Contents

Sl.No	Date	Program Name	Page No
1		Program to implement linear and binary search to locate the first occurrence of element	
2		Program to sort the array in ascending or descending order using Quick and Merge Sort	
3		Program to convert the infix expressions to its postfix form	
4		Program to create a linked list and perform insert and delete operations	
5		Program to perform insert and delete operations on a circular linked list	
6		Program to perform operations on a stack (linked list implementation) (push, pop and peek)	
7		Program to perform operations on a circular queue (enqueue, dequeue and peek)	
8		Program to compute the transitive closure of a given directed group using Warshall's Algorithm	
9		Program to find Minimum Cost Spanning Tree of an undirected graph using Prim's Algorithm	
10		Program to implement 0/1 Knapsack Problem using Dynamic Programming	
11		Program to find shortest paths to other vertices using Dijkstra's algorithm, from a given vertex in a weighted connected graph.	
12		Program to implement N Queen's problem using Backtracking and prove its efficiency.	
13		Program to find a subset of a given set 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.	

PROGRAM: 1 **Date:**

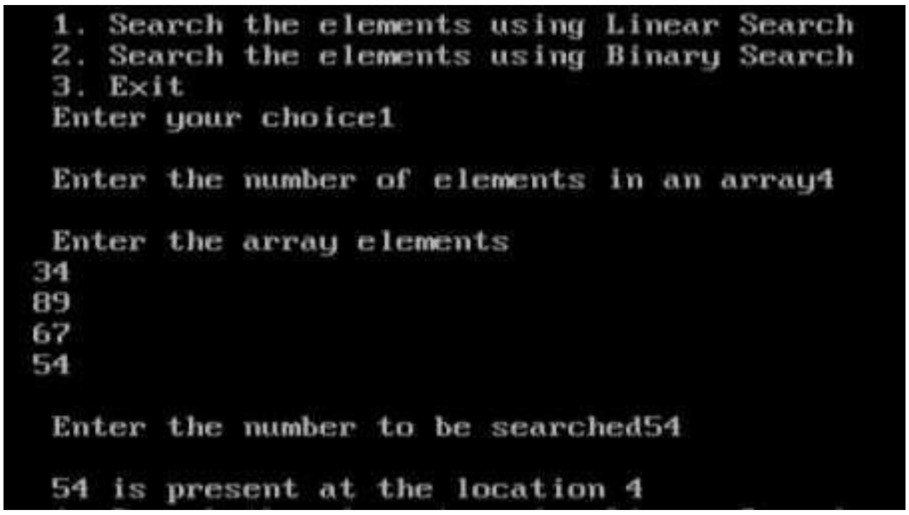
Write a menu driven program to implement linear and binary search to find the location of the first occurrence of an item.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void linear();
void binary();
int main()
{
	int choice;
	do
	{
		printf("\n 1. Search the elements using Linear Search");
		printf("\n 2. Search the elements using Binary Search");
		printf("\n 3. Exit");
		printf("\n Enter your choice");
		scanf("%d",&choice);
		switch(choice)
		{
			case 1:
				linear();
				break;
			case 2:
				binary();
				break;
			case 3:

				exit(0);
				break;
		}
	}while((choice>=1)&&(choice<=2));
	return 0;
	}
void linear()
```

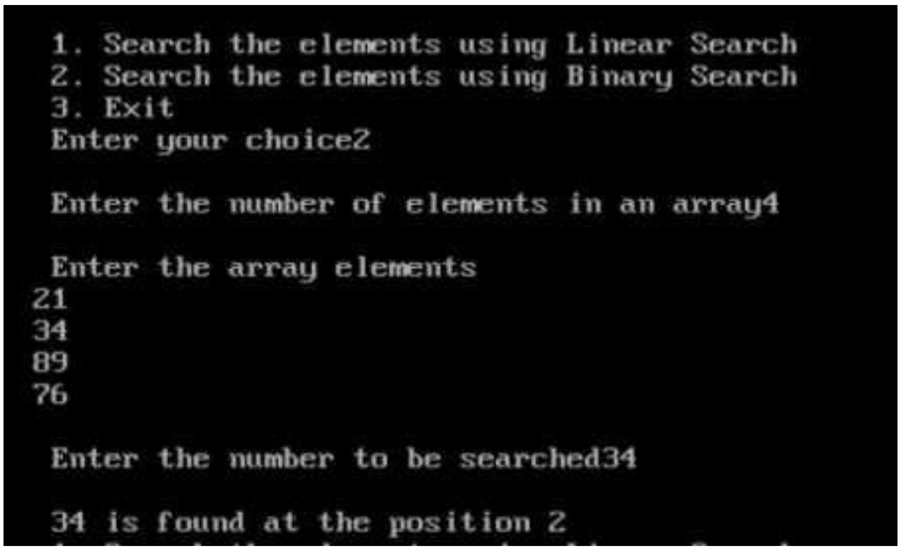
```
{
	int n,c,array[100],search;
	printf("\n Enter the number of elements in an array");
	scanf("%d",&n);
	printf("\n Enter the array elements");
	for(c=0;c<n;c++)
		scanf("%d",&array[c]);
	printf("\n Enter the number to be searched");
	scanf("%d",&search);
	for(c=0;c<n;c++)
	{
		if(array[c]==search)
		{
			printf("\n %d is present at the location %d",search,c+1);
			break;
		}
	}
	if(c==n)
	{
		printf("\n %d is not present in the array",search);
	}
}
void binary()
{
	int array[100],i,j,n,mid,temp,flag=0,low=0,high,item,c;
	printf("\n Enter the number of elements in an array");
	scanf("%d",&n);
	printf("\n Enter the array elements");
	for(c=0;c<n;c++)
		scanf("%d",&array[c]);
	printf("\n Enter 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 the position %d",item,mid);
                        flag=1;
                        break;
                }
                else if(item>array[mid])
                {
                        low=mid+1;
                }
                else
                {
                        high = mid-1;
                }
        }
        if(flag==0)
        {
                printf("\n %d is not found",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. 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
```

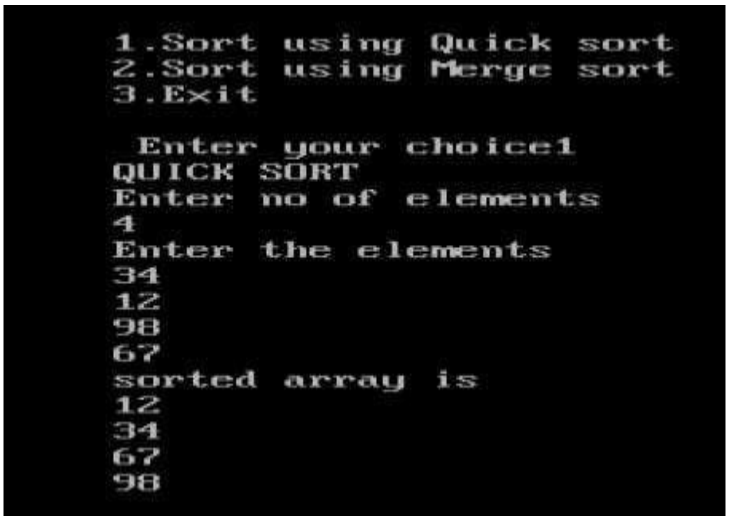
PROGRAM: 2 **Date:**

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

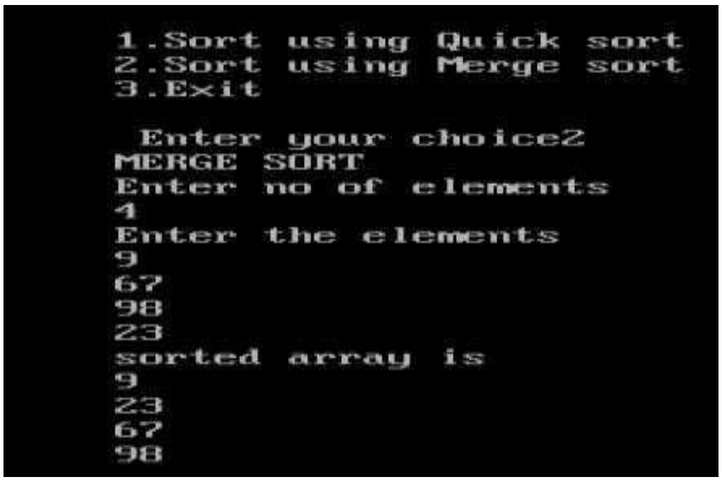
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void quick_sort(int[],int,int);
int partition(int[],int,int);
void mergesort(int[],int,int);
void merge(int[],int,int,int,int);
void quick_sort(int a[100],int l,int u)
{
        int j;
        if(l<u)
        {
                j=partition(a,l,u);
                quick_sort(a,l,j-1);
                quick_sort(a,j+1,u);
        }
}
int partition(int a[100],int l,int u)
{
        int v,i,j,temp;
        v=a[l];
        i=l;
        j=u+1;
        do
        {
                do
                {
                        i++;
                }while(a[i]<v&&i<=u);
                do
                {
                        j--;
                }while(a[j]>v);
                if(i<j)
                {
                        temp=a[i];
                        a[i]=a[j];
                        a[j]=temp;
                }
        }while(i<j);
        a[l]=a[j];
        a[j]=v;
        return(j);
}
void mergesort(int a[100],int i, int j)
{
        int mid;
        if(i<j)
        {
                mid=(i+j)/2;
                mergesort(a,i,mid);
                mergesort(a,mid+1,j);
                merge(a,i,mid,mid+1,j);
        }
}
void merge(int a[100],int i1,int j1, int i2,int j2)
{
        int temp[100];
        int i,j,k;
        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++];
          }
          while(i<=j1)
          temp[k++]=a[i++];
          while(j<=j2)
          temp[k++]=a[j++];
          for(i=i1,j=0;i<=j2;i++,j++)
                    a[i]=temp[j];
}
void main()
{
          int a[100],n,i,op;
          clrscr();
          while(1)
          {
                    printf("\n1.Sort using Quick sort");
                    printf("\n2.Sort using Merge sort");
                    printf("\n3.Exit\n");
                    printf("\n Enter your choice");
                    scanf("%d",&op);
                    switch(op)
                    {
                              case 1: printf("QUICK SORT\n");
                                         printf("Enter no of elements\n");
                                         scanf("%d",&n);
                                         printf("Enter the elements\n");
                                        for(i=0;i<=n-1;i++)
                                        scanf("%d",&a[i]);
                                        quick_sort(a,0,n-1);
                                        printf("sorted array is\n");
                                        for(i=0;i<=n-1;i++)
                                        printf("%d\n",a[i]);
                                        break;
                              case 2: printf("MERGE SORT\n");
                                        printf("Enter no of elements\n");
                                        scanf("%d",&n);
                                        printf("Enter the elements\n");
                                        for(i=0;i<=n-1;i++)
                                        scanf("%d",&a[i]);
                                        mergesort(a,0,n-1);
                                        printf("sorted array is\n");
                                        for(i=0;i<=n-1;i++)
                                        printf("%d\n",a[i]);
                                        break;
                                case 3: exit(1);
                                default: printf("Invalid Choice\n");
                    }
          }
}
```

Output:



```
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.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 **Date:**

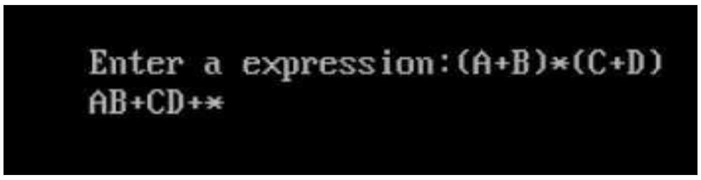
Write a program to convert the given infix expression into its postfix form.

```
#include<stdio.h>
#include<stdlib.h>
int prec(char c);
char stack[100];
int top=-1;
void push(int num);
char pop();
void infixtopostfix(char str[]);
void main()
{
	char str[100];
	int ch;
	clrscr();
	printf("\nEnter a expression:");
	scanf("%s",str);
	infixtopostfix(str);
}
int prec(char c)
{
	if(c=='^')
		return 3;
	else if(c=='/'|| c=='*')
		return 2;
	else if(c=='+'|| c=='-')
		return 1;
	else
		return -1;
}
void push(int num)
{
	stack[++top]=num;
}
char pop()
```

```
{
        char c=stack[top];
        stack[top--]='\0';
        return(c);
}
void infixtopostfix(char str[])
{
        int i=0;
        while(str[i]!='\0')
        {
                if(str[i]>=65 && str[i]<=90 || str[i]>=97 && str[i]<=122)
                        printf("%c",str[i]);
                else if(str[i]=='(')
                        push(str[i]);
                else if(str[i]==')')
                {
                        char c;
                        do
                        {
                                c=pop();
                                if(c!='(')
                                        printf("%c",c);
                        } while(c!='(');
                }
                else
                {
                        if(stack[top]=='\0'|| prec(str[i])>prec(stack[top]))
                        {
                                push(str[i]);
                        }
                        else
                        {
                                while(prec(str[i])<=prec(stack[top]))
                                {
                                        char c=pop();
```

```
                    if(c!='('|| c!=')')
                        printf("%c",c);
                    else
                        break;
                }
                push(str[i]);
            }
        }
        i++;
    }
    while(top!=-1)
    printf("%c",pop());
    getch();
}
```

Output:



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

PROGRAM: 4 **Date:**

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

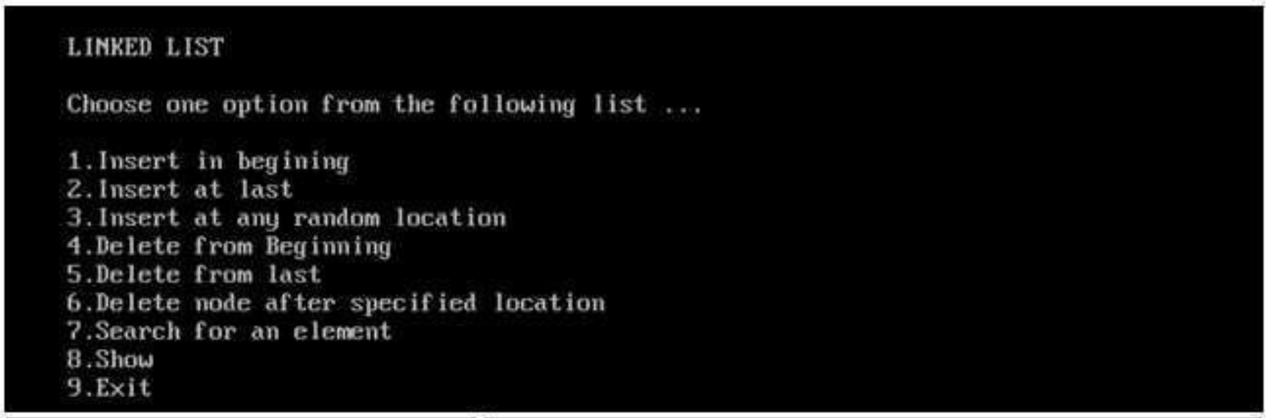
```
				}
				else
				{
					temp = head;
					while (temp -> next != NULL)
					{
						temp = temp -> next;
					}
					temp->next = ptr;
					ptr->next = NULL;
					printf("\nNode inserted");

				}
		}
}
void randominsert()
{
		int i,loc,item;
		struct node *ptr, *temp;
		ptr = (struct node *) malloc (sizeof(struct node));
		if(ptr == NULL)
		{
				printf("\nOVERFLOW");
		}
		else
		{
				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++)
				{
					temp = temp->next;
```

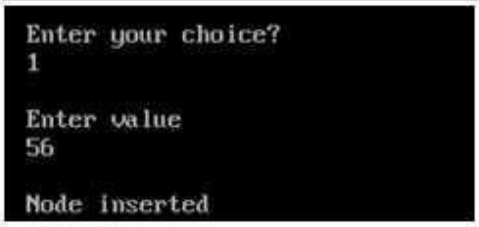
```
                    if(temp == NULL)
                    {
                            printf("\ncan't insert\n");
                            return;
                    }


            }
            ptr ->next = temp ->next;
            temp ->next = ptr;
            printf("\nNode inserted");
        }
}
void begin_delete()
{
        struct node *ptr;
        if(head == NULL)
        {
                printf("\nList is empty\n");
        }
        else
        {
                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 if(head -> next == NULL)
```

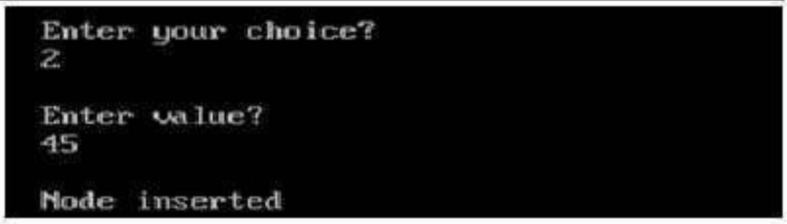
```
	{
		head = NULL;
		free(head);
		printf("\nOnly node of the list deleted ...\n");
	}
	else
	{
		ptr = head;
		while(ptr->next != NULL)
		{
			ptr1 = ptr;
			ptr = ptr ->next;
		}
		ptr1->next = NULL;
		free(ptr);
		printf("\nDeleted Node from the last ...\n");
	}
}
void random_delete()
{
	struct 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;
		}
	}
```

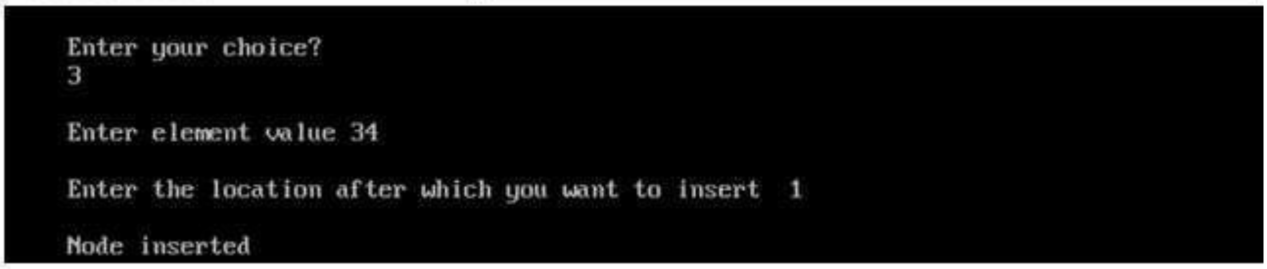
```
        ptr1 ->next = ptr ->next;
        free(ptr);
        printf("\nDeleted node %d ",loc+1);
}
void search()
{
        struct node *ptr;
        int item,i=0,flag;
        ptr = head;
        if(ptr == NULL)
        {
                printf("\nEmpty List\n");
        }
        else
        {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

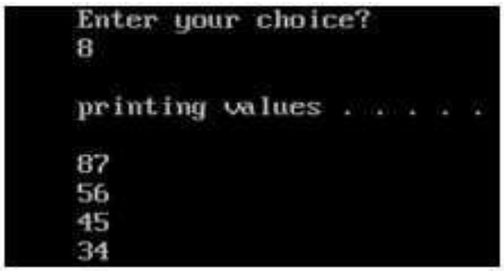
Output:

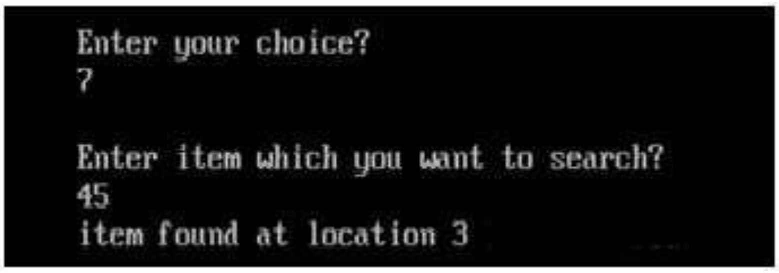

```
LINKED LIST

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
```

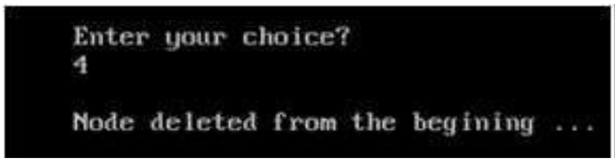

```
Enter your choice?
1

Enter value
56

Node inserted
```

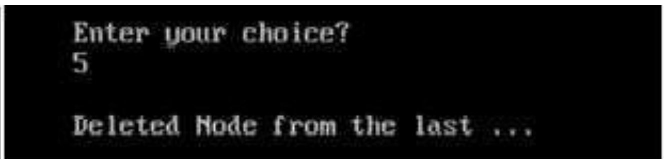

```
Enter your choice?
2

Enter value?
45

Node inserted
```

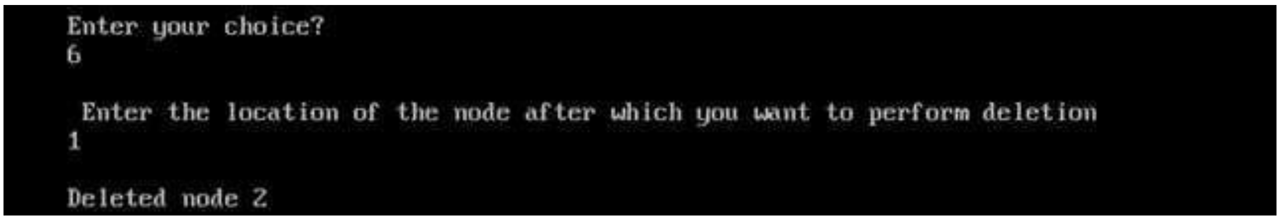

```
Enter your choice?
3

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?
4

Node deleted from the begining ...
```


```
Enter your choice?
5

Deleted Node from the last ...
```


```
Enter your choice?
6

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

Deleted node 2
```

PROGRAM:5 **Date:**

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

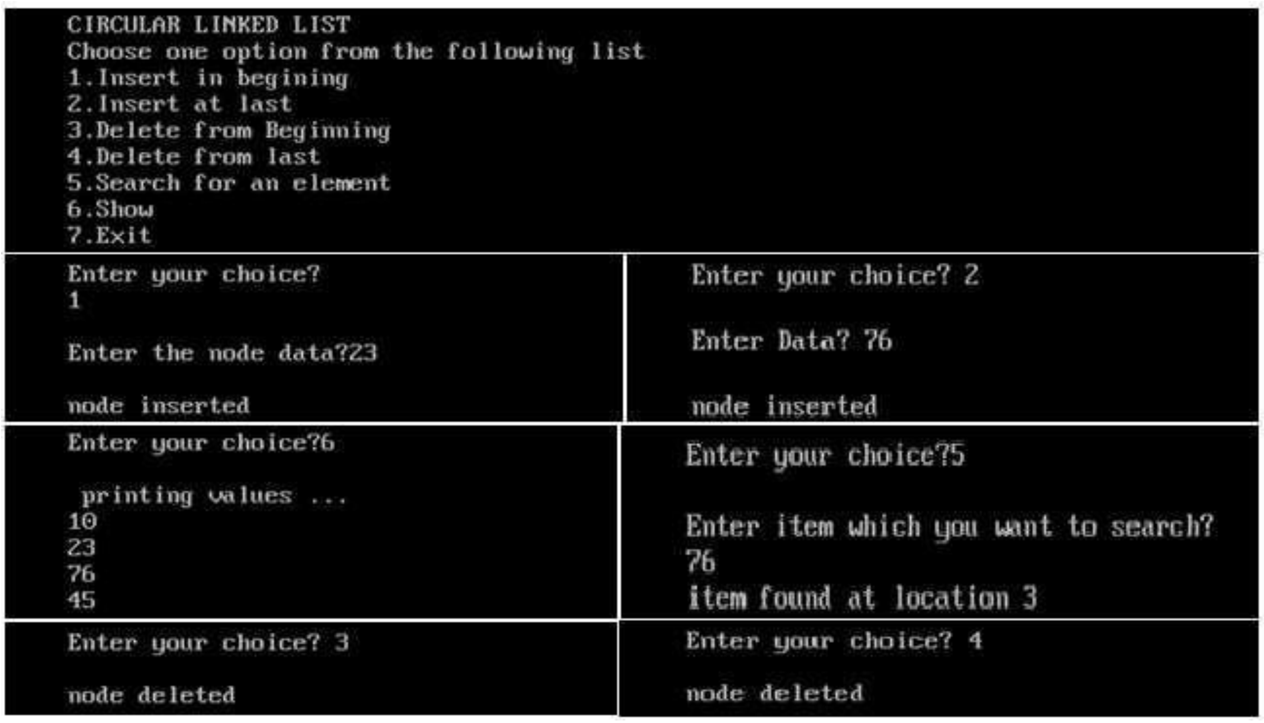
```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int data;
        struct node *next;
};
struct node *head;
void beginsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
void main ()
{
        int 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:
                                beginsert();
                                break;
                        case 2:
```

```
                                        lastinsert();
                break;
                        case 3:
                                        begin_delete();
                                        break;
                        case 4:
                                        last_delete();
                                        break;
                        case 5:
                                        search();
                                        break;
                        case 6:
                                        display();
                                        break;
                        case 7:
                                        exit(0);
                                        break;
                        default:
                                        printf("Please enter valid choice..");
                }
        }
}
void beginsert()
{
        struct node *ptr,*temp;
        int item;
        ptr = (struct node *)malloc(sizeof(struct node));
        if(ptr == NULL)
        {
                printf("\nOVERFLOW");
        }
        else
        {
                printf("\nEnter the node data?");
                scanf("%d",&item);
                ptr -> data = item;
                if(head == NULL)
                {
                        head = ptr;
                        ptr -> next = head;
                }
                else
                {
                        temp = head;
                        while(temp->next != head)
                                temp = temp->next;
                        ptr->next = head;
                        temp -> next = ptr;
                        head = ptr;
                }
                printf("\nnode inserted\n");
        }
}
void lastinsert()
{
        struct node *ptr,*temp;
        int item;
        ptr = (struct node *)malloc(sizeof(struct node));
        if(ptr == NULL)
        {
                printf("\nOVERFLOW\n");
        }
        else
        {
                printf("\nEnter Data?");
                scanf("%d",&item);
                ptr->data = item;
                if(head == NULL)
                {
                        head = ptr;
                ptr -> next = head;
            }
            else
            {
                temp = head;
                while(temp -> next != head)
                {
                    temp = temp -> next;
                }
                temp -> next = ptr;
                ptr -> next = head;
            }
            printf("\nnode inserted\n");
        }
}
void begin_delete()
{
        struct node *ptr;
        if(head == NULL)
        {
            printf("\nUNDERFLOW");
        }
        else if(head->next == head)
        {
            head = NULL;
            free(head);
            printf("\nnode deleted\n");
        }
        else
        {
            ptr = head;
            while(ptr -> next != head)
                ptr = ptr -> next;
            ptr->next = head->next;
            free(head);
```

```
                head = ptr->next;
                printf("\nnode deleted\n");
        }
}
void last_delete()
{
        struct node *ptr, *preptr;
        if(head==NULL)
        {
                printf("\nUNDERFLOW");
        }
        else if (head ->next == head)
        {
                head = NULL;
                free(head);
                printf("\nnode deleted\n");
        }
        else
        {
                ptr = head;
                while(ptr ->next != head)
                {
                        preptr=ptr;
                        ptr = ptr->next;
                }
                preptr->next = ptr -> next;
                free(ptr);
                printf("\nnode deleted\n");
        }
}
 void search()
{
        struct node  *ptr;
        int item,i=0,flag=1;
        ptr = head;
```

```
if(ptr == NULL)
{
        printf("\nEmpty List\n");
}
else
{
        printf("\nEnter item which you want to search?\n");
        scanf("%d",&item);
        if(head ->data == item)
        {
                printf("item found at location %d",i+1);
                flag=0;
        }
        else
        {
                while (ptr->next != head)
                {
                        if(ptr->data == item)
                        {
                                printf("item found at location %d ",i+1);
                                flag=0;
                                break;
                        }
                        else
                        {
                                flag=1;
                        }
                        i++;
                        ptr = ptr -> next;
                }
        }
        if(flag != 0)
        {
                printf("Item not found\n");
        }
        }
}
 void display()
{
        struct node *ptr;
        ptr=head;
        if(head == NULL)
        {
                printf("\nnothing to print");
        }
        else
        {
                printf("\n printing values ... \n");
                while(ptr -> next != head)
                {
                        printf("%d\n", ptr -> data);
                        ptr = ptr -> next;
                }
                printf("%d\n", ptr -> data);
        }
}
```

Output:



```
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 **Date:**

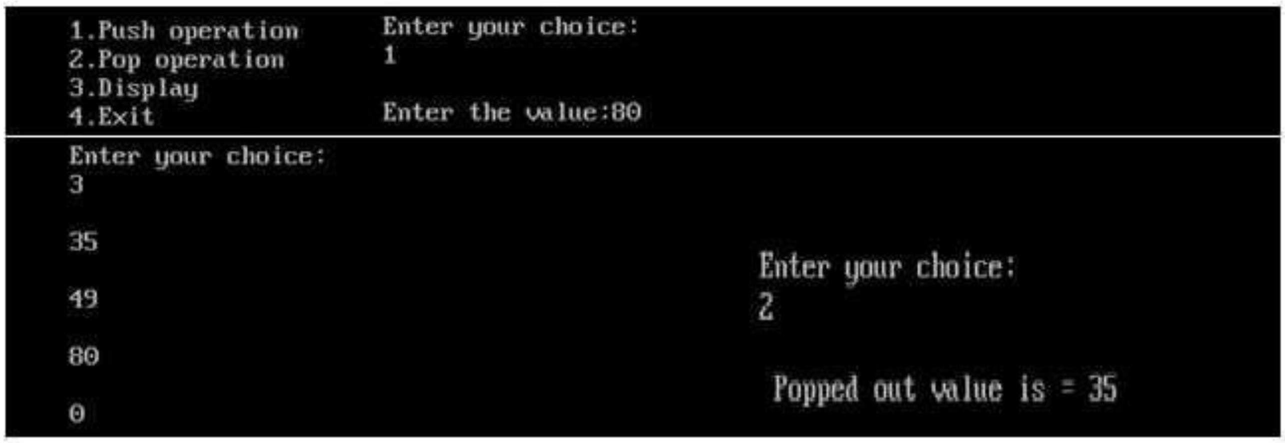
Write a menu driven program to perform operations on a stack (linked list implementation) (push, pop, and peek).

```
#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)
	{
		printf("\nUNDERFLOW COMDITION");
		return;
	}
	else
	{
		printf("\n Popped oyt value is = %d",top->num);
```

```
                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

49

80

0
```

```
Enter your choice:
2

 Popped out value is = 35
```

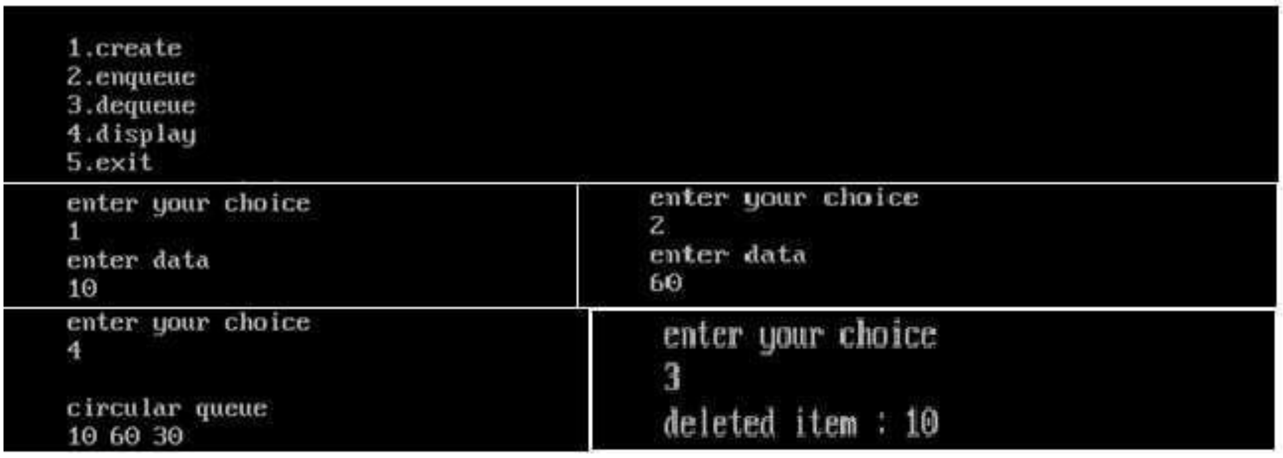
PROGRAM: 7 **Date:**

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)
				{
					rear=front=create();
				}
				else
					printf("queue already exists\n");
				break;
			case 2: enqueue();
				break;
			case 3: if(front==NULL)
					printf("queue underflow\n");
```

```
                                        else
                                                printf("deleted item : %d ",dequeue());
                                        break;
                                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
```

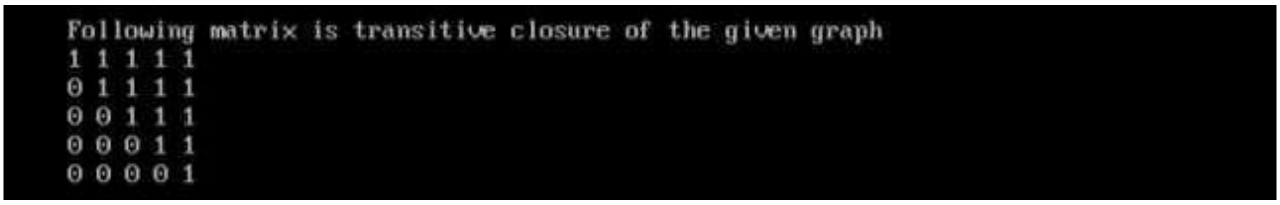
```
enter your choice
3
deleted item : 10
```

PROGRAM:8 Date:

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

```
#define V 5
void printmatrix(int reach[][V]);
void transitiveClosure(int graph[][V])
{
        int reach[V][V], i, j, k;
        for (i = 0; i < V; i++)
                for (j = 0; j < V; j++)
                        reach[i][j] = graph[i][j];
        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);
}
void printmatrix(int reach[][V])
{
        int i,j;
         printf ("\nFollowing matrix is transitive closure of given graph");
        for (i = 0; i < V; i++)
        {
                for (j = 0; j < V; j++)
                {


                                if(i == j)
                                        printf("1 ");
                                        else
                                                  printf ("%d ", reach[i][j]);
                    }
                    printf("\n");
                    getch();
          }
}
int main()
{
          /* Let us create the following weighted graph
          10
      (0)------->(3)
       |           /|\
     5 |            |
       |            | 1
      \|/           |
      (1)------->(2)
          3          */
          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:



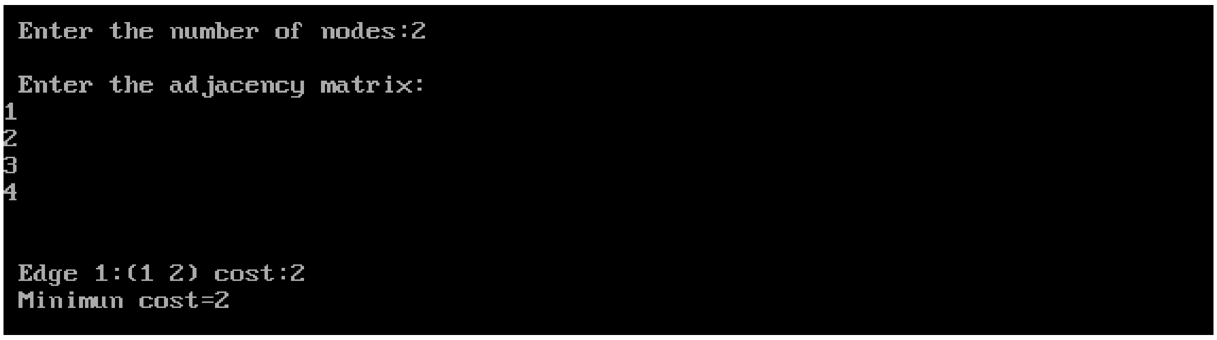
```
Following matrix is transitive closure of the given graph
1 1 1 1 1
0 1 1 1 1
0 0 1 1 1
0 0 0 1 1
0 0 0 0 1
```

PROGRAM: 9 **Date:**

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

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]= {0}
,min,mincost=0,cost[10][10];
void main() {
	clrscr();
	printf("\n Enter the number of nodes:");
	scanf("%d",&n);
	printf("\n Enter 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;
	}
	visited[1]=1;
	printf("\n");
	while(ne<n) {
		for (i=1,min=999;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 Minimun cost=%d",mincost);
	getch();
}
```

Output:



```
 Enter the number of nodes:2

 Enter the adjacency matrix:
1
2
3
4



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

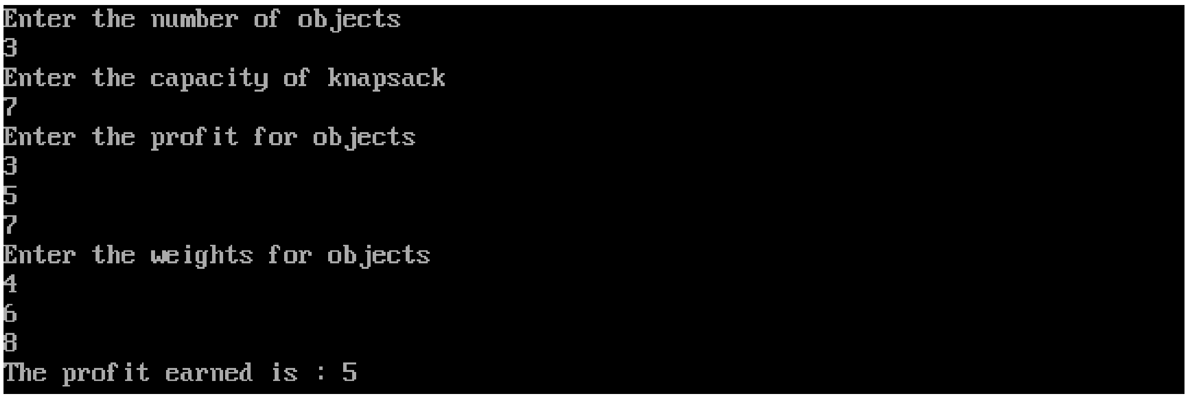
PROGRAM: 10 **Date:**

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

```
#include  <stdio.h>
#include <conio.h>
void main()
{
	int i,j,n,m,p[50],w[50],v[50][50];
	clrscr();
	printf("Enter the number of objects \n");
	scanf("%d",&n);
	printf("Enter the capacity of knapsack \n");
	scanf("%d",&m);
	printf("Enter the profit for objects \n");
	for (i=1;i<=n;i++)
		scanf("%d",&p[i]);
	printf("Enter the weights for objects \n");
	for (i=1;i<=n;i++)
		scanf("%d",&w[i]);
	for (i=1;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
			{
				if(v[i-1][j]>v[i-1][j-w[i]]+p[i])
				{
					v[i][j]=v[i-1][j];
				}
				else
				{
					v[i][j]=v[i-1][j-w[i]]+p[i];
```

```
                        }
                    }
                }
        printf("The profit earned is : %d",v[n][m]);
        getch();
}
```

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
```

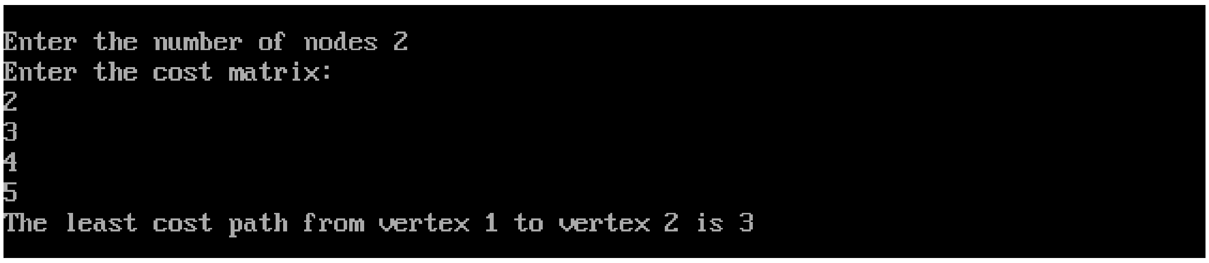
PROGRAM: 11 **Date:**

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>
int dijk(int c[10][10],int d[10],int n)
{
        int v[10]={0}, node,i,j,count,min;
        for (i=1;i<=n;i++)
        {
                d[i]=c[1][i];
                v[i]=0;
         }
         v[1]=1;
         count=1;
         while(count<=n-1)
         {
                for(i=1,min=9999;i<=n;i++)
                {
                        if(d[i]<min && v[i]==0)
                        {
                                min=d[i];
                                node=i;
                         }
                }
                v[node]=1;
                 for (i=1;i<=n;i++)
                {
                        if(!v[i])
                        {
                                if (min + c[node][i]<d[i])
                                d[i]=min+c[node][i];
                        }
                }
                 count++;
         }
```

```
            for(i=2;i<=n;i++)
            {
                        printf("The least cost path from vertex 1 to vertex %d is %d",i,d[i]);
              }
}
int main()
{
            int i,j;
             int n,d[10],c[10][10];
             printf("Enter the number of nodes");
             scanf("%d",&n);
            printf("Enter the cost matrix:");
             for(i=1;i<=n;i++)
            {
                        for (j=1;j<=n;j++)
                                    scanf("%d",&c[i][j]);
            }
              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
```

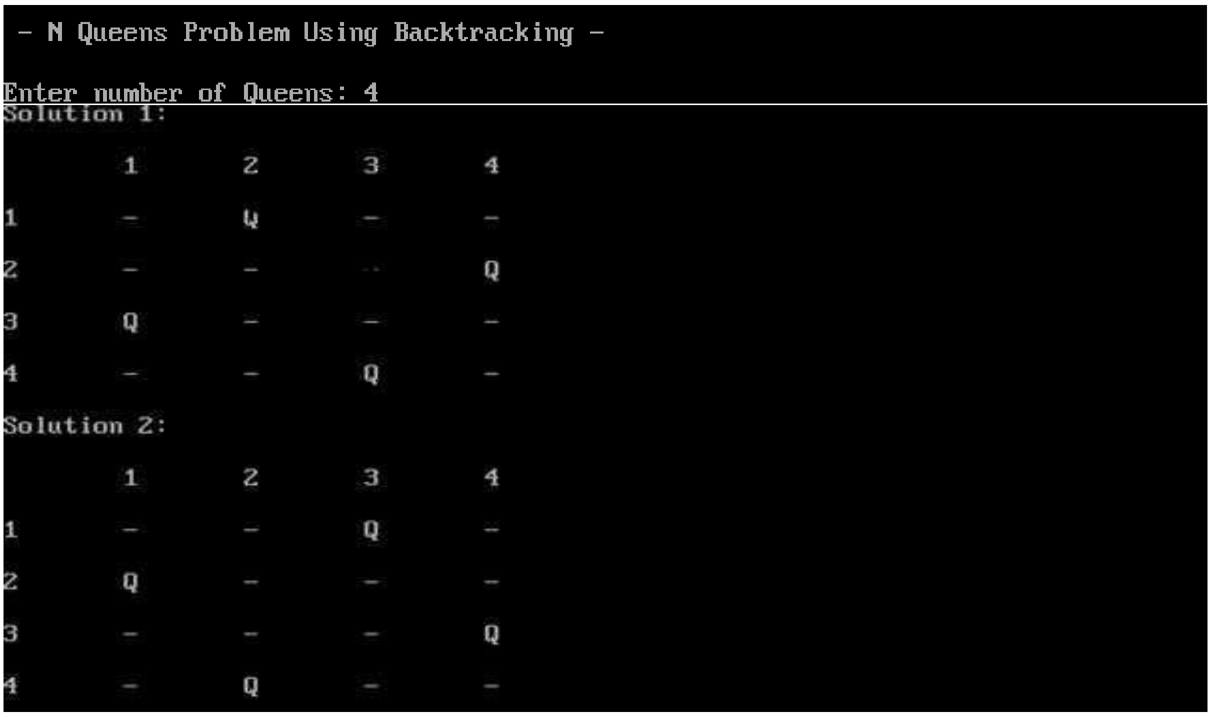
PROGRAM: 12 **Date:**

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;
int main(){
             int n,i,j;
             void queen(int row,int n);
             printf(" - N Queens Problem Using Backtracking -");
            printf("\n\nEnter number of Queens:");
            scanf("%d",&n);
            queen(1,n);
            return 0;
}
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++) //for nxn board
                        {
                                     if(board[i]==j)
                                                 printf("\tQ"); //queen at i,j position
                                    else
                                                printf("\t-"); //empty slot
                        }
            }
  }
int place(int row,int column)
{
```

```
            int i;
            for(i=1;i<=row-1;i++) {
                        if(board[i]==column)
                                    return 0;
                        else if(abs(board[i]-column)==abs(i-row))
                         return 0;
             }
            return 1; //no conflicts
}
void queen(int row,int n){
             int column;
            for(column=1;column<=n;column++) {
                        if(place(row,column)){
                                     board[row]=column; //no conflicts so place queen
                         if(row==n) //dead end
                                     print(n); //printing the board configuration
                        else //try queen with next position
                                    queen(row+1,n);
                        }
             }
    }
```

Output:



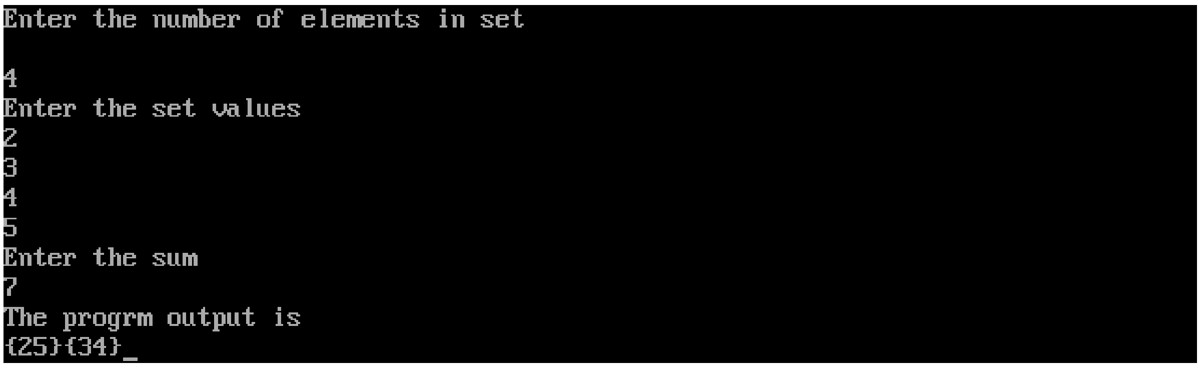
```
 - N Queens Problem Using Backtracking -

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 **Date:**

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>
#include<conio.h>
int s[10],d,n,set[10],count=0;
void display(int);
int subset(int,int);
int flag=0;
int main()
{
        int i;
        printf("Enter the number of elements in set\n");
        scanf("%d",&n);
        printf("Enter the set values\n");
        for(i=0;i<n;++i)
        scanf("%d",&s[i]);
        printf("Enter the sum\n");
        scanf("%d",&d);
        printf("The progrm output is\n");
        subset(0,0);
        if(flag==0)
                printf("there is no solution");
        getch();
        return 0;
}
int subset(int sum,int i)
{
        if(sum==d)
        {
                flag=1;
                display(count);
                return;
        }
        if(sum>d||i>=n)
```

```
                return;
        else
        {
                set[count]=s[i];
                count++;
                subset(sum+s[i],i+1);
                count--;
                subset(sum,i+1);
        }
}
void display(int count)
{
        int i;
        printf("{");
        for(i=0;i<count;i++)
                printf("%d",set[i]);
                printf("}");
}
```



```
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}_
```