KRISTU JAYANTI
(DEEMED TO BE UNIVERSITY)
Under Section 3 of UGC Act 1956
A CMI INSTITUTION | BENGALURU | INDIA

Department of Computer Science (PG)

Practical Record on
DATA STRUCTURES AND ALGORITHMS PRACTICAL

(25MCC1L211)

I Semester MCA

For the academic year 2025 - 2026

Submitted by

ADYAPRANA PRADHAN

25MCAC57

KRISTU JAYANTI (DEEMED TO BE UNIVERSITY)

K. Narayanapura, Kothanur Post, Bengaluru 560077

KRISTU JAYANTI
(DEEMED TO BE UNIVERSITY)
Under Section 3 of UGC Act 1956
A CMI INSTITUTION | BENGALURU | INDIA

DEPARTMENT OF COMPTUER SCIENCE (PG)

Master of Computer Applications

CERTIFICATE

This is to certify that **ADYAPRANA PRADHAN** bearing roll number **25MCAC57** has satisfactorily completed the practical programs for the course **DATA STRUCTURES AND ALGORITHMS PRACTICAL, 25MCC1L211** during the
academic year 2025 – 2026.

Date: Faculty incharge

Head of the Department

Examiners (Name & Signature)

1. ______________________ Center: Kristu Jayanti University

Date:

2. ______________________

KRISTU JAYANTI
(DEEMED TO BE UNIVERSITY)
Under Section 3 of UGC Act 1956
A CMI INSTITUTION | BENGALURU | INDIA

Table of Contents

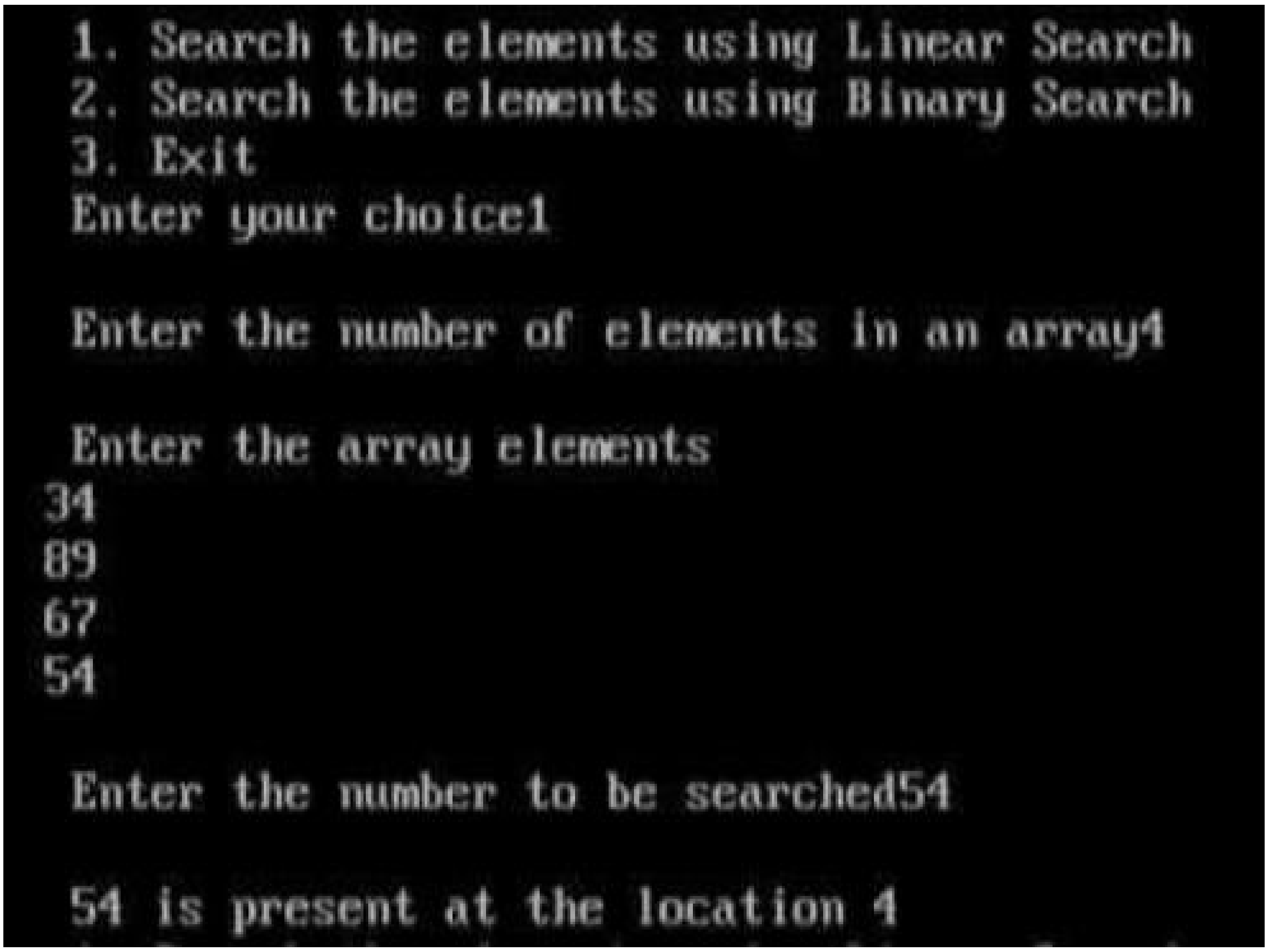
PROGRAM 1:

Write a menu driven program to implement linear and binary search to find the location of the first occurrence of an item.

```
#include <stdio.h>

#include <stdlib.h>


void linear();

void binary();


int main()

  { int

  choice; do

  {

    printf("\n1. Search the elements using Linear Search");

    printf("\n2. Search the elements using Binary Search");

    printf("\n3. Exit");

    printf("\nEnter your choice: ");

    scanf("%d", &choice);


    switch (choice)

      { case 1:

        linear();  // Calling the linear search function

        break;

      case 2:

        binary();  // Calling the binary search function

        break;

      case 3:

        printf("\nExiting program...\n");

        exit(0);  // Exit the program

        break;

      default:

        printf("\nInvalid choice. Please choose between 1, 2, or 3.\n");
    }
    } while (choice != 3);  // Continue until user chooses to exit

    return 0;
}

void linear() {
    int n, c, array[100], search;
    printf("\nEnter the number of elements in the array: ");
    scanf("%d", &n);
    printf("\nEnter the array elements: ");
    for (c = 0; c < n; c++) {
        scanf("%d", &array[c]);
    }
    printf("\nEnter the number to be searched: ");
    scanf("%d", &search);

    for (c = 0; c < n; c++) {
        if (array[c] == search) {
            printf("\n%d is present at location %d\n", search, c + 1);
            return;  // Exit once the element is found
        }
    }

    printf("\n%d is not present in the array\n", search);
}

void binary() {
    int array[100], i, j, n, mid, temp, flag = 0, low = 0, high, item, c;
    printf("\nEnter the number of elements in the array: ");
    scanf("%d", &n);
    printf("\nEnter the array elements: ");
    for (c = 0; c < n; c++) {
        scanf("%d", &array[c]);
    }

    // Sorting the array to perform binary search
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - 1 - i; j++)
            { if (array[j] > array[j + 1])
            {
                temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }

    printf("\nEnter the number to be searched: ");
    scanf("%d", &item);

    high = n - 1;
    while (low <= high)
        { mid = (low + high) /
        2;
        if (array[mid] == item) {
            printf("\n%d is found at position %d\n", item, mid + 1);  // Print position (1-based index)
            flag = 1;
            break;
        } else if (item > array[mid]) {
            low = mid + 1;  // Move the low pointer
        } else {
            high = mid - 1;  // Move the high pointer
        }
    }
```

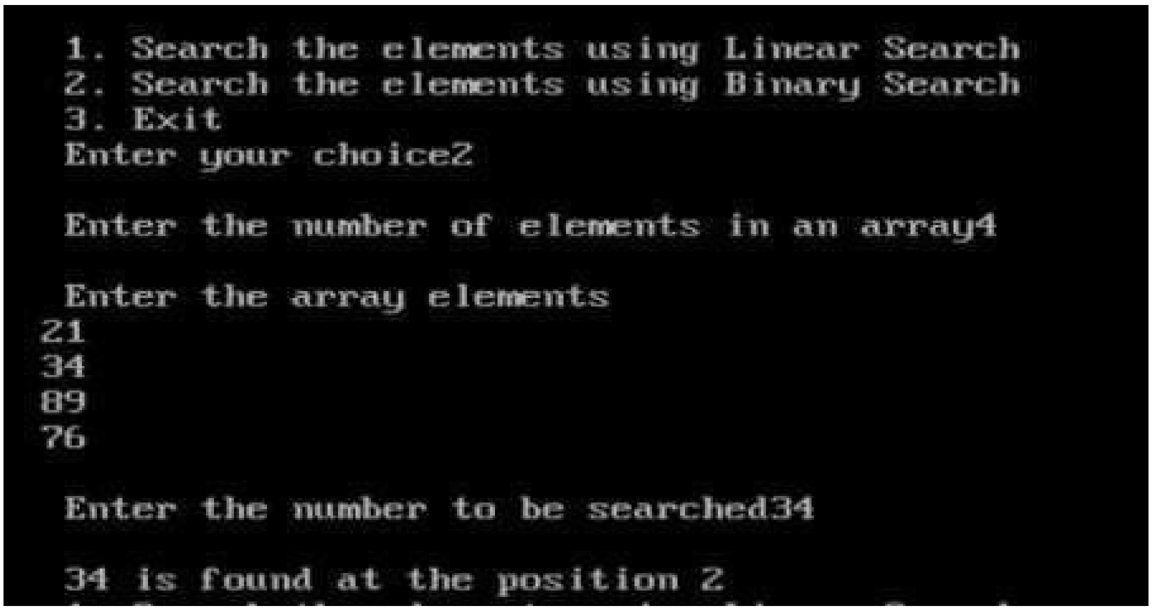
```
  if (flag == 0) {
        printf("\n%d is not found\n", item);
    }
}
```

OUTPUT:



```
 1. Search the elements using Linear Search
 2. Search the elements using Binary Search
 3. Exit
 Enter your choice1

 Enter the number of elements in an array4

 Enter the array elements
34
89
67
54

 Enter the number to be searched54

 54 is present at the location 4
```

1.1 Searching using linear search



```
 1. Search the elements using Linear Search
 2. Search the elements using Binary Search
 3. Exit
 Enter your choice2

 Enter the number of elements in an array4

 Enter the array elements
21
34
89
76

 Enter the number to be searched34

 34 is found at the position 2
```

1.2 Seaching using Binary Search

PROGRAM 2:

Write a menu driven program to sort the array in ascending / descending order using Quick Sort and Merge Sort.

```
#include <stdio.h>
#include <conio.h>  // For clrscr() and getch()
#include <stdlib.h>

// Function declarations
void quick_sort(int[], int, int);
int partition(int[], int, int);
void mergesort(int[], int, int);
void merge(int[], int, int, int, int);

// Quick Sort Function
void quick_sort(int a[100], int l, int u)
  { int j;
  if (l < u) {
      j = partition(a, l, u);   // Partitioning the array quick_sort(a,
      l, j - 1);  // Recursively sort the left subarray
      quick_sort(a, j + 1, u);  // Recursively sort the right subarray
  }
}

// Partition function for Quick Sort
int partition(int a[100], int l, int u) {
  int v, i, j, temp;
  v = a[l];  // Pivot element
  i = l;
  j = u + 1;
  do {
      do {
          i++;  // Move right until we find an element larger than the pivot
```

```
        } while (a[i] < v && i <= u);

        do {
            j--;  // Move left until we find an element smaller than the pivot
        } while (a[j] > v);

        if (i < j) {
            // Swap the elements
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    } while (i < j);

    // Swap pivot element with the element at index j
    a[l] = a[j];
    a[j] = v;
    return j;  // Return the pivot position
}

// Merge Sort Function
void mergesort(int a[100], int i, int j)
    { int mid;
    if (i < j) {
        mid = (i + j) / 2;  // Find the middle index
        mergesort(a, i, mid);  // Recursively sort the left half
        mergesort(a, mid + 1, j);  // Recursively sort the right half
        merge(a, i, mid, mid + 1, j);  // Merge the two sorted halves
    }
}

// Merge function to merge two sorted subarrays
```

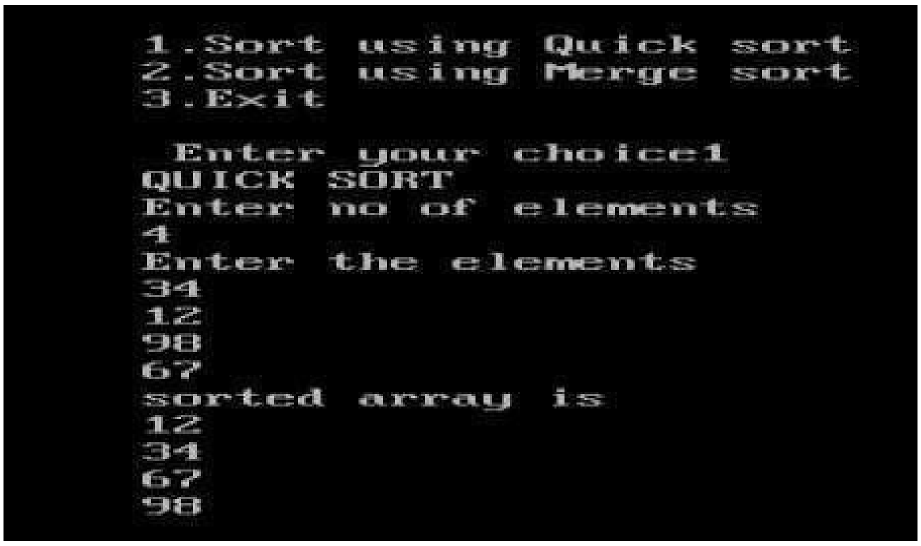
```
void merge(int a[100], int i1, int j1, int i2, int j2)
  { int temp[100];
  int i = i1, j = i2, k = 0;

  while (i <= j1 && j <= j2)
     { if (a[i] < a[j]) {
        temp[k++] = a[i++];
     } else {
        temp[k++] = a[j++];
     }
  }

  // Copy remaining elements of the left subarray, if any
  while (i <= j1) {
     temp[k++] = a[i++];
  }

  // Copy remaining elements of the right subarray, if any
  while (j <= j2) {
     temp[k++] = a[j++];
  }

  // Copy the sorted elements back to the original array
  for (i = i1, j = 0; i <= j2; i++, j++) {
     a[i] = temp[j];
  }
}

// Main function
void main() {
  int a[100], n, i, op;
```

```
clrscr();  // Clear the screen (specific to Turbo C)

while (1) {
    // Display menu
    printf("\n1. Sort using Quick sort");
    printf("\n2. Sort using Merge sort");
    printf("\n3. Exit\n");
    printf("\nEnter your choice: ");
    scanf("%d", &op);

    switch (op)
        { case 1:
            printf("QUICK SORT\n");
            printf("Enter number of elements: ");
            scanf("%d", &n);
            printf("Enter the elements: ");
            for (i = 0; i < n; i++) {
                scanf("%d", &a[i]);
            }
            quick_sort(a, 0, n - 1);
            printf("Sorted array is:\n");
            for (i = 0; i < n; i++) {
                printf("%d ", a[i]);
            }
            printf("\n");
            break;

        case 2:
            printf("MERGE SORT\n");
            printf("Enter number of elements: ");
            scanf("%d", &n);
            printf("Enter the elements: ");
```

```
                for (i = 0; i < n; i++)
                    { scanf("%d", &a[i]);
                }
                mergesort(a, 0, n - 1);
                printf("Sorted array is:\n");
                for (i = 0; i < n; i++) {
                    printf("%d ", a[i]);
                }
                printf("\n");
                break;

            case 3:
                exit(0);  // Exit the program

            default:
                printf("Invalid Choice\n");
        }
    }
}
```

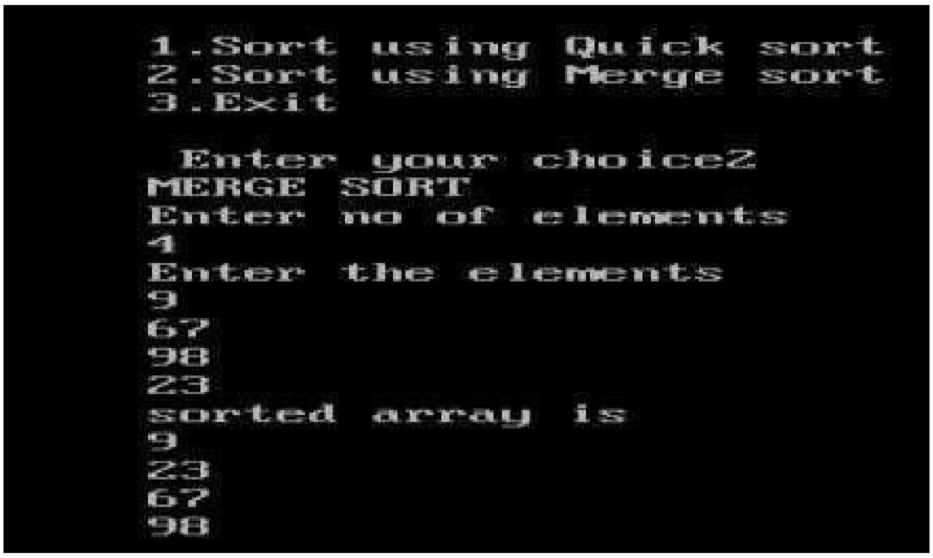
Output:

1.1 QUICK SORT



```
1.Sort using Quick sort
2.Sort using Merge sort
3.Exit

 Enter your choice1
QUICK SORT
Enter no of elements
4
Enter the elements
34
12
98
67
sorted array is
12
34
67
98
```

1.2 MERGE SORT



```
1.Sort using Quick sort
2.Sort using Merge sort
3.Exit

 Enter your choice2
MERGE SORT
Enter no of elements
4
Enter the elements
9
67
98
23
sorted array is
9
23
67
98
```

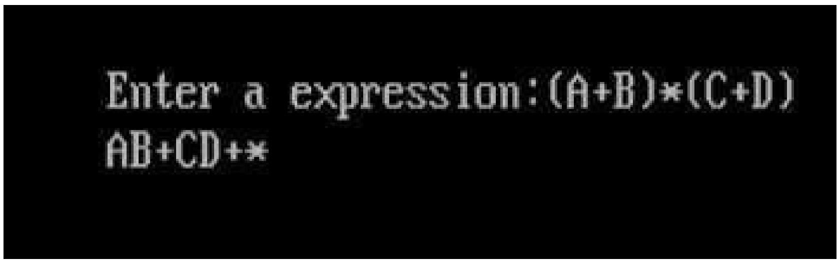
PROGRAM 3:

Write a program to convert the given infix expression into its postfix form.

```
#include <stdio.h>

#include <stdlib.h>

#include <ctype.h> // For isalpha()


// Stack to hold operators

char stack[100];

int top = -1;


// Function declarations

int prec(char c);

void push(char c);

char pop();

void infixtopostfix(char str[]);


int main()

    { char

    str[100];

    printf("\nEnter an infix expression: ");

    scanf("%s", str);

    infixtopostfix(str);

    return 0;

}


// Function to return precedence of operators

int prec(char c) {

    if (c == '^')

        return 3;

    else if (c == '/' || c == '*')

        return 2;

    else if (c == '+' || c == '-')
```

```
        return 1;
    else
        return -1;  // For operands or invalid characters
}

// Push function to add an element to the stack
void push(char c) {
    stack[++top] = c;
}

// Pop function to remove and return the top element from the stack
char pop() {
    if (top == -1) {
        return -1; // Return -1 if stack is empty
    }
    return stack[top--];
}

// Function to convert infix to postfix
void infixtopostfix(char str[]) {
    int i = 0;
    while (str[i] != '\0') {
        // If the character is an operand (a letter or digit), print it
        if (isalnum(str[i])) {
            printf("%c", str[i]);
        }
        // If the character is '(', push it to the stack
        else if (str[i] == '(') {
            push(str[i]);
        }
        // If the character is ')', pop from the stack and print until '(' is found
        else if (str[i] == ')') {
            while (top != -1 && stack[top] != '(') {
                printf("%c", pop());
            }
            if (top != -1) {
                pop(); // Pop '(' from the stack
            }
        }
        // If the character is an operator, handle precedence and push it to the stack
        else {
            while (top != -1 && prec(str[i]) <= prec(stack[top]))
                { printf("%c", pop());
            }
            push(str[i]);
        }
        i++;
    }

    // Pop all remaining operators from the stack
    while (top != -1) {
        printf("%c", pop());
    }
}
```

Output:



```
Enter a expression:(A+B)*(C+D)
AB+CD+*
```

PROGRAM: 4

Write a menu driven program to create a linked list and to preform insert and delete operations.

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
struct node
{
int data;
struct node *next;
};
struct node *head;
void beginsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
void main ()
{
int choice =0;
clrscr();
while(choice != 9)
{
printf("\nLINKED LIST\n");
printf("\nChoose one option from the following list ...\n");
printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete from
Beginning\n5.Delete from last\n6.Delete node after specified location\n7.Search for an
element\n8.Show\n9.Exit\n");
```

```
printf("\nEnter your choice?\n");
scanf("\n%d",&choice);
switch(choice)
{
case 1: beginsert();
break;
case 2: lastinsert();
break;

case 3: randominsert();
break;

case 4: begin_delete();
break;

case 5: last_delete();
break;

case 6: random_delete();
break;

case 7: search();
break;

case 8: display();
break;

case 9: exit(0);
break;
default:
printf("Please enter valid choice..");
}
```

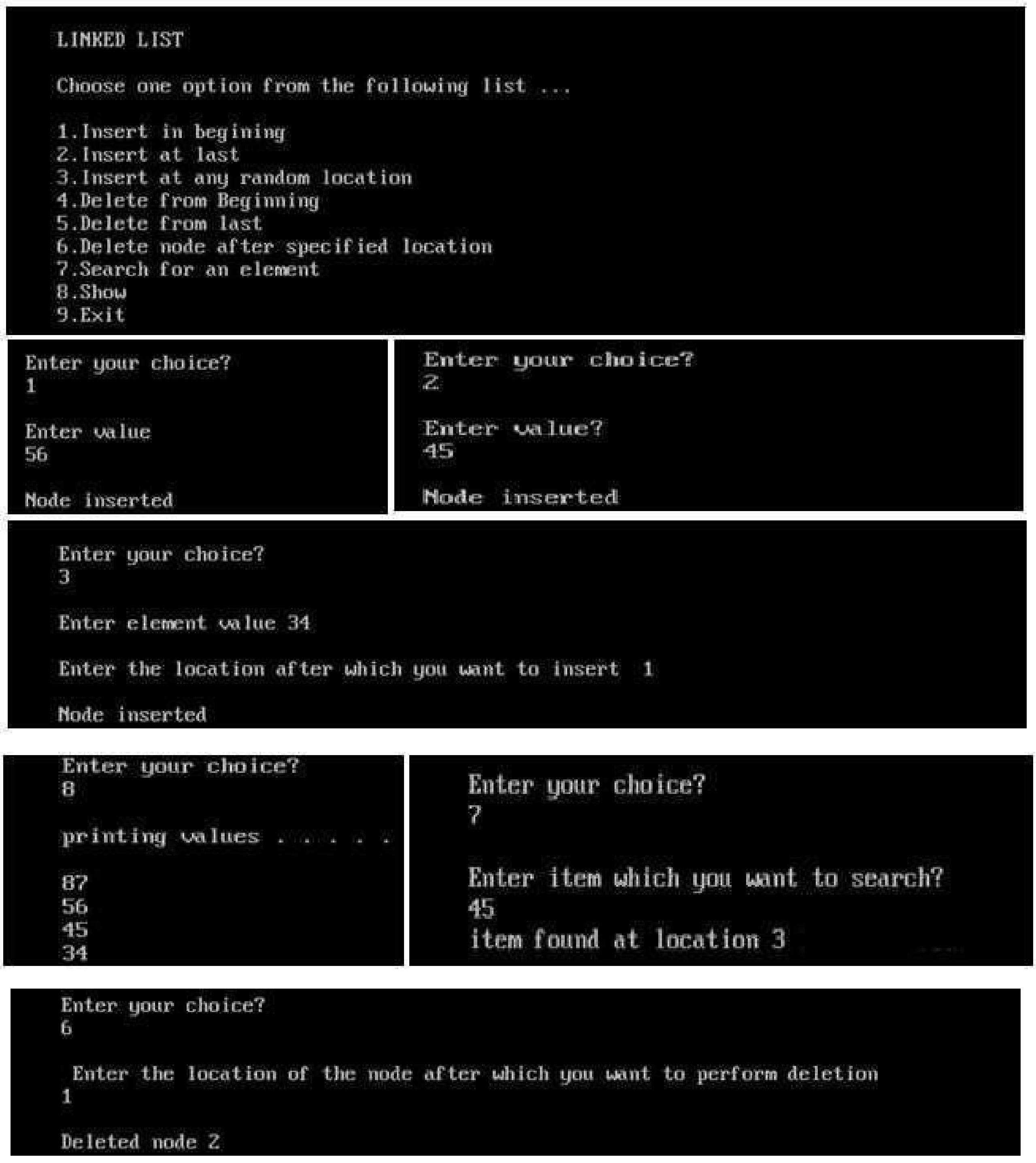
```
}
}
void beginsert()
{
struct node *ptr;
intptr = (struct node *) malloc(sizeof(struct node *));
if(ptr == NULL)
{
}
else
{
}
printf("\nOVERFLOW");
printf("\nEnter  value\n");
scanf("%d",&item);
ptr->data  =  item;
ptr->next = head;
head = ptr;
printf("\nNode inserted");
}
void lastinsert()
{
struct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

```
scanf("%d",&item);
ptr->data = item;
if(head == NULL)
{
ptr -> next = NULL;
head = ptr;
printf("}
else
{
temp = head;
while (temp -> next != NULL)
{
temp = temp -> next;
}
temp->next = ptr;
ptr->next = NULL;
printf("\nNode inserted");
}
}
}
void randominsert()
{
int i,loc,item;
struct node *ptr, *temp;
ptr = (struct node *) malloc (sizeof(struct node));
if(ptr == NULL)
{
}
else
{
printf("\nOVERFLOW");
printf("\nEnter element value");
scanf("%d",&item);
ptr->data = item;
printf("\nEnter the location after which you want to insert ");
scanf("\n%d",&loc);
temp=head;
for(i=0;i<loc;i++)
{
tempif(temp == NULL)
{
printf("\ncan't insert\n");
return;
}
}
ptr ->next = temp ->next;
temp ->next = ptr;
printf("\nNode inserted");
}
}
void begin_delete()
{
struct node *ptr;
if(head == NULL)
{
}
else
{
}
}
printf("\nList is empty\n");
ptr = head;
head = ptr->next;
free(ptr);
```

```
printf("\nNode deleted from the begining ...\n");
void last_delete()
{
struct node *ptr,*ptr1;
if(head == NULL)
{
printf("\nlist is empty");
}
else{
}
else
{
head = NULL;
free(head);
printf("\nOnly node of the list deleted ...\n");
ptr = head;
while(ptr->next != NULL)
{
ptr1 = ptr;
ptr = ptr ->next;
}
ptr1->next = NULL;
free(ptr);
printf("\nDeleted Node from the last ...\n");
}
}
void random_delete()
{
struct node *ptr,*ptr1;
int loc,i;
printf("\n Enter the location of the node after which you want to perform deletion \n");
scanf("%d",&loc);
```

```
ptr=head;
for(i=0;i<loc;i++)
{
ptr1 = ptr;
ptr = ptr->next;
if(ptr == NULL)
{
printf("\nCan't delete");
return;
}
}ptr1 ->next = ptr ->next;
free(ptr);
printf("\nDeleted node %d ",loc+1);
}
void search()
{
struct node *ptr;
int item,i=0,flag;
ptr = head;
if(ptr == NULL)
{
}
else
{
printf("\nEmpty List\n");
printf("\nEnter item which you want to search?\n");
scanf("%d",&item);
while (ptr!=NULL)
{
if(ptr->data == item)
{
}
else
{
}
i++;
printf("item found at location %d ",i+1);
flag=0;
flag=1;
ptr = ptr -> next;
}
if(flag==1)
{
printf("Item not found\n");
}}
}
void display()
{
struct node *ptr;
ptr = head;
if(ptr == NULL)
{
}
else
{
printf("Nothing to print");
printf("\nprinting values...........\n");
while (ptr!=NULL)
{
printf("\n%d",ptr->data);
ptr = ptr -> next;
}
}
}
```

Output:



```
LINKED LIST

Choose one option from the following list ...

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete node after specified location
7.Search for an element
8.Show
9.Exit
```

```
Enter your choice?
1

Enter value
56

Node inserted
```

```
Enter your choice?
2

Enter value?
45

Node inserted
```

```
Enter your choice?
3

Enter element value 34

Enter the location after which you want to insert  1

Node inserted
```

```
Enter your choice?
8

printing values . . . . .

87
56
45
34
```

```
Enter your choice?
7

Enter item which you want to search?
45
item found at location 3
```

```
Enter your choice?
6

 Enter the location of the node after which you want to perform deletion
1

Deleted node 2
```

PROGRAM:5

Write a menu driven program to perform insert and delete operations in a circular linked list.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *head;
void beginsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
void main ()
{
int choice =0;
while(choice != 7)
{
printf("\nCIRCULAR LINKED LIST");
printf("\nChoose one option from the following list");
printf("\n1.Insert in begining\n2.Insert at last\n3.Delete from Beginning\n4.Delete from
last\n5.Search for an element\n6.Show\n7.Exit\n");
printf("\nEnter your choice?");
scanf("\n%d",&choice);
switch(choice)
{
case 1:
case 2:
beginsert();
break;case 3:
case 4:
case 5:
case 6:
case 7:
lastinsert();
break;
begin_delete();
break;
last_delete();
break;
search();
break;
display();
break;
exit(0);
break;
default:
printf("Please enter valid choice..");
}
}
}
```

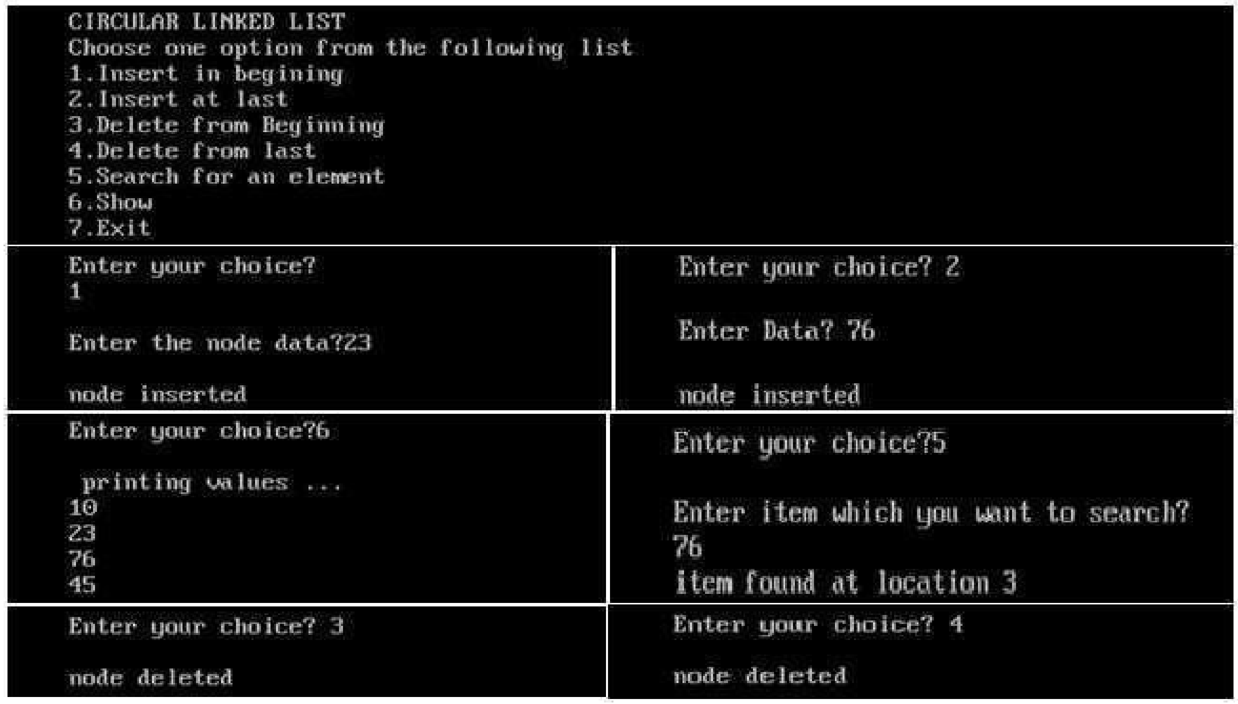
```
void beginsert()
{
struct node *ptr,*temp;
int item;
ptr = (struct node *)malloc(sizeof(struct node));
if(ptr == NULL)
{
}
else
{
printf("\nOVERFLOW");
printf("\nEnter the node data?");
scanf("%d",&item);ptr -> data = item;
if(head == NULL)
{
}
else
{
}
head = ptr;
ptr -> next = head;
temp = head;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

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

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

```
if(flag != 0)
{
printf("Item not found\n");
}}
}
void display()
{
struct node *ptr;
ptr=head;
if(head == NULL)
{
}
else
{
printf("\nnothing to print");
printf("\n printing values ... \n");
while(ptr -> next != head)
{
printf("%d\n", ptr -> data);
ptr = ptr -> next;
}
printf("%d\n", ptr -> data);
}
}
```

Output:



```
CIRCULAR LINKED LIST
Choose one option from the following list
1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit
```

```
Enter your choice?
1

Enter the node data?23

node inserted
```

```
Enter your choice? 2

Enter Data? 76

node inserted
```

```
Enter your choice?6

 printing values ...
10
23
76
45
```

```
Enter your choice?5

Enter item which you want to search?
76
item found at location 3
```

```
Enter your choice? 3

node deleted
```

```
Enter your choice? 4

node deleted
```

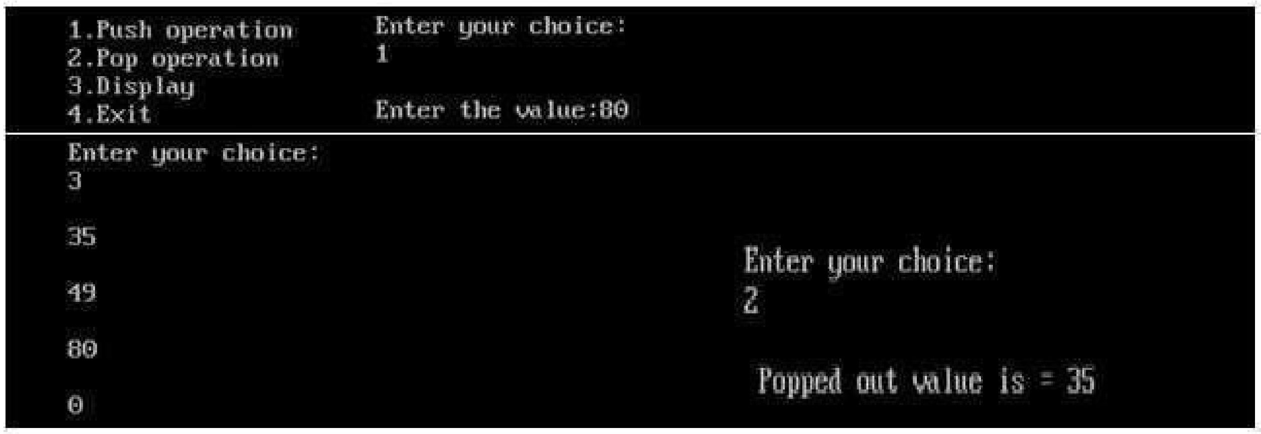
PROGRAM 6:

Write a menu driven program to perform operations on a stack (linked list implementation) (push, pop, and peek).

```
#include <stdio.h>
#include<stdlib.h>
struct stack
{
int num;
struct stack *next;
};
struct stack *top;
void push();
void pop();
void display();
int main()
{
int choice;
do
{
printf("\n1.Push operation");
printf("\n2.Pop operation");
printf("\n3.Display ");
printf("\n4.Exit ");
printf("\nEnter your choice:\n");
scanf("%d",&choice);
switch(choice)
{
case 1: push();
break;
case 2: pop();
break;
case 3: display();
break;
case 4: exit(0);
}}while(choice>=1 && choice<=4);
return 0;
}
void push()
{
struct stack *temp;
temp=(struct stack*)malloc(sizeof(struct stack));
if(temp==NULL)
{
puts("\n OVERFLOW CONDITION");
return;
}
printf("\nEnter the value:");
scanf("%d",&temp->num);
if(top==NULL)
{
top=temp;
}
else
{
temp->next=top;
top=temp;
}
```

```
}
void pop()
{
struct stack *temp;
if(top==NULL)
{
}
else
{
printf("\nUNDERFLOW COMDITION");
return;
printf("temp=top;
top=top->next;
free(temp);
}
}
void display()
{
struct stack *temp;
if(top==NULL)
{
printf("\n NO ELEMENTS TO DISPLAY>>\n STACK IS EMPTY");
return;
}
for(temp=top;temp!=NULL;temp=temp->next)
printf("\n%d\n",temp->num);
}
```

Output:



```
1.Push operation      Enter your choice:
2.Pop operation       1
3.Display
4.Exit                Enter the value:80
```

```
Enter your choice:
3

35                                        Enter your choice:
                                          2
49

80                                         Popped out value is = 35

0
```

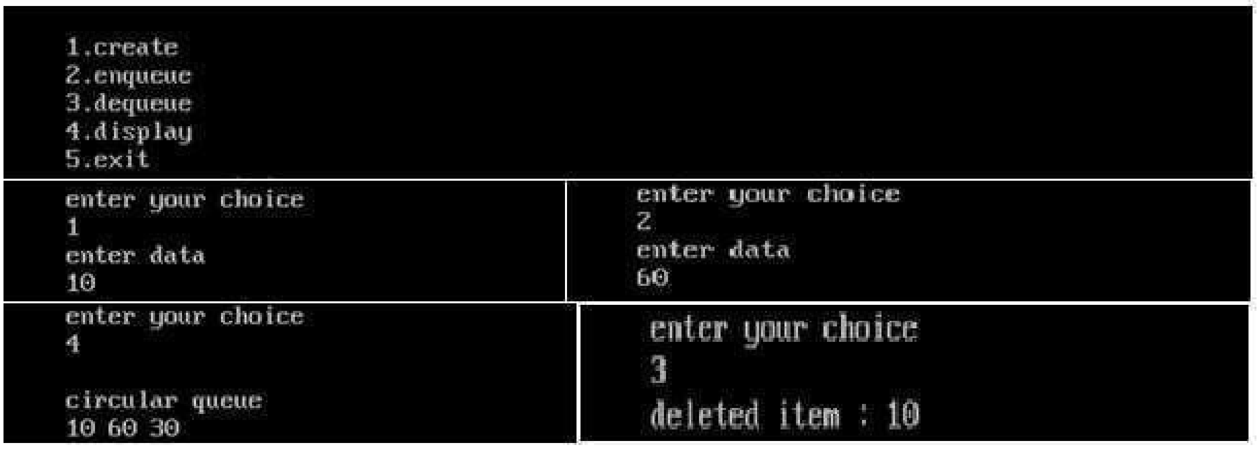
PROGRAM 7:

Write a menu driven program to perform operations on a circular queue (enqueue, dequeue and peek).

```
#include<stdio.h>
#include<stdlib.h>
typedef struct queue
{
int data;
struct queue *link;
}queue;
queue *front=NULL,*rear=NULL;
void enqueue();
int dequeue();
void display();
queue *create();
int main()
{
int ch=6;
while(ch!=5)
{
printf("\n1.create\n2.enqueue\n3.dequeue\n4.display\n5.exit\nenter your choice\n");
scanf("%d",&ch);
switch (ch)
{
case 1: if(front==NULL)
{
}
else
break;
rear=front=create();
printf("queue already exists\n");
case 2: enqueue();
break;
case 3: if(front==NULL)
printf("queue underflow\n");
break;
printf("deleted item : %d ",dequeue());
case 4 :display();
break;
default:
break;
}
}
return 0;
}
queue *create()
{
queue *temp=(queue*)malloc(sizeof(queue));
printf("enter data\n");
scanf("%d",&temp->data);
temp->link=front;
return temp;
}
void enqueue()
{
rear->link=create();
rear=rear->link;
```

```
}
int dequeue()
{
queue *temp=front;
int ch=front->data;
if(front==rear)
front=rear=NULL;
else
{
}
front=front->link;
rear->link=front;
free(temp);
return ch;
}
void display()
{
queue *temp=front;
printf("\ncircular queue\n");
if(temp==rear)
printf("%d",temp->data);
else if(front==NULL)
printf("queue is empty\n");
else
{
int i=0;
do
{
printf("%d ",temp->data);
temp=temp->link;i++;
} while (temp!=front);
}
}
```

Output:



```
1.create
2.enqueue
3.dequeue
4.display
5.exit
```

```
enter your choice
1
enter data
10
```

```
enter your choice
2
enter data
60
```

```
enter your choice
4

circular queue
10 60 30
```

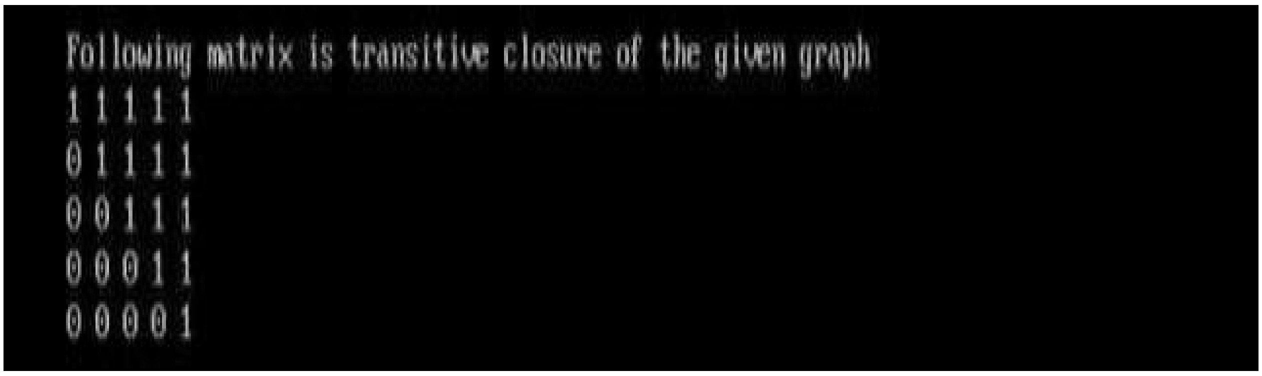
```
enter your choice
3
deleted item : 10
```

PROGRAM 8:

Write a program to compute the transitive closure of a given direted graph using Warshall's algorithm and prove its efficiency.

```
#include <stdio.h>
#define V 5

void printMatrix(int reach[V][V]);

// Function to compute transitive closure using Warshall's Algorithm
void transitiveClosure(int graph[V][V]) {
    int reach[V][V], i, j, k;

    // Initialize reachability matrix same as input graph matrix
    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            reach[i][j] = graph[i][j];

    // Warshall's algorithm
    for (k = 0; k < V; k++) {
        for (i = 0; i < V; i++)
            { for (j = 0; j < V; j++)
            {
                reach[i][j] = reach[i][j] || (reach[i][k] && reach[k][j]);
            }
        }
    }

    printMatrix(reach);
}

// Function to print the matrix
void printMatrix(int reach[V][V]) {
    int i, j;
    printf("\nTransitive closure of the given graph is:\n");
    for (i = 0; i < V; i++) {
        for (j = 0; j < V; j++)
            { printf("%d ", reach[i][j]);
        }
        printf("\n");
    }
}

int main() {
    /* Example directed graph represented as adjacency matrix
       0 → 1, 0 → 2
       1 → 2, 1 → 4
       2 → 3
       3 → 4
    */

    int graph[V][V] = {
        {1, 1, 1, 0, 0},
        {0, 1, 1, 0, 1},
        {0, 0, 1, 1, 0},
        {0, 0, 0, 1, 1},
        {0, 0, 0, 0, 1}
    };

    transitiveClosure(graph);
    return 0;
}
```

Output:



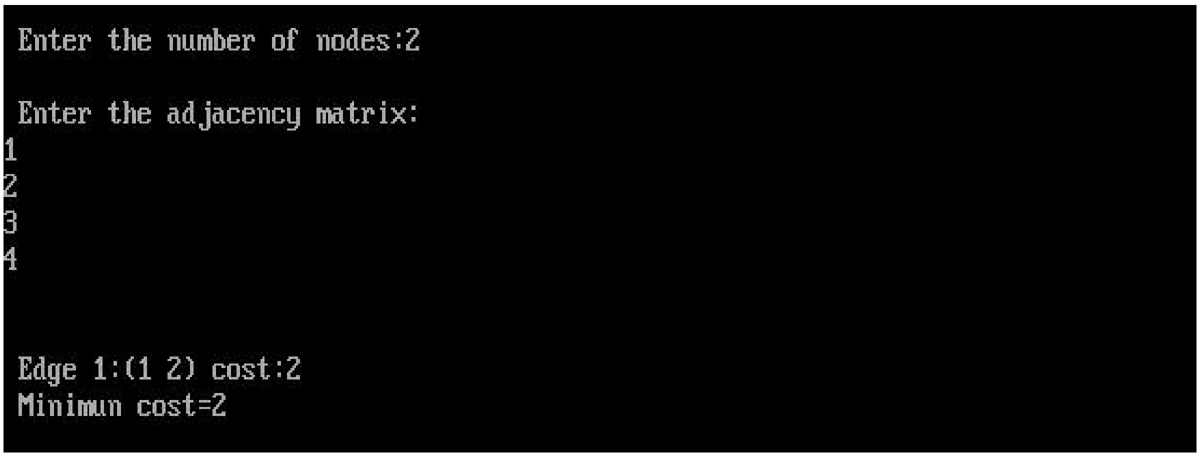
```
Following matrix is transitive closure of the given graph
1 1 1 1 1
0 1 1 1 1
0 0 1 1 1
0 0 0 1 1
0 0 0 0 1
```

PROGRAM 9:

Write a program to find the Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.

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

int a, b, u, v, n, i, j, ne = 1;
int visited[10] = {0};
int min, mincost = 0, cost[10][10];

int main() {
   printf("\nEnter the number of nodes: ");
   scanf("%d", &n);

   printf("\nEnter the adjacency matrix:\n");
   for (i = 1; i <= n; i++) {
      for (j = 1; j <= n; j++)
         { scanf("%d",
         &cost[i][j]); if (cost[i][j]
         == 0)
            cost[i][j] = 999; // Represent infinity
      }
   }

   visited[1] = 1;

   printf("\nThe edges of the Minimum Spanning Tree are:\n");

   while (ne < n) {
        min = 999;

        for (i = 1; i <= n; i++)
            { for (j = 1; j <= n; j++)
            {
                if (cost[i][j] < min) {
                    if (visited[i] != 0) {
                        min = cost[i][j];
                        a = u = i;
                        b = v = j;
                    }
                }
            }
        }

        if (visited[u] == 0 || visited[v] == 0) {
            printf("\n Edge %d: (%d -> %d)  cost: %d", ne++, a, b, min);
            mincost += min;
            visited[b] = 1;
        }

        cost[a][b] = cost[b][a] = 999;
    }

    printf("\n\nMinimum cost = %d\n", mincost);

    return 0;
}
```

Output:



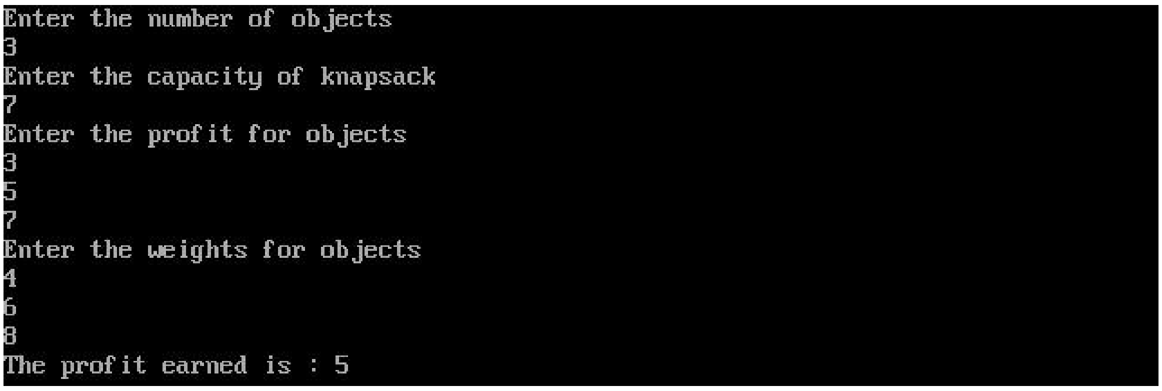
```
 Enter the number of nodes:2

 Enter the adjacency matrix:
1
2
3
4


 Edge 1:(1 2) cost:2
 Minimun cost=2
```

PROGRAM 10:

Write a program to implement o/1 Knapsack problem using dynamic programming and prove its efficiency.

```
#include <stdio.h>


int max(int a, int b)
    { return (a > b) ? a : b;
}


int main()
    {  int i, j, n,
    m;
    int p[50], w[50], v[50][50];

    printf("Enter the number of objects: ");
    scanf("%d", &n);

    printf("Enter the capacity of knapsack: ");
    scanf("%d", &m);

    printf("Enter the profit for each object:\n");
    for (i = 1; i <= n; i++)
        scanf("%d", &p[i]);

    printf("Enter the weight for each object:\n");
    for (i = 1; i <= n; i++)
        scanf("%d", &w[i]);
    // Build DP table
    for (i = 0; i <= n; i++)
        {  for (j = 0; j <= m; j++)
        {
            if (i == 0 || j == 0)
                v[i][j] = 0;
            else if (w[i] > j) v[i][j]
                = v[i - 1][j];
            else
                v[i][j] = max(v[i - 1][j], v[i - 1][j - w[i]] + p[i]);
        }
    }

    printf("\nMaximum profit earned: %d\n", v[n][m]);

    // (Optional) Display DP Table
    printf("\nDP Table:\n");
    for (i = 0; i <= n; i++)
        { for (j = 0; j <= m;
        j++)
            printf("%3d ", v[i][j]);
        printf("\n");
    }

    return 0;
}
```

Output:



```
Enter the number of objects
3
Enter the capacity of knapsack
7
Enter the profit for objects
3
5
7
Enter the weights for objects
4
6
8
The profit earned is : 5
```

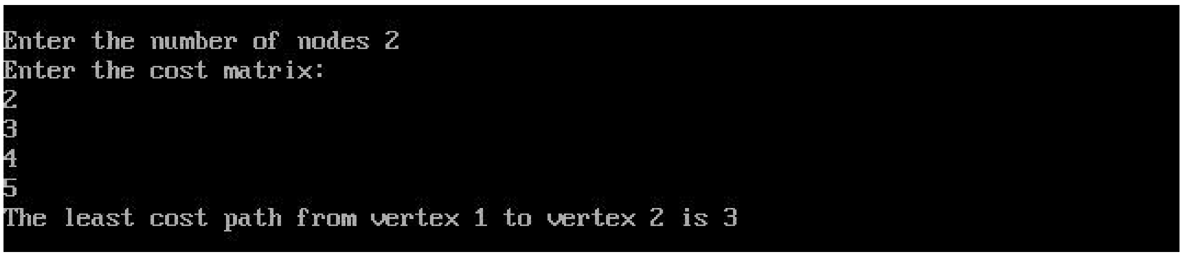
PROGRAM 11:

Write a program to find shortest paths to other vertices using Dijkstra’s algorithm, from a given vertex in a weighted connected graph.

```
#include <stdio.h>
#define INF 9999  // Infinity constant

void dijk(int c[10][10], int d[10], int n)
   { int v[10] = {0};
   int node = 0, i, j, count, min;

   // Initialize distances from source (vertex 1)
   for (i = 1; i <= n; i++) {
      d[i] = c[1][i];
      v[i] = 0;
   }

   v[1] = 1; // Start vertex is visited
   count = 1;

   while (count <= n - 1)
      { min = INF;

      // Find the next node with minimum distance
      for (i = 1; i <= n; i++) {
         if (d[i] < min && v[i] == 0)
            { min = d[i];
            node = i;
         }
```

```
        }

        v[node] = 1; // Mark node as visited

        // Update distances of adjacent nodes
        for (i = 1; i <= n; i++) {
            if (!v[i]) {
                if (min + c[node][i] < d[i]) {
                    d[i] = min + c[node][i];
                }
            }
        }

        count++;
    }

    // Print shortest distances
    printf("\nShortest distances from vertex 1:\n");
    for (i = 2; i <= n; i++) {
        if (d[i] >= INF)
            printf("Vertex 1 -> Vertex %d : No path\n", i);
        else
            printf("Vertex 1 -> Vertex %d : %d\n", i, d[i]);
    }
}

int main() {
    int i, j, n, d[10], c[10][10];
    printf("Enter the number of nodes: ");
    scanf("%d", &n);

    printf("Enter the cost matrix (0 for no edge):\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++)
            { scanf("%d",
            &c[i][j]);
            if (c[i][j] == 0 && i != j)
                c[i][j] = INF; // Replace 0 with infinity except self-loops
        }
    }

    dijk(c, d, n);
    return 0;
}
```

Output:



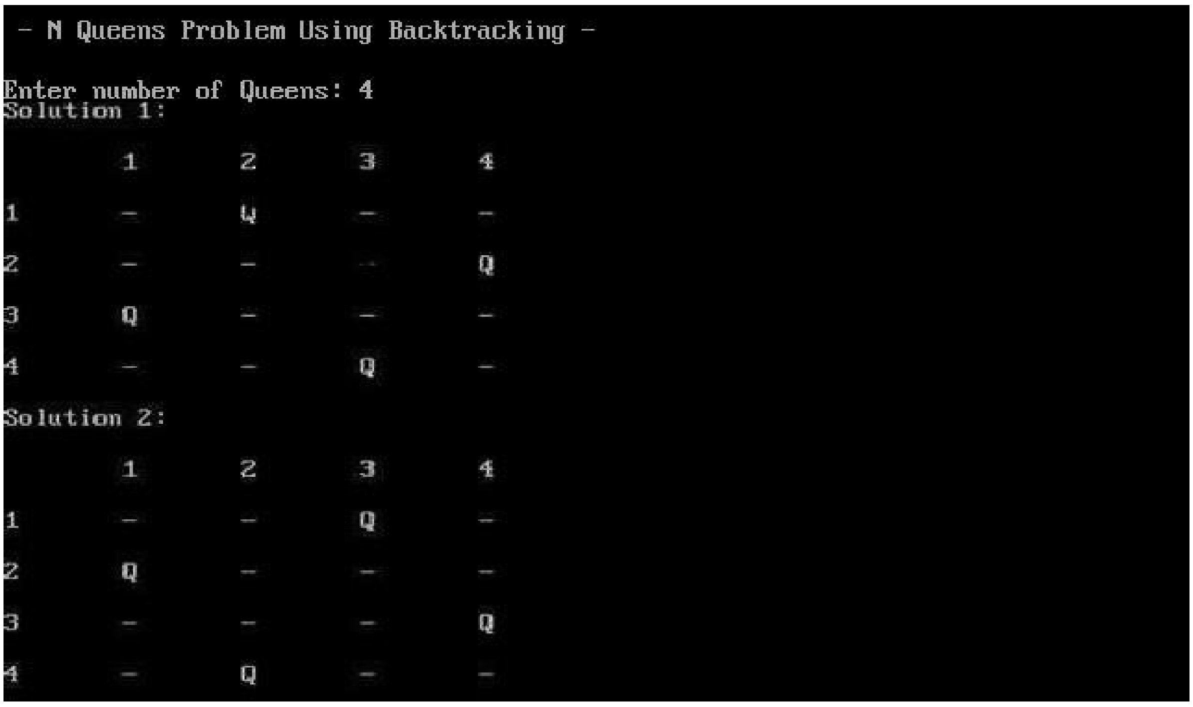
```
Enter the number of nodes 2
Enter the cost matrix:
2
3
4
5
The least cost path from vertex 1 to vertex 2 is 3
```

PROGRAM 12:

Write a program to implement N Queen’s problem using Backtacking and prove its efficiency.

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

int board[20], count = 0;

int place(int row, int column);
void print(int n);
void queen(int row, int n);

int main()
   { int n;
   printf(" - N Queens Problem Using Backtracking -\n\n");
   printf("Enter number of Queens: ");
   scanf("%d", &n);
   queen(1, n);

   if (count == 0)
      printf("\nNo solutions exist for %d Queens.\n", n);
   return 0;
}

// Check if the queen can be placed
int place(int row, int column) {
   int i;
   for (i = 1; i <= row - 1; i++) {
        // Check column or diagonal conflicts
        if (board[i] == column || abs(board[i] - column) == abs(i - row))
            return 0;
    }
    return 1; // No conflict
}

// Print a solution
void print(int n) {
    int i, j;
    printf("\n\nSolution %d:\n\n", ++count);
    for (i = 1; i <= n; i++)
        printf("\t%d", i);
    for (i = 1; i <= n; i++)
        { printf("\n\n%d", i);
        for (j = 1; j <= n; j++)
            { if (board[i] == j)
                printf("\tQ"); // Queen
            else
                printf("\t-"); // Empty cell
        }
    }
    printf("\n");
}

// Backtracking function
void queen(int row, int n) {
    int column;
    for (column = 1; column <= n; column++) {
        if (place(row, column)) {
            board[row] = column; // Place queen
            if (row == n)
                print(n); // Found a solution
            else
                queen(row + 1, n); // Recurse
        }
    }
}
```

Output:



```
 - N Queens Problem Using Backtracking -

Enter number of Queens: 4
Solution 1:

        1       2       3       4

1       -       Q       -       -

2       -       -       -       Q

3       Q       -       -       -

4       -       -       Q       -

Solution 2:

        1       2       3       4

1       -       -       Q       -

2       Q       -       -       -

3       -       -       -       Q

4       -       Q       -       -
```

PROGRAM 13:

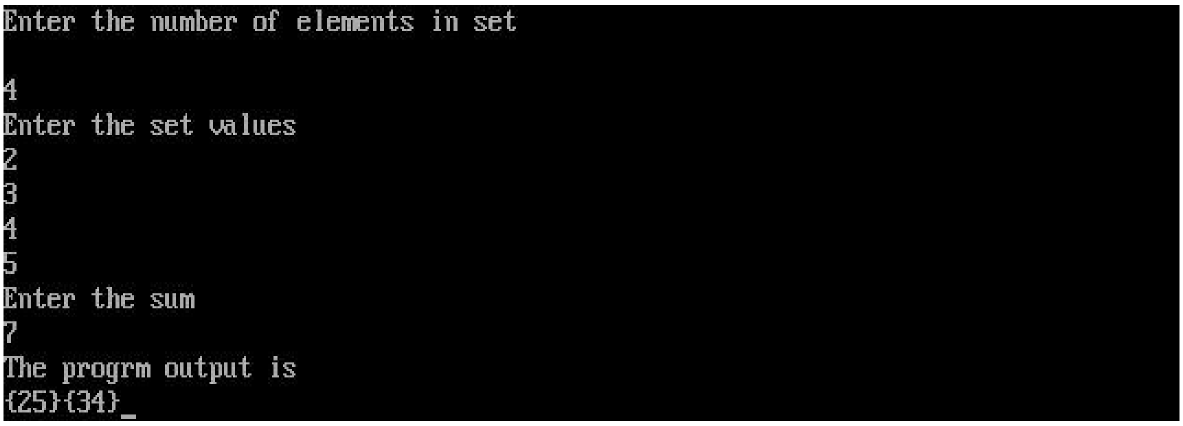
Write a program to find a subset of a given set S={s1,s2,….,sn} of n positive integers whose sum is equal to a given positive integer d. A suitable message is to be displayed if the given problem instance doesn't have a solution and prove its efficiency.

```
#include <stdio.h>


int s[10], n, d;

int set[10], count = 0;

int flag = 0;


void display(int count);

void subset(int sum, int i);


int main()

   { int i;


   printf("Enter the number of elements in the set: ");

   scanf("%d", &n);


   printf("Enter the set values: ");

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

       scanf("%d", &s[i]);


   printf("Enter the desired sum: ");

   scanf("%d", &d);


   printf("\nSubsets with sum %d are:\n", d);

   subset(0, 0);


   if (flag == 0)
```

```
        printf("No subset found with the given sum.\n");

    return 0;
}

void subset(int sum, int i) {
    // Base cases
    if (sum == d)
        { flag = 1;
        display(count);
        return;
    }

    if (sum > d || i >= n)
        return;

    // Include current element
    set[count] = s[i]; count++;
    subset(sum + s[i], i + 1);

    // Exclude current element
    count--;
    subset(sum, i + 1);
}

void display(int count)
    { int i;
    printf("{ ");
    for (i = 0; i < count; i++)
        printf("%d ", set[i]);
    printf("}\n");
```

}

Output:



```
Enter the number of elements in set

4
Enter the set values
2
3
4
5
Enter the sum
7
The progrm output is
{25}{34}_
```