("%d <===> ",temp -> data);
      }
      printf("%d ---> NULL", temp -> data);
   }
}
```

Output:


DOSBox 0.74, 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 inserted: 10

Select from the following Inserting options
1. At Beginning
2. At End
3. After a Node
4. Cancel
Enter your choice: 1

Insertion success!!!
Enter the value to be inserted: 20

Select from the following Inserting options
1. At Beginning
2. At End
3. After a Node
4. Cancel
Enter your choice: 2
```


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Insertion success!!!
Enter the value to be inserted: 4

Select from the following Inserting options
1. At Beginning
2. At End
3. After a Node
4. Cancel
Enter your choice: 4

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

List elements are:
NULL <--- 10 <===> 20 ---> NULL
************ MENU **************
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
```

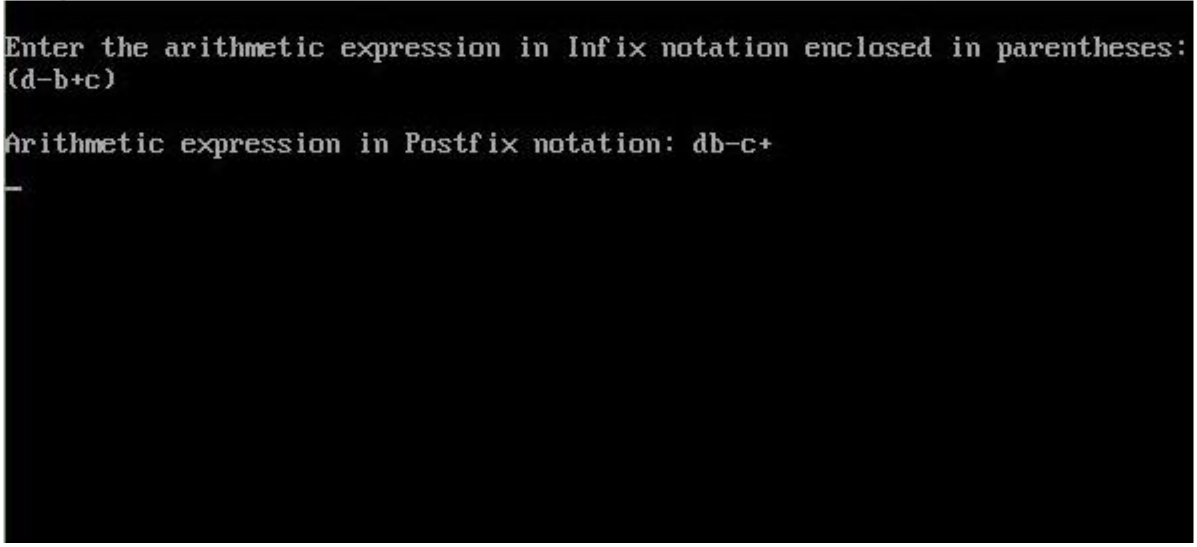
Week3:

Write a C program that uses stack operations to convert a given infix expression into its postfix Equivalent, Implement the stack using an array.

Solution:

```
#include<stdio.h>
#include<conio.h>
#define SIZE 100
int top = -1;
char stack[SIZE];
void push(char item);
char pop();
int is_operator(char symbol);
int precedence(char symbol);
void main(){
 int i;
 int j;
 char infix_exp[SIZE], postfix_exp[SIZE];
 char item;
 char x;
 clrscr();
 printf("\nEnter Infix expression in parentheses: \n");
 gets(infix_exp);
 i=0;
 j=0;
 item=infix_exp[i++];
 while(item != '\0') {
  if(item == '(') {
   push(item);
  }
  else if((item >= 'A'  && item <= 'Z') ||  (item >= 'a' && item <= 'z')){
   postfix_exp[j++] = item;
  }
  else if(is_operator(item) == 1){
   x=pop();
   while(is_operator(x) == 1 && precedence(x)>= precedence(item)){
    postfix_exp[j++] = x;
    x = pop();
   }
   push(x);
   push(item);
  }
  else if(item == ')'){
   x = pop();
   while(x != '('){
    postfix_exp[j++] = x;
    x = pop();
   }
  }
  else{
   printf("\nInvalid Arithmetic Expression.\n");
   getch();
   exit(0);
  }
  item = infix_exp[i++];
 }
  postfix_exp[j++] = '\0';
  printf("\nArithmetic expression in Postfix notation: ");
  puts(postfix_exp);
 getch();
}
void push(char item){
 if(top >= SIZE-1){
  printf("\nStack Overflow. Push not possible.\n");
 }
 else{
  top = top+1;
  stack[top] = item;
 }
}
char pop(){
 char item = NULL;
 if(top <= -1){
  printf("\nStack Underflow. Pop not possible.\n");
 }
 else {
  item = stack[top];
  stack[top] = NULL;
  top = top-1;
 }
 return(item);
}
int is_operator(char symbol){
if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol == '-'){
  return 1;
 }
 else{
  return 0;
 }
}
int precedence(char symbol){
 if(symbol == '^'){
  return(3);
 }
 else if(symbol == '*' || symbol == '/'){
  return(2);
 }
 else if(symbol == '+' || symbol == '-'){
  return(1);
 }
 else {
  return(0);
 }
}
```

Output:



```
Enter the arithmetic expression in Infix notation enclosed in parentheses:
(d-b+c)

Arithmetic expression in Postfix notation: db-c+
_
```

Week 4:

Write C programs to implement a double ended queue ADT using i)array and ii)doubly linked list respectively.

Solution: Double ended queue ADT using array

```
#define MAX 10
int q[MAX],front=0,rear=0;
void add_rear();
void add_front();
void delete_rear();
void delete_front();
void display();
void main() {
    int ch;
    clrscr();
    do {
        printf("\n DQueue Menu\n1. Add at Rear\n2. Add at Front\n3. Delete from Rear\n4. Delete from Front\n5. Display\n6. Exit");
        printf(“\n Enter your choice : ");
        scanf("%d",&ch);
        switch(ch) {
            case 1:
                add_rear();
                printf("\n Queue after insert at rear");
                display();
                break;
            case 2:
                add_front();
                printf("\n Queue after insert at front");
                display();
                break;
            case 3:
                delete_rear();
                printf("\n Queue after delete at rear");
                display();
                break;
            case 4:
                delete_front();
                printf("\n Queue after delete at front");
                display();
                break;
            case 5:
                display();
                break;
            case 6:
                exit(0);
            default: printf("\n Wrong Choice\n");
        }
    } while(ch!=6);
}
```

```
void add_rear() {
    int no;
    printf("\n Enter value to insert : ");
    scanf("%d",&no);
    if(rear==MAX) {
        printf("\n Queue is Overflow");
        return;
    }
    else {
        rear++;
        q[rear]=no;
         if(rear==0)
            rear=1;
        if(front==0)
            front=1;
     }
 }
void add_front() {
    int no;
    printf("\n Enter value to insert:-");
    scanf("%d",&no);
    if(front<=1) {
        printf("\n Cannot add value at front end");
        return;
    }
    else {
        front--;
        q[front]=no;
    }
}
void delete_front() {
    int no;
    if(front==0) {
        printf("\n Queue is Underflow\n");
        return;
    }
    else {
        no=q[front];
        printf("\n Deleted element is %d\n",no);
        if(front==rear) {
            front=0;
            rear=0;
        }
        else {
            front++;
        }
    }
}
```

```
void delete_rear() {
      int no;
      if(rear==0) {
            printf("\n Cannot delete value at rear end\n");
            return;
      else {
            no=q[rear];
            if(front==rear) {
                  front=0;
                  rear=0;
            }
            else {
                  rear--;
                  printf("\n Deleted element is %d\n",no);
            }
      }
}
void display() {
      int i;
      if(front==0) {
            printf("\n Queue is Underflow\n");
            return;
      }
      else {
            printf("\n Output");
            for(i=front;i<=rear;i++) {
                  printf("\n %d",q[i]);
            }
      }
}
```

Output:

Solution: Double ended queue ADT using doubly linked list

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *previous;
    struct node *next;
};
struct node *front, *rear;
int count;
void display();
void insert_begin(int x);
void insert_last(int x);
int delete_begin();
int delete_last();

int main()
{
    int ch, ele;
    printf("\n1. Insert-begin\n2. Insert-last\n3. Delete-begin\n4. Delete-last\n5. Display \n6.exit");
    while(1)
    {
        printf("Enter your choice:");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                printf("Enter value for insertion :");
                scanf("%d",&ele);
                insert_begin(ele);
                break;
            case 2:
                printf(" Enter the value for insertion:");
                scanf("%d",&ele);
                insert_last(ele);
                break;
            case 3:
                ele = delete_begin();
                if(ele!=-1)
                printf("%d is deleted .",ele);
                break;
            case 4:
                ele = delete_last();
                if(ele!=-1)
                printf("%d is deleted .",ele);
                break;
            case 5:
                display();
                break;
            case 6: exit(0);
        }
    }
}

void display()
{
    struct node * ptr;
    ptr = front;
    if(front==NULL || rear==NULL)
    {
        printf("List is empty");
        return;
    }
    while(ptr != NULL)
    {
        printf( "%d -> ",ptr ->data);
        ptr = ptr->next;
    }
    printf("\n");
}
void insert_begin(int x)
{
    struct node *new1;
    new1 = (struct node*)malloc(sizeof(struct node));
    new1 -> data =x;
    new1 ->previous = new1 ->next =NULL;
    if(front == NULL||rear==NULL)
        front = rear = new1;
    else
    {
        new1 ->next = front;
        front ->previous = new1;
        front = new1;
    }
}

void insert_last(int x)
{
    struct node *new1;
    new1 = (struct node*)malloc(sizeof(struct node));
    new1 ->data = x;
    new1 -> previous = new1 ->next = NULL;
    if (front == NULL||rear==NULL)
        front = rear = new1;
    else
    {
        rear ->next = new1;
        new1 ->previous = rear;
        rear = new1;
    }
}
int delete_begin()
{
    int x;
    struct node *temp;
    if (front == NULL || rear==NULL)
    {
        printf( " LIST IS EMPTY ");
        return -1;
    }
    else
    {
        temp = front;
        x= temp->data;
        if(front==rear)
        {
            front=NULL;
            rear=NULL;
        }
        else
        {
            front = front->next;
            front->previous = NULL;
        }
        count --;
        free(temp);
        return x;
    }
}
int delete_last( ) {
    int x;
    struct node *temp;
    if(rear == NULL || front==NULL)
    {
        printf( " LIST IS EMPTY ");
        return -1;
    }
    else
    {
        temp = rear;
        if(front==rear)
        {
            front=NULL;
            rear=NULL;
        }
        else
        {
            rear = rear->previous;
            rear -> next = NULL;
        }
        x= temp ->data;
        free(temp);
        count --;
        return x;
    }
}
```

Output:

Week 5:

Write a C program that uses functions to perform the following:

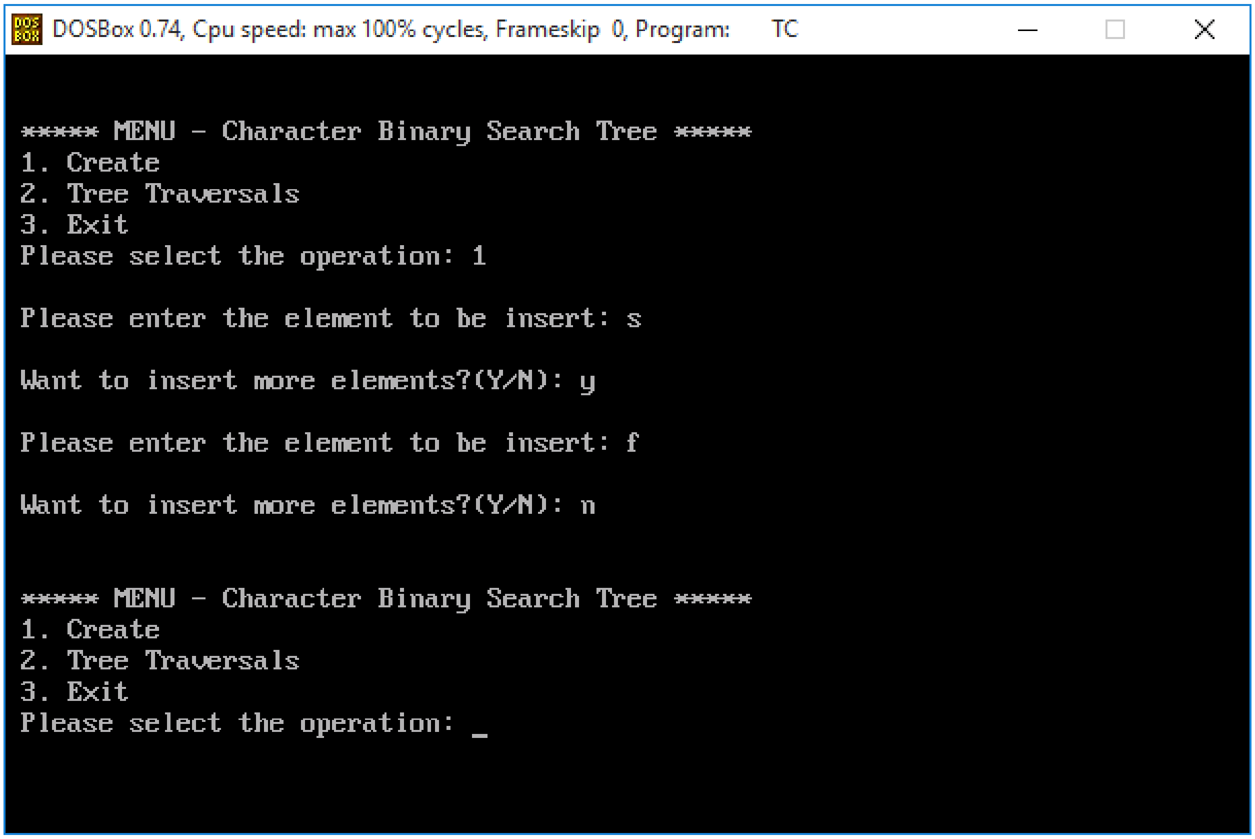
a) Create a binary search tree of characters

b) Traverse the above Binary search tree recursively in Postorder.

Solution:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct BST {
  char d; /*declaring a structure to create a node*/
  struct BST *lc,*rc;
}node;

void insert(node *root,node *nn) {
  int c,d;
  c=nn->d;
  d=root->d;
  if(c<d) {
    if(root->lc==NULL)
      root->lc=nn;
    else
      insert(root->lc,nn);
  }
}

void inorder(node *temp) {
  if(temp!=NULL) {
    inorder(temp->lc);
    printf(" %c",temp->d);
    inorder(temp->rc);
  }
}

void preorder(node *temp) {
  if(temp!=NULL) {
    printf(" %c",temp->d);
    preorder(temp->lc);
    preorder(temp->rc);
  }
}

void postorder(node *temp) {
  if(temp!=NULL) {
    postorder(temp->lc);
    postorder(temp->rc);
    printf(" %c",temp->d);
  }
}

/*main program*/

void main() {
  int choice;
  char ans='N';
  int key;
  node *nn,*root,*parent;
  root=NULL;
  while(1) {
    printf("\n\n ***** MENU - Binary search tree *****");
    printf("\n 1. Create\n 2. Tree Traversals\n 3. Exit");
    printf("\n Please select the operation: ");
    scanf("%d",&choice);
    switch(choice) {
      case 1: do {
                        nn=(node *)malloc(sizeof(node));
                        printf("\n Please enter the element to be insert: ");
                        nn->lc=NULL;
                        nn->rc=NULL;
                        scanf(" %c",&nn->d);
                        if(root==NULL)
                          root=nn;
                        else
                          insert(root,nn);
                        printf("\n Want to insert more elements?(Y/N): ");
                        scanf(" %c",&ans);
                  }while(ans=='y');
                  break;
      case 2: if(root==NULL)
                  printf("\n\n Tree is not created");
                else {
                  printf("\n\n The inorder display  : ");
                  inorder(root);
                  printf("\n\n The preorder display : ");
                  preorder(root);
                  printf("\n\n The postorder display:");
                  postorder(root);
                }
                break;
      case 3: exit(0);
    }
  }
}
```

Output:



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
***** MENU - Character Binary Search Tree *****
1. Create
2. Tree Traversals
3. Exit
Please select the operation: 1

Please enter the element to be insert: s

Want to insert more elements?(Y/N): y

Please enter the element to be insert: f

Want to insert more elements?(Y/N): n


***** MENU - Character Binary Search Tree *****
1. Create
2. Tree Traversals
3. Exit
Please select the operation: _
```

Week 6:

Write a C program that uses functions to perform the following:

a) Create a binary search tree of integers.

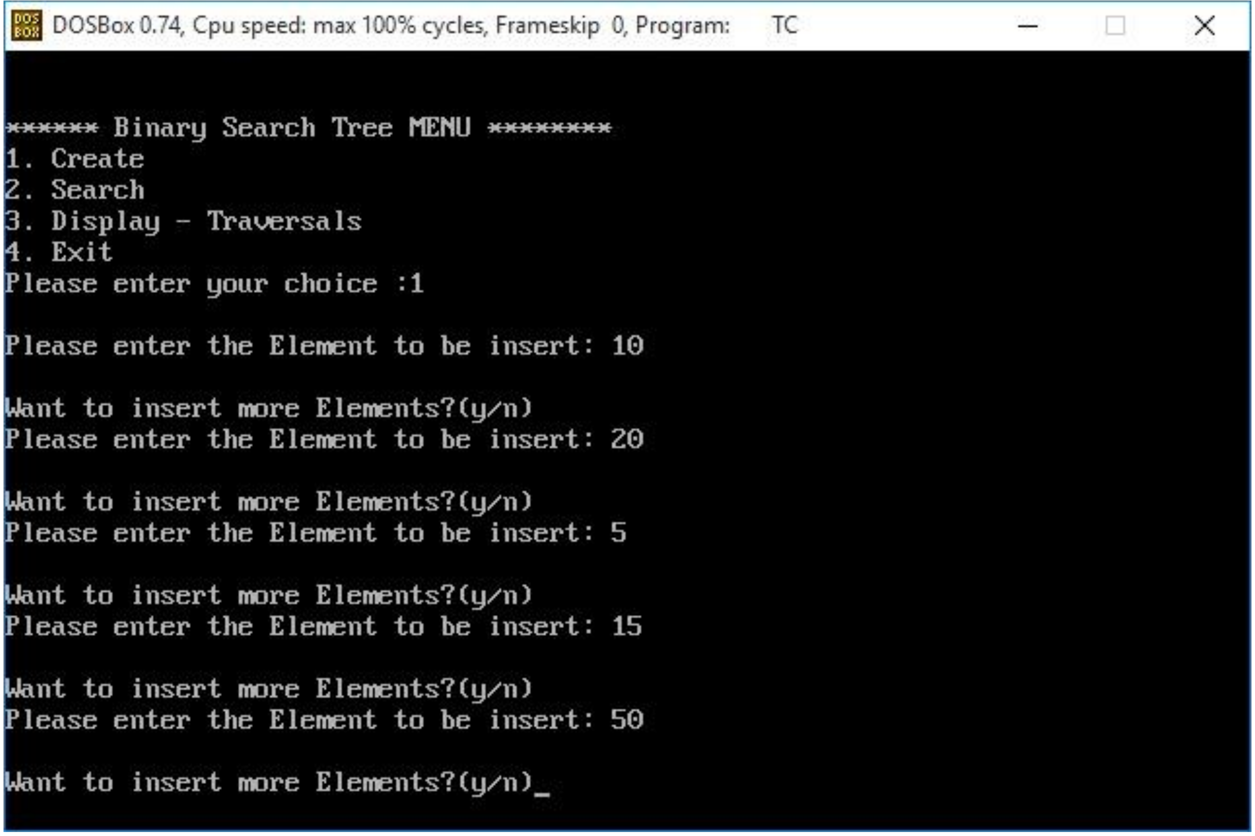
b) Traverse the above Binary search tree non recursively in inorder.

Solution:

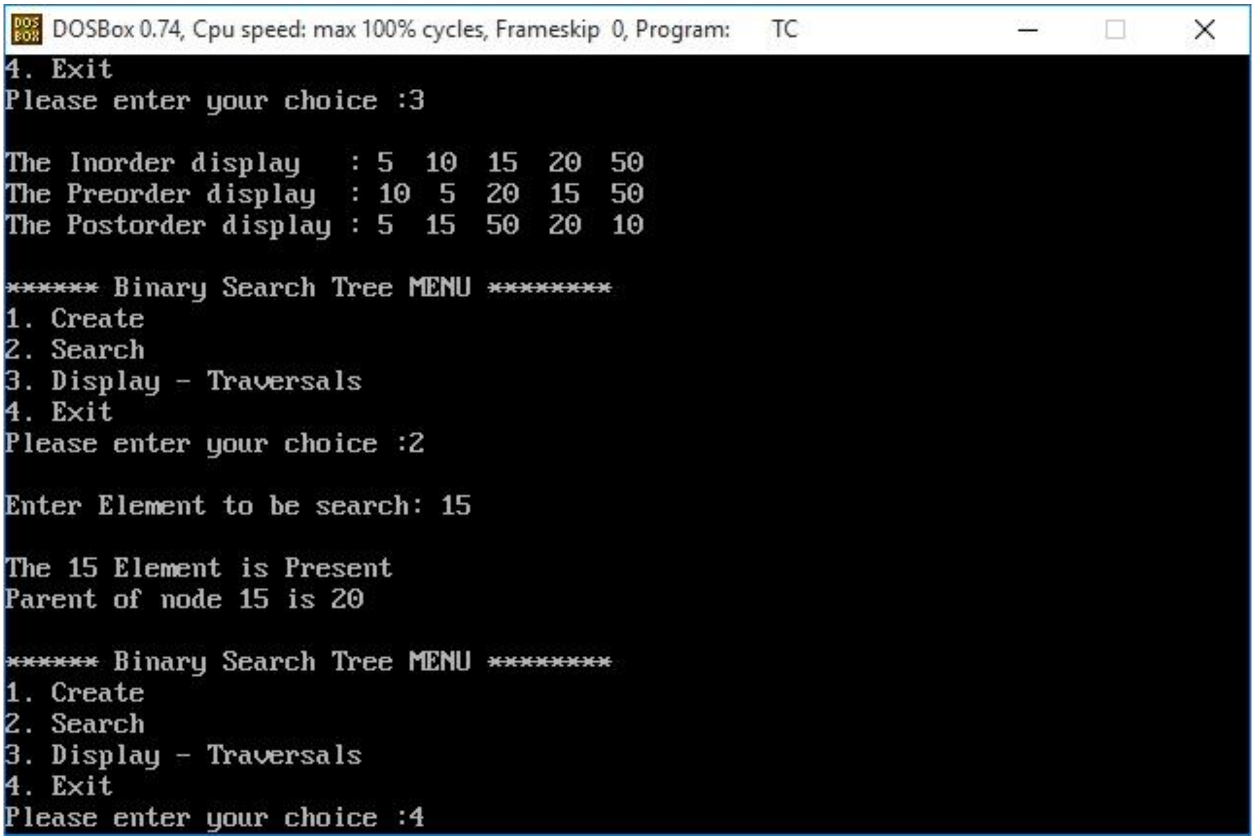
```
# include <stdio.h>
# include <conio.h>
# include <stdlib.h>

typedef struct BST {
  int data;
  struct BST *leftChild, *rightChild;
} node;

void insert(node *, node *);
void inorder(node *);
void preorder(node *);
void postorder(node *);
node *search(node *, int, node **);

void main() {
  int choice;
  char ans = 'N';
  int key;
  node *newNode, *root, *temp, *parent;
  node *getNode();
  root = NULL;
  clrscr();
  while(1){
    printf("\n\n****** Binary Search Tree MENU ********");
    printf("\n1. Create");
    printf("\n2. Search");
    printf("\n3. Display - Traversals");
    printf("\n4. Exit");
    printf("\nPlease enter your choice :");
    scanf("%d", &choice);

    switch (choice) {
    case 1:
        do {
          newNode = getNode();
          printf("\nPlease enter the Element to be insert: ");
          scanf("%d", &newNode->data);

          if (root == NULL) /* Tree is not Created */
            root = newNode;
          else
            insert(root, newNode);

          printf("\nWant to insert more Elements?(y/n)");
          ans = getch();
        } while (ans == 'y');
        break;

    case 2:
        printf("\nEnter Element to be search: ");
        scanf("%d", &key);

        temp = search(root, key, &parent);
        printf("\nParent of node %d is %d", temp->data, parent->data);
        break;

    case 3:
        if (root == NULL)
          printf("\nTree Is Not Created");
        else {
          printf("\nThe Inorder display   : ");
          inorder(root);
          printf("\nThe Preorder display  : ");
          preorder(root);
          printf("\nThe Postorder display : ");
          postorder(root);
        }
        break;
    case 4: exit(0);
    default: printf("\nPlease select correct operations!!!");
    }
  }
}
/* Creating a new Node */
node *getNode() {
  node *temp;
  temp = (node *) malloc(sizeof(node));
  temp->leftChild = NULL;
  temp->rightChild = NULL;
  return temp;
}
/* Inserting new Node into binary search tree */
void insert(node *root, node *newNode) {
  if (newNode->data < root->data) {
    if (root->leftChild == NULL)
        root->leftChild = newNode;
    else
         insert(root->leftChild, newNode);
  }

  if (newNode->data > root->data) {
    if (root->rightChild == NULL)
         root->rightChild = newNode;
    else
         insert(root->rightChild, newNode);
  }
}
/* Searching the node in binary Search Tree */
node *search(node *root, int key, node **parent) {
  node *temp;
  temp = root;
  while (temp != NULL) {
    if (temp->data == key) {
         printf("\nThe %d Element is Present", temp->data);
         return temp;
    }
    *parent = temp;

    if (temp->data > key)
         temp = temp->leftChild;
    else
         temp = temp->rightChild;
  }
  return NULL;
}
/* Inorder traversal display */
void inorder(node *temp) {
  if (temp != NULL) {
    inorder(temp->leftChild);
    printf("%d  ", temp->data);
    inorder(temp->rightChild);
  }
}
/* Preorder traversal display */
void preorder(node *temp) {
  if (temp != NULL) {
    printf("%d  ", temp->data);
    preorder(temp->leftChild);
    preorder(temp->rightChild);
  }
}
```

```
/* Postorder traversal display */
void postorder(node *temp) {
   if (temp != NULL) {
      postorder(temp->leftChild);
      postorder(temp->rightChild);
      printf("%d  ", temp->data);
   }
}
```

Output:


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
****** Binary Search Tree MENU ********
1. Create
2. Search
3. Display - Traversals
4. Exit
Please enter your choice :1

Please enter the Element to be insert: 10

Want to insert more Elements?(y/n)
Please enter the Element to be insert: 20

Want to insert more Elements?(y/n)
Please enter the Element to be insert: 5

Want to insert more Elements?(y/n)
Please enter the Element to be insert: 15

Want to insert more Elements?(y/n)
Please enter the Element to be insert: 50

Want to insert more Elements?(y/n)_
```



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
4. Exit
Please enter your choice :3

The Inorder display   : 5  10  15  20  50
The Preorder display  : 10  5  20  15  50
The Postorder display : 5  15  50  20  10

****** Binary Search Tree MENU ********
1. Create
2. Search
3. Display - Traversals
4. Exit
Please enter your choice :2

Enter Element to be search: 15

The 15 Element is Present
Parent of node 15 is 20

****** Binary Search Tree MENU ********
1. Create
2. Search
3. Display - Traversals
4. Exit
Please enter your choice :4
```

Week 7:

Write C programs for implementing the following sorting methods to arrange a list of integers in Ascending order :

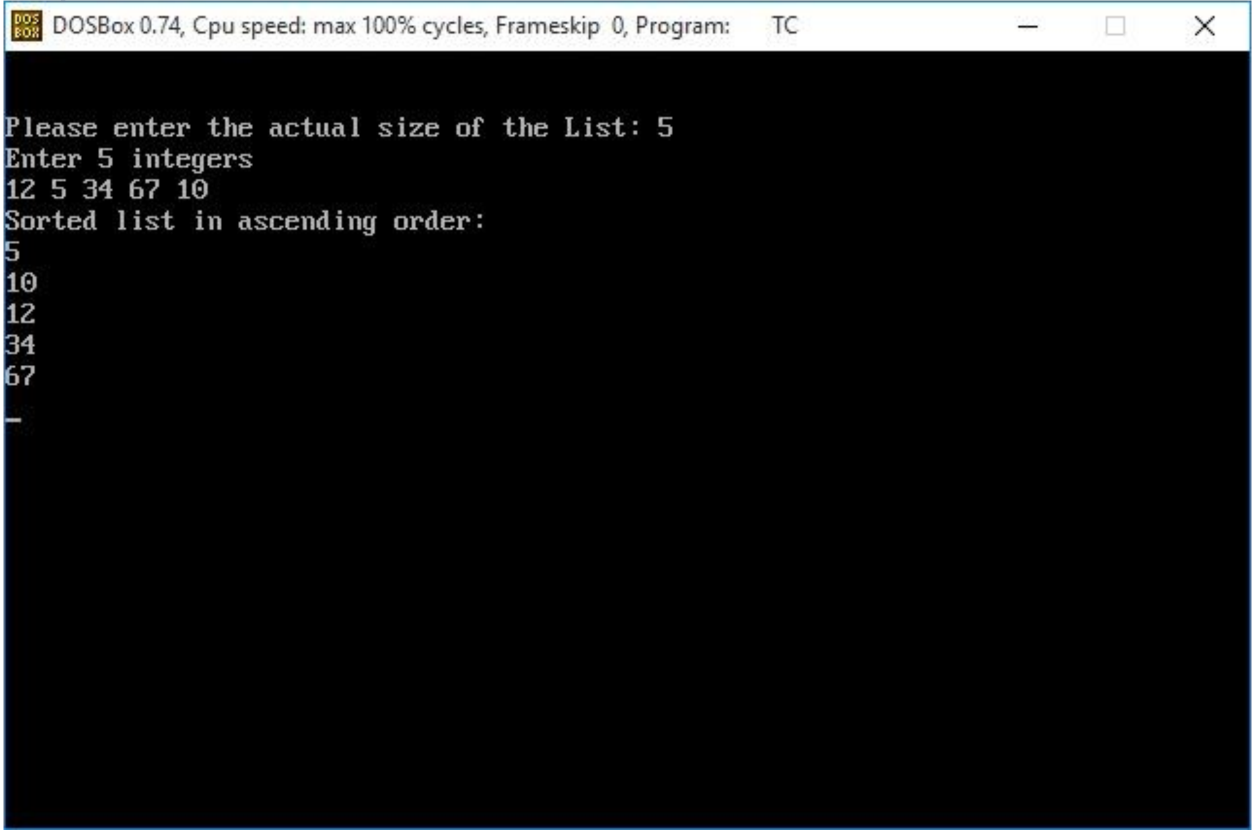
a) Insertion sort
b) Merge sort

Solution:

- Insertion Sort

```
# include <stdio.h>
# include <conio.h>
#define MAXSIZE 100

void main()
{
  int list[MAXSIZE],size, count, i, temp;
  clrscr();
  printf("Please enter the actual size of the List: ");
  scanf("%d", &size);

  printf("Enter %d integers\n", size);

  for (count = 0; count < size; count++) {
    scanf("%d", &list[count]);
  }

  for (count = 1 ; count <= size - 1; count++) {
    i = count;
    while ( i > 0 && list[i] < list[i-1]) {
      temp = list[i];
      list[i]   = list[i-1];
      list[i-1] = temp;
      d--;
    }
  }

  printf("Sorted list in ascending order:\n");

  for (count = 0; count <= size - 1; count++) {
    printf("%d\n", list[count]);
  }

  getch();
}
```

Output:



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

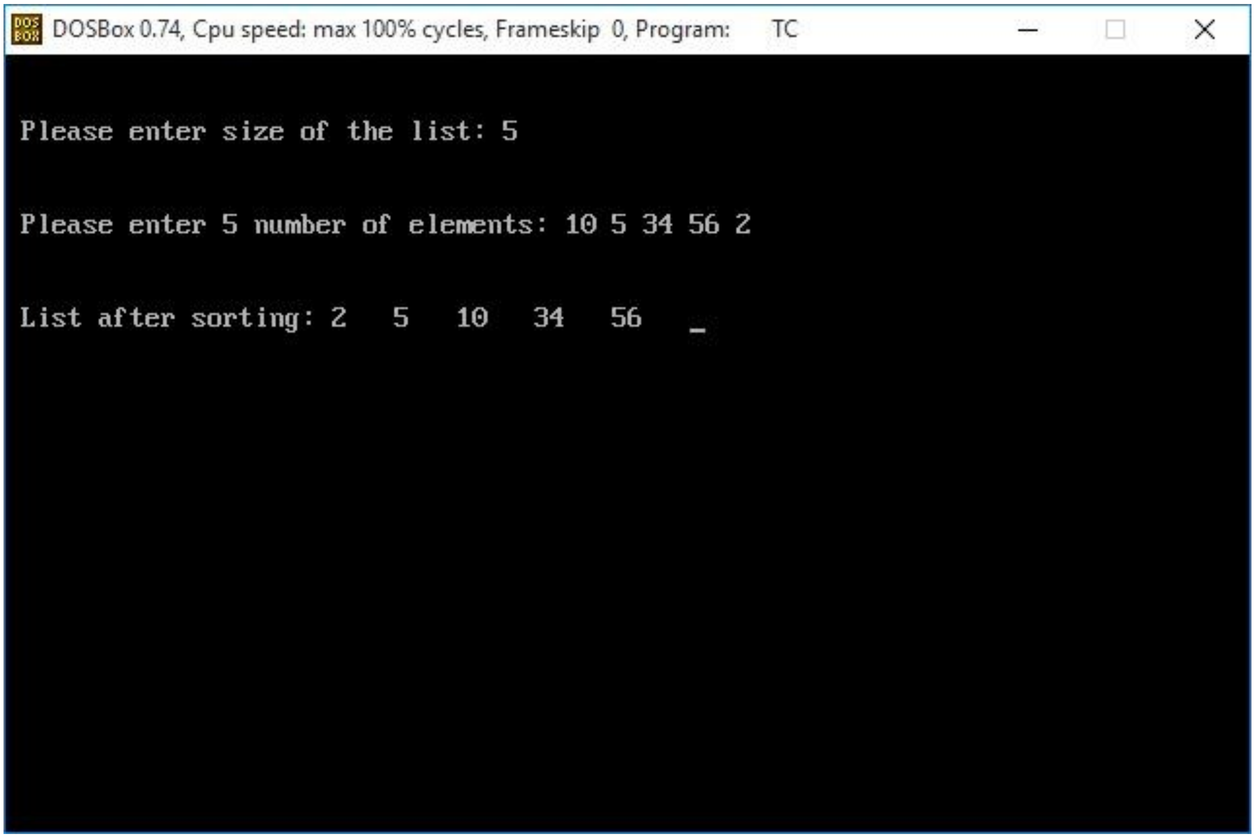
```
Please enter the actual size of the List: 5
Enter 5 integers
12 5 34 67 10
Sorted list in ascending order:
5
10
12
34
67
_
```

- **Merge Sort**

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

#define MAX 100

void mergesort(int[],int,int);
void mergearray(int[],int,int,int);
void main() {
  int list[MAX],size,i;
  clrscr();
  printf("\n\n Please enter size of the list: ");
  scanf("%d",&size);
  printf("\n\n Please enter %d number of elements: ",size);
  for(i=0;i<size;i++)
    scanf("%d",&list[i]);
  mergesort(list,0,size-1);
  printf("\n\n List after sorting: ");
  for(i=0;i<size;i++)
    printf("%d   ",list[i]);
  getch();
}

void mergesort(int list[],int beg,int end) {
  int mid;
  if(beg<end) {
    mid=(beg+end)/2;
    mergesort(list,beg,mid);
    mergesort(list,mid+1,end);
    mergearray(list,beg,mid,end);
  }
}

void mergearray(int list[],int beg,int mid,int end) {
  int i,leftend,num,temp,j,k,subList[MAX];
  for(i=beg;i<=end;i++)
    subList[i]=list[i];
  i=beg;
  j=mid+1;
  k=beg;
  while((i<=mid)&&(j<=end)) {
    if(subList[i]<+subList[j]) {
         list[k]=subList[i];
         i++;
         k++;
    }
```

```
          else {
                 list[k]=subList[j];
                 j++;
                 k++;
          }
       }
       if(i<=mid) {
          while(i<=mid) {
                 list[k]=subList[i];
                 i++;
                 k++;
          }
       }
       else {
          while(j<=end) {
                 list[k]=subList[j];
                 j++;
                 k++;
          }
       }
    }
```

Output:


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Please enter size of the list: 5

Please enter 5 number of elements: 10 5 34 56 2

List after sorting: 2   5   10   34   56   _
```

Week 8:

Write C programs for implementing the following sorting methods to arrange a list of integers in ascending order:

a) Quick sort
b) Selection sort

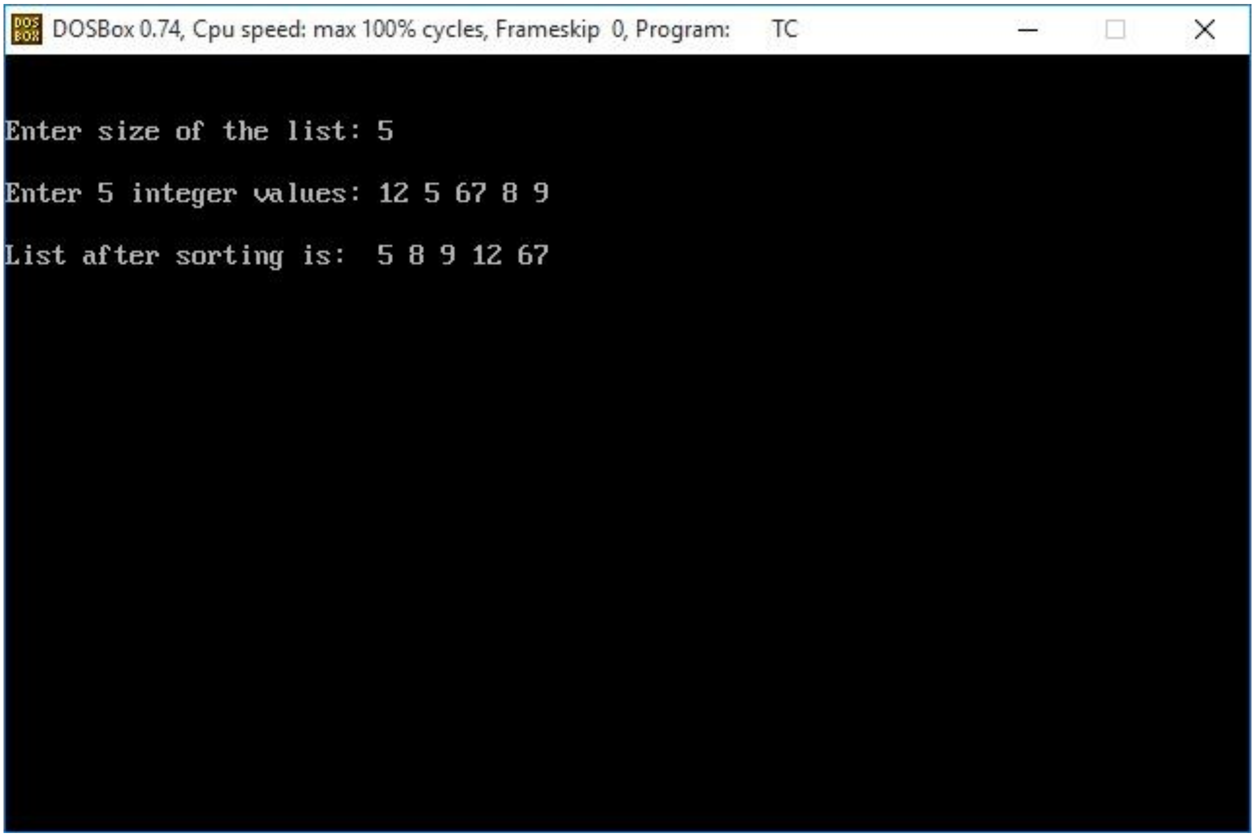
Solution:

- Quick Sort

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

void quickSort(int [10],int,int);

void main(){
  int list[20],size,i;
  clrscr();
  printf("\n\nEnter size of the list: ");
  scanf("%d",&size);

  printf("\nEnter %d integer values: ",size);
  for(i = 0; i < size; i++)
    scanf("%d",&list[i]);

  quickSort(list,0,size-1);

  printf("\nList after sorting is: ");
  for(i = 0; i < size; i++)
    printf(" %d",list[i]);

  getch();
}

void quickSort(int list[10],int first,int last){
    int pivot,i,j,temp;

     if(first < last){
         pivot = first;
         i = first;
         j = last;

         while(i < j){
             while(list[i] <= list[pivot] && i < last)
                 i++;
             while(list[j] > list[pivot])
                 j--;
             if(i < j){
                  temp = list[i];
                  list[i] = list[j];
```

```
                list[j] = temp;
            }
        }

        temp = list[pivot];
        list[pivot] = list[j];
        list[j] = temp;
        quickSort(list,first,j-1);
        quickSort(list,j+1,last);

    }
}
```

Output:



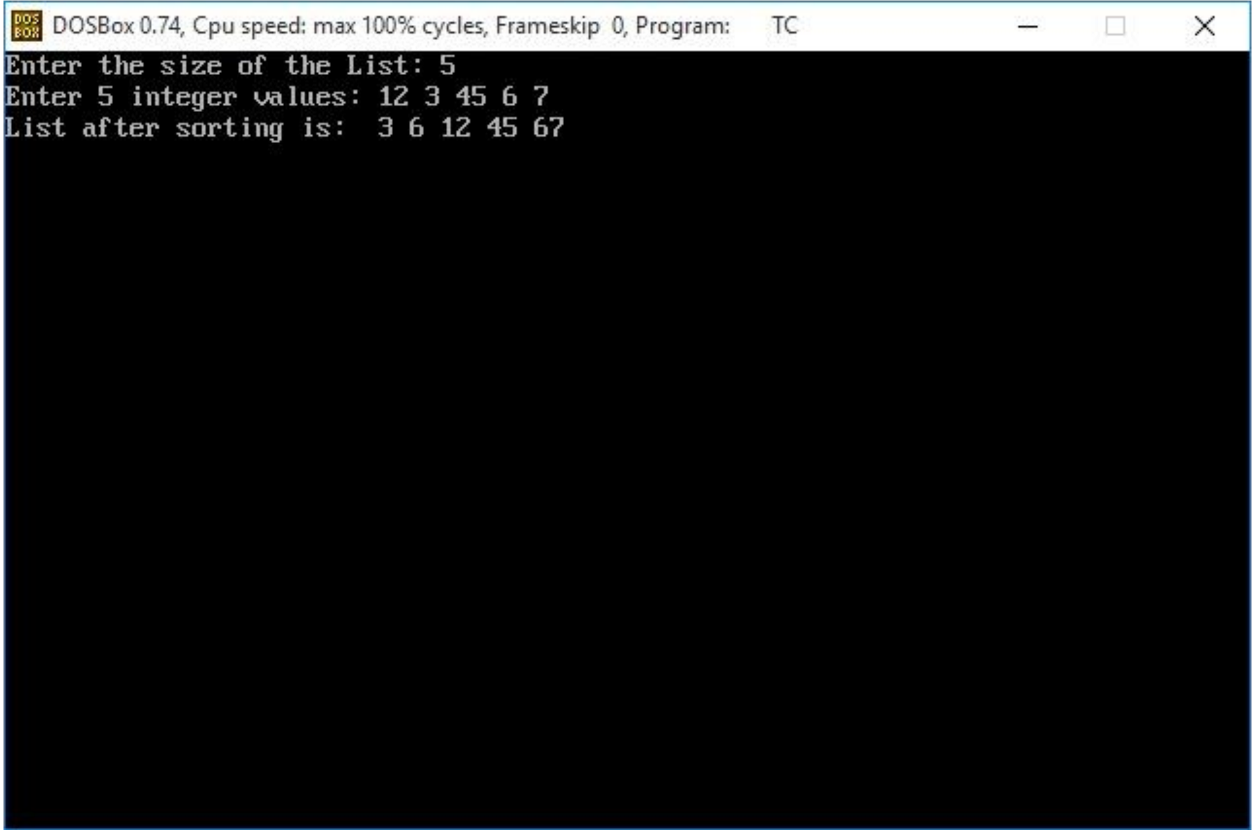
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter size of the list: 5

Enter 5 integer values: 12 5 67 8 9

List after sorting is:  5 8 9 12 67
```

- **Selection Sort**

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

void main(){

  int size,i,j,temp,list[100];
  clrscr();

  printf("Enter the size of the List: ");
  scanf("%d",&size);

  printf("Enter %d integer values: ",size);
  for(i=0; i<size; i++)
      scanf("%d",&list[i]);

  //Selection sort logic

  for(i=0; i<size; i++){
      for(j=i+1; j<size; j++){
           if(list[i] > list[j])
               {
               temp=list[i];
               list[i]=list[j];
               list[j]=temp;
           }
      }
  }

  printf("List after sorting is: ");
  for(i=0; i<size; i++)
      printf(" %d",list[i]);

  getch();
}
```

Output:


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the size of the List: 5
Enter 5 integer values: 12 3 45 6 7
List after sorting is:  3 6 12 45 67
```

Week 9:

i) Write a C program to perform the following operation:
a)Insertion into a B-tree.
ii) Write a C program for implementing Heap sort algorithm for sorting a given list of integers in ascending order.

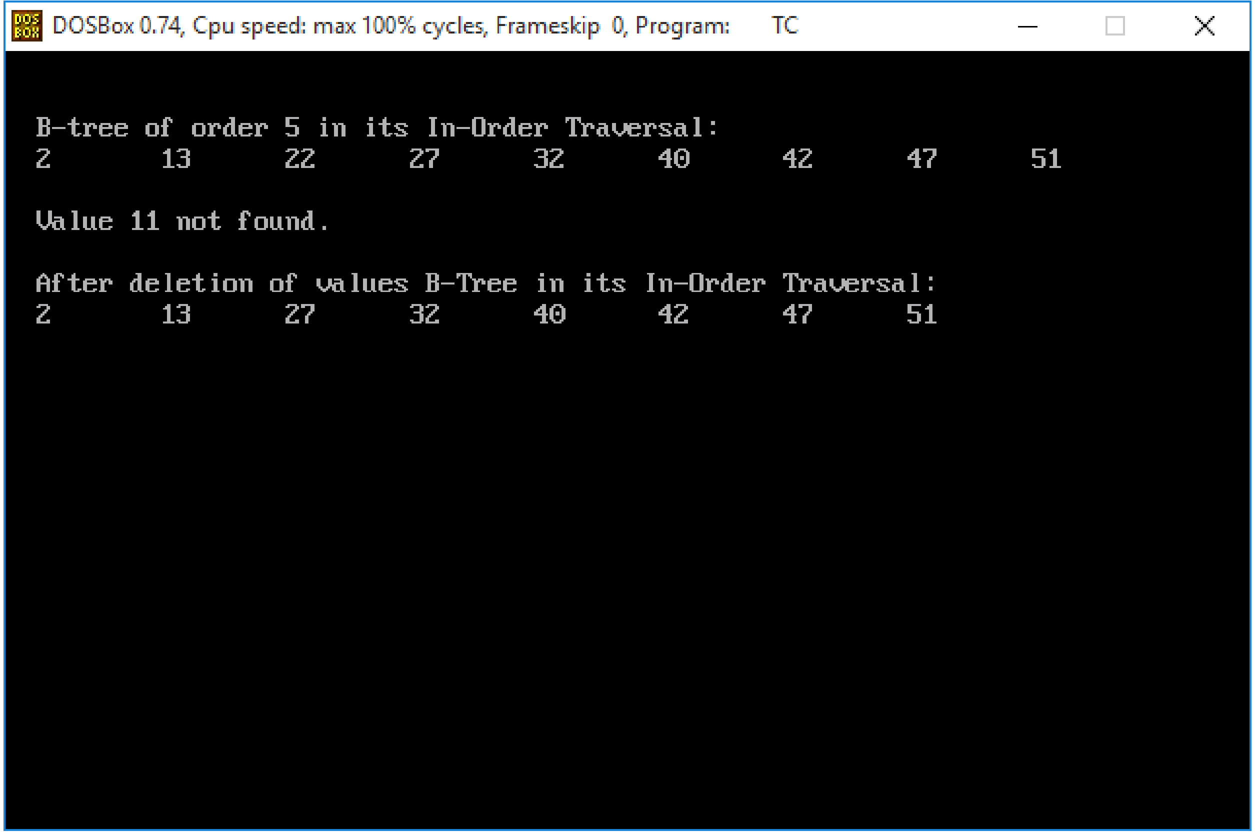
Solution: B- Tree

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

#define MAX 4
#define MIN 2

struct btnode
{
   int count ;
   int value[MAX + 1] ;
   struct btnode *child[MAX + 1] ;
} ;

struct btnode * insert ( int, struct btnode * ) ;
int setval ( int, struct btnode *, int *, struct btnode ** ) ;
struct btnode * search ( int, struct btnode *, int * ) ;
int searchnode ( int, struct btnode *, int * ) ;
void fillnode ( int, struct btnode *, struct btnode *, int ) ;
void split ( int, struct btnode *, struct btnode *,int, int *, struct btnode ** ) ;
struct btnode * delete ( int, struct btnode * ) ;
int delhelp ( int, struct btnode * ) ;
void clear ( struct btnode *, int ) ;
void copysucc ( struct btnode *, int ) ;
void restore ( struct btnode *, int ) ;
void rightshift ( struct btnode *, int ) ;
void leftshift ( struct btnode *, int ) ;
void merge ( struct btnode *, int ) ;
void display ( struct btnode * ) ;

void main( )
{
   struct node *root ;
   root = NULL ;

   clrscr( ) ;

   root = insert(27, root);
   root = insert(42, root);
```

```
    root = insert(22, root);
    root = insert(47, root);
    root = insert(32, root);
    root = insert(2, root);
    root = insert(51, root);
    root = insert(40, root);
    root = insert(13, root);

    printf ( "B-tree of order 5:\n" ) ;
    display ( root ) ;

    root = delete ( 22, root ) ;
    root = delete ( 11, root ) ;

    printf ( "\n\nAfter deletion of values:\n" ) ;
    display ( root ) ;

    getch( ) ;
}

/* inserts a value in the B-tree*/
struct btnode * insert ( int val, struct btnode *root )
{
    int i ;
    struct btnode *c, *n ;
    int flag ;

    flag = setval ( val, root, &i, &c ) ;
    if ( flag )
    {
        n = ( struct btnode * ) malloc ( sizeof ( struct btnode ) ) ;
        n -> count = 1 ;
        n -> value [1] = i ;
        n -> child [0] = root ;
        n -> child [1] = c ;
        return n ;
    }
    return root ;
}

/* sets the value in the node */
int setval ( int val, struct btnode *n, int *p, struct btnode **c )
{
    int k ;
    if ( n == NULL )
    {
        *p = val ;
        *c = NULL ;
        return 1 ;
    }
    else
    {
        if ( searchnode ( val, n, &k ) )
            printf ( "\nKey value already exists.\n" ) ;
        if ( setval ( val, n -> child [k], p, c ) )
        {
            if ( n -> count < MAX )
            {
                fillnode ( *p, *c, n, k ) ;
                return 0 ;
            }
            else
            {
                split ( *p, *c, n, k, p, c ) ;
                return 1 ;
            }
        }
        return 0 ;
    }
}

/* searches value in the node */
struct btnode * search ( int val, struct btnode *root, int *pos )
{
    if ( root == NULL )
        return NULL ;
    else
    {
        if ( searchnode ( val, root, pos ) )
            return root ;
            else
                return search ( val, root -> child [*pos], pos ) ;
    }
}

/* searches for the node */
int searchnode ( int val, struct btnode *n, int *pos )
{
    if ( val < n -> value [1] )
    {
        *pos = 0 ;
        return 0 ;
    }
    else
    {
        *pos = n -> count ;
        while ( ( val < n -> value [*pos] ) && *pos > 1 )
            ( *pos )-- ;
        if ( val == n -> value [*pos] )
            return 1 ;
            else
                return 0 ;
    }
}


/* adjusts the value of the node */
void fillnode ( int val, struct btnode *c, struct btnode *n, int k )
{
    int i ;
    for ( i = n -> count ; i > k ; i-- )
    {
        n -> value [i + 1] = n -> value [i] ;
        n -> child [i + 1] = n -> child [i] ;
    }
    n -> value [k + 1] = val ;
    n -> child [k + 1] = c ;
    n -> count++ ;
}

/* splits the node */
void split ( int val, struct btnode *c, struct btnode *n,
                int k, int *y, struct btnode **newnode )
{
    int i, mid ;

    if ( k <= MIN )
        mid = MIN ;
    else
        mid = MIN + 1 ;

    *newnode = ( struct btnode * ) malloc ( sizeof ( struct btnode ) ) ;

    for ( i = mid + 1 ; i <= MAX ; i++ )
    {
        ( *newnode ) -> value [i - mid] = n -> value [i] ;
        ( *newnode ) -> child [i - mid] = n -> child [i] ;
    }
    ( *newnode ) -> count = MAX - mid ;
    n -> count = mid ;

    if ( k <= MIN )
        fillnode ( val, c, n, k ) ;
    else
        fillnode ( val, c, *newnode, k - mid ) ;

    *y = n -> value [n -> count] ;
    ( *newnode ) -> child [0] = n -> child [n -> count] ;
    n -> count-- ;
}

/* deletes value from the node */
struct btnode * delete ( int val, struct btnode *root )
{
    struct btnode * temp ;
    if ( ! delhelp ( val, root ) )
        printf ( "\nValue %d not found.", val ) ;
    else
    {
        if ( root -> count == 0 )
        {
            temp = root ;
            root = root -> child [0] ;
            free ( temp ) ;
        }
    }
    return root ;
}

/* helper function for delete( ) */
int delhelp ( int val, struct btnode *root )
{
    int i ;
    int flag ;
    if ( root == NULL )
        return 0 ;
    else
    {
        flag = searchnode ( val, root, &i ) ;
        if ( flag )
        {
            if ( root -> child [i - 1] )
            {
                copysucc ( root, i ) ;
                flag = delhelp ( root -> value [i], root -> child [i] ) ;
                if ( !flag )
                    printf ( "\nValue %d not found.", val ) ;
            }
            else
                clear ( root, i ) ;
        }
        else
            flag = delhelp ( val, root -> child [i] ) ;

        if ( root -> child [i] != NULL )
        {
            if ( root -> child [i] -> count < MIN )
                restore ( root, i ) ;
        }
        return flag ;
    }
}

/* removes the value from the node and adjusts the values */
void clear ( struct btnode *node, int k )
{
    int i ;
    for ( i = k + 1 ; i <= node -> count ; i++ )
    {
        node -> value [i - 1] = node -> value [i] ;
        node -> child [i - 1] = node -> child [i] ;
    }
    node -> count-- ;
}

/* copies the successor of the value that is to be deleted */
void copysucc ( struct btnode *node, int i )
{
    struct btnode *temp ;

    temp = node -> child [i] ;

    while ( temp -> child[0] )
        temp = temp -> child [0] ;

    node -> value [i] = temp -> value [1] ;
}

/* adjusts the node */
```

```
void restore ( struct btnode *node, int i )
{
    if ( i == 0 )
    {
        if ( node -> child [1] -> count > MIN )
            leftshift ( node, 1 ) ;
        else
            merge ( node, 1 ) ;
    }
    else
    {
        if ( i == node -> count )
        {
            if ( node -> child [i - 1] -> count > MIN )
                rightshift ( node, i ) ;
            else
                merge ( node, i ) ;
        }
        else
        {
            if ( node -> child [i - 1] -> count > MIN )
                rightshift ( node, i ) ;
            else
            {
                if ( node -> child [i + 1] -> count > MIN )
                    leftshift ( node, i + 1 ) ;
                else
                    merge ( node, i ) ;
            }
        }
    }
}

/* adjusts the values and children while shifting the value from parent to right
    child */
void rightshift ( struct btnode *node, int k )
{
    int i ;
    struct btnode *temp ;

    temp = node -> child [k] ;

    for ( i = temp -> count ; i > 0 ; i-- )
    {
        temp -> value [i + 1] = temp -> value [i] ;
        temp -> child [i + 1] = temp -> child [i] ;
    }

    temp -> child [1] = temp -> child [0] ;
    temp -> count++ ;
    temp -> value [1] = node -> value [k] ;

    temp = node -> child [k - 1] ;
    node -> value [k] = temp -> value [temp -> count] ;
    node -> child [k] -> child [0] = temp -> child [temp -> count] ;
    temp -> count-- ;
}

/* adjusts the values and children while shifting the value from parent to left
    child */
void leftshift ( struct btnode *node, int k )
{
    int i ;
    struct btnode *temp ;

    temp = node -> child [k - 1] ;
    temp -> count++ ;
    temp -> value [temp -> count] = node -> value [k] ;
    temp -> child [temp -> count] = node -> child [k] -> child [0] ;

    temp = node -> child [k] ;
    node -> value [k] = temp -> value [1] ;
    temp -> child [0] = temp -> child [1] ;
    temp -> count-- ;

    for ( i = 1 ; i <= temp -> count ; i++ )
    {
        temp -> value [i] = temp -> value [i + 1] ;
        temp -> child [i] = temp -> child [i + 1] ;
    }
}

/* merges two nodes */
void merge ( struct btnode *node, int k )
{
    int i ;
    struct btnode *temp1, *temp2 ;

    temp1 = node -> child [k] ;
    temp2 = node -> child [k - 1] ;
    temp2 -> count++ ;
    temp2 -> value [temp2 -> count] = node -> value [k] ;
    temp2 -> child [temp2 -> count] = node -> child [0] ;

    for ( i = 1 ; i <= temp1 -> count ; i++ )
    {
        temp2 -> count++ ;
        temp2 -> value [temp2 -> count] = temp1 -> value [i] ;
        temp2 -> child [temp2 -> count] = temp1 -> child [i] ;
    }
    for ( i = k ; i < node -> count ; i++ )
    {
        node -> value [i] = node -> value [i + 1] ;
        node -> child [i] = node -> child [i + 1] ;
    }
    node -> count-- ;
    free ( temp1 ) ;
}

/* displays the B-tree */
void display ( struct btnode *root )
{
    int i ;

    if ( root != NULL )
    {
        for ( i = 0 ; i < root -> count ; i++ )
        {
            display ( root -> child [i] ) ;
            printf ( "%d\t", root -> value [i + 1] ) ;
        }
        display ( root -> child [i] ) ;
    }
}
```

Output:



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

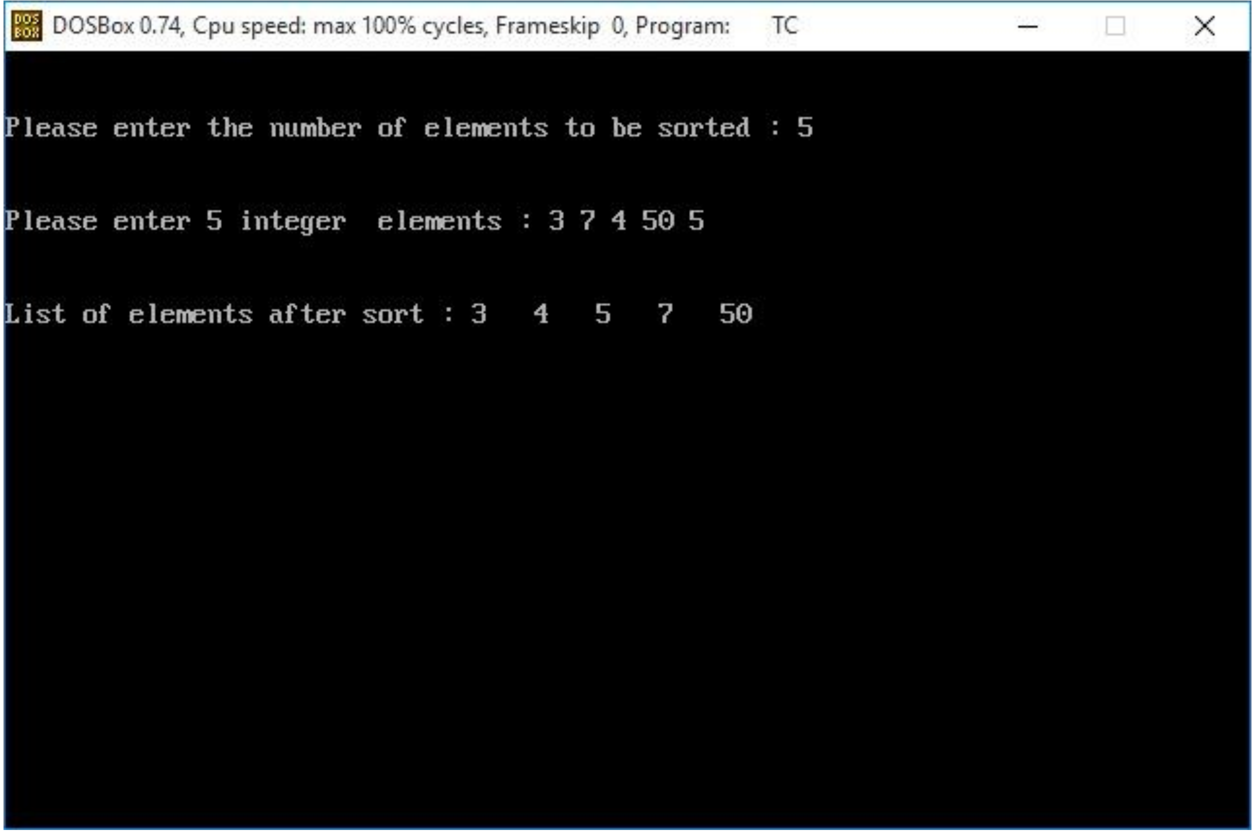
```
B-tree of order 5 in its In-Order Traversal:
2       13      22      27      32      40      42      47      51

Value 11 not found.

After deletion of values B-Tree in its In-Order Traversal:
2       13      27      32      40      42      47      51
```

Solution: Heap Sort

```
#include <stdio.h>
#include <conio.h>
int p(int);
int left(int);
int right(int);
void heapify(int[],int,int);
void buildheap(int[],int);
void heapsort(int[],int);
void main() {
  int x[20],n,i;
  clrscr();
  printf("\n\nPlease enter the number of elements to be sorted : ");
  scanf("%d",&n);
  printf("\n\nPlease enter %d integer elements : ",n);
  for(i=0;i<n;i++)
    scanf("%d",&x[i]);
  heapsort(x,n);
  printf("\n\nList of elements after sort : ");
  for(i=0;i<n;i++)
  printf("%d   ",x[i]);
  getch();
}
int p(int i) {
  return i/2;
}
int left(int i)
{
  return 2*i+1;
}
int right(int i) {
  return 2*i+2;
}
void heapify(int a[],int i,int n) {
  int l,r,large,t;
  l=left(i);
  r=right(i);
  if((l<=n-1)&&(a[l]>a[i]))
    large=l; else large=i;
  if((r<=n-1)&&(a[r]>a[large]))
    large=r;
  if(large!=i) {
    t=a[i];
    a[i]=a[large];
    a[large]=t;
```

```
      heapify(a,large,n);
    }
}
void buildheap(int a[],int n) {
   int i;
   for(i=(n-1)/2;i>=0;i--)
      heapify(a,i,n);
}
void heapsort(int a[],int n) {
   int i,m,t;
   buildheap(a,n);
   m=n ;
   for(i=n-1;i>=1;i--) {
      t=a[0];
      a[0]=a[i];
      a[i]=t;
      m=m-1;
      heapify(a,0,m);
   }
}
```

Output:


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Please enter the number of elements to be sorted : 5

Please enter 5 integer  elements : 3 7 4 50 5

List of elements after sort : 3   4   5   7   50
```

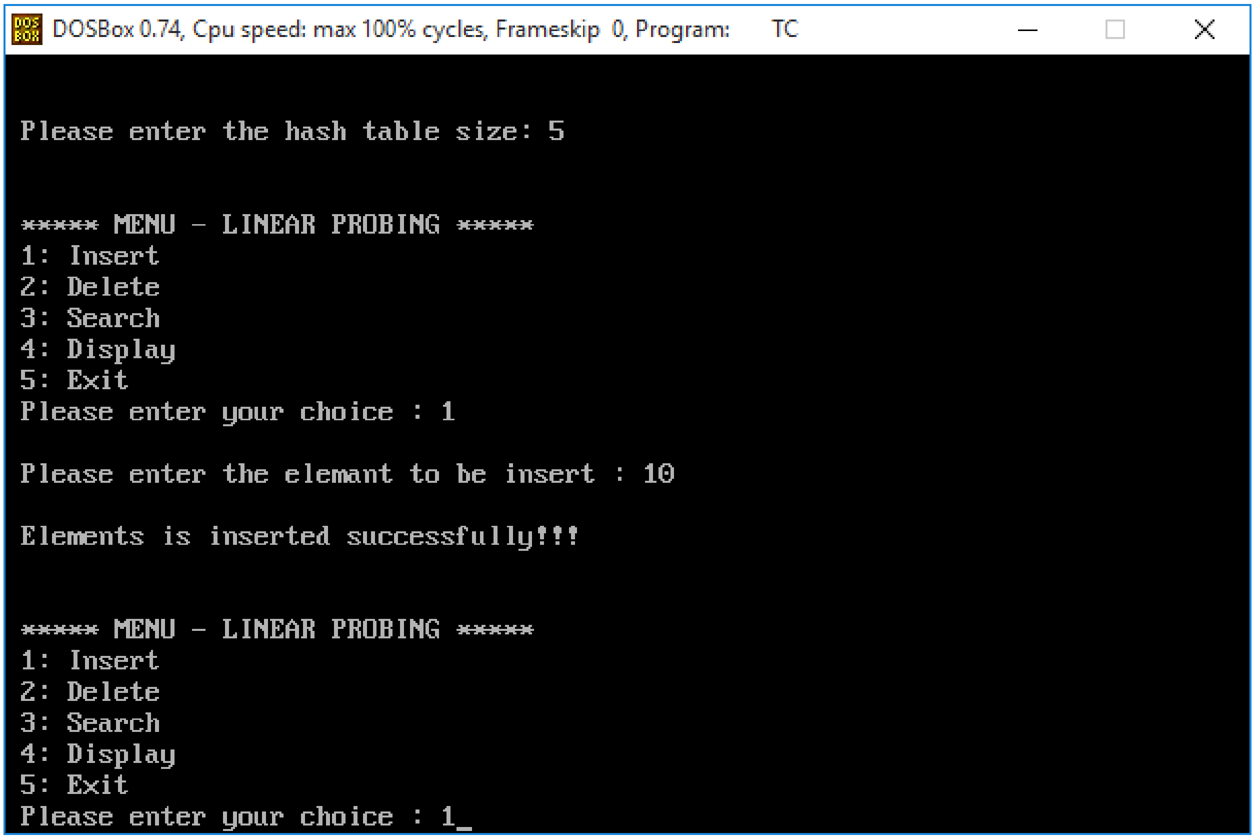
Week 10:

Write a C program to implement all the functions of a dictionary (ADT) using hashing.

Solution:

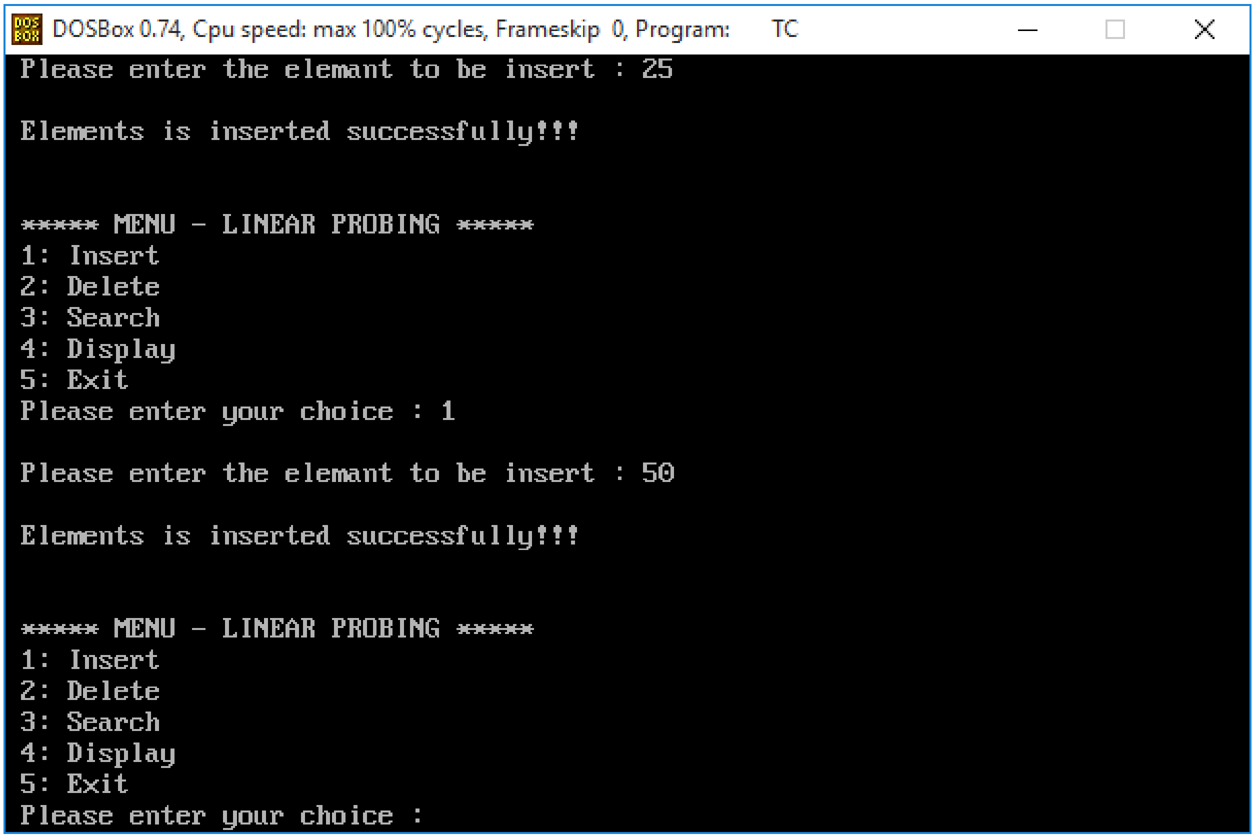
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
int b;
int hsearch(int key,int d,int *ht,int *empty) {
  int i=key%(d);
  int j=i, c=0;
  do {
    if(empty[j]||(*(ht+j)==key))
      return j;
    c++;
    j=(i+c)%(d);
  }while(j!=i);
  return 0;
}
int search(int key,int d,int *ht,int *empty) {
  b=hsearch(key,d,ht,empty);
  if(empty[b]==1)
    return -1;
  else if(b==0)
    return 1;
  else
    return b;
}
void insert(int key,int d,int *ht,int *empty) {
  b=hsearch(key,d,ht,empty);
  if(empty[b]) {
    empty[b]=0;
    *(ht+b)=key;
    printf("\n Elements is inserted successfully!!!\n");
  }
}
void delete(int key,int d,int *ht,int *empty) {
  int b=hsearch(key,d,ht,empty);
  *(ht+b)=0;
  empty[b]=1;
  printf("\n Element is deleted\n");
}
void display(int d,int *ht,int *empty) {
  int i;
  printf("\n Hash table elements are\n");
  for(i=0;i<d;i++) {
    if(empty[i])
      printf(" 0");
    else
      printf("%5d",*(ht+i));
  }
  printf("\n");
}
void main() {
  int choice=1, key, d,i,s, *empty,*ht;
  clrscr();
  printf("\n\n Please enter the hash table size: ");
  scanf("%d",&d);
  ht=(int *)malloc(d *sizeof(int));
  empty=(int *)malloc(d *sizeof(int));
  for(i=0;i<d;i++)
     empty[i]=1;
  while(1) {
    printf("\n\n ***** MENU - LINEAR PROBING *****");
    printf("\n 1: Insert\n 2: Delete\n 3: Search\n 4: Display\n 5: Exit");
    printf("\n Please enter your choice : ");
    scanf("%d",&choice);
      switch(choice) {
        case 1: printf("\n Please enter the elemant to be insert : ");
                  scanf("%d",&key);
                  insert(key,d,ht,empty);
                  break;
          case 2: printf("\n Please enter the element to be remove : ");
                  scanf("%d",&key);
                  delete(key,d,ht,empty);
                  break;
          case 3: printf("\n Please enter the search element to be search : ");
                  scanf("%d",&key);
                  s=search(key,d,ht,empty);
                  if(s==-1||s==0)
                    printf("\n Given element is not found\n");
                  else
                    printf("\n Given element is found at index %d",hsearch(key,d,ht,empty));
                  break;
          case 4: display(d,ht,empty);
                  break;
          case 5: exit(0);
         }
  }
}
```

Output:

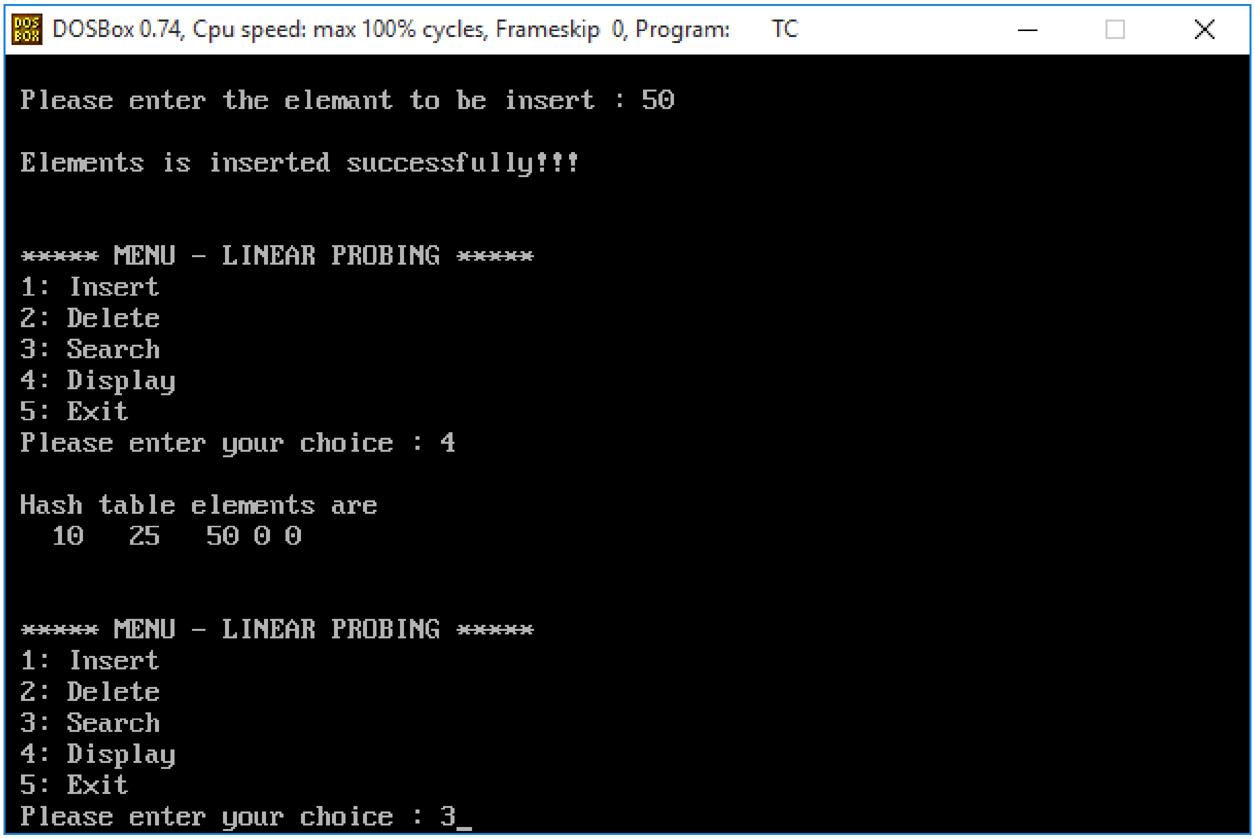


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

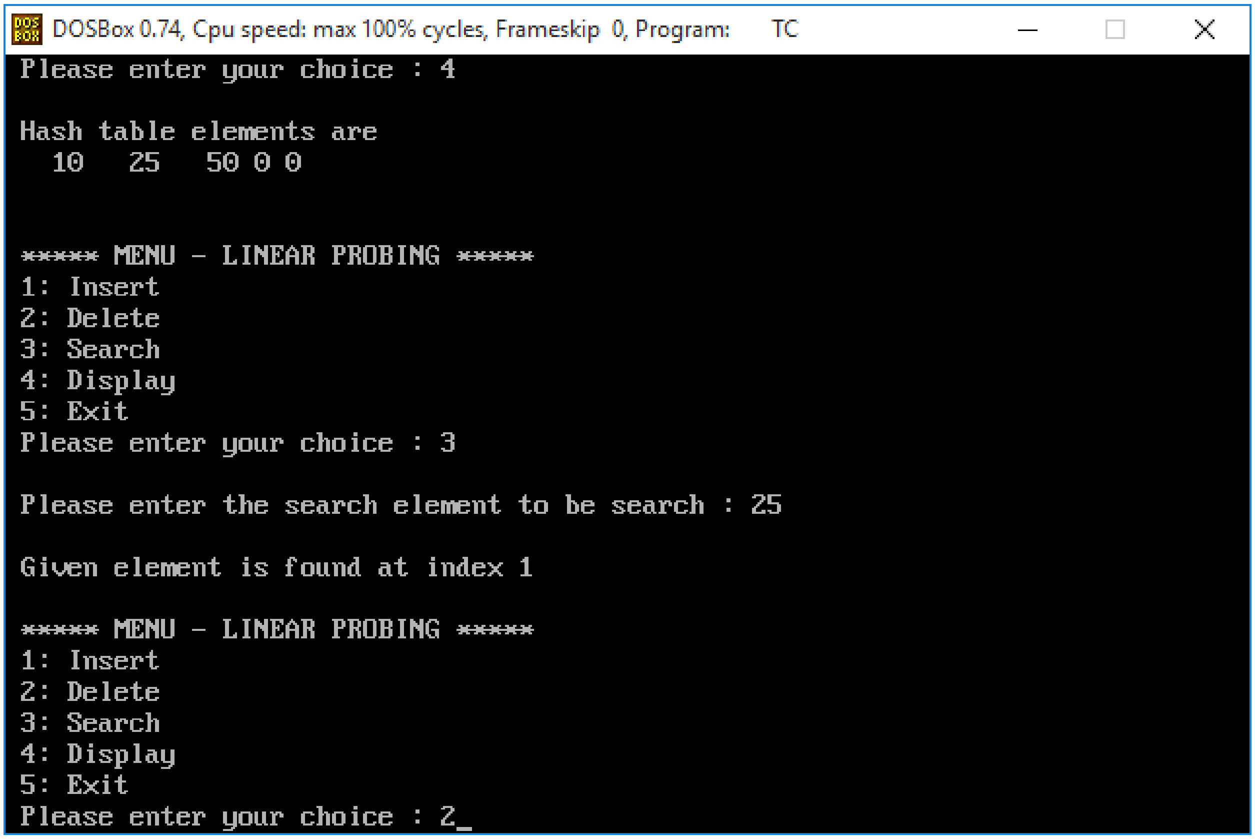
```
Please enter the hash table size: 5


***** MENU - LINEAR PROBING *****
1: Insert
2: Delete
3: Search
4: Display
5: Exit
Please enter your choice : 1

Please enter the elemant to be insert : 10

Elements is inserted successfully!!!


***** MENU - LINEAR PROBING *****
1: Insert
2: Delete
3: Search
4: Display
5: Exit
Please enter your choice : 1_
```


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

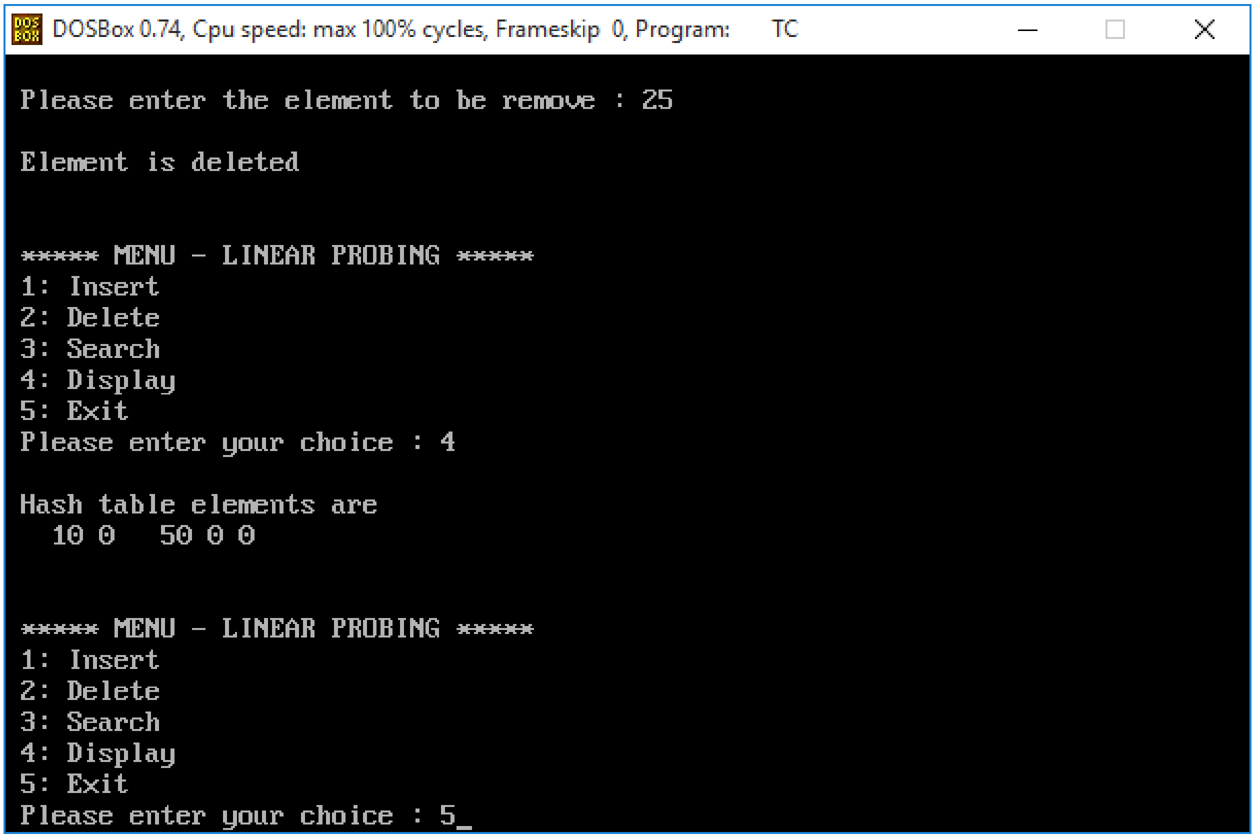
```
Please enter the elemant to be insert : 25

Elements is inserted successfully!!!


***** MENU - LINEAR PROBING *****
1: Insert
2: Delete
3: Search
4: Display
5: Exit
Please enter your choice : 1

Please enter the elemant to be insert : 50

Elements is inserted successfully!!!


***** MENU - LINEAR PROBING *****
1: Insert
2: Delete
3: Search
4: Display
5: Exit
Please enter your choice :
```


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```

Please enter the elemant to be insert : 50

Elements is inserted successfully!!!


***** MENU - LINEAR PROBING *****
1: Insert
2: Delete
3: Search
4: Display
5: Exit
Please enter your choice : 4

Hash table elements are
  10   25   50 0 0


***** MENU - LINEAR PROBING *****
1: Insert
2: Delete
3: Search
4: Display
5: Exit
Please enter your choice : 3_
```


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Please enter your choice : 4

Hash table elements are
   10    25    50 0 0


***** MENU - LINEAR PROBING *****
1:  Insert
2: Delete
3: Search
4: Display
5: Exit
Please enter your choice : 3

Please enter the search element to be search : 25

Given element is found at index 1

***** MENU - LINEAR PROBING *****
1:  Insert
2: Delete
3: Search
4: Display
5: Exit
Please enter your choice : 2_
```



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Please enter the element to be remove : 25

Element is deleted


***** MENU - LINEAR PROBING *****
1: Insert
2: Delete
3: Search
4: Display
5: Exit
Please enter your choice : 4

Hash table elements are
  10 0   50 0 0


***** MENU - LINEAR PROBING *****
1: Insert
2: Delete
3: Search
4: Display
5: Exit
Please enter your choice : 5_
```

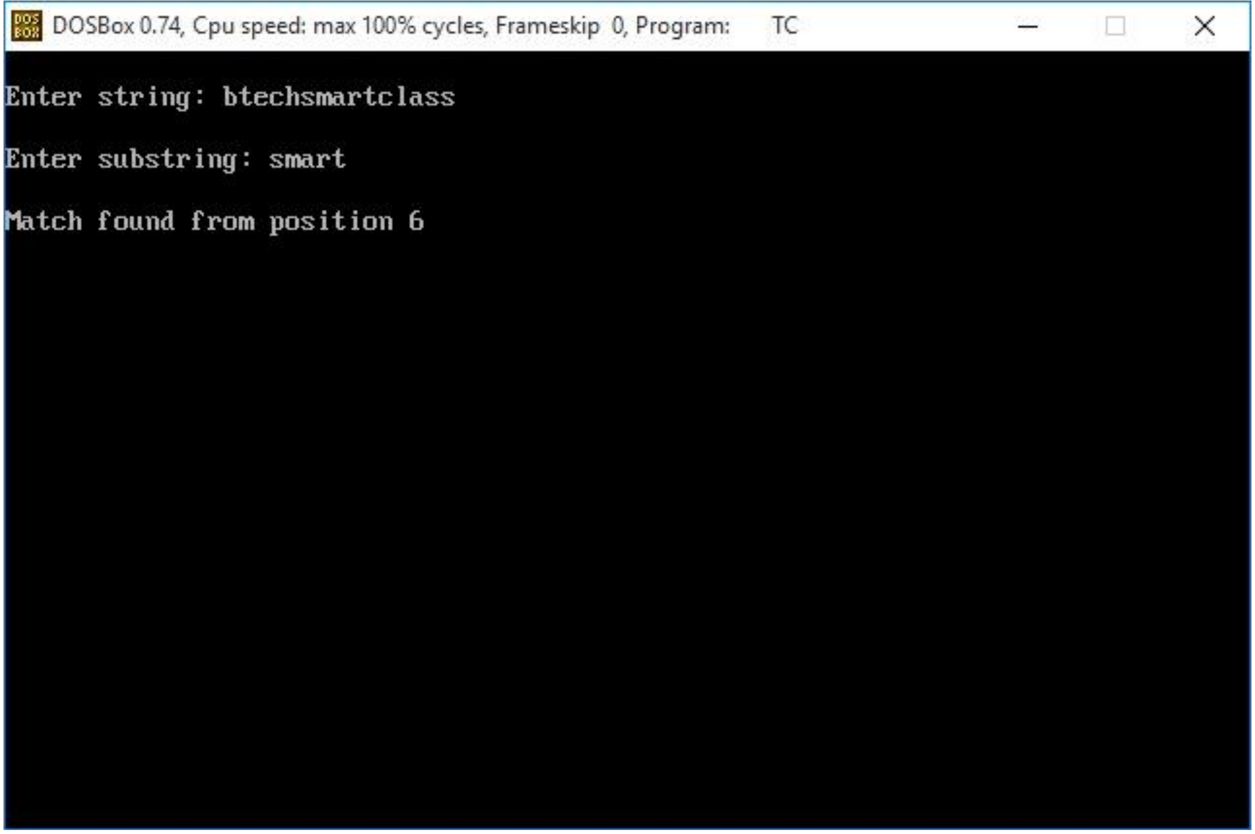
Week 11:

Write a C program for implementing Knuth-Morris- Pratt pattern matching algorithm.

Solution:

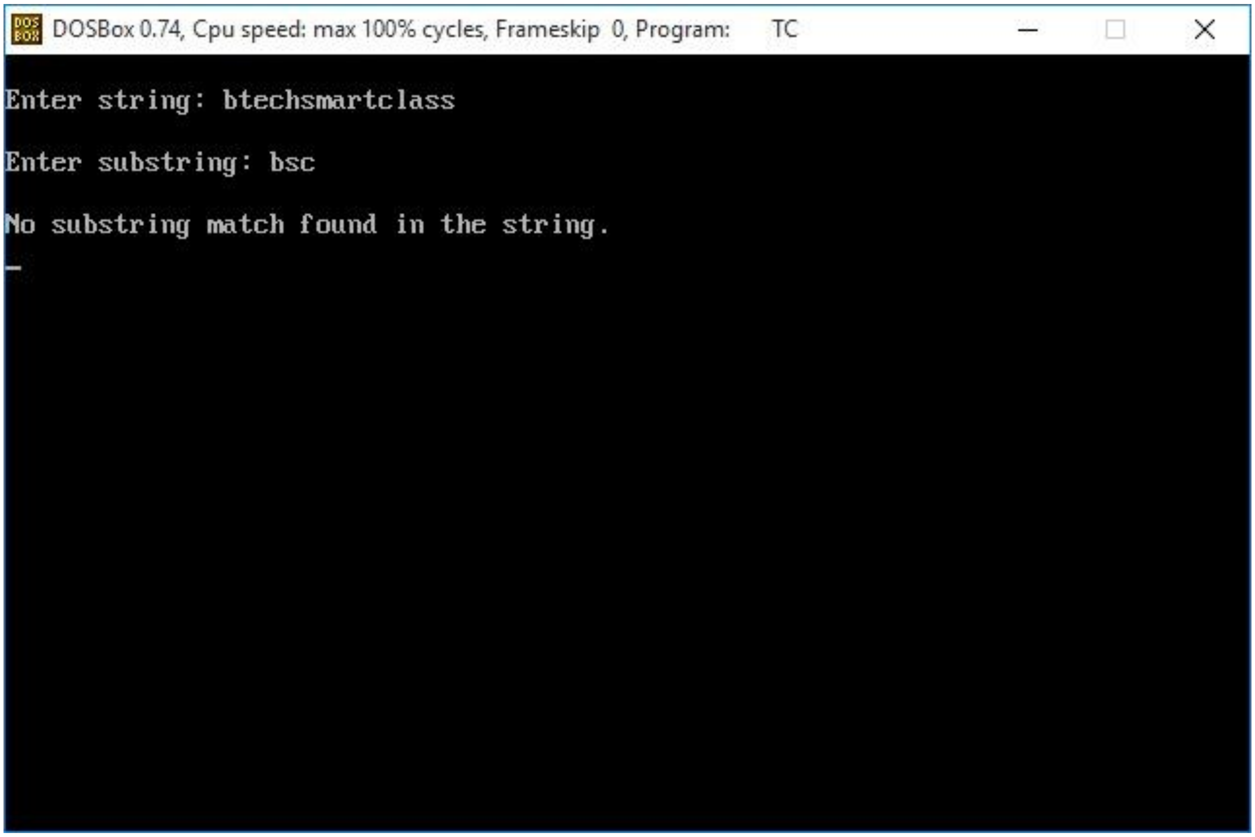
```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <ctype.h>

void main()
{
    char string[100], matchcase[20], c;
    int i = 0, j = 0, index;
    clrscr();
    /*Reading string*/
    printf("\nEnter string: ");
    scanf("%s",string);
    i = strlen(string);
    string[i - 1] = '\0';
    /* Reading pattern to be search*/
    printf("\nEnter substring: ");
    scanf("%s",matchcase);
    i = strlen(matchcase);
    matchcase[i - 1] = '\0';
    for (i = 0; i < strlen(string) - strlen(matchcase) + 1; i++)
    {
            index = i;
            if (string[i] == matchcase[j])
            {
                do
                {
                        i++;
                        j++;
                } while(j != strlen(matchcase) && string[i] == matchcase[j]);
                if (j == strlen(matchcase))
                {
                        printf("\nMatch found from position %d\n", index + 1);
                        goto end;
                }
                else
                {
                        i = index + 1;
                        j = 0;
                }
            }
    }
    printf("\nNo substring match found in the string.\n");
    end: getch();
}
```

Output:



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter string: btechsmartclass

Enter substring: smart

Match found from position 6
```



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter string: btechsmartclass

Enter substring: bsc

No substring match found in the string.
_
```

Week 12:

Write C programs for implementing the following graph traversal algorithms:

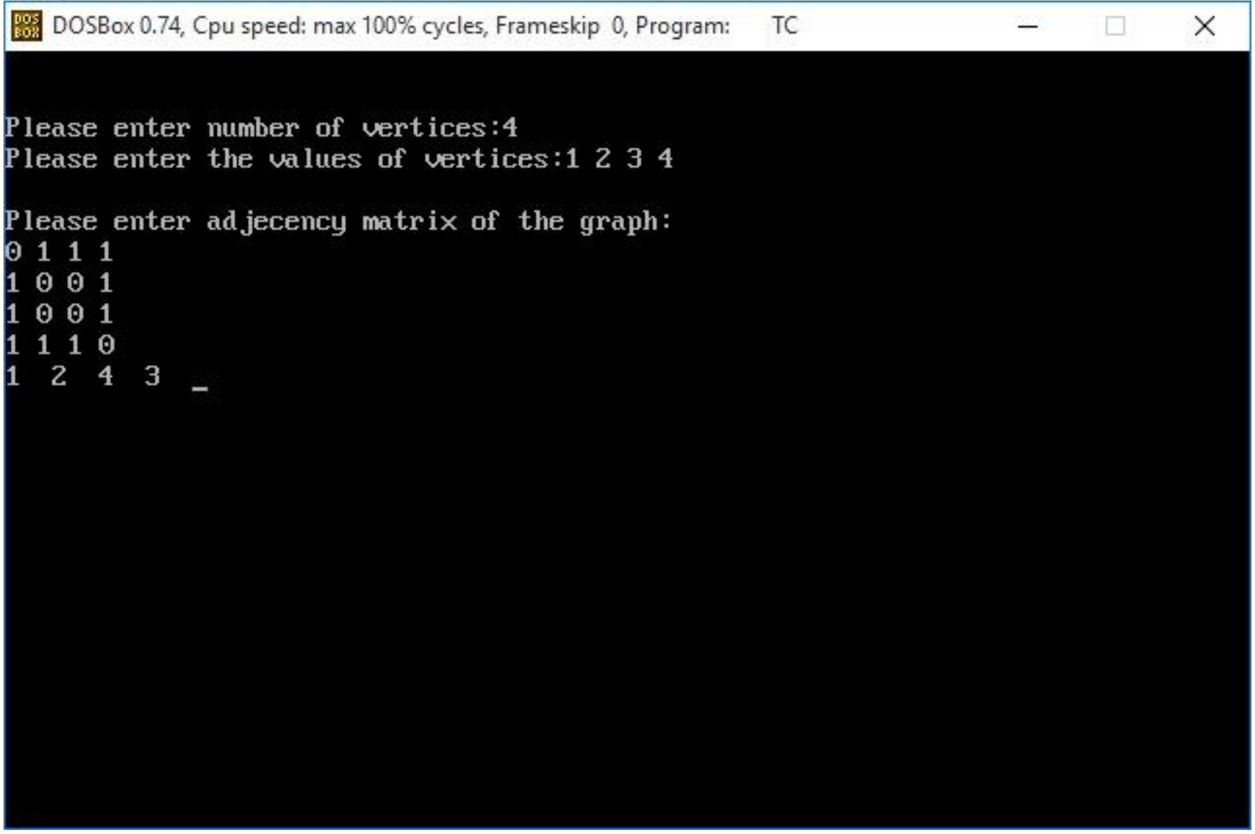
a)Depth first traversal

b)Breadth first traversal

Solution: DFS

```
#include <stdio.h>
#include<conio.h>
void dfs(int);
int g[10][10],visited[10],n,vertex[10];
void main() {
  int i,j;
  clrscr();
  printf("\n\nPlease enter number of vertices:");
  scanf("%d",&n);
  printf("Please enter the values of vertices:");
  for(i=0;i<n;i++)
    scanf("%d",&vertex[i]);
  printf("\nPlease enter adjecency matrix of the graph:\n");
  for(i=0;i<n;i++)
    for(j=0;j<n;j++)
         scanf("%d",&g[i][j]);
  for(i=0;i<n;i++)
    visited[i]=0;
  dfs(0);
  getch();
}
void dfs(int i) {
  int j;
  printf("%d  ",vertex[i]);
  visited[i]=1;
  for(j=0;j<n;j++)
    if(!visited[j]&&g[i][j]==1)
  dfs(j);
}
```

Output:



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Please enter number of vertices:4
Please enter the values of vertices:1 2 3 4

Please enter adjecency matrix of the graph:
0 1 1 1
1 0 0 1
1 0 0 1
1 1 1 0
1  2  4  3  _
```