ARRAY IMPLEMENTATION OF LIST ADT

PROGRAM:

```
#include <stdio.h>
#include <conio.h>
void create();
void insert();
void search();
void deletion();
void display();
int i, e, n, pos;
static int b[50];
main()
{
int ch;
char g = 'y';
create();
do
{
printf("\n List Operations");
printf("\n 1.Deletion\n 2.Insert\n 3.Search\n 4.Exit\n");
printf("Enter your choice: ");
scanf("%d", &ch);
switch(ch)
{
case 1:
deletion();
break;
case 2:
insert();
break;
case 3:
```

```c
search();
break;
case 4:
exit(0);
default:
printf("\n Enter the correct choice:");
}
printf("Do you want to continue: ");
fflush(stdin);
scanf("\n %c",&g);
} while(g=='y' || g=='Y');
getch();
}
void create()
{
printf("\n Enter the number of elements:");
scanf("%d",&n);
printf("\n Enter list elements: ");
for(i=0; i<n; i++)
scanf("%d", &b[i]);
}
void deletion()
{
printf("\n enter the position you want to delete: ");
scanf("%d", &pos);
if(pos >= n)
printf("\n Invalid location");
else
{
for(i=pos+1; i<n; i++)
b[i-1] = b[i];
```

```
n--;
printf("List elements after deletion");
display();
}
}
void search()
{
int flag = 0;
printf("\n Enter the element to be searched: ");
scanf("%d", &e);
for(i=0; i<n; i++)
{
if(b[i] == e)
{
flag = 1;
printf("Element is in the %d position", i);
break;
}
}
if(flag == 0)
printf("Value %d is not in the list", e);
}
void insert()
{
printf("\n Enter the position you need to insert: ");
scanf("%d", &pos);
if(pos >= n)
printf("\n Invalid location");
else
{
++n;
```

```
for(i=n; i>pos; i--)
b[i] = b[i-1];
printf("\n Enter the element to insert: ");
scanf("%d", &e);
b[pos] = e;
}
printf("\n List after insertion:");
display();
}
void display()
{
for(i=0; i<n; i++)
printf("\n %d", b[i]);
}
```

OUTPUT:

```
Enter the number of elements:3
 Enter list elements: 11
22
33
 List Operations
 1.Deletion
 2.Insert
 3.Search
 4.Exit
Enter your choice: 2
 Enter the position you need to insert: 2
 Enter the element to insert: 10
 List after insertion:
 11
 22
 10
 33Do you want to continue: y
 List Operations
 1.Deletion
 2.Insert
 3.Search
 4.Exit
Enter your choice: 3
 Enter the element to be searched: 33
Element is in the 3 positionDo you want to continue: y
 List Operations
 1.Deletion
 2.Insert
 3.Search
 4.Exit
```

```
Enter your choice: 1

 enter the position you want to delete: 2

List elements after deletion

 11

 22

 33Do you want to continue: n
```

ARRAY IMPLEMENTATION OF STACK ADT

PROGRAM:

```
#include <stdio.h>
int stack[100],i,j,choice=0,n,top=-1;
void push();
void pop();
void show();
void main ()
{
  printf("Enter the number of elements in the stack ");
  scanf("%d",&n);
   printf("*********Stack operations using array*********");

 printf("\n----------------------------------------------\n");
 while(choice != 4)
 {
   printf("Choose one from the below options...\n");
   printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");
   printf("\n Enter your choice \n");
   scanf("%d",&choice);
   switch(choice)
   {
     case 1:
     {
       push();
       break;
      }
      case 2:
      {
        pop();
         break;
```

```
            }
            case 3:
            {
                show();
                break;
            }
            case 4:
            {
                printf("Exiting....");
                break;
            }
            default:
            {
                printf("Please Enter valid choice ");
            }
        };
    }
}
void push ()
{
    int val;
    if (top == n )
    printf("\n Overflow");
    else
    {
        printf("Enter the value?");
        scanf("%d",&val);
        top = top +1;
        stack[top] = val;
    }
}
```

```
void pop ()
{
   if(top == -1)
   printf("Underflow");
   else
   top = top -1;
}
void show()
{
   for (i=top;i>=0;i--)
   {
      printf("%d\n",stack[i]);
   }
   if(top == -1)
   {
      printf("Stack is empty");
   }
}
```

OUTPUT:

```
Enter the number of elements in the stack 3
*********Stack operations using array*********
---------------------------------------------
Choose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
1
Enter the value?11
Choose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
1
Enter the value?22
Choose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
1
Enter the value?33
```

```
Choose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
3
33
22
11
Choose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
2
Choose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
3
22
11
Choose one from the below options...
```

```
1.Push

2.Pop

3.Show

4.Exit

 Enter your choice
```

ARRAY IMPLEMENTATION OF QUEUE ADT

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#define maxsize 5
void insert();
void delete();
void display();
int front = -1, rear = -1;
int queue[maxsize];
void main ()
{
   int choice;
   while(choice != 4)
   {
      printf("\n***************************Main Menu*******************************\n");
      printf("\n========================================================================\n");
      printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");
      printf("\nEnter your choice ?");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1:
         insert();
         break;
         case 2:
         delete();
         break;
         case 3:
         display();
         break;
```

```
            case 4:
            exit(0);
            break;
            default:
            printf("\nEnter valid choice??\n");
        }
    }
}
void insert()
{
    int item;
    printf("\nEnter the element\n");
    scanf("\n%d",&item);
    if(rear == maxsize-1)
    {
        printf("\nOVERFLOW\n");
        return;
    }
    if(front == -1 && rear == -1)
    {
        front = 0;
        rear = 0;
    }
    else
    {
        rear = rear+1;
    }
    queue[rear] = item;
    printf("\nValue inserted ");

}
```

```
void delete()
{
   int item;
   if (front == -1 || front > rear)
   {
      printf("\nUNDERFLOW\n");
      return;

   }
   else
   {
      item = queue[front];
      if(front == rear)
      {
         front = -1;
         rear = -1 ;
      }
      else
      {
         front = front + 1;
      }
      printf("\nvalue deleted ");
   }


}

void display()
{
   int i;
   if(rear == -1)
```

```
    {
        printf("\nEmpty queue\n");
    }
    else
    {   printf("\nprinting values .....\n");
        for(i=front;i<=rear;i++)
        {
            printf("\n%d\n",queue[i]);
        }
    }
}
```

OUTPUT:

```
************************Main Menu*****************************

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

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter the element

55

Value inserted

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

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

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter the element

44

Value inserted

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

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

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter the element

66

Value inserted
```

```
*************************Main Menu*****************************

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

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter the element

77

Value inserted

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

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

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?3

printing values .....

55

44

66

77

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

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

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?2

value deleted
```

```
************************Main Menu****************************
=================================================================
1.insert an element
2.Delete an element
3.Display the queue
4.Exit
Enter your choice ?3
printing values .....
44
66
77
************************Main Menu****************************
=================================================================
1.insert an element
2.Delete an element
3.Display the queue
4.Exit
Enter your choice ?
```

LINKED LIST IMPLEMENTATION OF STACK

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
void push();
void pop();
void display();
struct node
{
int val;
struct node *next;
};
struct node *head;
void main ()
{
   int choice=0;
   printf("\n*********Stack operations using linked list*********\n");
   printf("\n---------------------------------------------\n");
   while(choice != 4)
   {
      printf("\n\nChose one from the below options...\n");
      printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");
      printf("\n Enter your choice \n");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1:
         {
            push();
            break;
         }
```

```
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
                display();
                break;
            }
            case 4:
            {
                printf("Exiting....");
                break;
            }
            default:
            {
                printf("Please Enter valid choice ");
            }
    };
}
}
void push ()
{
    int val;
    struct node *ptr = (struct node*)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("not able to push the element");
    }
    else
    {
        printf("Enter the value");
        scanf("%d",&val);
        if(head==NULL)
        {
            ptr->val = val;
            ptr -> next = NULL;
            head=ptr;
        }
        else
        {
            ptr->val = val;
            ptr->next = head;
            head=ptr;
        }
        printf("Item pushed");
    }
}
void pop()
{
    int item;
    struct node *ptr;
    if (head == NULL)
    {
        printf("Underflow");
    }
    else
    {
        item = head->val;
        ptr = head;
        head = head->next;
        free(ptr);
        printf("Item popped");
    }
}
void display()
{
    int i;
    struct node *ptr;
    ptr=head;
    if(ptr == NULL)
    {
        printf("Stack is empty\n");
    }
    else
    {
        printf("Printing Stack elements \n");
        while(ptr!=NULL)
        {
            printf("%d\n",ptr->val);
            ptr = ptr->next;
        }
    }
}
```

OUTPUT:

```
*********Stack operations using linked list********
----------------------------------------------
Chose one from the below options...
1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
1
Enter the value11
Item pushed
Chose one from the below options...
1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
1
Enter the value22
Item pushed
Chose one from the below options...
1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
1
Enter the value33
Item pushed
```

```
Chose one from the below options...

1.Push

2.Pop

3.Show

4.Exit

 Enter your choice

1

Enter the value44

Item pushed

Chose one from the below options...

1.Push

2.Pop

3.Show

4.Exit

 Enter your choice

3

Printing Stack elements

44

33

22

11

Chose one from the below options...

1.Push

2.Pop

3.Show

4.Exit

 Enter your choice

2

Item popped
```

```
Chose one from the below options...
1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
3
Printing Stack elements
33
22
11
Chose one from the below options...
1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
```

LINKED LIST IMPLEMENTATION OF QUEUE

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *front;
struct node *rear;
void insert();
void delete();
void display();
void main ()
{
   int choice;
   while(choice != 4)
   {
      printf("\n*************************Main Menu*****************************\n");
      printf("\n=================================================================\n");
      printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");
      printf("\nEnter your choice ?");
      scanf("%d",& choice);
      switch(choice)
      {
         case 1:
         insert();
         break;
         case 2:
         delete();
         break;
         case 3:
         display();
         break;
         case 4:
         exit(0);
         break;
         default:
         printf("\nEnter valid choice??\n");
      }
   }
}
void insert()
{
   struct node *ptr;
   int item;

   ptr = (struct node *) malloc (sizeof(struct node));
   if(ptr == NULL)
   {
      printf("\nOVERFLOW\n");
      return;
   }
   else
   {
      printf("\nEnter value?\n");
      scanf("%d",&item);
      ptr -> data = item;
      if(front == NULL)
      {
         front = ptr;
```

```
         rear = ptr;
         front -> next = NULL;
         rear -> next = NULL;
      }
      else
      {
         rear -> next = ptr;
         rear = ptr;
         rear->next = NULL;
      }
   }
}
void delete ()
{
   struct node *ptr;
   if(front == NULL)
   {
      printf("\nUNDERFLOW\n");
      return;
   }
   else
   {
      ptr = front;
      front = front -> next;
      free(ptr);
   }
}
void display()
{
   struct node *ptr;
   ptr = front;
```

```
    if(front == NULL)
    {
        printf("\nEmpty queue\n");
    }
    else
    {   printf("\nprinting values .....\n");
        while(ptr != NULL)
        {
            printf("\n%d\n",ptr -> data);
            ptr = ptr -> next;
        }
    }
}
```

OUTPUT:

```
************************Main Menu****************************
===============================================================
1.insert an element
2.Delete an element
3.Display the queue
4.Exit
Enter your choice ?1
Enter value?
11
************************Main Menu****************************
===============================================================
1.insert an element
2.Delete an element
3.Display the queue
4.Exit
Enter your choice ?1
Enter value?
22
************************Main Menu****************************
===============================================================
1.insert an element
2.Delete an element
3.Display the queue
4.Exit
Enter your choice ?1
Enter value?
33
************************Main Menu****************************
===============================================================
1.insert an element
```

```
2.Delete an element
3.Display the queue
4.Exit
Enter your choice ?3
printing values .....
11
22
33
*************************Main Menu*****************************
==================================================================
1.insert an element
2.Delete an element
3.Display the queue
4.Exit
Enter your choice ?2
*************************Main Menu*****************************
==================================================================
1.insert an element
2.Delete an element
3.Display the queue
4.Exit
Enter your choice ?3
printing values .....
22
33
*************************Main Menu*****************************
==================================================================
1.insert an element
2.Delete an element
3.Display the queue
4.Exit
```

IMPLEMENTATION OF POLYNOMIAL ADDITION USING STACK

```
#include<stdio.h>
/* declare structure for polynomial */
struct poly
{
        int coeff;
        int expo;
};
/* declare three arrays p1, p2, p3 of type structure poly.
* each polynomial can have maximum of ten terms
* addition result of p1 and p2 is stored in p3 */
struct poly p1[10],p2[10],p3[10];
/* function prototypes */
int readPoly(struct poly []);
int addPoly(struct poly [],struct poly [],int ,int ,struct poly []);
void displayPoly( struct poly [],int terms);
int main()
{
        int t1,t2,t3;
        /* read and display first polynomial */
        t1=readPoly(p1);
        printf(" \n First polynomial : ");
        displayPoly(p1,t1);
        /* read and display second polynomial */
        t2=readPoly(p2);
        printf(" \n Second polynomial : ");
        displayPoly(p2,t2);
        /* add two polynomials and display resultant polynomial */
        t3=addPoly(p1,p2,t1,t2,p3);
        printf(" \n\n Resultant polynomial after addition : ");
        displayPoly(p3,t3);
```

```
    printf("\n");
    return 0;
}
int readPoly(struct poly p[10])
{
    int t1,i;
    printf("\n\n Enter the total number of terms in the polynomial:");
    scanf("%d",&t1);
    printf("\n Enter the COEFFICIENT and EXPONENT in DESCENDING ORDER\n");
    for(i=0;i<t1;i++)
    {
        printf("   Enter the Coefficient(%d): ",i+1);
        scanf("%d",&p[i].coeff);
        printf("      Enter the exponent(%d): ",i+1);
        scanf("%d",&p[i].expo);       /* only statement in loop */
    }
    return(t1);
}
int addPoly(struct poly p1[10],struct poly p2[10],int t1,int t2,struct poly p3[10])
{
    int i,j,k;
    i=0;
    j=0;
    k=0;
    while(i<t1 && j<t2)
    {
        if(p1[i].expo==p2[j].expo)
        {
            p3[k].coeff=p1[i].coeff + p2[j].coeff;
            p3[k].expo=p1[i].expo;
            i++;
```

```
                j++;
                k++;
        }
        else if(p1[i].expo>p2[j].expo)
        {
                p3[k].coeff=p1[i].coeff;
                p3[k].expo=p1[i].expo;
                i++;
                k++;
        }
        else
        {
                p3[k].coeff=p2[j].coeff;
                p3[k].expo=p2[j].expo;
                j++;
                k++;
        }
}
/* for rest over terms of polynomial 1 */
while(i<t1)
{
        p3[k].coeff=p1[i].coeff;
        p3[k].expo=p1[i].expo;
        i++;
        k++;
}
/* for rest over terms of polynomial 2 */
while(j<t2)
{
        p3[k].coeff=p2[j].coeff;
        p3[k].expo=p2[j].expo;
```

```
                j++;
                k++;
        }
        return(k); /* k is number of terms in resultant polynomial*/
}
void displayPoly(struct poly p[10],int term)
{
        int k;
        for(k=0;k<term-1;k++)
        printf("%d(x^%d)+",p[k].coeff,p[k].expo);
        printf("%d(x^%d)",p[term-1].coeff,p[term-1].expo);
}
```

OUTPUT:

```
Enter the total number of terms in the polynomial:3
 Enter the COEFFICIENT and EXPONENT in DESCENDING ORDER
   Enter the Coefficient(1): 5
      Enter the exponent(1): 3
   Enter the Coefficient(2): 4
      Enter the exponent(2): 2
   Enter the Coefficient(3): 3
      Enter the exponent(3): 1
 First polynomial : 5(x^3)+4(x^2)+3(x^1)

 Enter the total number of terms in the polynomial:3
 Enter the COEFFICIENT and EXPONENT in DESCENDING ORDER
   Enter the Coefficient(1): 8
      Enter the exponent(1): 3
   Enter the Coefficient(2): 6
      Enter the exponent(2): 2
   Enter the Coefficient(3): 7
      Enter the exponent(3): 1
 Second polynomial : 8(x^3)+6(x^2)+7(x^1)

 Resultant polynomial after addition : 13(x^3)+10(x^2)+10(x^1)
```

CONVERSION OF INFIX TO POSTFIX EXPRESSION BY USING STACK

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>      /* for exit() */
#include<ctype.h>     /* for isdigit(char ) */
#include<string.h>
#define SIZE 100
/* declared here as global variable because stack[]
* is used by more than one fucntions */
char stack[SIZE];
int top = -1;
/* define push operation */
void push(char item)
{
        if(top >= SIZE-1)
        {
                printf("\nStack Overflow.");
        }
        else
        {
                top = top+1;
                stack[top] = item;
        }
}
/* define pop operation */
char pop()
{
    char item ;
    if(top <0)
    {
        printf("stack under flow: invalid infix expression");
        getchar();
        /* underflow may occur for invalid expression */
        /* where ( and ) are not matched */
        exit(1);
    }
    else
    {
        item = stack[top];
        top = top-1;
        return(item);
    }
}
/* define function that is used to determine whether any symbol is operator or not
(that is symbol is operand)
* this fucntion returns 1 if symbol is opreator else return 0 */
int is_operator(char symbol)
{
    if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol =='-')
    {
        return 1;
    }
    else
    {
    return 0;
    }
```

```
}
/* define fucntion that is used to assign precendence to operator.
* Here ^ denotes exponent operator.
* In this fucntion we assume that higher integer value
* means higher precendence */
int precedence(char symbol)
{
        if(symbol == '^')/* exponent operator, highest precedence*/
        {
                return(3);
        }
        else if(symbol == '*' || symbol == '/')
        {
                return(2);
        }
        else if(symbol == '+' || symbol == '-')         /* lowest precedence */
        {
                return(1);
        }
        else
        {
                return(0);
        }
}
void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
        int i, j;
        char item;
        char x;
```

```
    push('(');                    /* push '(' onto stack */
    strcat(infix_exp,")");              /* add ')' to infix expression */
    i=0;
    j=0;
    item=infix_exp[i];        /* initialize before loop*/
    while(item != '\0')       /* run loop till end of infix expression */
    {
        if(item == '(')
        {
            push(item);
        }
        else if( isdigit(item) || isalpha(item))
        {
            postfix_exp[j] = item;          /* add operand symbol to postfix expr */
            j++;
        }
        else if(is_operator(item) == 1)       /* means symbol is operator */
        {
            x=pop();
            while(is_operator(x) == 1 && precedence(x)>= precedence(item))
            {
                postfix_exp[j] = x;               /* so pop all higher precendence
operator and */
                j++;
                x = pop();                   /* add them to postfix expresion */
            }
            push(x);
            /* because just above while loop will terminate we have
            oppped one extra item
```

```
                for which condition fails and loop terminates, so that one*/
                push(item);             /* push current oprerator symbol onto stack */
        }
        else if(item == ')')        /* if current symbol is ')' then */
        {
                x = pop();              /* pop and keep popping until */
                while(x != '(')             /* '(' encounterd */
                {
                        postfix_exp[j] = x;
                        j++;
                        x = pop();
                }
        }
        else
        { /* if current symbol is neither operand not '(' nor ')' and nor
                operator */
                printf("\nInvalid infix Expression.\n");        /* the it is illegeal  symbol*/
                getchar();
                exit(1);
        }
        i++;
        item = infix_exp[i]; /* go to next symbol of infix expression */
} /* while loop ends here */
if(top>0)
{
        printf("\nInvalid infix Expression.\n");        /* the it is illegeal  symbol */
        getchar();
        exit(1);
}
```

```
	if(top>0)
	{
		printf("\nInvalid infix Expression.\n");	/* the it is illegeal  symbol */
		getchar();
		exit(1);
	}
	postfix_exp[j] = '\0'; /* add sentinel else puts() fucntion */
	/* will print entire postfix[] array upto SIZE */
/* main function begins */
int main()
{
	char infix[SIZE], postfix[SIZE];	/* declare infix string and postfix string */
	/* why we asked the user to enter infix expression
	* in parentheses ( )
	* What changes are required in porgram to
	* get rid of this restriction since it is not
	* in algorithm
	* */
	printf("ASSUMPTION: The infix expression contains single letter variables and single digit constants only.\n");
	printf("\nEnter Infix expression : ");
	gets(infix);
	InfixToPostfix(infix,postfix);		/* call to convert */
	printf("Postfix Expression: ");
	puts(postfix);			/* print postfix expression */
	return 0;
}
```

OUTPUT:

```
ASSUMPTION: The infix expression contains single letter variables and single digit constants only.

Enter Infix expression : (A+B)*(C+D)

Postfix Expression: AB+CD+*
```

IMPLEMENTATION OF BINARY TREE AND IT'S OPERATION

PROGRAM:

```
/* Tree Traversal */
#include <stdio.h>
#include <stdlib.h>
typedef struct node
{
int data;
struct node *left;
struct node *right;
}node;
int count=1;
node *insert(node *tree,int digit)
{
if(tree == NULL)
{
tree = (node *)malloc(sizeof(node));
tree->left = tree->right=NULL;
tree->data = digit;
count++;
}
else if(count%2 == 0)
tree->left = insert(tree->left, digit);
else
tree->right = insert(tree->right, digit);
return tree;
}
void preorder(node *t)
{
if(t != NULL)
{
```

```
printf(" %d", t->data);
preorder(t->left);
preorder(t->right);
}
}
void postorder(node *t)
{
if(t != NULL)
{
postorder(t->left);
postorder(t->right);
printf(" %d", t->data);
}
}
void inorder(node *t)
{
if(t != NULL)
{
inorder(t->left);
printf(" %d", t->data);
inorder(t->right);
}
}
main()
{
node *root = NULL;
int digit;
puts("Enter integer:To quit enter 0");
scanf("%d", &digit);
while(digit != 0)
{
```

```
root=insert(root,digit);
scanf("%d",&digit);
}
printf("\nThe preorder traversal of tree is:\n");
preorder(root);
printf("\nThe inorder traversal of tree is:\n");
inorder(root);
printf("\nThe postorder traversal of tree is:\n");
postorder(root);
getch();
}
```

OUTPUT:

```
Enter integer:To quit enter 0

11

22

33

44

55

66

77

88

99

0

The preorder traversal of tree is:

 11 22 44 66 88 33 55 77 99

The inorder traversal of tree is:

 88 66 44 22 11 33 55 77 99

The postorder traversal of tree is:

 88 66 44 22 99 77 55 33 11
```

IMPLEMENTATION OF BINARY SEARCH TREE

PROGRAM:

```
/*
 * C Program to Construct a Binary Search Tree and perform deletion, inorder traversal on it
 */
#include <stdio.h>
#include <stdlib.h>
struct btnode
{
   int value;
   struct btnode *l;
   struct btnode *r;
}*root = NULL, *temp = NULL, *t2, *t1;
void delete1();
void insert();
void delete();
void inorder(struct btnode *t);
void create();
void search(struct btnode *t);
void preorder(struct btnode *t);
void postorder(struct btnode *t);
void search1(struct btnode *t,int data);
int smallest(struct btnode *t);
int largest(struct btnode *t);
int flag = 1;
void main()
{
   int ch;
   printf("\nOPERATIONS ---");
   printf("\n1 - Insert an element into tree\n");
   printf("2 - Delete an element from the tree\n");
   printf("3 - Inorder Traversal\n");
   printf("4 - Preorder Traversal\n");
   printf("5 - Postorder Traversal\n");
   printf("6 - Exit\n");
   while(1)
   {
     printf("\nEnter your choice : ");
     scanf("%d", &ch);
     switch (ch)
     {
     case 1:
       insert();
       break;
     case 2:
       delete();
       break;
     case 3:
       inorder(root);
       break;
     case 4:
       preorder(root);
       break;
     case 5:
       postorder(root);
       break;
     case 6:
       exit(0);
     default :
       printf("Wrong choice, Please enter correct choice  ");
       break;
     }
```

```
    }
}
/* To insert a node in the tree */
void insert()
{
    create();
    if (root == NULL)
        root = temp;
    else
        search(root);
}
/* To create a node */
void create()
{
    int data;
    printf("Enter data of node to be inserted : ");
    scanf("%d", &data);
    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
    temp->value = data;
    temp->l = temp->r = NULL;
}
/* Function to search the appropriate position to insert the new node */
void search(struct btnode *t)
{
    if ((temp->value > t->value) && (t->r != NULL))      /* value more than root node value insert at
right */
        search(t->r);
    else if ((temp->value > t->value) && (t->r == NULL))
        t->r = temp;
    else if ((temp->value < t->value) && (t->l != NULL))    /* value less than root node value insert at
left */
        search(t->l);
```

```
   else if ((temp->value < t->value) && (t->l == NULL))
      t->l = temp;
}
/* recursive function to perform inorder traversal of tree */
void inorder(struct btnode *t)
{
   if (root == NULL)
   {
      printf("No elements in a tree to display");
      return;
   }
   if (t->l != NULL)
      inorder(t->l);
   printf("%d -> ", t->value);
   if (t->r != NULL)
      inorder(t->r);
}
/* To check for the deleted node */
void delete()
{
   int data;
   if (root == NULL)
   {
      printf("No elements in a tree to delete");
      return;
   }
   printf("Enter the data to be deleted : ");
   scanf("%d", &data);
   t1 = root;
   t2 = root;
   search1(root, data);
```

```
}
/* To find the preorder traversal */
void preorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display");
        return;
    }
    printf("%d -> ", t->value);
    if (t->l != NULL)
        preorder(t->l);
    if (t->r != NULL)
        preorder(t->r);
}
/* To find the postorder traversal */
void postorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display ");
        return;
    }
    if (t->l != NULL)
        postorder(t->l);
    if (t->r != NULL)
        postorder(t->r);
    printf("%d -> ", t->value);
}
/* Search for the appropriate position to insert the new node */
void search1(struct btnode *t, int data)
```

```
{
   if ((data>t->value))
   {
      t1 = t;
      search1(t->r, data);
   }
   else if ((data < t->value))
   {
      t1 = t;
      search1(t->l, data);
   }
   else if ((data==t->value))
   {
      delete1(t);
   }
}
/* To delete a node */
void delete1(struct btnode *t)
{
   int k
   /* To delete leaf node */
   if ((t->l == NULL) && (t->r == NULL))
   {
      if (t1->l == t)
      {
         t1->l = NULL;
      }
      else
      {
         t1->r = NULL;
      }
```

```
    t = NULL;
    free(t);
    return;
}
/* To delete node having one left hand child */
else if ((t->r == NULL))
{
    if (t1 == t)
    {
        root = t->l;
        t1 = root;
    }
    else if (t1->l == t)
    {
        t1->l = t->l;
    }
    else
    {
        t1->r = t->l;
    }
    t = NULL;
    free(t);
    return;
}
/* To delete node having right hand child */
else if (t->l == NULL)
{
    if (t1 == t)
    {
        root = t->r;
        t1 = root;
        }
        else if (t1->r == t)
            t1->r = t->r;
        else
            t1->l = t->r;
        t == NULL;
        free(t);
        return;
    }
    /* To delete node having two child */
    else if ((t->l != NULL) && (t->r != NULL))
    {
        t2 = root;
.        if (t->r != NULL)
        {
            k = smallest(t->r);
            flag = 1;
        }
        else
        {
            k =largest(t->l);
            flag = 2;
        }
        search1(root, k);
        t->value = k;
    }
}
/* To find the smallest element in the right sub tree */
int smallest(struct btnode *t)
{
    t2 = t;
    if (t->l != NULL)
    {
        t2 = t;
        return(smallest(t->l));
    }
    else
        return (t->value);
}
/* To find the largest element in the left sub tree */
int largest(struct btnode *t)
{
    if (t->r != NULL)
    {
        t2 = t;
        return(largest(t->r));
    }
    else
        return(t->value);
}
```

OUTSIDE:

```
OPERATIONS ---
1 - Insert an element into tree
2 - Delete an element from the tree
3 - Inorder Traversal
4 - Preorder Traversal
5 - Postorder Traversal
6 - Exit
Enter your choice : 1
Enter data of node to be inserted : 12
Enter your choice : 1
Enter data of node to be inserted : 24
Enter your choice : 1
Enter data of node to be inserted : 36
Enter your choice : 1
Enter data of node to be inserted : 1
Enter your choice : 1
Enter data of node to be inserted : 2
Enter your choice : 1
Enter data of node to be inserted : 7
Enter your choice : 3
1 -> 2 -> 7 -> 12 -> 24 -> 36 ->
Enter your choice : 4
12 -> 1 -> 2 -> 7 -> 24 -> 36 ->
Enter your choice : 5
7 -> 2 -> 1 -> 36 -> 24 -> 12 ->
Enter your choice : 2
Enter the data to be deleted : 36
Enter your choice : 3
1 -> 2 -> 7 -> 12 -> 24 ->
Enter your choice : 4
12 -> 1 -> 2 -> 7 -> 24 ->
```

```
Enter your choice : 5

7 -> 2 -> 1 -> 24 -> 12 ->

Enter your choice :
```

IMPLEMENTATION OF AVL TREE

PROGRAM:

```
#include<stdio.h>
typedef struct node
{
        int data;
        struct node *left,*right;
        int ht;
}node;
node *insert(node *,int);
node *Delete(node *,int);
void preorder(node *);
void inorder(node *);
int height( node *);
node *rotateright(node *);
node *rotateleft(node *);
node *RR(node *);
node *LL(node *);
node *LR(node *);
node *RL(node *);
int BF(node *);
int main()
{
        node *root=NULL;
        int x,n,i,op;
        do
        {
                printf("\n1)Create:");
                printf("\n2)Insert:");
                printf("\n3)Delete:");
                printf("\n4)Print:");
```

```
		printf("\n5)Quit:");
		printf("\n\nEnter Your Choice:");
		scanf("%d",&op);
		switch(op)
		{
			case 1: printf("\nEnter no. of elements:");
					scanf("%d",&n);
					printf("\nEnter tree data:");
					root=NULL;
					for(i=0;i<n;i++)
					{
						scanf("%d",&x);
						root=insert(root,x);
					}
					break;
			case 2: printf("\nEnter a data:");
					scanf("%d",&x);
					root=insert(root,x);
					break;
			case 3: printf("\nEnter a data:");
					scanf("%d",&x);
					root=Delete(root,x);
					break;
			case 4: printf("\nPreorder sequence:\n");
					preorder(root);
					printf("\n\nInorder sequence:\n");
					inorder(root);
					printf("\n");
					break;
		}
	}while(op!=5);
```

```
    return 0;
}
node * insert(node *T,int x)
{
    if(T==NULL)
    {
        T=(node*)malloc(sizeof(node));
        T->data=x;
        T->left=NULL;
        T->right=NULL;
    }
    else
        if(x > T->data)          // insert in right subtree
        {
            T->right=insert(T->right,x);
            if(BF(T)==-2)
                if(x>T->right->data)
                    T=RR(T);
                else
                    T=RL(T);
        }
        else
            if(x<T->data)
            {
                T->left=insert(T->left,x);
                if(BF(T)==2)
                    if(x < T->left->data)
                        T=LL(T);
                    else
                        T=LR(T);
            }
        T->ht=height(T);
        return(T);
}
node * Delete(node *T,int x)
{
    node *p;
    if(T==NULL)
    {
        return NULL;
    }
    else
        if(x > T->data)         // insert in right subtree
        {
            T->right=Delete(T->right,x);
            if(BF(T)==2)
                if(BF(T->left)>=0)
                    T=LL(T);
                else
                    T=LR(T);
        }
        else
            if(x<T->data)
            {
                T->left=Delete(T->left,x);
                if(BF(T)==-2)    //Rebalance during windup
                    if(BF(T->right)<=0)
                        T=RR(T);
                    else
                        T=RL(T);
            }
            else
            {
                //data to be deleted is found
                if(T->right!=NULL)
                {       //delete its inorder succesor
                    p=T->right;
                    while(p->left!= NULL)
                        p=p->left;
                    T->data=p->data;
                    T->right=Delete(T->right,p->data);
                    if(BF(T)==2)//Rebalance during windup
                        if(BF(T->left)>=0)
                            T=LL(T);
                        else
                            T=LR(T);\
                }
                else
                    return(T->left);
            }
    T->ht=height(T);
    return(T);
}
int height(node *T)
{
    int lh,rh;
    if(T==NULL)
        return(0);
    if(T->left==NULL)
        lh=0;
    else
        lh=1+T->left->ht;
        if(T->right==NULL)
                rh=0;
        else
                rh=1+T->right->ht;
        if(lh>rh)
                return(lh);
        return(rh);
}
node * rotateright(node *x)
{
        node *y;
        y=x->left;
        x->left=y->right;
        y->right=x;
        x->ht=height(x);
        y->ht=height(y);
        return(y);
}
node * rotateleft(node *x)
{
        node *y;
        y=x->right;
        x->right=y->left;
        y->left=x;
        x->ht=height(x);
        y->ht=height(y);
        return(y);
}
node * RR(node *T)
{
        T=rotateleft(T);
        return(T);
}
node * LL(node *T)
{
        T=rotateright(T);
        return(T);
}
node * LR(node *T)
{
        T->left=rotateleft(T->left);
        T=rotateright(T);
        return(T);
}
node * RL(node *T)
{
        T->right=rotateright(T->right);
        T=rotateleft(T);
        return(T);
}
int BF(node *T)
{
        int lh,rh;
        if(T==NULL)
                return(0);
        if(T->left==NULL)
                lh=0;
        else
                lh=1+T->left->ht;
        if(T->right==NULL)
                rh=0;
        else
                rh=1+T->right->ht;
        return(lh-rh);
}
void preorder(node *T)
{
        if(T!=NULL)
        {
                printf("%d(Bf=%d)",T->data,BF(T));
                preorder(T->left);
                preorder(T->right);
        }
}
void inorder(node *T)
{
        if(T!=NULL)
        {
                inorder(T->left);
                printf("%d(Bf=%d)",T->data,BF(T));
                inorder(T->right);
        }
}
```

OUTPUT:

```
1)Create:

2)Insert:

3)Delete:

4)Print:

5)Quit:

Enter Your Choice:2

Enter a data:11

1)Create:

2)Insert:

3)Delete:

4)Print:

5)Quit:

Enter Your Choice:2

Enter a data:22

1)Create:

2)Insert:

3)Delete:

4)Print:

5)Quit:

Enter Your Choice:2

Enter a data:33

1)Create:

2)Insert:

3)Delete:

4)Print:

5)Quit:

Enter Your Choice:2

Enter a data:44

1)Create:

2)Insert:
```

```
3)Delete:

4)Print:

5)Quit:

Enter Your Choice:2

Enter a data:55

1)Create:

2)Insert:

3)Delete:

4)Print:

5)Quit:

Enter Your Choice:4

Preorder sequence:

22(Bf=-1)11(Bf=0)44(Bf=0)33(Bf=0)55(Bf=0)

Inorder sequence:

11(Bf=0)22(Bf=-1)33(Bf=0)44(Bf=0)55(Bf=0)

1)Create:

2)Insert:

3)Delete:

4)Print:

5)Quit:

Enter Your Choice:3

Enter a data:55

1)Create:

2)Insert:

3)Delete:

4)Print:

5)Quit:

Enter Your Choice:4

Preorder sequence:

22(Bf=-1)11(Bf=0)44(Bf=1)33(Bf=0)
```

```
Inorder sequence:

11(Bf=0)22(Bf=-1)33(Bf=0)44(Bf=1)

1)Create:

2)Insert:

3)Delete:

4)Print:

5)Quit:

Enter Your Choice:
```

IMPLEMENTATION OF HEAP USING PRIORITY QUEUE

PROGRAM:

```
/* Priority Queue using Heap */
#include<stdio.h>
#include<math.h>
#define MAX 100
void swap(int*, int*);
void display(int[],int);
void insert(int[],int,int,int);
int del_hi_priori(int[],int,int);
int main()
{
   int lb,choice,num,n,a[MAX],data,s;
   choice = 0;
   n=0; //Represents number of nodes in the queue
   lb=0; //Lower bound of the array is initialized to 0
   while(choice != 4)
   {
      printf(".....MAIN MENU.....\n");
      printf("\n1.Insert.\n");
      printf("2.Delete.\n");
      printf("3.Display.\n");
      printf("4.Quit.\n");
      printf("\nEnter your choice : ");
      scanf("%d",&choice);
      switch(choice)
      {
      case 1:
         printf("Enter data to be inserted : ");
         scanf("%d",&data);
         insert(a,n,data,lb);
```

```
            n++;
            break;
        case 2:
            s=del_hi_priori(a,n+1,lb);
            if(s!=0)
            printf("The deleted value is : %d",s);
            if(n>0)
                n--;
            break;
        case 3:
            printf("\n");
            display(a,n);
            break;
        case 4:
            return 0;
        default:
            printf("Invalid choice\n");
        }
        printf("\n\n");
    }
return 0;
}
//-------INSERT-------
void insert(int a[],int heapsize,int data,int lb)
{
    int i,p;
    int parent(int);
    if(heapsize==MAX)
    {
        printf("Queue Is Full!!n");
        return;
    }
    i=lb+heapsize;
    a[i]=data;
    while(i>lb&&a[p=parent(i)]<a[i])
    {
        swap(&a[p],&a[i]);
        i=p;
    }
}
//-------DELETE-------
int del_hi_priori(int a[],int heapsize,int lb)
{
    int data,i,l,r,max_child,t;
    int left(int);
    int right(int);
    if(heapsize==1)
    {
        printf("Queue Is Empty!!\n");
        return 0;
    }
    t=a[lb];
    swap(&a[lb],&a[heapsize-1]);
    i=lb;
    heapsize--;
    while(1)
    {
        if((l=left(i))>=heapsize)
            break;
        if((r=right(i))>=heapsize)
            max_child=l;
        else
```

```
            max_child=(a[l]>a[r])?l:r;
         if(a[i]>=a[max_child])
            break;
         swap(&a[i],&a[max_child]);
         i=max_child;
      }
   return t;
   }
   //Returns Parent Index
   int parent(int i)
   {
      float p;
      p=((float)i/2.0)-1.0;
      return ceil(p);
   }
   //Return Leftchild Index
   int left(int i)
   {
      return 2*i+1;
   }
   //Return Rightchild Index
   int right(int i)
   {
      return 2*i+2;
   }
   //-------DISPLAY-------
   void display(int a[],int n)
   {
      int i;
      if(n==0)
      {
        printf("Queue Is Empty!!\n");
        return;
    }
    for(i=0;i<n;i++)
        printf("%d ",a[i]);
    printf("\n");
}
//-------SWAP-------
void swap(int*p,int*q)
{
    int temp;
    temp=*p;
    *p=*q;
    *q=temp;
}
```

OUTPUT:

```
.....MAIN MENU.....

1.Insert.

2.Delete.

3.Display.

4.Quit.

Enter your choice : 1

Enter data to be inserted : 11

.....MAIN MENU.....

1.Insert.

2.Delete.

3.Display.

4.Quit.

Enter your choice : 1

Enter data to be inserted : 45

.....MAIN MENU.....

1.Insert.

2.Delete.

3.Display.

4.Quit.

Enter your choice : 1

Enter data to be inserted : 23

.....MAIN MENU.....

1.Insert.

2.Delete.

3.Display.

4.Quit.

Enter your choice : 1

Enter data to be inserted : 32

.....MAIN MENU.....

1.Insert.
```

```
2.Delete.

3.Display.

4.Quit.

Enter your choice : 3

45 32 23 11

.....MAIN MENU.....

1.Insert.

2.Delete.

3.Display.

4.Quit.

Enter your choice : 2

The deleted value is : 45

.....MAIN MENU.....

1.Insert.

2.Delete.

3.Display.

4.Quit.

Enter your choice : 3

32 11 23

.....MAIN MENU.....

1.Insert.

2.Delete.

3.Display.

4.Quit.

Enter your choice :
```

IMPLEMENTATION OF GRAPH REPRESENTATION AND TRAVERSAL METHODS(BFS)

PROGRAM:

```
/* Graph Traversal – BFS */

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

#define initial 1

#define waiting 2

#define visited 3

int n;

int adj[MAX][MAX];

int state[MAX];

void create_graph();

void BF_Traversal();

void BFS(int v);

int queue[MAX], front = -1,rear = -1;

void insert_queue(int vertex);

int delete_queue();

int isEmpty_queue();

int main()

{

create_graph();

BF_Traversal();

return 0;

}

void BF_Traversal()

{

int v;

for(v=0; v<n; v++)

state[v] = initial;
```

```
printf("Enter Start Vertex for BFS: ");
scanf("%d", &v);
BFS(v);
}
void BFS(int v)
{
int i;
insert_queue(v);
state[v] = waiting;
printf("BFS Traversal : ");
while(!isEmpty_queue())
{
v = delete_queue( );
printf("%d ", v);
state[v] = visited;
for(i=0; i<n; i++)
{
if(adj[v][i] == 1 && state[i] == initial)
{
insert_queue(i);
state[i] = waiting;
}
}
}
printf("\n");
}
void insert_queue(int vertex)
{
if(rear == MAX-1)
printf("Queue Overflow\n");
else
```

```
{
if(front == -1)
front = 0;
rear = rear+1;
queue[rear] = vertex ;
}
}
int isEmpty_queue()
{
if(front == -1 || front > rear)
return 1;
else
return 0;
}
int delete_queue()
{
int delete_item;
if(front == -1 || front > rear)
{
printf("Queue Underflow\n");
exit(1);
}
delete_item = queue[front];
front = front+1;
return delete_item;
}
void create_graph()
{
int count,max_edge,origin,destin;
printf("Enter number of vertices : ");
scanf("%d", &n);
```

```
max_edge = n * (n-1);

for(count=1; count<=max_edge; count++)

{

printf("Enter edge %d( -1 -1 to quit ) : ",count);

scanf("%d %d", &origin, &destin);

if((origin == -1) && (destin == -1))

break;

if(origin>=n || destin>=n || origin<0 || destin<0)

{

printf("Invalid edge!\n");

count--;

}

else

adj[origin][destin] = 1;

}

}
```

OUTPUT:

```
Enter number of vertices : 9

Enter edge 1( -1 -1 to quit ) : 0 1

Enter edge 2( -1 -1 to quit ) : 0 3

Enter edge 3( -1 -1 to quit ) : 0 4

Enter edge 4( -1 -1 to quit ) : 1 2

Enter edge 5( -1 -1 to quit ) : 1 4

Enter edge 6( -1 -1 to quit ) : 2 5

Enter edge 7( -1 -1 to quit ) : 3 4

Enter edge 8( -1 -1 to quit ) : 3 6

Enter edge 9( -1 -1 to quit ) : 4 5

Enter edge 10( -1 -1 to quit ) : 4 7

Enter edge 11( -1 -1 to quit ) : 6 4

Enter edge 12( -1 -1 to quit ) : 6 7

Enter edge 13( -1 -1 to quit ) : 7 8

Enter edge 14( -1 -1 to quit ) : -1 -1

Enter Start Vertex for BFS: 0

BFS Traversal : 0 1 3 4 2 6 5 7 8
```

IMPLEMENTATION OF GRAPH REPRESENTATION AND TRAVERSAL METHODS(DFS)

PROGRAM:

```
/* DFS on undirected graph */
#include <stdio.h>
#include <stdlib.h>
#define true 1
#define false 0
#define MAX 5
struct Vertex
{
char label;
int visited;
};
int stack[MAX];
int top = -1;
struct Vertex* lstVertices[MAX];
static int adjMatrix[MAX][MAX];
int vertexCount = 0;
void push(int item)
{
stack[++top] = item;
}
int pop()
{
return stack[top--];
}
int peek()
{
return stack[top];
}
```

```
int isStackEmpty()
{
return top == -1;
}
void addVertex(char label)
{
struct Vertex* vertex = (struct Vertex*)
malloc(sizeof(struct Vertex));
vertex->label = label;
vertex->visited = false;
lstVertices[vertexCount++] = vertex;
}
void addEdge(int start, int end)
{
adjMatrix[start][end] = 1;
adjMatrix[end][start] = 1;
}
void displayVertex(int vertexIndex)
{
printf("%c ", lstVertices[vertexIndex]->label);
}
int getAdjUnvisitedVertex(int vertexIndex)
{
int i;
for(i = 0; i < vertexCount; i++)
{
if(adjMatrix[vertexIndex][i] == 1 &&
lstVertices[i]->visited == false)
return i;
}
return -1;
```

```
}
void depthFirstSearch()
{
int i;
lstVertices[0]->visited = true;
displayVertex(0);
push(0);
while(!isStackEmpty())
{
int unvisitedVertex = getAdjUnvisitedVertex(peek());
if(unvisitedVertex == -1)
pop();
else
{
lstVertices[unvisitedVertex]->visited = true;
displayVertex(unvisitedVertex);
push(unvisitedVertex);
}
}
for(i = 0;i < vertexCount;i++)
lstVertices[i]->visited = false;
}
main()
{
int i, j, n, edges, orgn, destn;
char ch;
printf("Enter no. of vertices : ");
scanf("%d", &n);
edges = n * (n - 1);
printf("Enter Vertex Labels : \n");
for (i=0; i<n; i++)
```

```
{
fflush(stdin);
scanf("%c", &ch);
addVertex(ch);
}
for(i=0; i<edges; i++)
{
printf("Enter edge ( -1 -1 to quit ) : ");
scanf("%d %d", &orgn, &destn);
if((orgn == -1) && (destn == -1))
break;
if(orgn>=n || destn>=n || orgn<0 || destn<0)
printf("Invalid edge!\n");
else
addEdge(orgn, destn);
}
printf("\nDepth First Search: ");
depthFirstSearch();
```

OUTPUT:

```
Enter no. of vertices : 5

Enter Vertex Labels :

S

A

B

C

D

Enter edge ( -1 -1 to quit ) : 0 1

Enter edge ( -1 -1 to quit ) : 0 3

Enter edge ( -1 -1 to quit ) : 0 2

Enter edge ( -1 -1 to quit ) : 1 4

Enter edge ( -1 -1 to quit ) : 2 4

Enter edge ( -1 -1 to quit ) : 3 4

Enter edge ( -1 -1 to quit ) : -1 -1


Depth First Search: S A D B C
```

IMPLEMENTATION OF LINEAR SEARCH

PROGRAM:

```
#include <stdio.h>
int main()
{
  int array[100], search, c, n;

  printf("Enter number of elements in array\n");
  scanf("%d", &n);

  printf("Enter %d integer(s)\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  printf("Enter a number to search\n");
  scanf("%d", &search);

  for (c = 0; c < n; c++)
  {
    if (array[c] == search)    /* If required element is found */
    {
      printf("%d is present at location %d.\n", search, c+1);
      break;
    }
  }
  if (c == n)
    printf("%d isn't present in the array.\n", search);

  return 0;
}
```

OUTPUT:

```
Enter number of elements in array

5

Enter 5 integer(s)

45

62

12

34

43

Enter a number to search

45

45 is present at location 1.
```

IMPLEMENTATION OF BINARY SEARCH

PROGRAM:

```
#include <stdio.h>
int main()
{
 int c, first, last, middle, n, search, array[100];
 printf("Enter number of elements\n");
 scanf("%d", &n);
 printf("Enter %d integers\n", n);
 for (c = 0; c < n; c++)
  scanf("%d", &array[c]);
 printf("Enter value to find\n");
 scanf("%d", &search);
 first = 0;
 last = n - 1;
 middle = (first+last)/2;
 while (first <= last) {
  if (array[middle] < search)
   first = middle + 1;
  else if (array[middle] == search) {
   printf("%d found at location %d.\n", search, middle+1);
   break;
  }
  else
   last = middle - 1;
  middle = (first + last)/2;}
 if (first > last)
  printf("Not found! %d isn't present in the list.\n", search);
 return 0;}
```

OUTPUT:

```
Enter number of elements

5

Enter 5 integers

100

34

26

75

84

Enter value to find

75

75 found at location 4.
```

IMPLEMENTATION OF INSERTION SORT

PROGRAM:

```
/* C Program to sort an array in ascending order using Insertion Sort */
#include <stdio.h>
int main()
{
   int n, i, j, temp;
   int arr[64];
   printf("Enter number of elements\n");
   scanf("%d", &n)
   printf("Enter %d integers\n", n);
   for (i = 0; i < n; i++)
   {
      scanf("%d", &arr[i]);
   }
   for (i = 1 ; i <= n - 1; i++)
   {   j = i;
         while ( j > 0 && arr[j-1] > arr[j])
         {
            temp     = arr[j];
            arr[j]   = arr[j-1];
            arr[j-1] = temp;
            j--;
         }
   }
   printf("Sorted list in ascending order:\n");
   for (i = 0; i <= n - 1; i++)
   { printf("%d\n", arr[i]);
   }
   return 0;
}
```

OUTPUT:

```
Enter number of elements

5

Enter 5 integers

12

1

32

23

17

Sorted list in ascending order:

1

12

17

23

32
```

IMPLEMANTATION OF MERGE SORT

PROGRAM:

```
/* Merge sort */
#include <stdio.h>
#include <conio.h>
void merge(int [],int ,int ,int );
void part(int [],int ,int );
int size;
main()
{
        int i, arr[30];
        printf("Enter total no. of elements : ");
        scanf("%d", &size);
        printf("Enter array elements : ");
        for(i=0; i<size; i++)
                scanf("%d", &arr[i]);
        part(arr, 0, size-1);
        printf("\n Merge sorted list : ");
        for(i=0; i<size; i++)
                printf("%d ",arr[i]);
        getch();
}
void part(int arr[], int min, int max)
{
        int i, mid;
        if(min < max)
        {
                mid = (min + max) / 2;
                part(arr, min, mid);
                part(arr, mid+1, max);
                merge(arr, min, mid, max);
        }
        if (max-min == (size/2)-1)
        {
                printf("\n Half sorted list : ");
                for(i=min; i<=max; i++)
                        printf("%d ", arr[i]);
        }
}
void merge(int arr[],int min,int mid,int max)
{
        int tmp[30];
        int i, j, k, m;
        j = min;
        m = mid + 1;
        for(i=min; j<=mid && m<=max; i++)
        {
                if(arr[j] <= arr[m])
                {
                        tmp[i] = arr[j];
                        j++;
                }
                else
                {
                        tmp[i] = arr[m];
                        m++;
                }
        }
        if(j > mid)
        {
                for(k=m; k<=max; k++)
                {
                        tmp[i] = arr[k];
                        i++;
                }
        }
        else
        {
                for(k=j; k<=mid; k++)
                {
                        tmp[i] = arr[k];
                        i++;
                }
        }
        for(k=min; k<=max; k++)
                arr[k] = tmp[k];
}
```

OUTPUT:

```
Enter total no. of elements : 5

Enter array elements : 32

11

26

77

9


 Half sorted list : 11 32

 Half sorted list : 9 77

 Merge sorted list : 9 11 26 32 77
```

IMPLEMENTATION OF QUICK SORT

PROGRAM:

```
/* Quick Sort */
#include <stdio.h>
#include <conio.h>
void qsort(int arr[20], int fst, int last);
main()
{
int arr[30];
int i, size;
printf("Enter total no. of the elements : ");
scanf("%d", &size);
printf("Enter total %d elements : \n", size);
for(i=0; i<size; i++)
scanf("%d", &arr[i]);
qsort(arr,0,size-1);
printf("\n Quick sorted elements \n");
for(i=0; i<size; i++)
printf("%d\t", arr[i]);
getch();
}
void qsort(int arr[20], int fst, int last)
{
int i, j, pivot, tmp;
if(fst < last)
{
pivot = fst;
i = fst;
j = last;
while(i < j)
{
```

```
while(arr[i] <=arr[pivot] && i<last)

i++;

while(arr[j] > arr[pivot])

j--;

if(i <j )

{

tmp = arr[i];

arr[i] = arr[j];

arr[j] = tmp;

}

}

tmp = arr[pivot];

arr[pivot] = arr[j];

arr[j] = tmp;

qsort(arr, fst, j-1);

qsort(arr, j+1, last);

}

}
```

OUTPUT:

```
Enter total no. of the elements : 5

Enter total 5 elements :

56

12

34

23

78


 Quick sorted elements

12     23     34     56     78
```

IMPLEMENTATION OF HASHING TECHNIQUE

PROGRAM:

```
/* Open hashing */
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
main()
{
int a[MAX], num, key, i;
char ans;
int create(int);
void linearprobing(int[], int, int);
void display(int[]);
printf("\nCollision handling by linear probing\n\n");
for(i=0; i<MAX; i++)
a[i] = -1;
do
{
printf("\n Enter number:");
scanf("%d", &num);
key = create(num);
linearprobing(a, key, num);
printf("\nwish to continue?(y/n):");
ans = getch();
} while( ans == 'y');
display(a);
}
int create(int num)
{
int key;
key = num % 10;
```

```
return key;
}
void linearprobing(int a[MAX], int key, int num)
{
int flag, i, count = 0;
void display(int a[]);
flag = 0;
if(a[key] == -1)
a[key] = num;
else
{
i=0;
while(i < MAX)
{
if(a[i] != -1)
count++;
i++;
}
if(count == MAX)
{
printf("hash table is full");
display(a);
getch();
exit(1);
}
for(i=key+1; i<MAX; i++)
if(a[i] == -1)
{
a[i] = num;
flag = 1;
break;
}
for(i=0; i<key && flag==0; i++ )
if(a[i] == -1)
{
a[i] = num;
flag = 1;
break;
}
}
}
void display(int a[MAX])
{
int i;
printf("\n Hash table is:");
for(i=0; i<MAX; i++)
printf("\n %d\t\t%d",i,a[i]);
}
```

OUTPUT:

```
Collision handling by linear probing
 Enter number:1
wish to continue?(y/n):
 Enter number:26
wish to continue?(y/n):
 Enter number:62
wish to continue?(y/n):
 Enter number:93
wish to continue?(y/n):
 Enter number:84
wish to continue?(y/n):
 Enter number:15
wish to continue?(y/n):
 Enter number:76
wish to continue?(y/n):
 Enter number:98
wish to continue?(y/n):
 Enter number:26
wish to continue?(y/n):
 Enter number:199
wish to continue?(y/n):
 Enter number:1234
hash table is full
 Hash table is:
 0          199
 1          1
 2          62
 3          93
 4          84
 5          15
```

6	26
7	76
8	98
9	26