VELALAR COLLEGE OF ENGINEERING AND TECHNOLOGY

(AUTONOMOUS)

DEPARTMENT OF COMUPTER SCIENCE AND ENGINEERING

REGULATION 2018

Name	: ………………………………………………...	Course	:…………..
Register No.	:…………………………………………………	Branch	:…………..
Class	:…………………………………………………	Semester	:…………..
Course Name	:…………………………………………………	Course Code	:…………..

Certified that this is a bonafide record of workdone by the above student during the academic year 20 - 20

Faculty In-charge **Head of the Department**

Submitted for the Practical Examination held on:….……………………………………..………..

Internal Examiner **External Examiner**

INDEX

Ex. No	Date	Experiments	Page No	Marks	Signature

Ex.No: 1 **Roll No:**

Date:

IMPLEMENTATION OF ORDERED LIST

There are lists where insertion should ensure the ordering of data elements. Since the elements are in ascending order the search can terminate once equal or greater element is found. Implement a singly linked list of ordered integers (ascending/descending) with insert, search and display operations.

Aim:

To write a C program to implement a singly linked list of ordered integers with insert, search and display operations.

Algorithm:

STEP 1 : Start the program.

STEP 2 : Allocate a memory space for a node using malloc function.

STEP 3 : Get the data to be inserted.

STEP 4 : Identify the position to insert the new node into the list so that the value of the nodes are present in ascending order. Rearrange the pointers correspondingly.

STEP 5 : To delete a node, get the data from the user and check whether the data is found in the list, if not display data not found.

STEP 6 : Searching for a value of the node stops once greater than or equal value comes in searching. If not, display data not found. Otherwise delete the node.

STEP 7 : To display the elements in the list, starting from the first element, print the value of each node of the list using a temporary variable to point to the first element of the list.

STEP 8 : Stop the program

Program:

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

struct node {
  int data;
```

```
   struct node *next;
};
int n=0,f=0,pos=0;

struct node *head = NULL;
struct node *current = NULL;

void display()
{
   struct node *ptr = head;
   printf("\n[ ");
   while(ptr != NULL)
   {
     printf("%d ",ptr->data);
     ptr = ptr->next;
   }
   printf(" ]");
}

void insert(int ele)
{
   struct node *ptr;
   struct node *p = (struct node*) malloc(sizeof(struct node));
   p->data = ele;
   p->next = NULL;
   if(head==NULL)
   head = p;
   else
   {
    ptr = head;
    if(ptr->data >ele)
    {
       p->next = head;
       head = p;
    }
    else
    {
       while(ptr->next != NULL &&ptr->next->data <= ele)
                                  ptr = ptr->next;
       p->next=ptr->next;
       ptr->next = p;
    }
   }

}

int find(int key)
{
   struct node *current = head;
   f=0;
   pos=0;
   if(head == NULL)
       return 0;
   while(current != NULL && current->data <= key)
   {
     if(current->data == key)
     {
       f=1;
       break;
     }
     else
     {
       pos++;
       current = current->next;
     }
   }
   return pos;
}
```

```
void main() {
   int ch,x;
   while(1)
   {
    printf("\n\nLIST");
    printf("\n1. Insert\n2. Search\n3. Display\n4. Exit");
    printf("\nEnter the choice 1 - 4 from above : ");
    scanf("%d",&ch);
    switch(ch)
    {
    case 1:printf("Enter the element to be inserted : ");
           scanf("%d",&x);
           insert(x);
           break;
    case 2:printf("Enter the element to be searched : ");
          scanf("%d",&x);
          pos=find(x);
          if(f == 1)
            printf("The element is present at %d",pos);
          else
            printf("The element is not present in the list");
          break;
   case 3:display();
          break;
   case 4:printf("End of the List Operations");
          exit(0);
  default:exit(0);
  }
 }
}
```

Output:

```
 List
1.Insert
2.Search
3.Display
4.Exit
Enter the choice 1-4 from above :1
Enter the element to be inserted:98
 List
1.Insert
2.Search
3.Display
4.Exit
Enter the choice 1-4 from above :2
Enter the element to be searched:98
The element is present at 0
 List
1.Insert
2.Search
3.Display
4.Exit
Enter the choice 1-4 from above :3
[98]
 List
1.Insert
2.Search
3.Display
4.Exit
Enter the choice 1-4 from above :4
End of the list operation
```

Result:

Thus the given C program to implement an ordered list has been executed successfully and the output is verified

Ex.No: 2a **Roll No:**

Date:

IMPLEMENTATION OF STACK USING ARRAY

Write a C Program to simulate Stack using array data structure.

Aim:

To write a C program to implement a Stack using array and to carry out its various operations.

Algorithm:

```
STEP 1 : Start the program.
STEP 2 : Initialize the stack and Top pointer.
STEP 3 : For Push operation,
        i) Get the data to be pushed into the Stack.
        ii) Check whether the Stack is full. If so print appropriate message.
        iii) If not, increment the top pointer and add it to the Stack array.
STEP 4 : For Pop operation
        i) Check whether the Stack is empty. If so print appropriate message.
        ii) If not, print the top element.
        iii) Decrement the Stack size by 1.
STEP 5 : For display operation, starting from the first element, print the elements of the Stack using a looping construct.
STEP 6 : Stop the program
```

Program:

```
  #include<stdio.h>
  #include<conio.h>
  #define size 10
  void main()
  {
int choice,element,i,item;
int s[size],top=-1; clrscr();
  do
  {
```

```
printf("\n1.Push 2.Pop 3.Display 4.Exit\n");
printf("Enter your choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1: if(top==size-1)
   printf("Sorry! Stack Overflows\n");
   else
   {
   printf("\nEnter the element to be inserted\n");
   scanf("%d",&element);
   top++;
   s[top]=element;
   }
   break;
case 2: if(top==-1)
   printf("Sorry! Stack Underflow\n");
   else
   {
   item=s[top];
   top--;
   printf("The popped item is%d\t",item);
   }
   break;
case 3: if(top==-1)
   printf("Sorry! Stack Underflow\n");
   else
   {
      for(i=top;i>=0;i--)
         printf("%d\t",s[i]);
   }
   break;
default:printf("\n\nProgram is over");
}
```

```
}while(choice!=4);
getch();
}
```

Output:

```
1.Push   2.Pop  3.Display   4.Exit
 Enter your choice
 1
 Enter the element to be inserted
 21
 1.Push   2.Pop  3.Display   4.Exit
 Enter your choice
 1
 Enter the element to be inserted
 33
 1.Push   2.Pop  3.Display   4.Exit
 Enter your choice
 1
 Enter the element to be inserted
 44
1.Push   2.Pop  3.Display   4.Exit
 Enter your choice
 2
 The popped item is: 44
 1.Push   2.Pop  3.Display   4.Exit
 Enter your choice
 3
 33 21
1.Push   2.Pop  3.Display   4.Exit
 Enter your choice
  4
  Program is over.
```

Result:

Thus the given C program to implement a Stack using array has been executed successfully and the output is verified

Ex.No: 2b **Roll No:**

Date:

IMPLEMENTATION OF STACK USING LINKED LIST

Write a C Program to simulate Stack using Linked List data structure.

Aim:

To write a C program to implement a Stack using linked list and to carry out its various operations.

Algorithm:

```
STEP 1 : Start the program.
STEP 2 : Define a structure for a Stack.
STEP 3 : For Push operation,
             i) Get the data to be pushed into the Stack.
             ii) Form a new node using malloc function.
             iii) Add the new node to the stack and make it as Top.
STEP 4 : For Pop operation

             i)   Check whether the Stack is empty. If so print appropriate message.
             ii)  If not, print the top element and make the next element as new Top.
             iii) Free the memory occupied by the deleted node

STEP 5 : For display operation, define a temporary variable to contain the address of the top
         element of the Stack and print the elements of the Stack.
STEP 6 : Stop the program
```

Program:

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
struct stack
  {
  int element;
  struct stack *next;
  }*top=NULL;
  void main()
  {
  int choice,num,item;
  clrscr();
  top=NULL;
  do
  {
  printf("\n1.Push 2.Pop 3.Display 4.Exit\n");
  printf("Enter your choice\n");
  scanf("%d",&choice);
  switch(choice)
  {
  case 1: push();
     break;
  case 2: pop();
     break;
  case 3: display();
     break;
case 4: break;
  default:printf("INVALID CHOICE\n");
  }
  }while(choice!=4);
  getch();
  }
```

```
push()
{
struct stack *tmpcell;
int num;
tmpcell=malloc(sizeof(struct stack));
if(tmpcell==NULL)
    printf("Sorry!memory cannot be allocated\n");
else
{
printf("\nEnter the element to be pushed : ");
scanf("%d",&num);
tmpcell->element=num;
tmpcell->next=top;
top=tmpcell;
}
}

pop()
{
struct stack *tmpcell;
int item;
if(top==NULL)
printf("stack is empty\n");
else
{
tmpcell=top;
item=tmpcell->element;
top=tmpcell->next;

free(tmpcell);
printf("\nThe deleted element is %d",item);
}
}
```

```
  display()

{
  struct stack *s;
  s=top;
  while(s!=NULL)
  {
  printf("%d\t",s->element);
  s=s->next;
  }
  }
{
  struct stack *s;
  s=top;
  while(s!=NULL)
  {
  printf("%d\t",s->element);
  s=s->next;
  }
  }
```

Output:

```
1.Push   2.Pop  3.Display   4.Exit
  Enter your choice
  1
Enter the element to  be pushed :11
1.Push   2.Pop  3.Display   4.Exit
Enter your choice
 1
Enter the element to  be pushed :12
1.Push   2.Pop  3.Display   4.Exit
Enter your choice
 1
Enter the element to  be pushed :13
1.Push   2.Pop  3.Display   4.Exit
Enter your choice
 1
Enter the element to  be pushed :14
1.Push   2.Pop  3.Display   4.Exit
```

```
Enter your choice
2
The deleted element :14
1.Push   2.Pop  3.Display   4.Exit
Enter your choice
 3
13 12 11
1.Push   2.Pop  3.Display   4.Exit
Enter your choice
4
```

Result:

Thus the given C program to implement a Stack using Linked List has been executed successfully and the output is verified

Ex.No: 3a **Roll No:**

Date:

IMPLEMENTATION OF QUEUE USING ARRAY

Write a C Program to simulate a Queue using array data structure.

Aim:

To write a C program to implement a Queue using array and to carry out its various operations.

Algorithm:

```
STEP 1 : Start the program.
STEP 2 : Initialize the queue, front and rear pointers.
STEP 3 : For Enqueue operation,
        i) Get the data to be added to the Queue.
        ii) Check whether the Queue is full. If so print appropriate message.
        iii) If not, increment the rear pointer and add it to the Queue array.
STEP 4 : For Dequeue operation
        i) Check whether the Queue is empty. If so print appropriate message.
        ii) If not, print the front element.
        iii) Increment the front pointer by 1 and reduce the size of the queue by 1.
STEP 5 : For display operation, starting from the front pointer, print the elements of the Queue
         using a looping construct.
STEP 6 : Stop the program
```

Program:

```
#include<stdio.h>
#include<conio.h>
# define max 10
void main()
{
int i,size=0,choice=0,num,val;
int front=-1;
int rear=-1,qu[max];
clrscr();
```

```
do
{
printf("\nMENU");
printf("\nEnter your choice(1-4)\n");
printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
scanf("%d",&choice);
   switch(choice)
    {
    case 1: if(rear>=max)
    printf("Queue is FULL\n");
        else
        {
    printf("Enter the element to be inserted\n");
    scanf("%d",&num);
        if(front==-1)
          ++front;
        ++rear;
    qu[rear]=num;
        ++size;
    printf("\nThe number of elements present in the queue is\t%d",size);
        }
        break;
   case 2: if(front==-1 || front==rear+1)
    printf("\n QUEUE UNDERFLOW");
        else
        {
    val=qu[front++];
        --size;
    printf("\nThe deleted element is\t%d",val);
    printf("\nThe size of the queue is\t%d",size);
        }
        break;
        case 3: if(front==-1 || front==rear+1)printf("\n QUEUE UNDERFLOW");
        else

        {

         for(i=front;i<=rear;i++)

     printf("%d\t",qu[i]);

        }

        break;

     }

  }while(choice!=4);

  }
```

Output:

```
MENU
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:1
Enter the element to be inserted:51
The number of elements present in the queue is 1
MENU
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:1
Enter the element to be inserted:64
The number of elements present in the queue is 2
MENU
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:2
The deleted element is 64
The number of elements present in the queue is 1
MENU
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:4
```

Result:
Thus the given C program to implement a Queue using array has been executed successfully andthe output is verified

Ex.No: 3b **Roll No:**

Date:

IMPLEMENTATION OF QUEUE USING LINKED LIST

Write a C Program to simulate Queue using Linked List data structure.

Aim:

To write a C program to implement a Queue using linked list and to carry out its various operations.

Algorithm:

```
STEP 1 : Start the program.
STEP 2 : Define a structure for a Queue.
STEP 3 : For Enqueue operation,
        i) Get the data to be added to the Queue.
        ii) Form a new node using malloc function.
        iii) Add the new node to the Queue and make it as new rear element.
STEP 4 : For Dequeue operation
        i) Check whether the Queue is empty. If so print appropriate message.
        ii) If not, print the element pointed out by front and make the next node as new Front.
        iii) Free the memory occupied by the deleted node.
STEP 5 : For display operation, define a temporary variable to contain the address of the front element of the Queue and print the elements of the Queue.
STEP 6 : Stop the program.
```

Program:

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
struct queue
{
int element;
struct queue *next;
};
struct queue *front=NULL,*rear=NULL;
```

```
void main()
{
int choice;
clrscr();
do
{
printf("\n1. ENQUEUE 2.DEQUEUE 3. DISPLAY 4. EXIT\n");
printf("enter your choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1: enqueue();
        break;
case 2: dequeue();
        break;
case 3: display();
        break;
case 4: break;
}
}while(choice!=4);
getch();
}
enqueue()
{
    struct queue *tmpcell;
    int num;
    printf("Enter the element to be added to the Queue\n");
    scanf("%d",&num);
    tmpcell=(struct queue *)malloc(sizeof(struct queue));
    tmpcell->element=num;
    tmpcell->next=NULL;


    if(front==NULL && rear==NULL)
    front=tmpcell;
    else
      rear->next=tmpcell;
    rear=tmpcell;
}
dequeue()
{
    struct queue *t;
    if(front==NULL)
      printf("\n\nQUEUE UNDERFLOW\n");
    else
    {
      t=front;
      printf("\nThe Element deleted from the Queue is %d\n",t->element);
      if(front==rear)
      {
        front=NULL;
        rear=NULL;
      }
      front=t->next;
      free(t);
    }
}

display()
{
    struct queue *tmp;
    if(front==NULL)
      printf("\n\nQUEUE UNDERFLOW\n");
    else
    {
    tmp=front;
```

```
        while(tmp!=NULL)
        {
        printf("%d\t",tmp->element);
        tmp=tmp->next;
        }
        }
    }
```

Output:

```
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:1
Enter the element to be added in the queue:64
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:1
Enter the element to be added in the queue:51
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:2
The element deleted from the queue:64
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:4
```

Result:

Thus the given C program to implement a Queue using Linked List has been executed successfully and the output is verified.

Ex.No: 4 **Roll No:**

Date:

IMPLEMENTATION OF TREE TRAVERSALS

Write a recursive C program, for traversing a binary tree in preorder, in-order and post-order.

Aim:

To write a C program to implement various types of Tree Traversals.

Algorithm:

STEP 1 : Start the program.

STEP 2 : Define a structure for a node of a tree.

STEP 3 : For inorder traversal, the left subtree is visited first, then the root and later the right sub-tree.

STEP 4 : For preorder traversal, the root node is visited first, then the left subtree and finally the right subtree.

STEP 5 : For postorder traversal the left subtree is vivited first, then the right subtree and finally the root node.

STEP 6 : Stop the program.

Program:

```
#include<stdio.h>
#include<conio.h>
typedef struct bin
  {
     int data;
     struct bin *left;
     struct bin *right;
  }node;
  void insert(node *,node *);
  void inorder(node *);

  void preorder(node *);
  void postorder(node *);
  node*get_node();
  void main()
  {
     int choice;
     char ans='y';
     node *new,*root;
     root=NULL;
     clrscr();
     do
     {
     printf("\n Program for Simple Binary Tree");
     printf("\n 1.Create");
  printf("\n 2.Inorder");
     printf("\n 3.Preorder");
     printf("\n 4.Postorder");
     printf("\n 5.Exit");
     printf("\n\t Enter Your Choice : ");
     scanf("%d",&choice);
     switch(choice)
     {
     case 1:root=NULL;
           do
           {
         new=get_node();
         printf("\n Enter the Element : ");
         scanf("%d",&new->data);
         if(root==NULL)
              root=new;
         else
              insert(root,new);
         printf("\n Do you want to enter more data(y/n): ");
         ans=getch();
          }while(ans=='y'||ans=='Y');
         clrscr();
          break;
     case 2:if(root==NULL)
         printf("Tree is not created");
         else
         inorder(root);
         break;
     case 3:if(root==NULL)
         printf("Tree is not created");
         else
         preorder(root);
         break;
     case 4:if(root==NULL)
         printf("Tree is not created");


         else
         postorder(root);
         break;
     }
     }while(choice!=5);
  }
  node *get_node()
  {
     node *temp;
     temp=(node*)malloc(sizeof(node));
     temp->left=NULL;
     temp->right=NULL;
     return temp;
  }
  void insert(node *root,node *New)
  {
     char ch;
     printf("\n Where to insert left or right of %d(l/r/L/R)",root->data);
     ch=getch();
     if((ch=='r')||(ch=='R'))
     {
         if(root->right==NULL)
         {
         root->right=New;
         }
         else
         insert(root->right,New);
     }
     else
     {
         if(root->left==NULL)
         {
         root->left=New;
         }
         else
         insert(root->left,New);
     }
  }


  void inorder(node *temp)
  {
     if(temp!=NULL)
     {
         inorder(temp->left);
         printf("%d",temp->data);
         inorder(temp->right);
     }
  }
```

```
void preorder(node *temp)
{
    if(temp!=NULL)
    {
        printf("%d",temp->data);
        preorder(temp->left);
        preorder(temp->right);
    }
}
void postorder(node *temp)
{
    if(temp!=NULL)
    {
        postorder(temp->left);
        postorder(temp->right);
        printf("%d",temp->data);
    }
}
```

Output:

```
Program for simple binary tree
        1.Create
        2.Inorder
        3.Preorder
        4.Postorder
        5.Exit
        Enter your choice:1
        Enter the element:90
        Do you want to enter more data(y/n):y
        Program for simple binary tree
        1.Create
        2.Inorder
        3.Preorder
        4.Postorder
        5.Exit
        Enter your choice:1
        Enter the element:12
```

```
Where to insert left or right of 90(l/r/R/r):l
Do you want to enter more data(y/n):y
Program for simple binary tree
1.Create
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter your choice:1
Enter the element:78
Where to insert left or right of 90(l/r/R/r):r
Do you want to enter more data(y/n):y\
Program for simple binary tree
1.Create
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter your choice:1
Enter the element:87
Where to insert left or right of 90(l/r/R/r):l

Where to insert left or right of 12(l/r/R/r):l
Do you want to enter more data(y/n):y
Program for simple binary tree
1.Create
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter your choice:1
Enter the element:21
Where to insert left or right of 90(l/r/R/r):l
Where to insert left or right of 12(l/r/R/r):r\
Do you want to enter more data(y/n):y
Program for simple binary tree
1.Create
2.Inorder
3.Preorder
4.Postorder
5.Exit
```

```
Enter your choice:1
Enter the element:41
Where to insert left or right of 90(l/r/R/r):r
Where to insert left or right of 78(l/r/R/r):r
Do you want to enter more data(y/n):y
Program for simple binary tree
1.Create
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter your choice:2
87 12 21 90 78 41
Program for simple binary tree
1.Create
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter your choice:3
90 12 87 21 78 41

Program for simple binary tree
1.Create
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter your choice:4
87 21 12 41 78 90
Program for simple binary tree
1.Create
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter your choice:5
```

Result:

Thus the given C program to implement various types of Tree Traversals has been executedsuccessfully and the output is verified.

Ex.No: 5 **Roll No:**

Date:

IMPLEMENTATION OF BINARY SEARCH TREES

Write a C program to insert, delete and search for a node in a Binary Search Tree.

Aim:

To write a C program to implement Binary Search Trees and its operations.

Algorithm:

```
STEP 1 : Start the program.
STEP 2 : Define a structure for a node of a tree.
STEP 3 : Enter the choice. Insert/ Delete / Search.
STEP 4 : If the choice is Insert, choose the position and place the node.
STEP 5 : If the choice is delete, and
            If leaf node just made its parent’s LST/RST to be NULL.
            If the node has one child, replace the node with its corresponding left/right child.
            If the node has two children, replace the node with the minimum element in the right
            sub tree.
STEP 6 : If the choice is display, print the nodes using inorder traversal.
STEP 7 : Stop the program.
```

Program:

```
  #include<stdio.h>
  #include<conio.h>
#include<alloc.h>
typedef struct bst
  {

    int data;
    struct bst *left,*right;
  }node;
```

```
void insert(node *,node *);
void inorder(node *);
node * search(node *,int);
node * findmin(node *);
node * del(node *, int);

void main()
{
  int ch,key;
  node *New,*root,*tmp,*parent;
  node *get_node();
  root=NULL;
  clrscr();
  do
  {
    printf("\n1->Create\n2->Search\n3->Delete\n4->Display\n5->exit\tchoice:");
    scanf("%d",&ch);
    switch(ch)
    {
      case 1: New=get_node();
           printf("\nEnter the element : ");
           scanf("%d",&New->data);
           if(root==NULL)
           root=New;
           else
           insert(root,New);
           break;
      case 2:  printf("enter the element to search:");
        scanf("%d",&key);
        tmp=search(root, key);
        if(tmp!=NULL)
          printf("\n The element %d is present",tmp->data);
          else
             printf("\n The element %d is not present",key);
            break;
      case 3:  printf("\n Enter the element to delete");
        scanf("%d",&key);
        root=del(root,key);
        break;
      case 4: if(root==NULL)
            printf("Tree is not Created");
        else
        {
            printf("\n Tree is...:");
            inorder(root);
        }
         break;
    }
  }while(ch!=5);
}

node *get_node()
{
  node *temp;
  temp=(node*)malloc(sizeof(node));
  temp->left=NULL;
  temp->right=NULL;
  return temp;
}
void insert(node *root, node *New)
{
  if(New->data < root->data)
  {
    if(root->left==NULL)
       root->left=New;
    else
       insert(root->left,New);
  }
  if(New->data > root->data)
  {
    if(root->right==NULL)
       root->right=New;
    else
       insert(root->right,New);
  }
}
node * search(node *root,int key)
{
  node *temp;
  temp=root;
  while(temp!=NULL)
  {
    if(temp->data==key)
    {
    return temp;
    }
    if(temp->data>key)
    temp=temp->left;
    else
      temp=temp->right;
  }
  return NULL;
}

node * findmin(node *t)
```

```
{
   if(t!=NULL)
   while(t->left!=NULL)
      t=t->left;
   return t;
}



node * del(node *t,int x)
{
     node *tmpcell;
   if(t==NULL)
      printf("Sorry!no element is there in the tree\n");
   else if(x<t->data)
      t->left=del(t->left,x);
   else if(x>t->data)
      t->right=del(t->right,x);
   else if(t->left && t->right)
   {
      tmpcell=findmin(t->right);
      t->data=tmpcell->data;
      t->right=del(t->right,t->data);
   }
   else
   {
      tmpcell=t;
      if(t->left==NULL)
         t=t->right;
      else if(t->right==NULL)
         t=t->left;
      free(tmpcell);
   }
   return t;
}
```

```
void inorder(node *temp)
{
  if(temp!=NULL)
  {
    inorder(temp->left);
    printf("%d ",temp->data);
    inorder(temp->right);
  }
}
```

Output:

```
1.Create
2.Search
3.Delete
4.Display
5.Exit
Enter your choice:1
Enter the element:98
1.Create
2.Search
3.Delete
4.Display
5.Exit
Enter your choice:1
Enter the element:47
1.Create
2.Search
3.Delete
4.Display
5.Exit
Enter your choice:1
Enter the element:23
1.Create
2.Search
3.Delete
4.Display
5.Exit
Enter your choice:1
Enter the element:100
```

```
1.Create
2.Search
3.Delete
4.Display
5.Exit
Enter your choice:2
Enter the element to be searched :23
The element 23 is present
1.Create
2.Search
3.Delete
4.Display
5.Exit
Enter your choice:2
Enter the element to be searched :46
The element 46 is not present
1.Create
2.Search
3.Delete
4.Display
5.Exit
Enter your choice:3
Enter the element to be deleted:23
1.Create
2.Search
3.Delete
4.Display
5.Exit
Enter your choice:4
Tree is…..47 98 100
1.Create
2.Search
3.Delete
4.Display
5.Exit
Enter your choice:5
```

Result:

Thus the given C program to implement various operations in a Binary Search Tree has been executed successfully and the output is verified.

Ex.No: 6a **Roll No:**

Date:

IMPLEMENTATION OF BREADTH FIRST SEARCH

Write a C program for Breadth First Search Graph Traversals

Aim:

To write a C program to implement Breadth First Search graph traversals.

Algorithm:

STEP 1 : Start the program.

STEP 2 : Get the graph in matrix form.

STEP 3 : One node is selected as starting vertex and printed(visited).

STEP 4 : Then all its adjacent nodes are printed(visited) one by one.

STEP 5 : Move further to check another vertex and visits its adjacent vertices.

STEP 6 : Above step is repeated until all the vertices are visited.

STEP 7 : Stop the program.

Program:

```
#include<stdio.h>
#include<conio.h>
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v)
{
 for(i=1;i<=n;i++)
  if(a[v][i] && !visited[i])
   q[++r]=i;
 if(f<=r)
 {
  visited[q[f]]=1;
  bfs(q[f++]);
 }
}
void main()
{
 int v;
 clrscr();
 printf("n Enter the number of vertices:");
 scanf("%d",&n);
 for(i=1;i<=n;i++)
 {
  q[i]=0;
  visited[i]=0;
 }
 printf("n Enter graph data in matrix form:n");
 for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
   scanf("%d",&a[i][j]);
 printf("n Enter the starting vertex:");
 scanf("%d",&v);
 bfs(v);
 printf("n The node which are reachable are:n");
 for(i=1;i<=n;i++)
  if(visited[i])
   printf("%dt",i);
  else
   printf("n Bfs is not possible");
 getch();
}
```

Output:

```
Enter the number of vertices:4
Enter graph data in matrix form:
0 1 1 0
1 0 1 1
1 1 0 1
0 1 1 0
Enter the starting vertex:1
The node which are reachable are:
1       2       3       4
```

Result:

Thus the given C program to implement Breadth First Search graph traversals has been executedsuccessfully and the output is verified.

Ex.No: 6b **Roll No:**

Date:

IMPLEMENTATION OF DEPTH FIRST SEARCH

Write a C program for Depth First Search Graph Traversals

Aim:

To write a C program to implement Depth First Search Graph Traversals.

Algorithm:

```
STEP 1 : Start the program.
STEP 2 : Get the graph in matrix form.
STEP 3 : One node is selected as starting vertex and printed(visited).
STEP 4 : Then its adjacent vertex is printed(visited).
STEP 5 : Above 2 steps are repeated. Once there is no adjacent vertex, backtrack and start visiting
         the next vertex which is not visited.
STEP 6 : Above 3 steps are repeated until all the vertices are visited.
STEP 7 : Stop the program.
```

Program:

```
#include<stdio.h>
void DFS(int);
int G[10][10],visited[10],n;//n is no of vertices and graph is sorted in array G[10][10]
void main()
{
int i,j;
printf("Enter number of vertices:");
scanf("%d",&n);
//read the adjecency matrix
printf("\nEnter adjecency matrix of the graph:");
for(i=0;i<n;i++)
for(j=0;j<n;j++)

scanf("%d",&G[i][j]);
//visited is initialized to zero
for(i=0;i<n;i++)
visited[i]=0;
DFS(0);
}
void DFS(int i)
{
int j;
printf("\n%d",i);
visited[i]=1;
for(j=0;j<n;j++)
if(!visited[j]&&G[i][j]==1)
DFS(j);
}
```

Output:

```
Enter number of vertices:4
Enter adjecency matrix of the graph:
0 1 1 0
1 0 1 0
1 0 0 1
0 1 1 0

0
1
2
3
```

Result:

Thus the given C program to implement Breadth First Search graph traversals has been executed successfully and the output is verified.

Ex.No: 7 **Roll No:**

Date:

IMPLEMENTATION OF HEAP SORT

Consider the motor racing game in which there are n participants. Get the points scored by each participant. Write a program to sort the positions of players in ascending order based on points scored using heap sort and print the highest score

Aim:

To write a C program to implement Heap Sort.

Algorithm:

STEP 1 : Start the program.

STEP 2 : Get the points scored by players in an array.

STEP 3 : Build a max heap using the elements of the array satisfying the properties of a heap.

STEP 4 : Replace the first item with the last item of the heap and reduce the size of heap by 1.

STEP 5 : Above 2 steps are repeated until the size of the heap is reduced to 1.

STEP 6 : Now the array is in sorted order and the player name with high score is printed.

STEP 7 : Stop the program.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define leftchild(i) (2*(i)+1)
struct player
{
  char name[10];
  int score;
}p[10];
void heapsort(struct player *,int);
void percdown(struct player *,int,int);
void main()
{
int n,i;
printf("\nEnter the number of participants : ");
scanf("%d",&n);
printf("\nEnter the scores of each participant\n");
for(i=0;i<n;i++)
{
```

```
printf("\nEnter player name :");
scanf("%s",p[i].name);
printf("\nEnter the score :");
scanf("%d",&p[i].score);
}
heapsort(p,n);

getch();
}
void heapsort(struct player p[],int n)
{
int i;
for(i=n/2;i>=0;i--)
percdown(p,i,n);
for(i=n-1;i>0;i--)
{
char name1[10];
int temp = p[0].score;
p[0].score = p[i].score;
p[i].score = temp;
strcpy(name1,p[0].name);
strcpy(p[0].name,p[i].name);
strcpy(p[i].name,name1);
percdown(p,0,i);
}
printf("Value of the array after sorting\n\n");
for(i=0;i<n;i++)
printf("%s\t%d\n",p[i].name,p[i].score);
}

void percdown(struct player p[],int i,int n)
{
int child;
int tmp;
char tname[10];
strcpy(tname,p[i].name);
for(tmp=p[i].score;leftchild(i)<n;i=child)
{
child=leftchild(i);
if(child!=n-1 && p[child+1].score>p[child].score)
child++;
if(tmp<p[child].score)
{
p[i].score=p[child].score;
strcpy(p[i].name,p[child].name);
}
else
break;
}
```

```
p[i].score=tmp;
strcpy(p[i].name,tname);
}
```

Output:

```
Enter the number of participants:2
Enter the score of each participant
Enter player name:john
Enter the score:1000
Enter the player name:Daniel
Enter the score:5000
Value of the array after sorting
Daniel    5000
John      1000
```

Result:

Thus the given C program to implement Heap Sort has been executed successfully and the output is verified.

Ex.No: 8 **Roll No:**

Date:

IMPLEMENTATION OF LINEAR SEARCH

Write a C program to use linear search technique to search for a character in a given message

Aim:

To write a C program to implement Linear Search.

Algorithm:

STEP 1 : Start the program.

STEP 2 : Get the message and the character to be searched from the user.

STEP 3 : Using a for loop check whether the character is present in the message read.

STEP 4 : If its present, print the position of the character in the message.

STEP 5 : Otherwise print that the character is not present in the message.

STEP 6 : Stop the program.

Program:

```
#include <stdio.h>
int main()
{
   char str[1000],ch;
   int i, found=0;
printf("Enter a string:");
gets(str);
printf("Enter a character to find:");
scanf("%c",&ch);
for(i=0;str[i]!='\0';++i)
{
    if(ch==str[i])
    {
        found = 1;
        break;
    }
}
if(found == 1)
    printf("Character %c is at the position =%d",ch,i+1);
else
    printf("\nThe character %c is not present in the given string",ch);
return 0;
}
```

Output:

```
Enter a string:Data Structures
Enter a character to find:S
Character S is at the position =5
```

Result:

Thus the given C program to implement Linear Search has been executed successfully and the output is verified.

Ex.No: 9 **Roll No:**

Date:

IMPLEMENTATION OF BINARY SEARCH

A person has registered for voter id, he received a voter number and he need to check whether it exist in the voter list or not. Use binary search in a recursive way to find whether the voter number exist in the list or not.

Aim:

To write a C program to implement Binary Search.

Algorithm:

```
STEP 1 : Start the program.
STEP 2 : Get the list of voter ID’s in sorted order from the user in an array and the voter-ID to be
         searched.
STEP 3 : Find the mid-index in the list.
STEP 4 : Compare the search ID with the middle element in the list.
STEP 5 : If both are matched, then print the position of voter ID in the array.
STEP 6 : Otherwise, check whether the search element is smaller or larger than the middle element.
STEP 7 : If the search element is smaller, then repeat steps 4,5,6 for the left sublist of the middle
         element.
STEP 8 : If the search element is larger, then repeat steps 4,5,6 for the right sublist of the middle
         element.
STEP 9 : Stop the program.
```

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int n[10],end,mid,key,i,f=0;
   int low=0,high;
   clrscr();
   printf("Enter the number of id:");
   scanf("%d",&end);
```

```
    printf("Enter voter ids:\n");
    for(i=0;i<end;i++)
    scanf("%d",&n[i]);
    printf("Enter the ID to be searched:");
    scanf("%d",&key);
    high=end-1;
    while(low<=high)
    {
    mid=(low+high)/2;
    if(n[mid]<key)
       low=mid+1;
    else if(n[mid]==key)
    {
       printf("Voter ID is found at %d",mid+1);
       f=1;
       break;
    }
    else
    {
       high=mid-1;
       mid=(low+high)/2;
    }
    }
    if(f==0)
     printf("Voter ID not found");
getch();
```

Output:

```
Enter the number of id:5
Enter voter ids:
11
12
13
14
```

```
15
Enter the element to be search:15
Element found at 5
```

Result:

Thus the given C program to implement Binary Search has been executed successfully and theoutput is verified.

Ex.No: 10 **Roll No:**

Date:

DIJKSTRA’S ALGORITHM

For the given route map with cost of transportation between different cities, find the shortest route from a source to all the other cities using Dijkstra’s Algorithm.

Aim:

To write a C program to implement Dijkstra’s Algorithm.

Algorithm:

```
STEP 1 : Start the program.
STEP 2 : Get the cost matrix and a source node.
STEP 3 : Mark all nodes unvisited
STEP 4 : Set the tentative distance to zero for initial node and to infinity for all other nodes
STEP 5 : For the initial node, calculate their tentative distances for its unvisited neighbors through
         the current node.
STEP 6 : Compare the newly calculated tentative distance to the current assigned value and assign
         the smaller one.
STEP 7 : Select an unvisited node with the smallest tentative distance as new current node, and go
         back to step 4.
STEP 8 : Print the edges selected.
STEP 9 : Stop the program.
```

Program:

```
#include  <stdio.h>
#include <stdlib.h>
#include<conio.h>
struct edge
{
   int st;
   int end;
   int dist;
} city [10];
int cost [10] [10], k,n,i,j,v;
int dist [10], parent [10];
void dij (int n,intv,int cost[10][10])
{
   int i,u,count,w,flag[10],min;
   for (i=1; i<=n;i++)
```

```
     flag[i]=0, parent[i]=v,dist[i]=cost[v][i];
    count =2;
    while (count<=n)
    {
min=31999;
        for (w=1; w<=n;w++)
            if(dist[w]<min &&! flag[w])
            min =dist[w], u=w;
        flag[u]=1;
        count++;
        for (w=1; w<=n;w++)
        {
            if((dist[u]+cost[u][w] <dist[w]) &&! flag[w])
            {
dist[w]=dist[u]+cost[u][w];
                parent [w]=u;
            }
        }
    }
parent[v]=-1;
}
path (int w)
{
if (parent [w]! =-1)
    {
        path(parent[w]);
printf ("->%d",w);
    }
}
void main ()
{
    int m,c;
printf ("\n enter the number of cities:");
scanf("%d",&n);
printf ("\n \n enter the number of roads connecting the cities:");
scanf("%d",&m);
getch ();
printf ("\n enter the distance between the cities\n");
printf("......................\n");
    for (k=1; k<=m;k++)
    {
printf ("enter the starting city (from 1 to %d:), n);
scanf("%d",&city[k].st);
printf ("enter the ending city (from 1 to %d:), n);
scanf("%d",&city[k].end);
pri
ntf ("enter the distance between %d and %d:", city[k].st,city[k].end);
scanf("%d",&city[k].dist);
        cost[city[k].st] [city[k]. end] =city[k]. dist;
        cost[city[k]. end] [city[k].st] =city[k]. dist;
    }
    for (i=1; i<=n;i++)
        for (j=1; j<=n; j++)
        if (cost[i][j] ==0)
        cost[i][j] =31999;
printf ("enter the source city (from 1 to %d) from which path has to be calculated:, n);
scanf("%d",&v);
dij(n,v,cost);
printf ("\n shortest path:\n");
printf ("source \t destination \t distance \n units \t path \n");
    for (i=1; i<=n;i++)
        if (i! =v)
    {
printf ("\n %d \t \t %d \t\t %d \t\t %d , v,i,dist[i],v);
        path(i);
    }
getch ();
            }
```

Output:

```
enter the number of cities:4

enter the number of roads connecting the cities:5

enter the distance between the cities

----------------

 enter the starting city (from 1 to 4:)1

enter the ending city (from 1 to 4:)2

enter the distance between 1 and 2 :1

enter the starting city (from 1 to 4:)1

enter the ending city (from 1 to 4:)3

enter the distance between 1 and 3 :4

enter the starting city (from 1 to 4:)1

enter the ending city (from 1 to 4:)4

enter the distance between 1 and 4 :10

enter the starting city (from 1 to 4:)2

enter the ending city (from 1 to 4:)4

enter the distance between 2 and 4 :3

enter the starting city (from 1 to 4:)4

enter the ending city (from 1 to 4:)3

enter the distance between 4 and 3 :5

enter the source city (from 1 to 4) from which path has to be calculated :1
```

shortest path:

sourceunits	destination path	distance	
1	2	1	1 -> 2
1	3	4	1 -> 3
1	4	4	1 -> 2-> 4

Result:

Thus the given C program to implement Dijkstra's Algorithm has been executed successfully and the output is verified.