DS LAB REPORT

Anitej Prasad

1BM19CS194

3-D

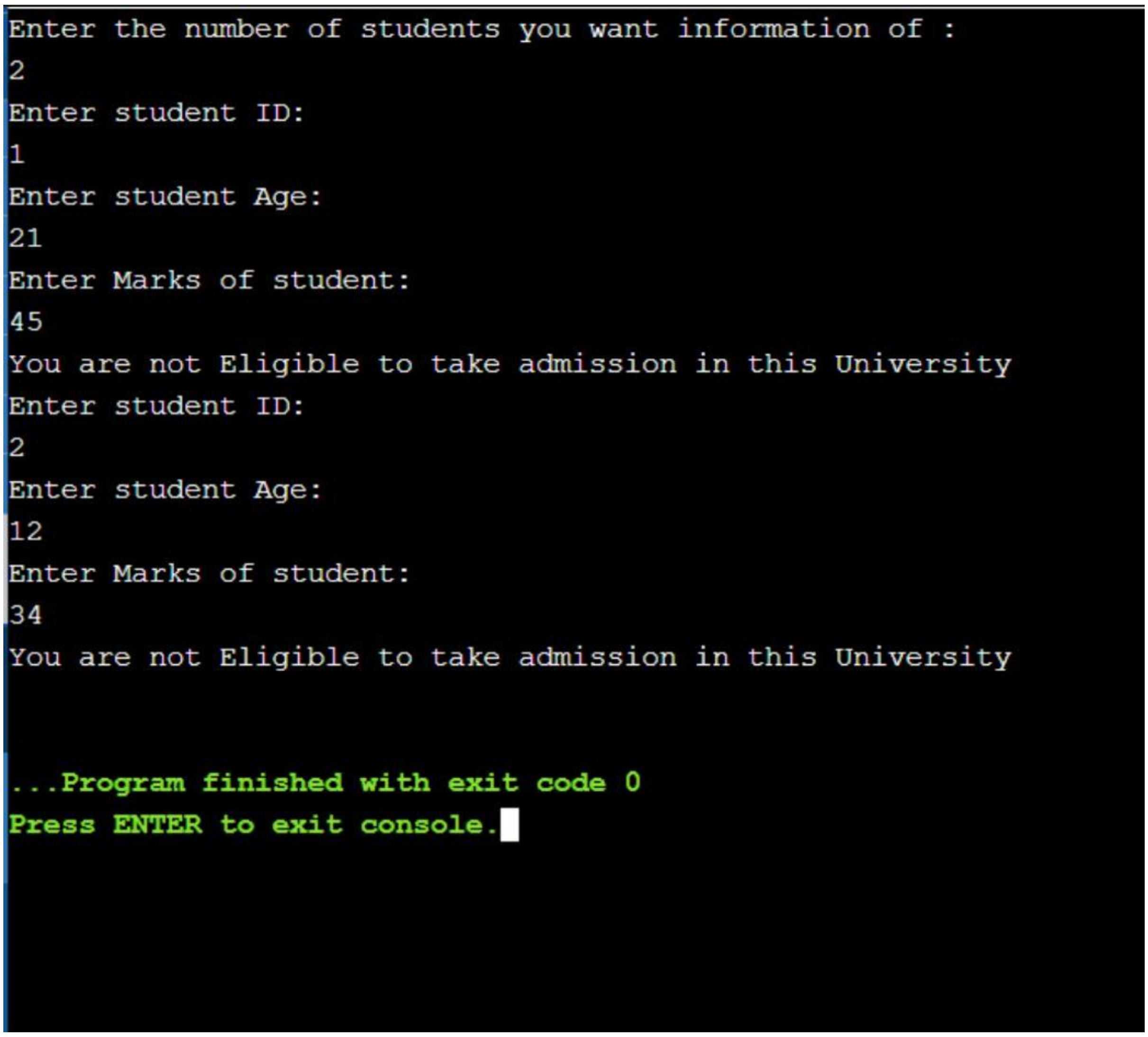
Semester: 3

Sep-Dec 2020

LAB 1:

WAP for the below given scenario: A university wants to automate their admission process. Students are admitted based on the marks scored in a qualifying exam. A student is identified by student id, age and marks in qualifying exam. Data are valid, if: ● Age is greater than 20 ● Marks is between 0 and 100 (both inclusive) A student qualifies for admission, if ● Age and marks are valid and ● Marks is 65 or more Write a program to represent the students seeking admission in the university

```
#include<stdio.h>
struct students {
int age;
int marks;
int id;
};
int main()
{
struct students s1;
int n;
int i;
printf("Enter the number of students you want information of :\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter student ID:\n");
scanf("%d",&s1.id);
printf("Enter student Age:\n");
scanf("%d",&s1.age);
printf("Enter Marks of student:\n");
scanf("%d",&s1.marks);
if(s1.age>20 && s1.marks<100 && s1.marks>0)
{
printf("The Entered data was valid.\n And You are Elligible to take the admission in the
university\n");
printf("Your ID is :%d\n",s1.id);
printf("Your Marks is :%d \n",s1.marks);
printf("Your Age is :%d\n",s1.age);
}
else
{
printf("You are not Eligible to take admission in this University\n");
} }
return 0;
}
```



```
Enter the number of students you want information of :
2
Enter student ID:
1
Enter student Age:
21
Enter Marks of student:
45
You are not Eligible to take admission in this University
Enter student ID:
2
Enter student Age:
12
Enter Marks of student:
34
You are not Eligible to take admission in this University


...Program finished with exit code 0
Press ENTER to exit console.
```

LAB 2:

Write a program to simulate the working of stack using an array with the following :

a) Push

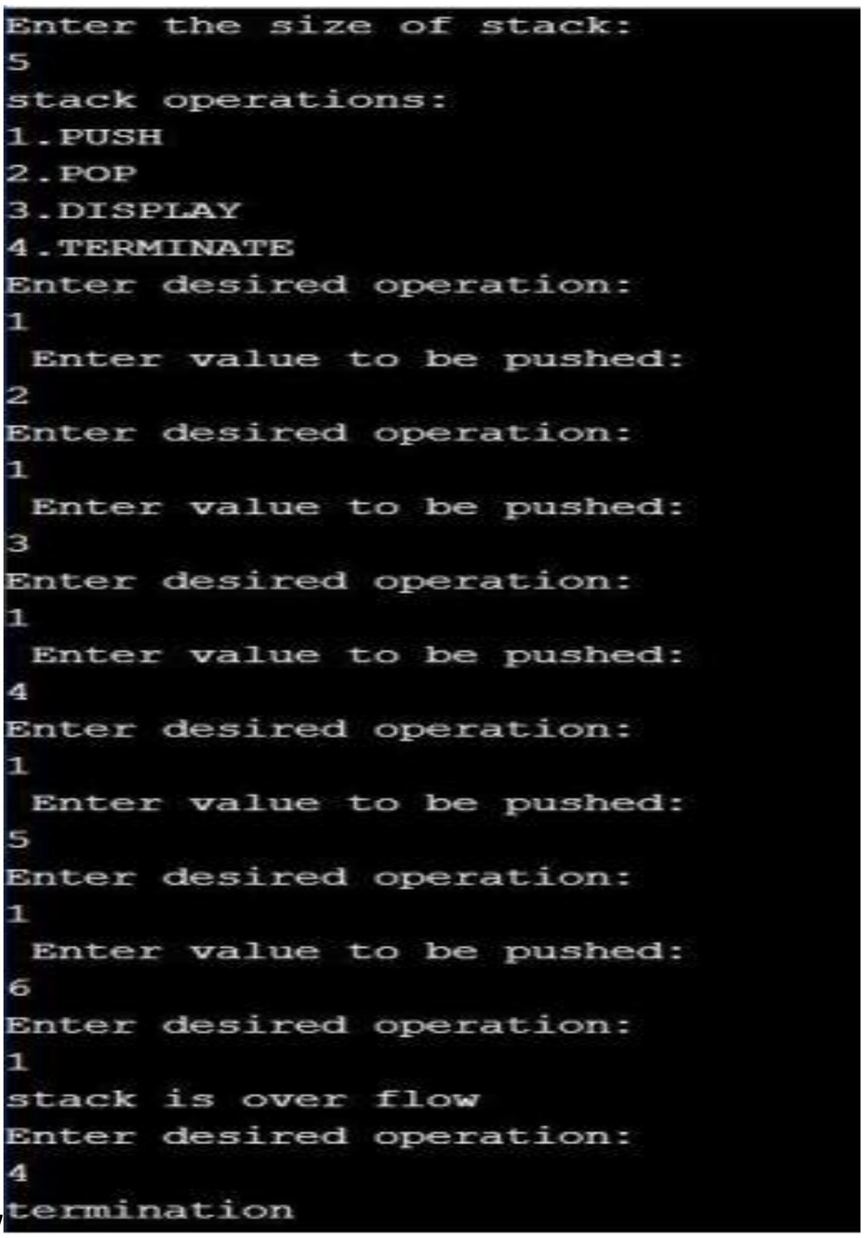
b) Pop

c) Display

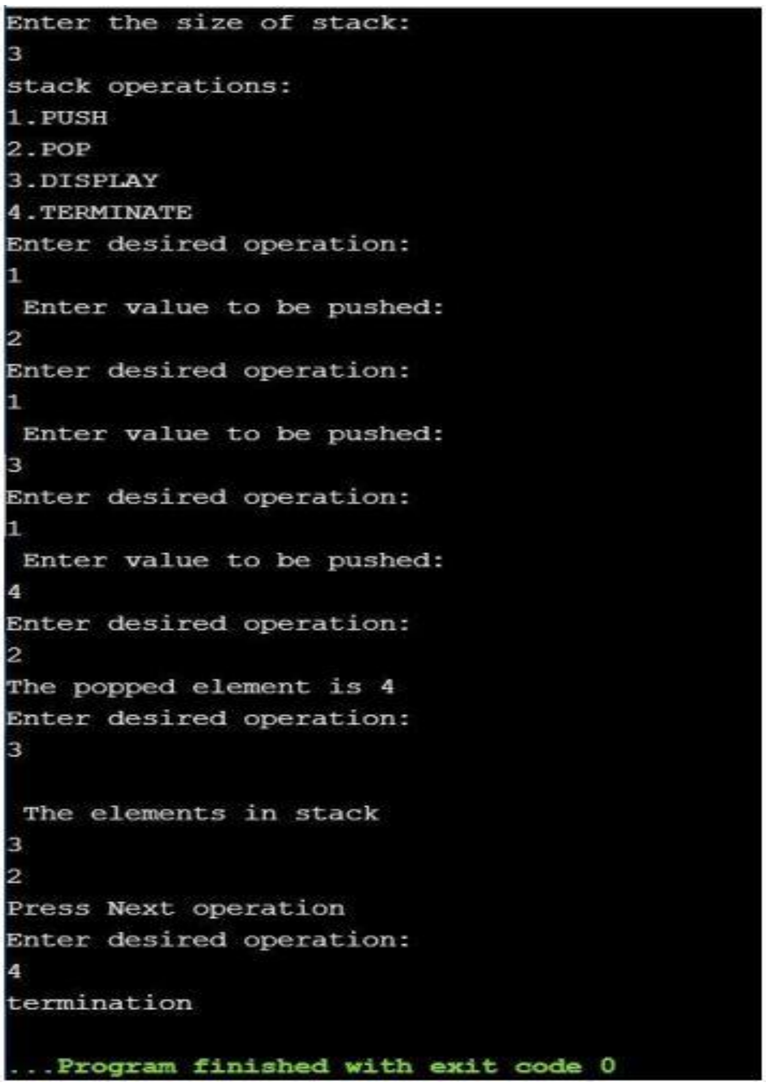
The program should print appropriate messages for stack overflow, stack underflow

```c
#include<stdio.h>
int stack[10],operation,n,top,x,i;
void push(void);
void pop(void);
void display(void);
int main()
{
top=-1;
printf("Enter the size of stack: \n");
scanf("%d",&n);
printf("stack operations: \n");
printf("1.PUSH\n");
printf("2.POP\n");
printf("3.DISPLAY\n");
printf("4.TERMINATE\n");
do
{
printf("Enter desired operation:\n");
scanf("%d",&operation);
switch(operation)
{
case 1:
{
push();
break;
}
case 2:
{
pop();
break;
}
case 3:
{
display();
break;
}
case 4:
{
printf("termination ");
break;
} }
}
while(operation!=4);
return 0;
}
void push()
{
if(top>=n-1)
{
printf("stack is over flow\n");
}
else
{
printf(" Enter value to be pushed:\n");
scanf("%d",&x);
top++;
stack[top]=x;
}
}
void pop()
{
if(top<=-1)
{
printf( "Stack is under flow");
}
else
{
printf("The popped elements is %d",stack[top]);
top--;
}
}
void display()
{
if(top>=0)
{
printf("\n The elements in stack \n");
for(i=top; i>=0; i--)
printf("%d\n",stack[i]);
printf("Press Next operation\n");
}
else
{
printf("The stack is empty\n");
}
}
```

overflow



```
Enter the size of stack:
5
stack operations:
1.PUSH
2.POP
3.DISPLAY
4.TERMINATE
Enter desired operation:
1
 Enter value to be pushed:
2
Enter desired operation:
1
 Enter value to be pushed:
3
Enter desired operation:
1
 Enter value to be pushed:
4
Enter desired operation:
1
 Enter value to be pushed:
5
Enter desired operation:
1
 Enter value to be pushed:
6
Enter desired operation:
1
stack is over flow
Enter desired operation:
4
termination
```

output 

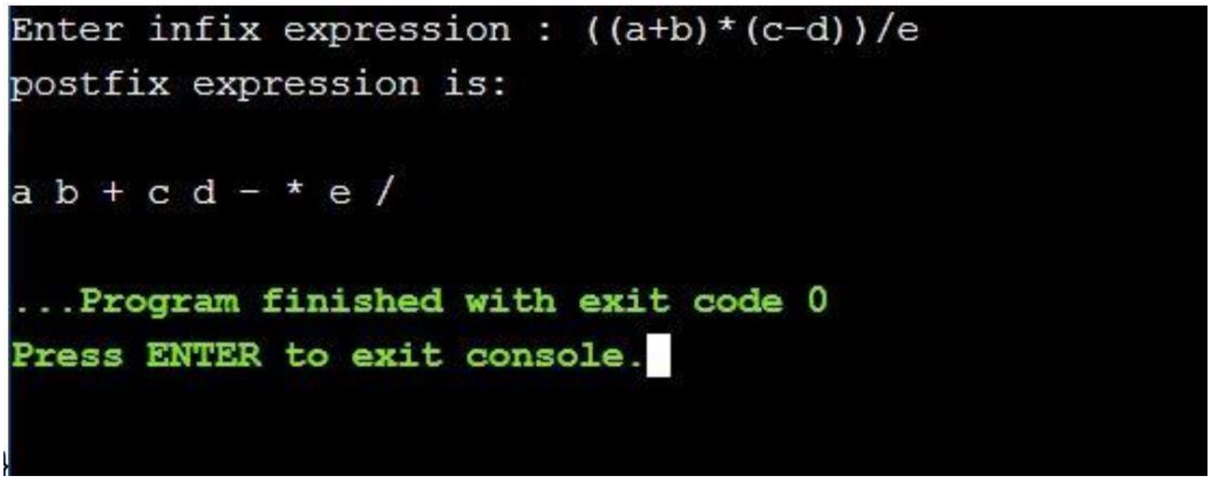
```
Enter the size of stack:
3
stack operations:
1.PUSH
2.POP
3.DISPLAY
4.TERMINATE
Enter desired operation:
1
 Enter value to be pushed:
2
Enter desired operation:
1
 Enter value to be pushed:
3
Enter desired operation:
1
 Enter value to be pushed:
4
Enter desired operation:
2
The popped element is 4
Enter desired operation:
3

 The elements in stack
3
2
Press Next operation
Enter desired operation:
4
termination

...Program finished with exit code 0
```

LAB 3:

WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

```
#include<stdio.h>
#include<ctype.h>
char stack[100];
int top = -1;
void push(char y)
{
stack[++top] = y;
}
char pop()
{
if(top == -1)
```

```
return -1;
else
return stack[top--];
}
int precedence(char y)
{
if(y == '(')
return 0;
if(y == '+' || y == '-')
return 1;
if(y == '*' || y == '/')
return 2;
return 0;
}
int main()
{
char exp[100];
char *ex, y;
printf("Enter infix expression : ");
scanf("%s",exp);
printf("postfix expression is:\n");
printf("\n");
ex = exp;
while(*ex != '\0')
{
if(isalnum(*ex))
printf("%c ",*ex);
else if(*ex == '(')
push(*ex);
else if(*ex == ')')
{
while((y = pop()) != '(')
printf("%c ", y);
}
else
{
while(precedence(stack[top]) >= precedence(*ex))
printf("%c ",pop());
push(*ex);
}
ex++;
}
while(top != -1)
{
printf("%c ",pop());
}return 0;
}
```



```
Enter infix expression : ((a+b)*(c-d))/e
postfix expression is:

a b + c d - * e /

...Program finished with exit code 0
Press ENTER to exit console.
```

LAB 4:

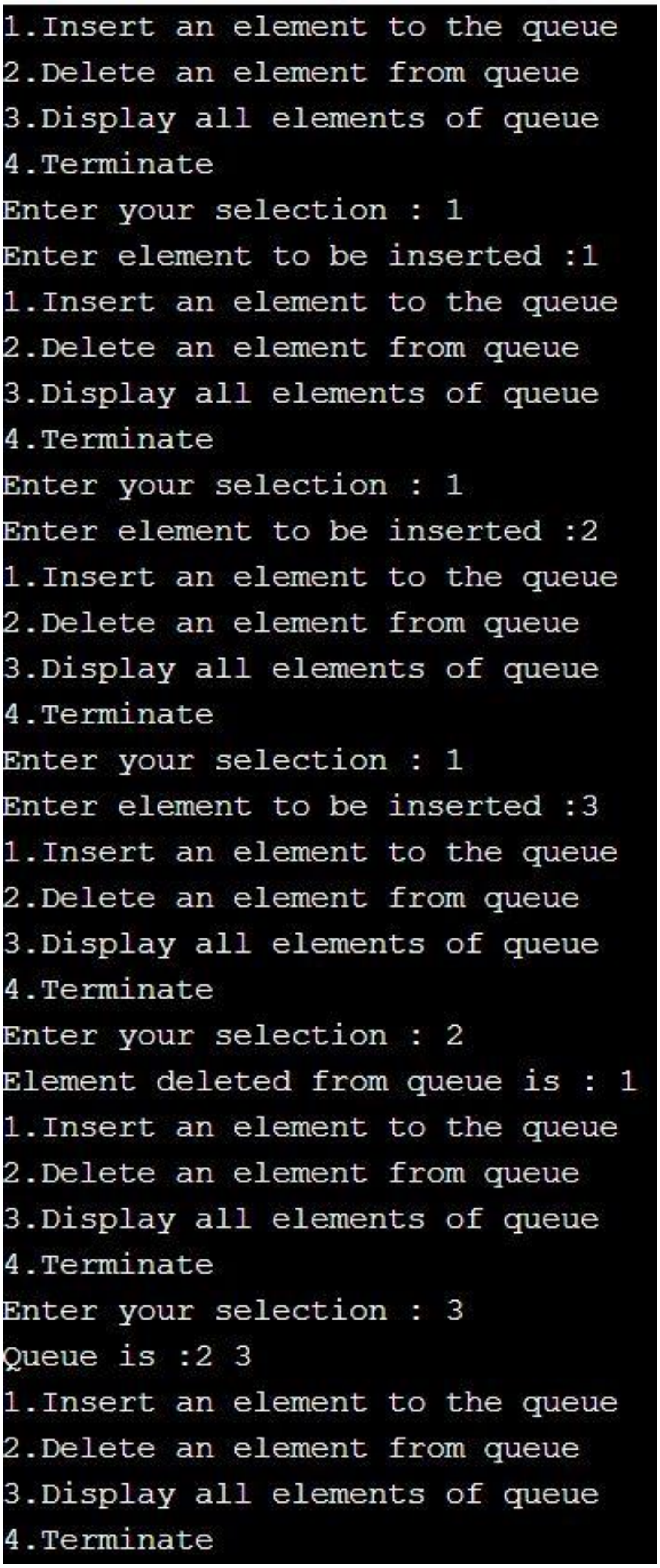
WAP to simulate the working of a queue of integers using an array. Provide the following operations a) Insert b) Delete c) Display

The program should print appropriate messages for queue empty and queue overflow conditions

```
#include <stdio.h>
#include<stdlib.h>
#define MAX 5
void insert();
void delete();
void display();
int array[MAX];
int rear = - 1;
int front = - 1;
int main()
{
int select;
while (1)
{
printf("1.Insert an element to the queue \n");
printf("2.Delete an element from queue \n");
printf("3.Display all elements of queue \n");
printf("4.Terminate \n");
printf("Enter your selection : ");
scanf("%d", &select);
switch (select)
{
case 1:
insert();
break;
case 2:
delete();
break;
case 3:
display();
break;
case 4:
exit(1);
}
}
}
void insert()
{
int add;
if (rear== MAX - 1)
printf("Queue Overflow \n");
else
{
if (front== - 1)
front = 0;
printf("Enter element to be inserted :");
scanf("%d", &add);
rear= rear + 1;
array[rear]= add;
}
}
void delete()
```

```
{
if (front== - 1 || front > rear)
{
printf("Queue Underflow \n");
return ;
}
else
{
printf("Element deleted from queue is : %d\n",array[front]);
front= front + 1;
}
}
void display()
{
int i;
if (front== - 1)
printf("Queue is empty \n");
else
{
printf("Queue is :");
for (i= front; i <= rear; i++)
printf("%d ",array[i]);
printf("\n");
}
}
```

Output:


```
1.Insert an element to the queue
2.Delete an element from queue
3.Display all elements of queue
4.Terminate
Enter your selection : 1
Enter element to be inserted :1
1.Insert an element to the queue
2.Delete an element from queue
3.Display all elements of queue
4.Terminate
Enter your selection : 1
Enter element to be inserted :2
1.Insert an element to the queue
2.Delete an element from queue
3.Display all elements of queue
4.Terminate
Enter your selection : 1
Enter element to be inserted :3
1.Insert an element to the queue
2.Delete an element from queue
3.Display all elements of queue
4.Terminate
Enter your selection : 2
Element deleted from queue is : 1
1.Insert an element to the queue
2.Delete an element from queue
3.Display all elements of queue
4.Terminate
Enter your selection : 3
Queue is :2 3
1.Insert an element to the queue
2.Delete an element from queue
3.Display all elements of queue
4.Terminate
```

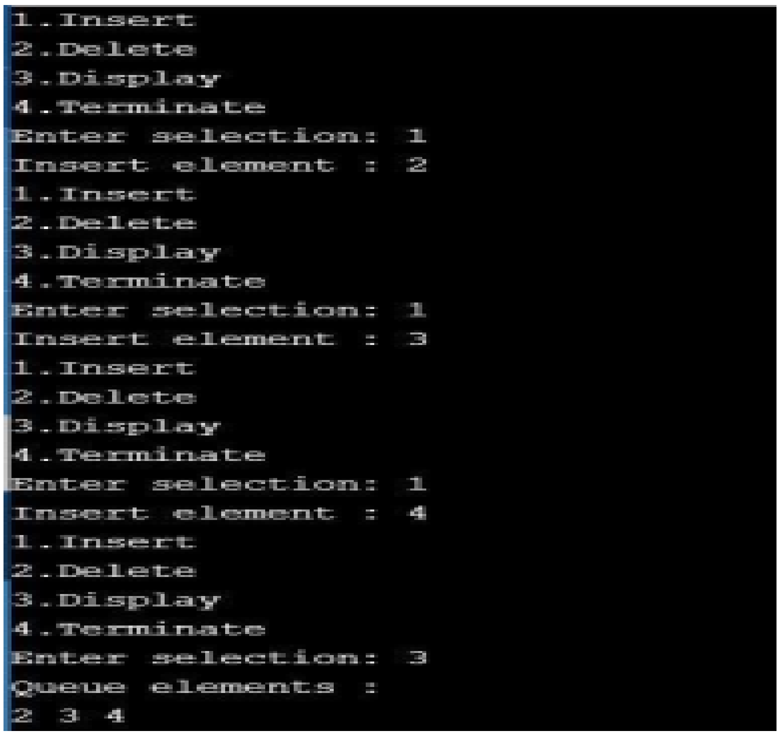
LAB 5:

Circular queue execution

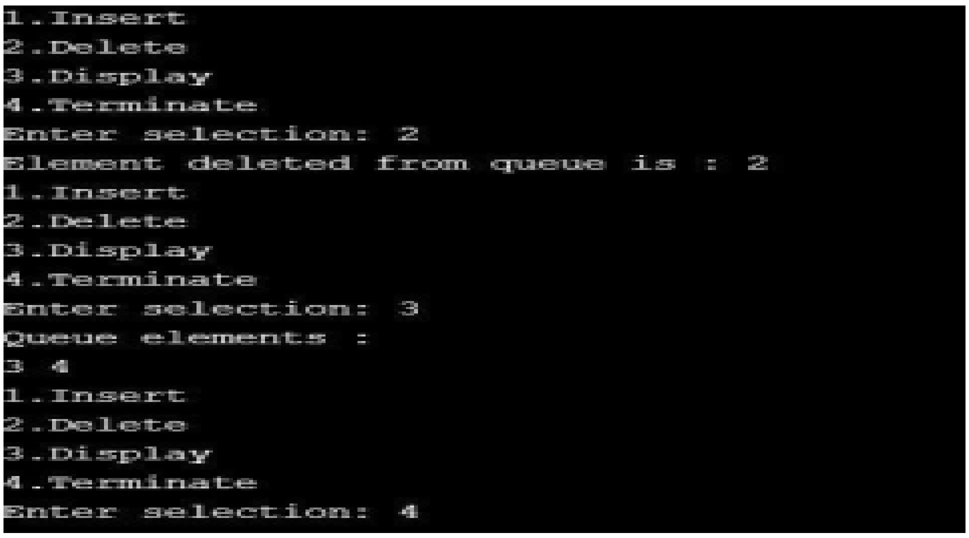
```
#include<stdio.h>
#define MAX 10
int cir_queue[MAX];
int front = -1;
int rear = -1;
void insert(int item)
{
if((front == 0 && rear == MAX-1) || (front == rear+1))
{
printf("Queue Overflow \n");
return;
}
if (front == -1)
{
front = 0;
rear = 0;
}
else
{
if(rear == MAX-1)
rear = 0;
else
rear = rear+1;
}
cir_queue[rear] = item ;
}
void delete()
{
if (front == -1)
{
printf("Queue Underflow\n");
return ;
}
printf("Element deleted from queue is : %d\n",cir_queue[front]);
```

```
if(front == rear)
{
front = -1;
rear=-1;
}
else
{
if(front == MAX-1)
front = 0;
else
front = front+1;
}
}
void display()
{
int fpos = front,rpos = rear;
if(front == -1)
{
printf("Queue is empty\n");
return;
}
printf("Queue elements :\n");
if( fpos <= rpos )
while(fpos <= rpos)
{
printf("%d ",cir_queue[fpos]);
fpos++;
}
else
{
while(fpos <= MAX-1)
{
printf("%d ",cir_queue[fpos]);
fpos++;
}
fpos = 0;
while(fpos <= rpos)
{
printf("%d ",cir_queue[fpos]);
fpos++;
}
}
printf("\n");
}
int main()
{
int select,item;
do
{
printf("1.Insert\n");
printf("2.Delete\n");
printf("3.Display\n");
printf("4.Terminate\n");
printf("Enter selection: ");
scanf("%d",&select);
switch(select)
{
case 1 :
printf("Insert element : ");
scanf("%d", &item);
insert(item);
break;
case 2 :
delete();
break;
case 3:
display();
break;
case 4:
break;
default:
printf("Wrong choice\n");
}
}while(select!=4);
```

```
return 0;
}
```



```
1.Insert
2.Delete
3.Display
4.Terminate
Enter selection: 1
Insert element : 2
1.Insert
2.Delete
3.Display
4.Terminate
Enter selection: 1
Insert element : 3
1.Insert
2.Delete
3.Display
4.Terminate
Enter selection: 1
Insert element : 4
1.Insert
2.Delete
3.Display
4.Terminate
Enter selection: 3
Queue elements :
2 3 4
```



```
1.Insert
2.Delete
3.Display
4.Terminate
Enter selection: 2
Element deleted from queue is : 2
1.Insert
2.Delete
3.Display
4.Terminate
Enter selection: 3
Queue elements :
3 4
1.Insert
2.Delete
3.Display
4.Terminate
Enter selection: 4
```

LAB 6:

WAP to Implement Singly Linked List with following operations

a) Create a linked list.

b) Insertion of a node at first position, at any position and at end of list.

c) Deletion of first element, specified element and last element in the list.

d) Display the contents of the linked list.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
struct node *prev;
struct node *next;
int data;
};
struct node *head;
void insertion_beginning();
void insertion_last();
void insertion_specified();
void deletion_beginning();
void deletion_last();
void deletion_specified();
void display();
void search();
void main ()
{
int choice =0;
while(choice != 9)
{
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 the node
after the given data\n7.Search\n8.Show\n9.Exit\n");
printf("\nEnter your choice?\n");
scanf("\n%d",&choice);
switch(choice)
{
case 1:
insertion_beginning();
break;
case 2:
```

```
insertion_last();
break;
case 3:
insertion_specified();
break;
case 4:
deletion_beginning();
break;
case 5:
deletion_last();
break;
case 6:
deletion_specified();
break;
case 7:
search();
break;
case 8:
display();
break;
case 9:
exit(0);
break;
default:
printf("Please enter valid choice..");
}
}
}
void insertion_beginning()
{
struct node *ptr;
int item;
ptr = (struct node *)malloc(sizeof(struct node));
if(ptr == NULL)
{
printf("\nOVERFLOW");
}
```

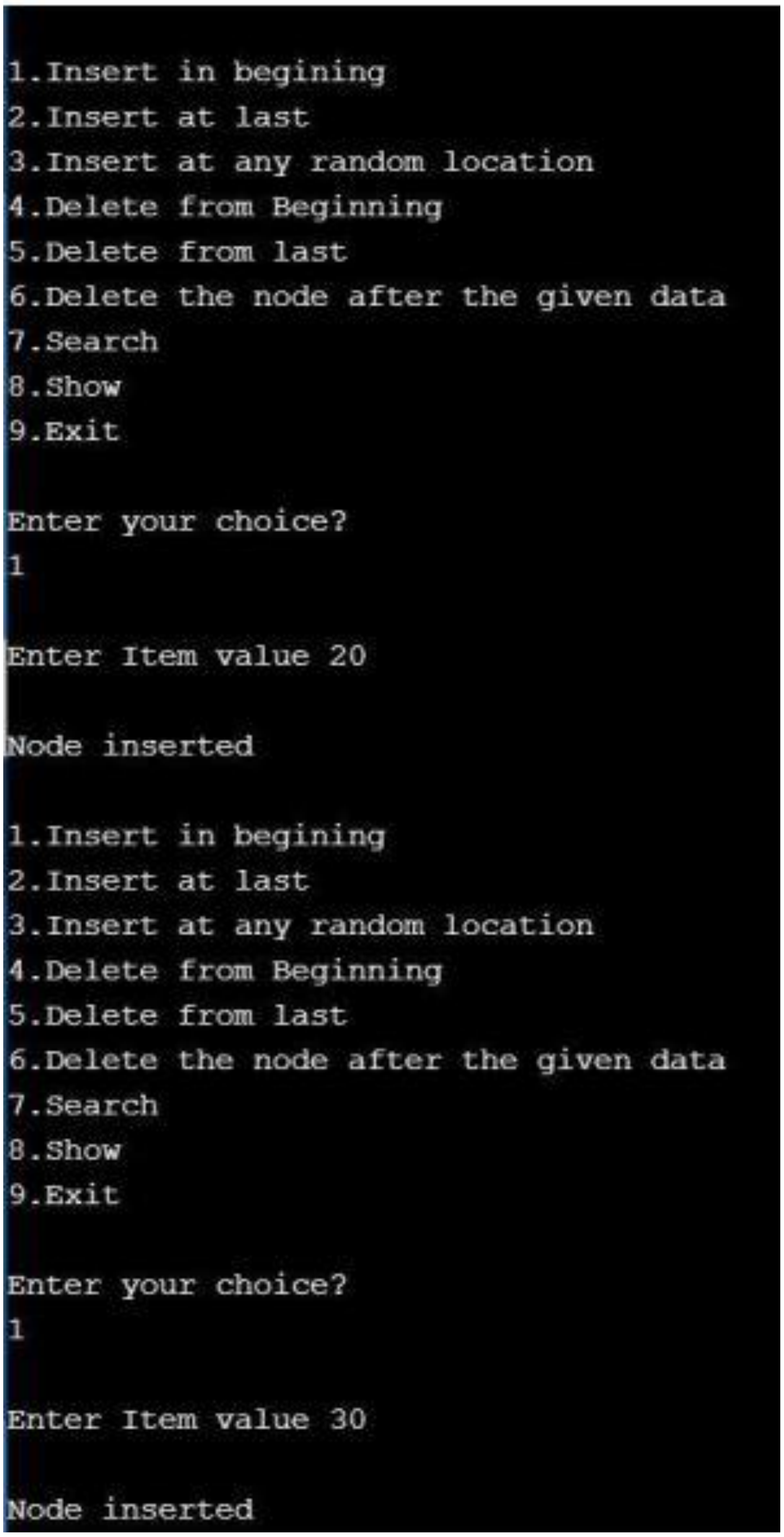
```
else
{
printf("\nEnter Item value");
scanf("%d",&item);
if(head==NULL)
{
ptr->next = NULL;
ptr->prev=NULL;
ptr->data=item;
head=ptr;
}
else
{
ptr->data=item;
ptr->prev=NULL;
ptr->next = head;
head->prev=ptr;
head=ptr;
}
printf("\nNode inserted\n");
}
}
void insertion_last()
{
struct node *ptr,*temp;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

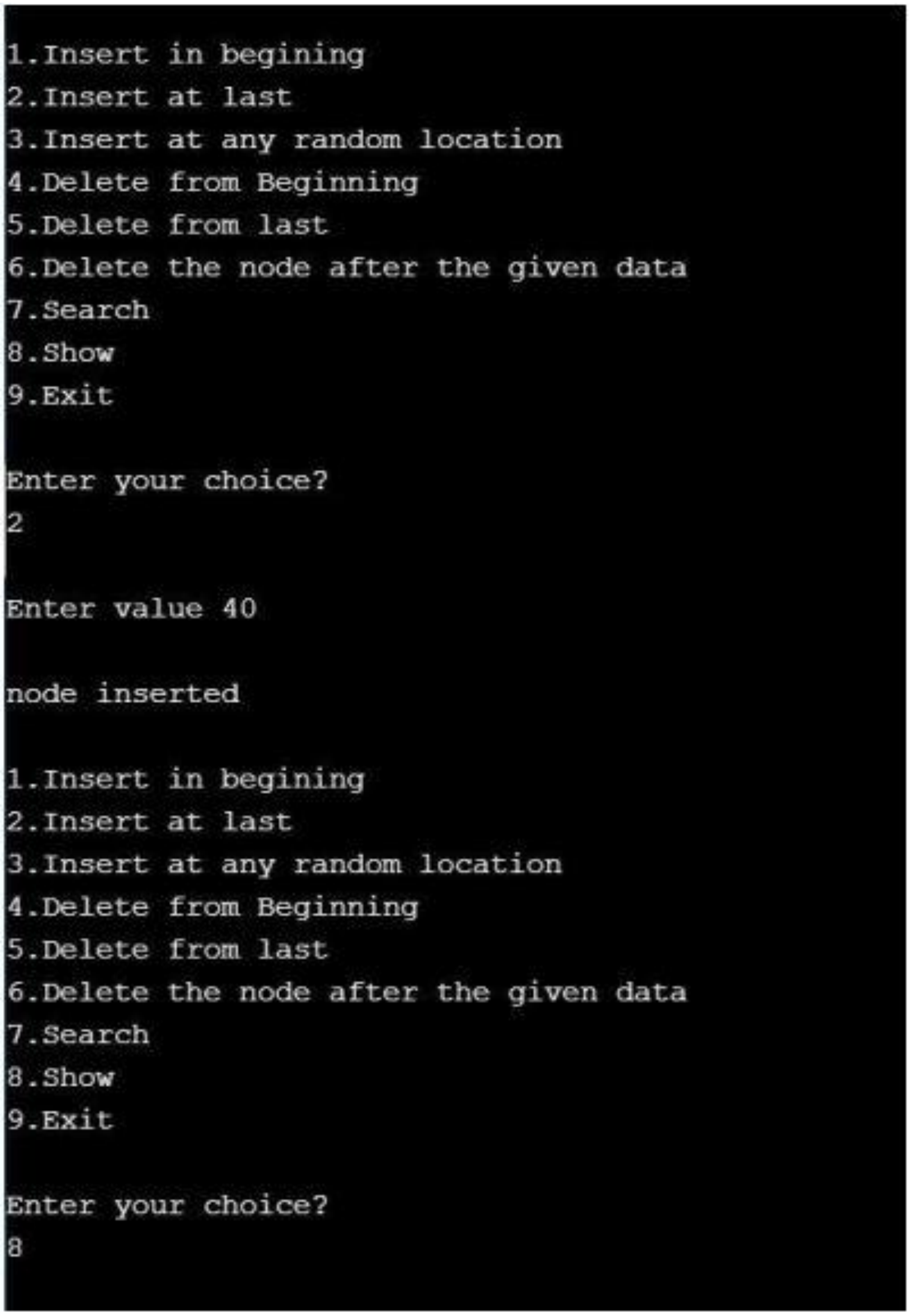
```
{
ptr->next = NULL;
ptr->prev = NULL;
head = ptr;
}
else
{
temp = head;
while(temp->next!=NULL)
{
temp = temp->next;
}
temp->next = ptr;
ptr ->prev=temp;
ptr->next = NULL;
}
}
printf("\nnode inserted\n");
}
void insertion_specified()
{
struct node *ptr,*temp;
int item,loc,i;
ptr = (struct node *)malloc(sizeof(struct node));
if(ptr == NULL)
{
printf("\n OVERFLOW");
}
else
{
temp=head;
printf("Enter the location");
scanf("%d",&loc);
for(i=0;i<loc;i++)
{
temp = temp->next;
if(temp == NULL)
{
printf("\n There are less than %d elements", loc);
return;
}
}
printf("Enter value");
scanf("%d",&item);
ptr->data = item;
ptr->next = temp->next;
ptr -> prev = temp;
temp->next = ptr;
temp->next->prev=ptr;
printf("\nnode inserted\n");
}
}
void deletion_beginning()
{
struct node *ptr;
if(head == NULL)
{
printf("\n UNDERFLOW");
}
else if(head->next == NULL)
{
head = NULL;
free(head);
printf("\nnode deleted\n");
}
else
{
ptr = head;
head = head -> next;
head -> prev = NULL;
free(ptr);
printf("\nnode deleted\n");
}
}
```

```
void deletion_last()
{
struct node *ptr;
if(head == NULL)
{
printf("\n UNDERFLOW");
}
else if(head->next == NULL)
{
head = NULL;
free(head);
printf("\nnode deleted\n");
}
else
{
ptr = head;
if(ptr->next != NULL)
{
ptr = ptr -> next;
}
ptr -> prev -> next = NULL;
free(ptr);
printf("\nnode deleted\n");
}
}
void deletion_specified()
{
struct node *ptr, *temp;
int val;
printf("\n Enter the data after which the node is to be deleted : ");
scanf("%d", &val);
ptr = head;
while(ptr -> data != val)
ptr = ptr -> next;
if(ptr -> next == NULL)
{
printf("\nCan't delete\n");
```

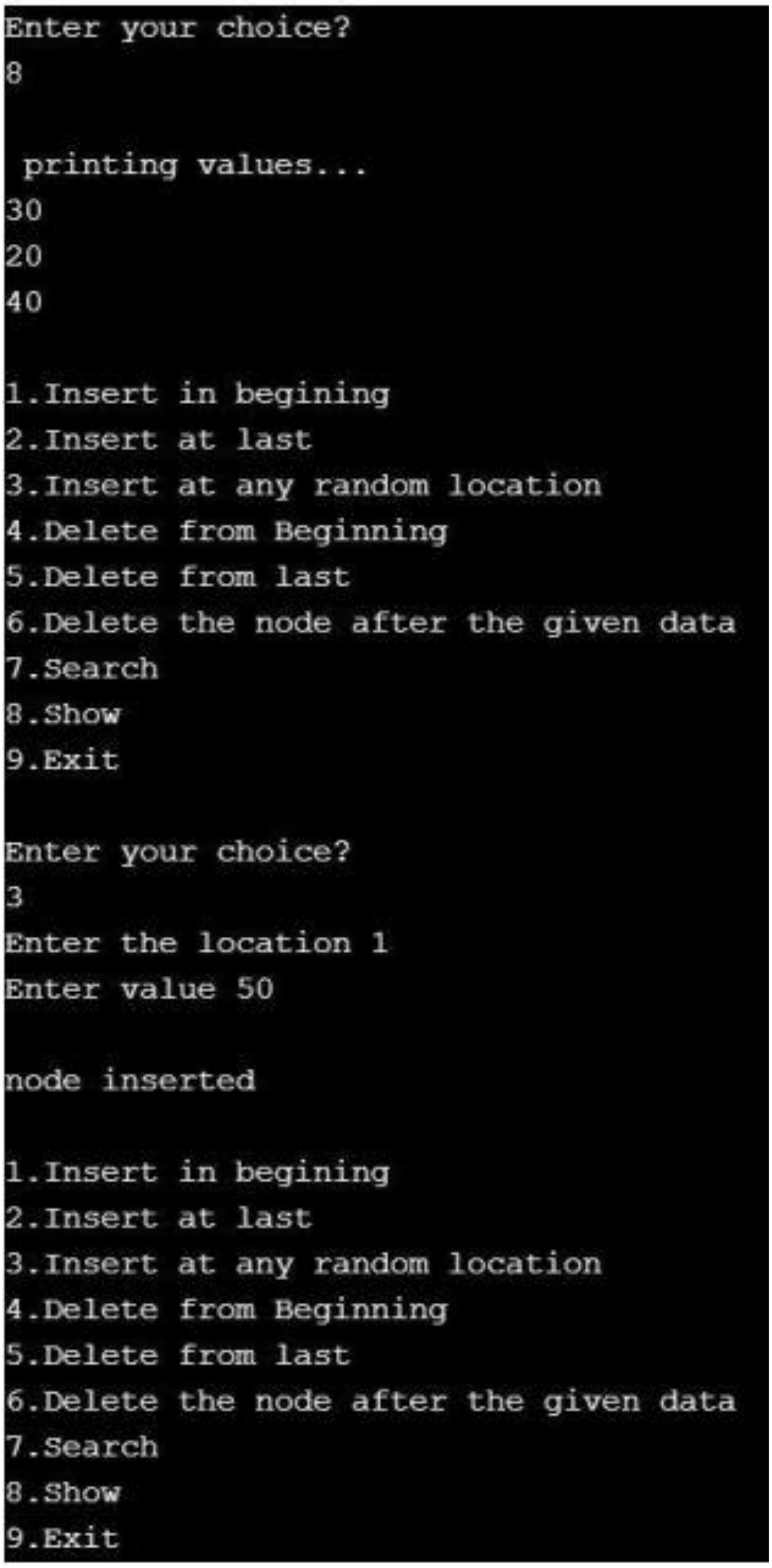
```
}
else if(ptr -> next -> next == NULL)
{
ptr ->next = NULL;
}
else
{
temp = ptr -> next;
ptr -> next = temp -> next;
temp -> next -> prev = ptr;
free(temp);
printf("\nnode deleted\n");
}
}
void display()
{
struct node *ptr;
printf("\n printing values...\n");
ptr = head;
while(ptr != NULL)
{
printf("%d\n",ptr->data);
ptr=ptr->next;
}
}
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("\nitem found at location %d ",i+1);
flag=0;
break;
}
else
{
flag=1;
}
i++;
ptr = ptr -> next;
}
if(flag==1)
{
printf("\nItem not found\n");
}
}
}
```

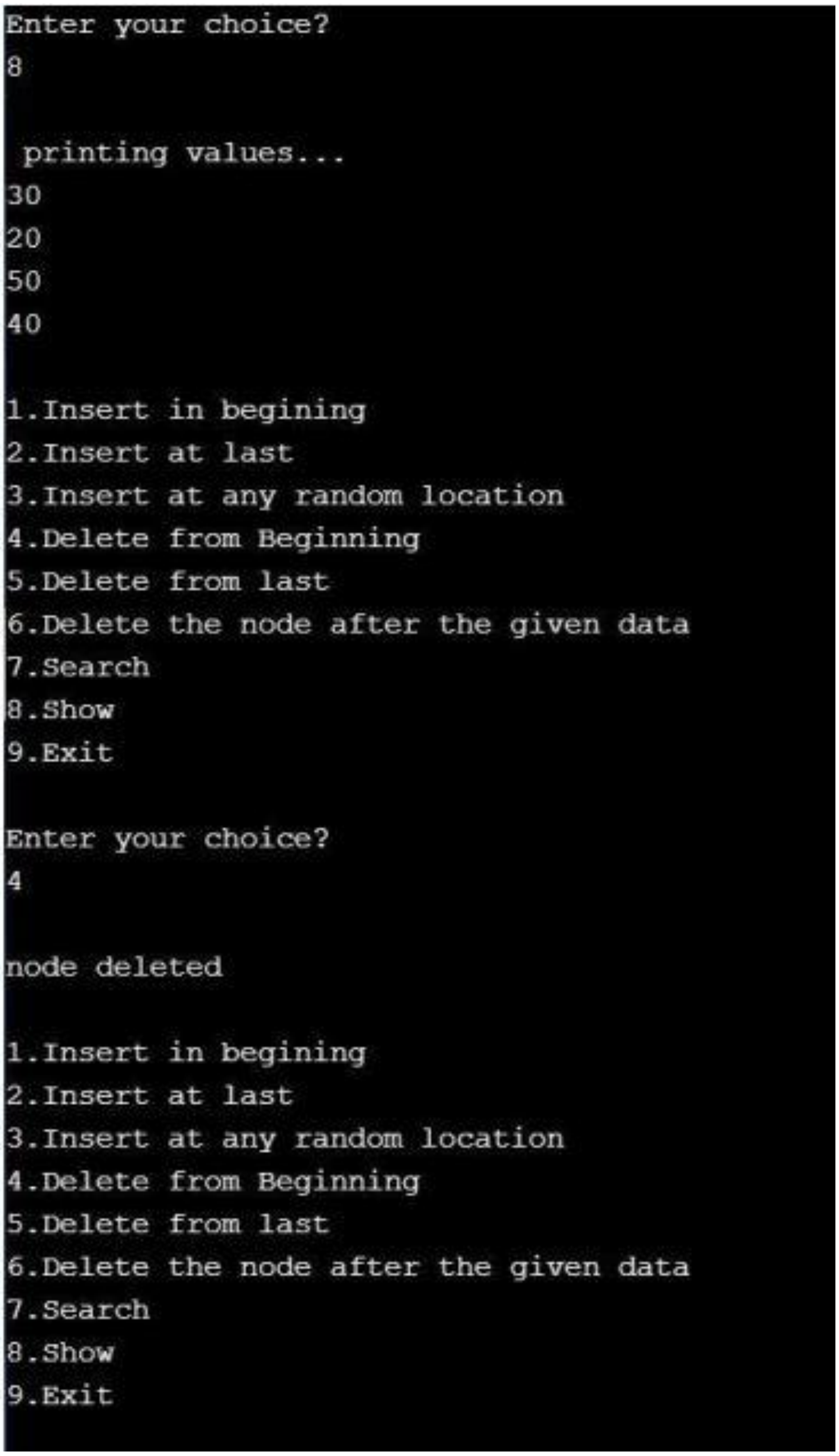

```
1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Search
8.Show
9.Exit

Enter your choice?
1

Enter Item value 20

Node inserted

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Search
8.Show
9.Exit

Enter your choice?
1

Enter Item value 30

Node inserted
```

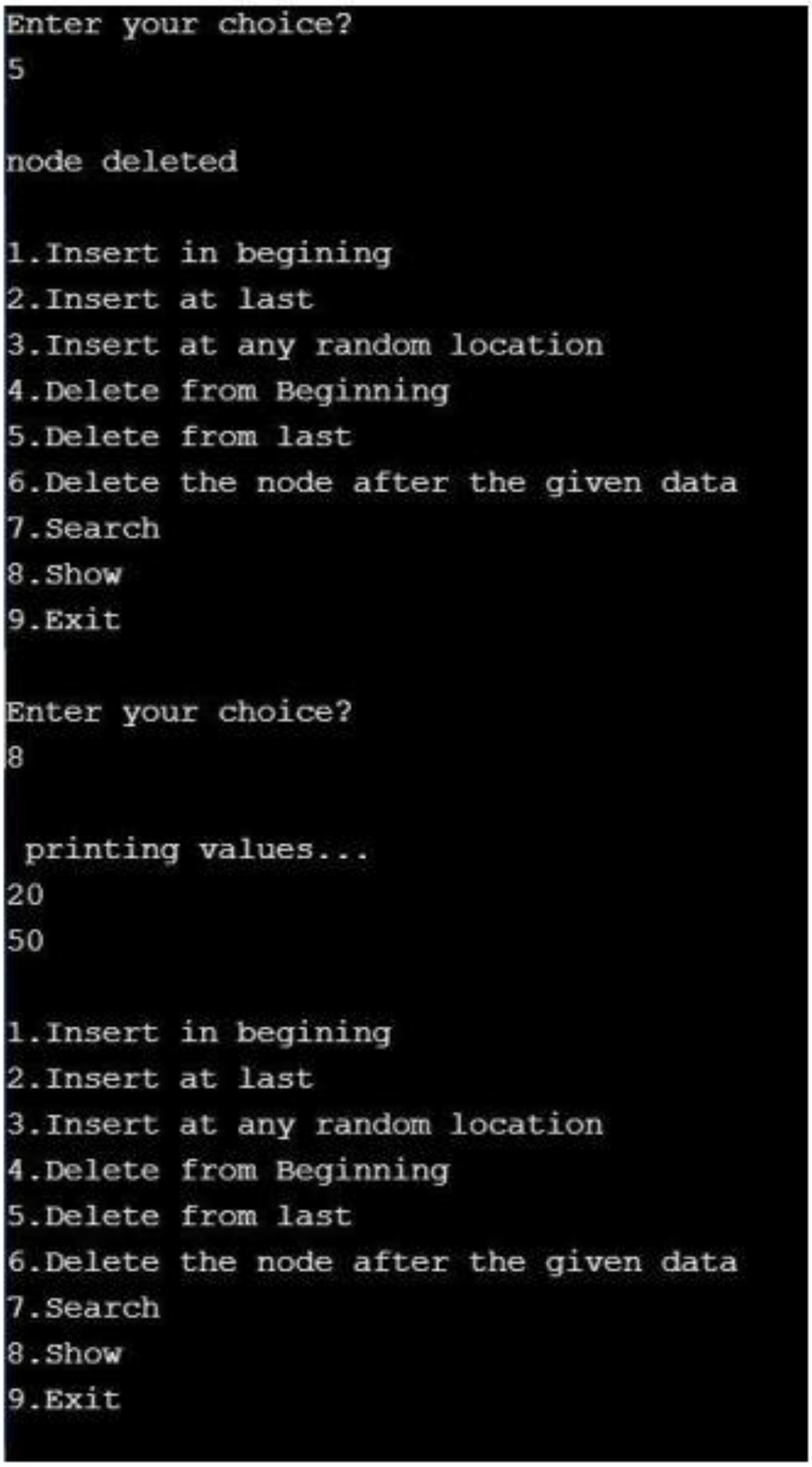


```
1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Search
8.Show
9.Exit

Enter your choice?
2

Enter value 40

node inserted

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Search
8.Show
9.Exit

Enter your choice?
8
```



```
Enter your choice?
8

 printing values...
30
20
40

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Search
8.Show
9.Exit

Enter your choice?
3
Enter the location 1
Enter value 50

node inserted

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Search
8.Show
9.Exit
```


```
Enter your choice?
8

 printing values...
30
20
50
40

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Search
8.Show
9.Exit

Enter your choice?
4

node deleted

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Search
8.Show
9.Exit
```



```
Enter your choice?
5

node deleted

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Search
8.Show
9.Exit

Enter your choice?
8

 printing values...
20
50

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Search
8.Show
9.Exit
```

LAB 7(stack and queue operations also done in this program):

WAP Implement Single Link List with following operations

a) Sort the linked list.

b) Reverse the linked list.

c) Concatenation of two linked lists

d) implement Stack & Queues using Linked Representation

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

struct node
{
        int data;
        struct node *next;
};
struct node *head;
struct node *head2;

//stack operations
void push()
{
        struct node *ptr;
        int new_data;
        ptr = (struct node *)malloc(sizeof(struct node));

        if(ptr == NULL)
        {
                printf("\nOVERFLOW!!!");
        }
        else
        {
                printf("\nEnter the Value to be inserted:");
                scanf("%d",&new_data);
                ptr->data = new_data;
                ptr->next = head;
                head = ptr;
                printf("\nNODE INSERTED AT THE TOP OF THE STACK\n");
        }
}
void pop()
{
        struct node *ptr;
        if(head == NULL)
        {
                printf("EMPTY LIST!!!");
        }
        else
        {
                ptr = head;
                head = ptr->next;
                free(ptr);
                printf("\nNODE DELETED FROM TOP OF THE STACK\n");
        }
}

//queue operations
void enqueue()
{
        struct node *ptr,*temp;
        int new_data;
        ptr = (struct node *)malloc(sizeof(struct node));

        printf("\nEnter the Value to be inserted:");
        scanf("%d",&new_data);
        ptr->data = new_data;
        if(head == NULL)
        {
                ptr->next = NULL;
                head = ptr;
                printf("\nNODE INSERTED AT REAR OF THE QUEUE\n");
        }
        else
        {
                temp = head;
                while(temp->next != NULL)
                {
                        temp = temp->next;
                }
                temp->next = ptr;
                ptr->next = NULL;
                printf("\nNODE INSERTED AT REAR OF THE QUEUE\n");
        }
}
void dequeue()
{
        struct node *ptr;
        if(head == NULL)
        {
                printf("EMPTY LIST!!!");
        }
        else
        {
                ptr = head;
                head = ptr->next;
                free(ptr);
                printf("\nNODE DELETED FROM FRONT OF THE QUEUE\n");
        }
}

//Display List
void display()
{
        struct node *ptr;
        ptr = head;
        if(ptr == NULL)
        {
                printf("EMPTY LIST!!!INSERT FEW ELEMENTS!!");
        }
        else
        {
                printf("\n\nLIST-->");
                while(ptr != NULL)
                {
                        printf("\t%d",ptr->data);
                        ptr = ptr->next;
                }
```

```
	}
}

//sort Linked list in ascending order
void sort()
{
	struct node *ptr = head;
	struct node *temp = NULL;
	int i;

	if(head == NULL)
	{
		return;
	}
	else
	{
		while(ptr != NULL)
		{
			temp = ptr->next;
			while(temp != NULL)
			{
				if(ptr->data >temp->data)
				{
					i = ptr->data;
					ptr->data = temp->data;
					temp->data = i;
				}
				temp = temp->next;
			}
```

```
                        ptr = ptr->next;
                }
        }
}

//reverse Linked List
void reverse()
{
        struct node *prev = NULL;
        struct node *next = NULL;
        struct node *ptr = head;
        while(ptr != NULL)
        {
                next = ptr->next;
                ptr->next = prev;
                prev = ptr;
                ptr = next;
        }
        head = prev;
}

//create list
struct node *create_list(struct node *head)
{
        struct node *ptr,*temp;
        int i,n,new_data;

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

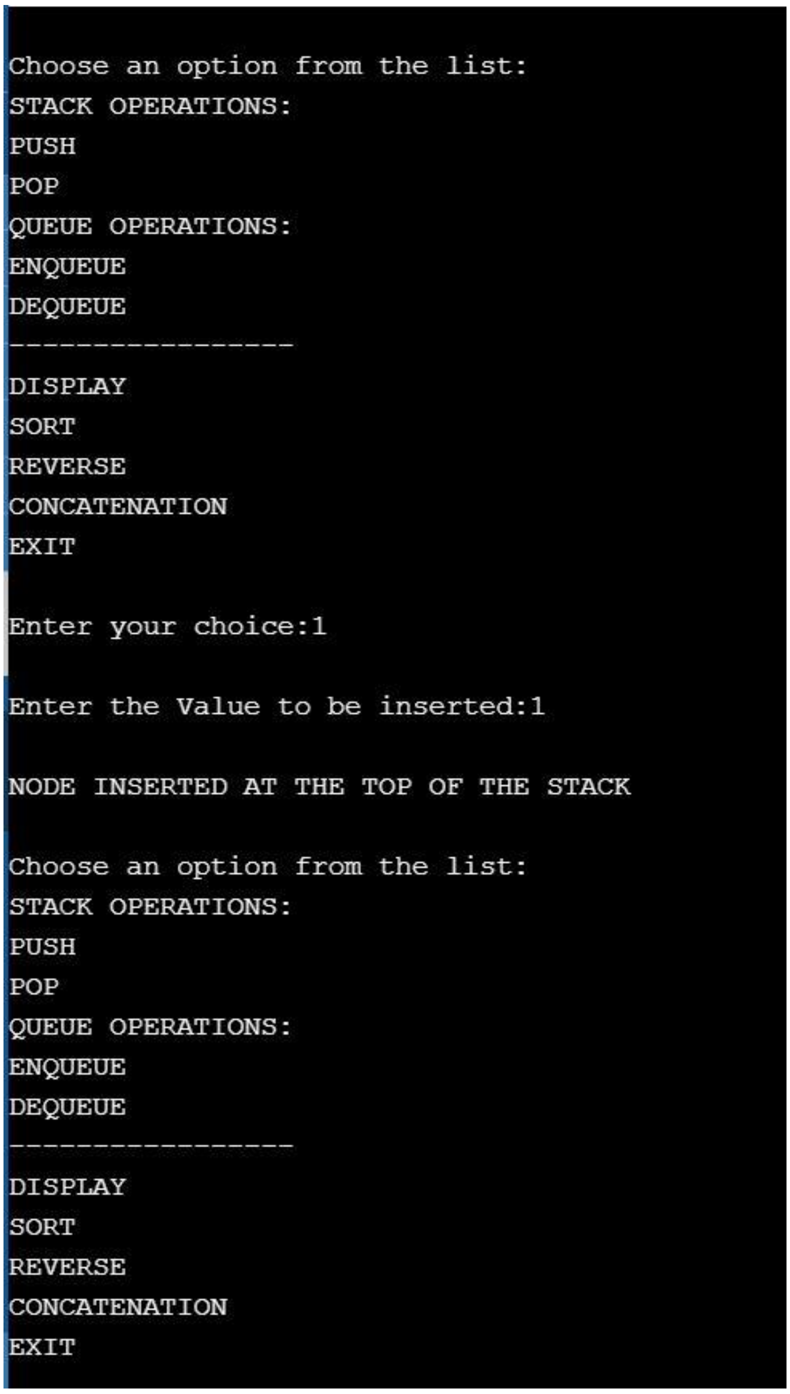
```
        head = NULL;

        if(n == 0)

        {

                return head;

        }

        for(i=1;i<=n;i++)

        {

                ptr = (struct node *)malloc(sizeof(struct node));

                printf("Enter the element to be inserted : ");

                scanf("%d",&new_data);

                ptr->data = new_data;

                if(head == NULL)

                {

                        ptr->next = NULL;

                        head = ptr;

                }

                else

                {

                        temp = head;

                        while(temp->next != NULL)

                        {

                                temp = temp->next;

                        }

                        temp->next = ptr;

                        ptr->next = NULL;

                }

        }

        return head;

}
```

```
//concatenate two lists
struct node *concatenate(struct node *head, struct node *head2)
{
        struct node *ptr;
        if(head == NULL)
        {
                head = head2;
                return head;
        }
        if(head2 == NULL)
        {
                return head;
        }
        ptr = head;
        while(ptr->next != NULL)
        {
                ptr = ptr->next;
        }
        ptr->next = head2;
        return head;
}



int main()
{
        int choice = 0;
```

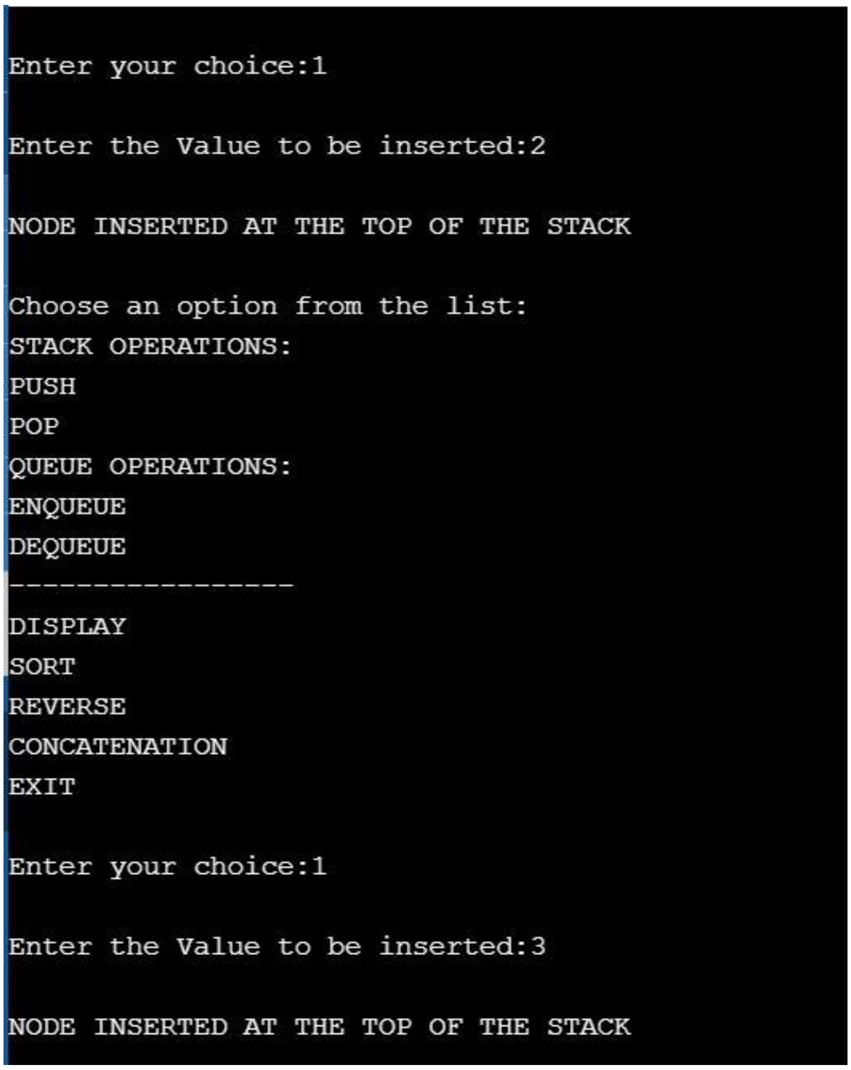
```
while(1)
{

        printf("Choose an option from the list:");
        printf("\nSTACK OPERATIONS:\nPUSH\nPOP");
        printf("\nQUEUE OPERATIONS:\nENQUEUE\nDEQUEUE");
        printf("\n-----------------");
        printf("\nDISPLAY\nSORT\nREVERSE\nCONCATENATION\nEXIT\n");
        printf("\nEnter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {
                case 1: push();
                                break;
                case 2: pop();
                                break;
                case 3: enqueue();
                                break;
                case 4: dequeue();
                                break;
                case 5: display();
                                break;
                case 6: sort();
                                printf("\nSorted List::");
                                display();
                                break;
                case 7: reverse();
                                printf("\nReversed List::");
                                display();
```

```
                    break;
            case 8: printf("\nCreate a Second list-->");
                    head2 = create_list(head2);
                    printf("\nList1:");
                    display();
                    struct node *ptr;
                    ptr = head2;
                    if(ptr == NULL)
                    {
                        printf("LIST2 IS EMPTY!!!");
                    }
                    else
                    {
                        printf("\n\nLIST2-->");
                        while(ptr != NULL)
                        {
                            printf("\t%d",ptr->data);
                            ptr = ptr->next;
                        }
                    }
                    head = concatenate(head,head2);
                    printf("\n\nConcatenated List:");
                    display();
                    break;
            case 9: exit(1);
            default:
                printf("\nINVALID CHOICE!!!\n");

        }
    }
```

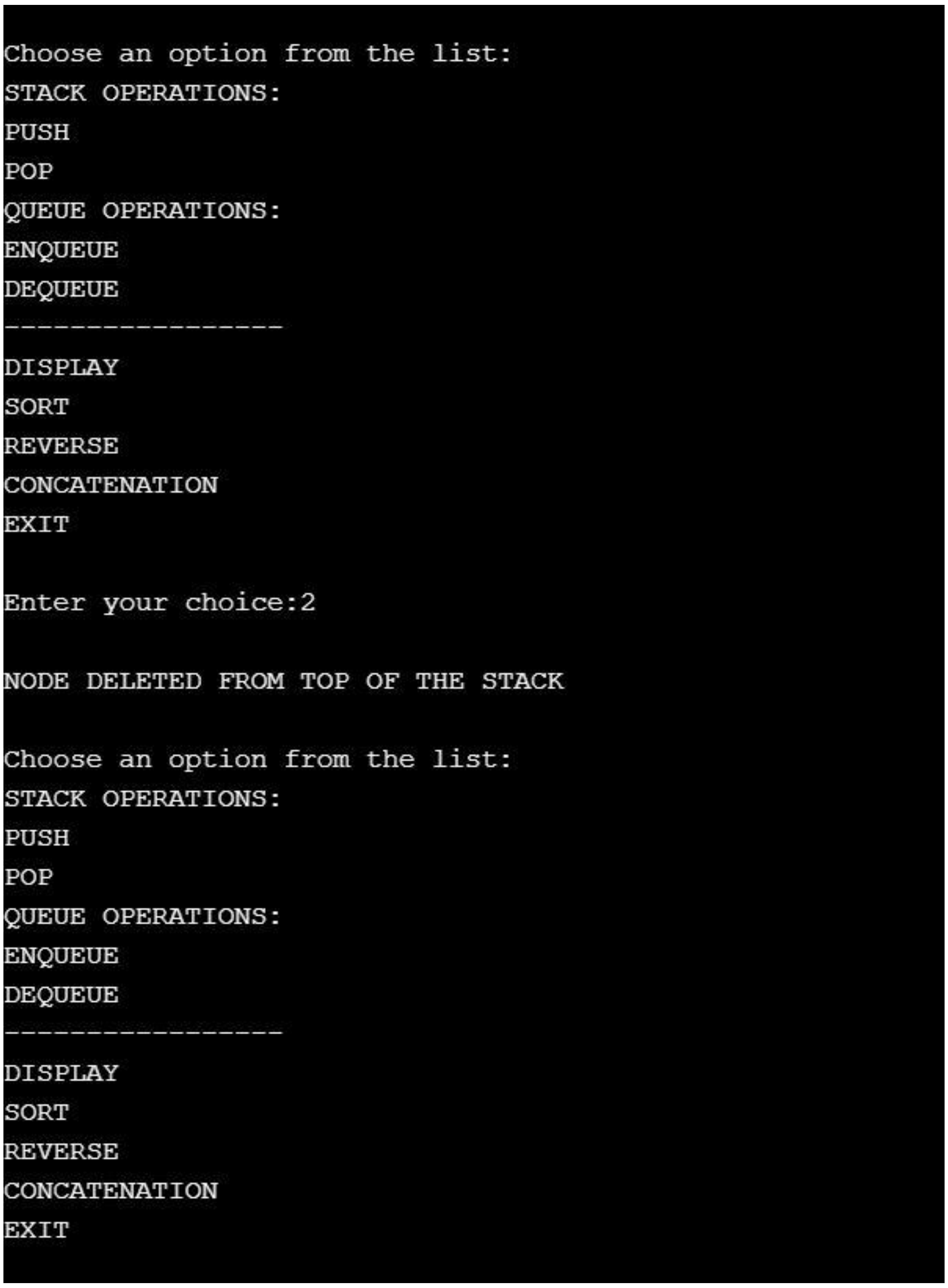
```
}
```



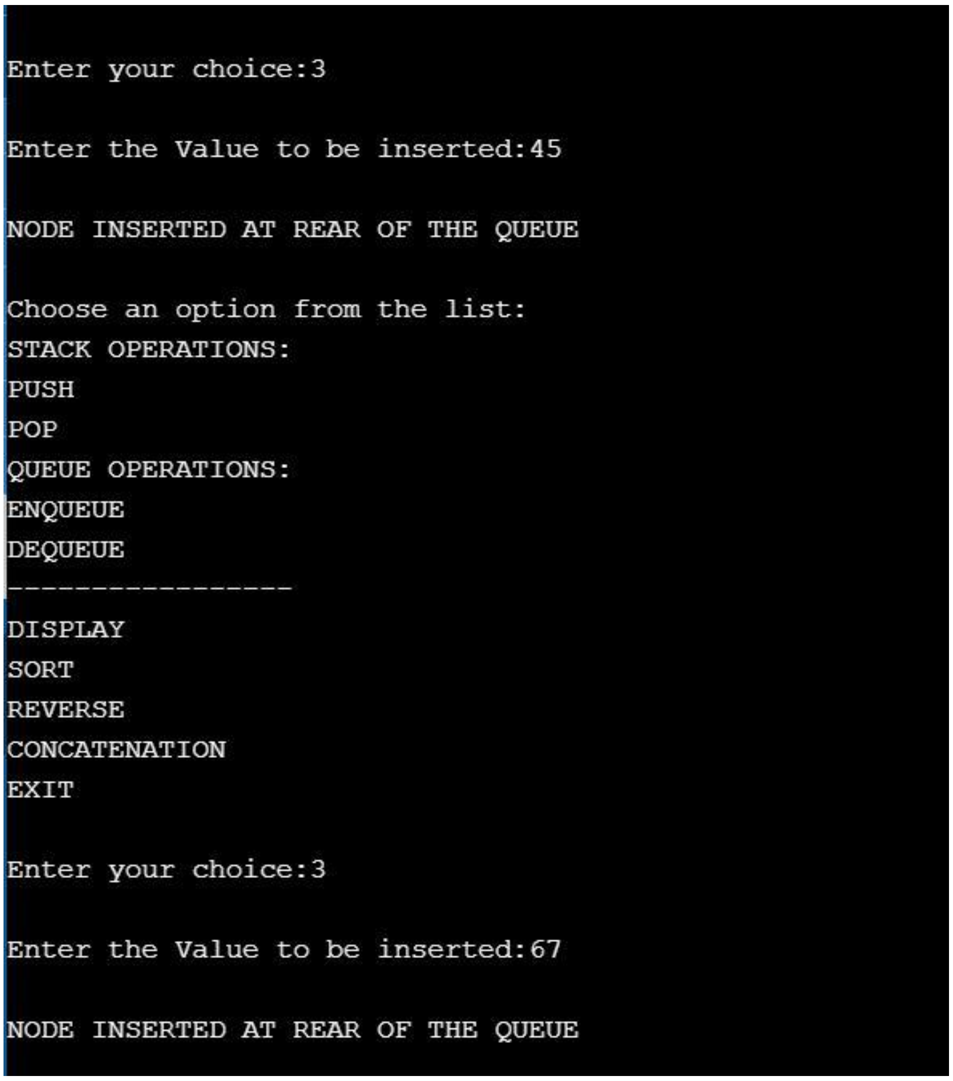
```
Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
-----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT

Enter your choice:1

Enter the Value to be inserted:1

NODE INSERTED AT THE TOP OF THE STACK

Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
-----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT
```



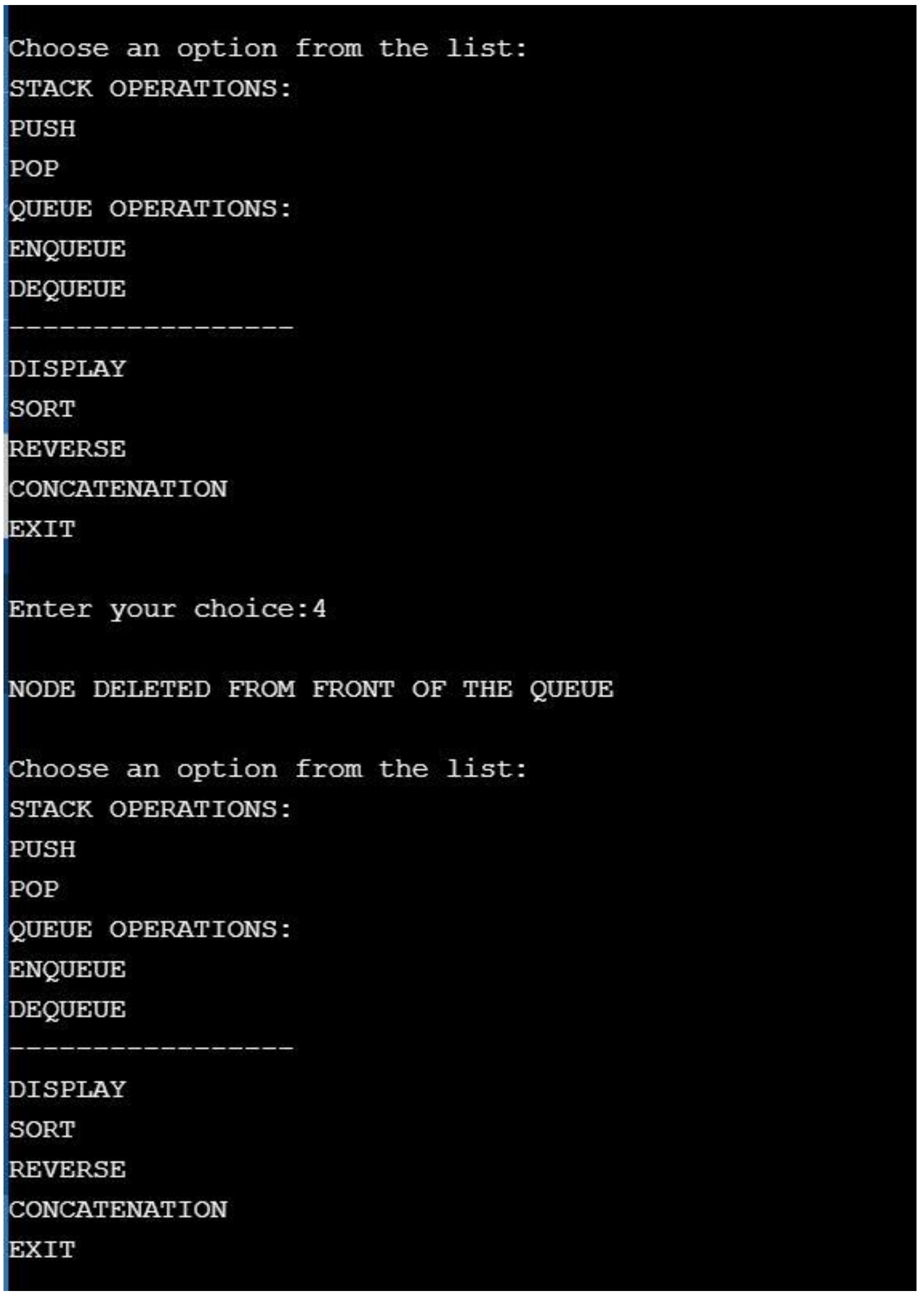
```
Enter your choice:1

Enter the Value to be inserted:2

NODE INSERTED AT THE TOP OF THE STACK

Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT

Enter your choice:1

Enter the Value to be inserted:3

NODE INSERTED AT THE TOP OF THE STACK
```



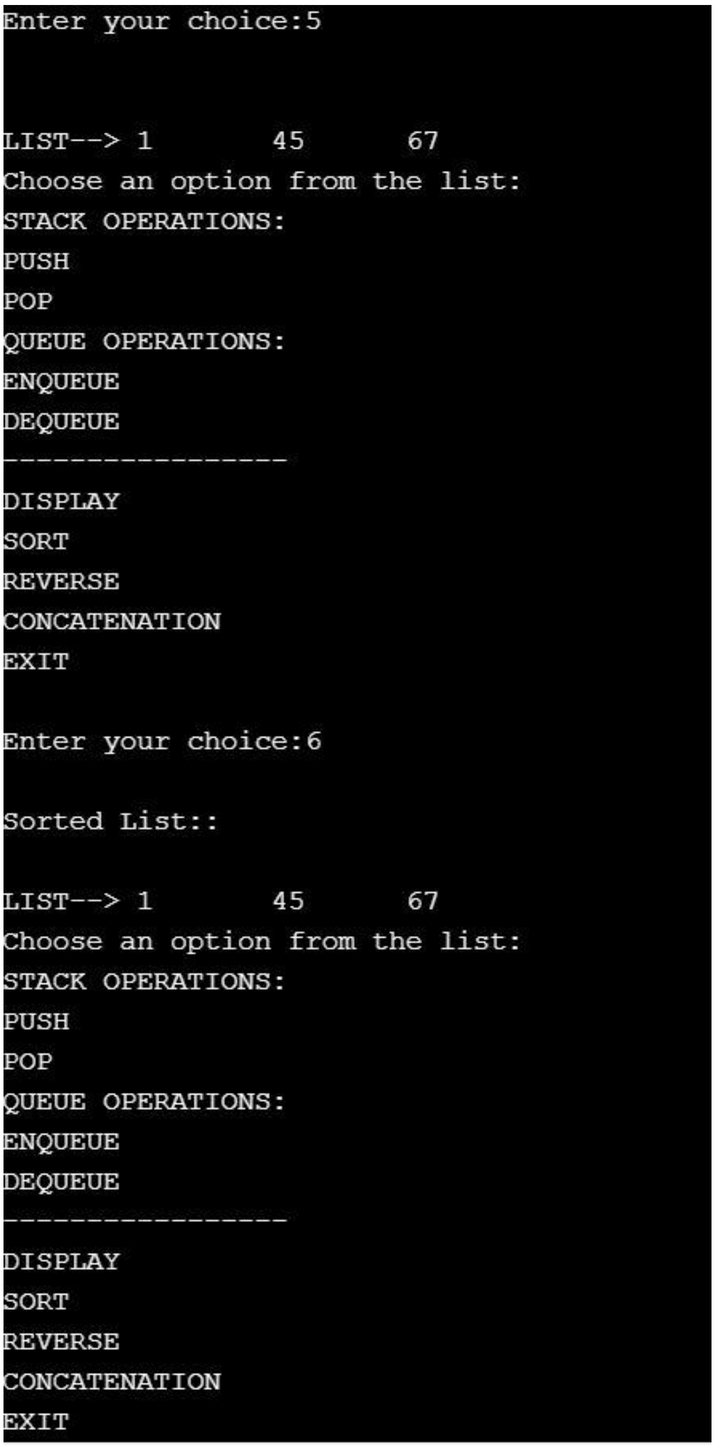
```
Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
-----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT

Enter your choice:2

NODE DELETED FROM TOP OF THE STACK

Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
-----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT
```



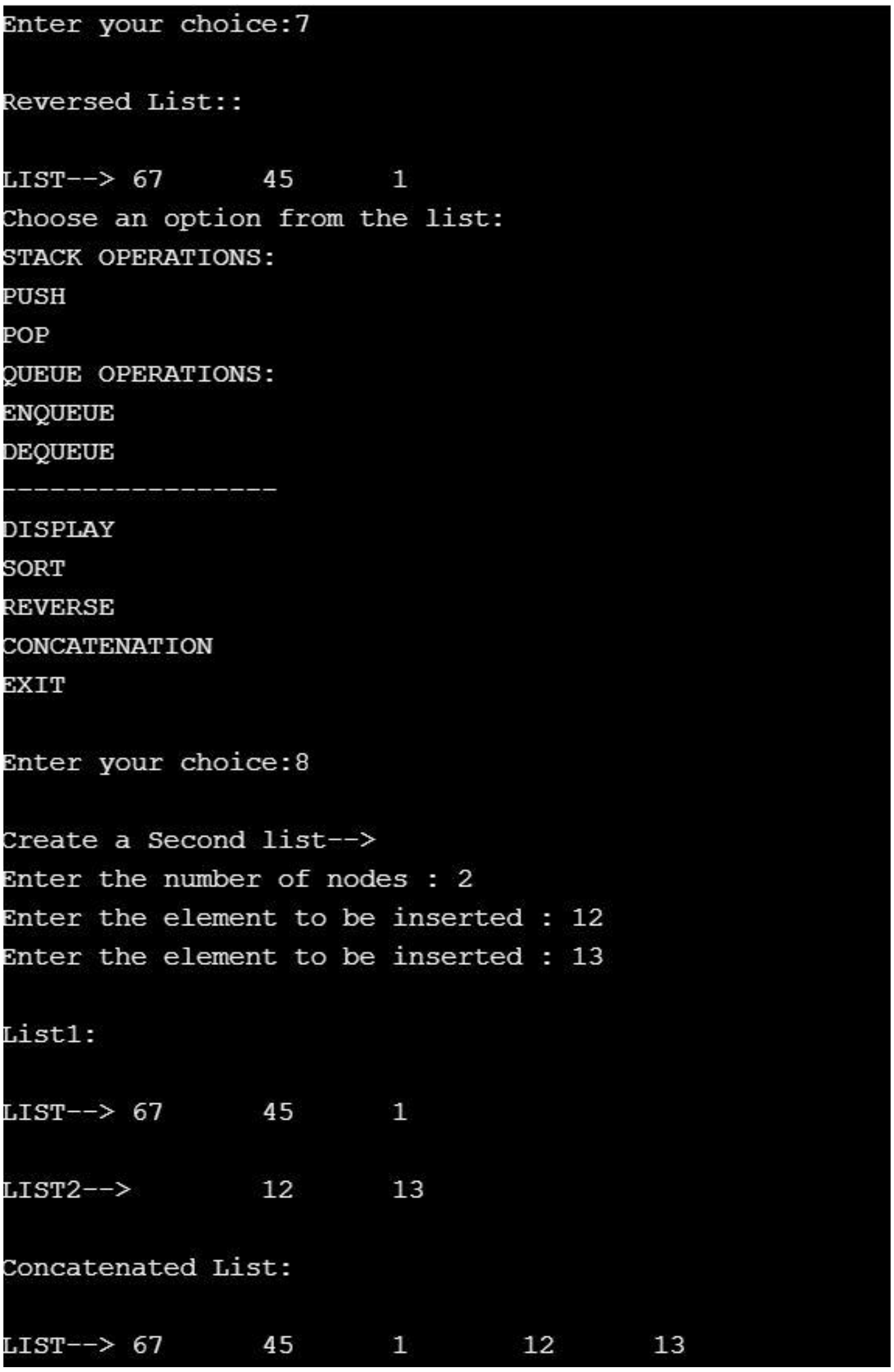
```
Enter your choice:3

Enter the Value to be inserted:45

NODE INSERTED AT REAR OF THE QUEUE

Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
-----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT

Enter your choice:3

Enter the Value to be inserted:67

NODE INSERTED AT REAR OF THE QUEUE
```



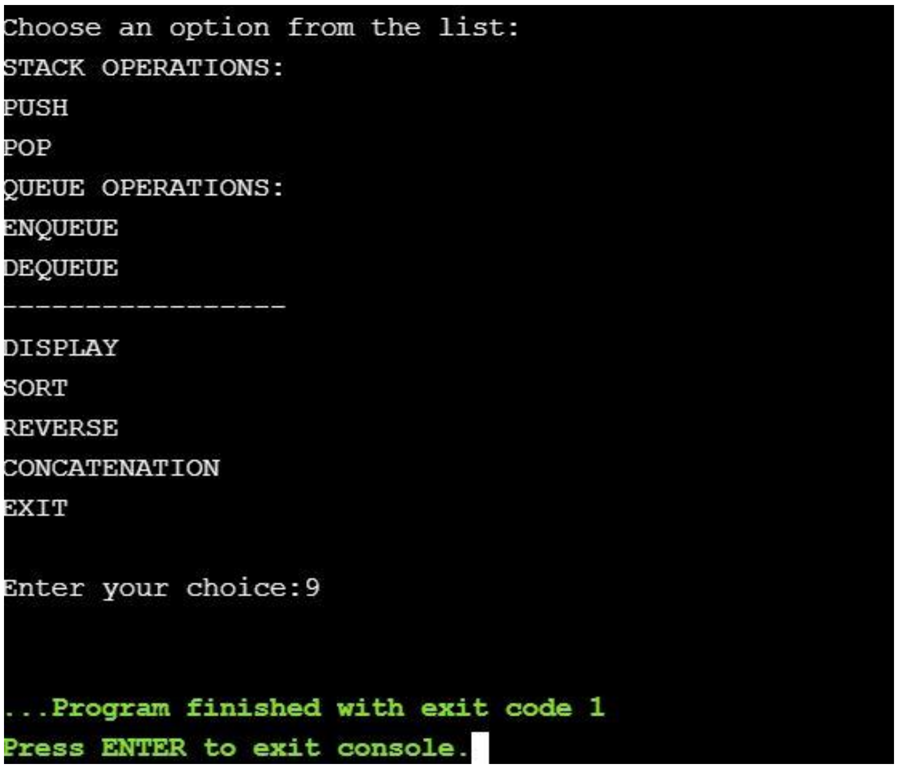
```
Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
-----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT

Enter your choice:4

NODE DELETED FROM FRONT OF THE QUEUE

Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
-----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT
```



```
Enter your choice:5


LIST--> 1       45      67
Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
-----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT

Enter your choice:6

Sorted List::

LIST--> 1       45      67
Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
-----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT
```



```
Enter your choice:7

Reversed List::

LIST--> 67      45      1
Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT

Enter your choice:8

Create a Second list-->
Enter the number of nodes : 2
Enter the element to be inserted : 12
Enter the element to be inserted : 13

List1:

LIST--> 67      45      1

LIST2-->        12      13

Concatenated List:

LIST--> 67      45      1       12      13
```



```
Choose an option from the list:
STACK OPERATIONS:
PUSH
POP
QUEUE OPERATIONS:
ENQUEUE
DEQUEUE
-----------------
DISPLAY
SORT
REVERSE
CONCATENATION
EXIT

Enter your choice:9


...Program finished with exit code 1
Press ENTER to exit console.
```

LAB 9:

WAP Implement doubly link list with primitive operations

a) Create a doubly linked list.

b) Insert a new node to the left of the node.

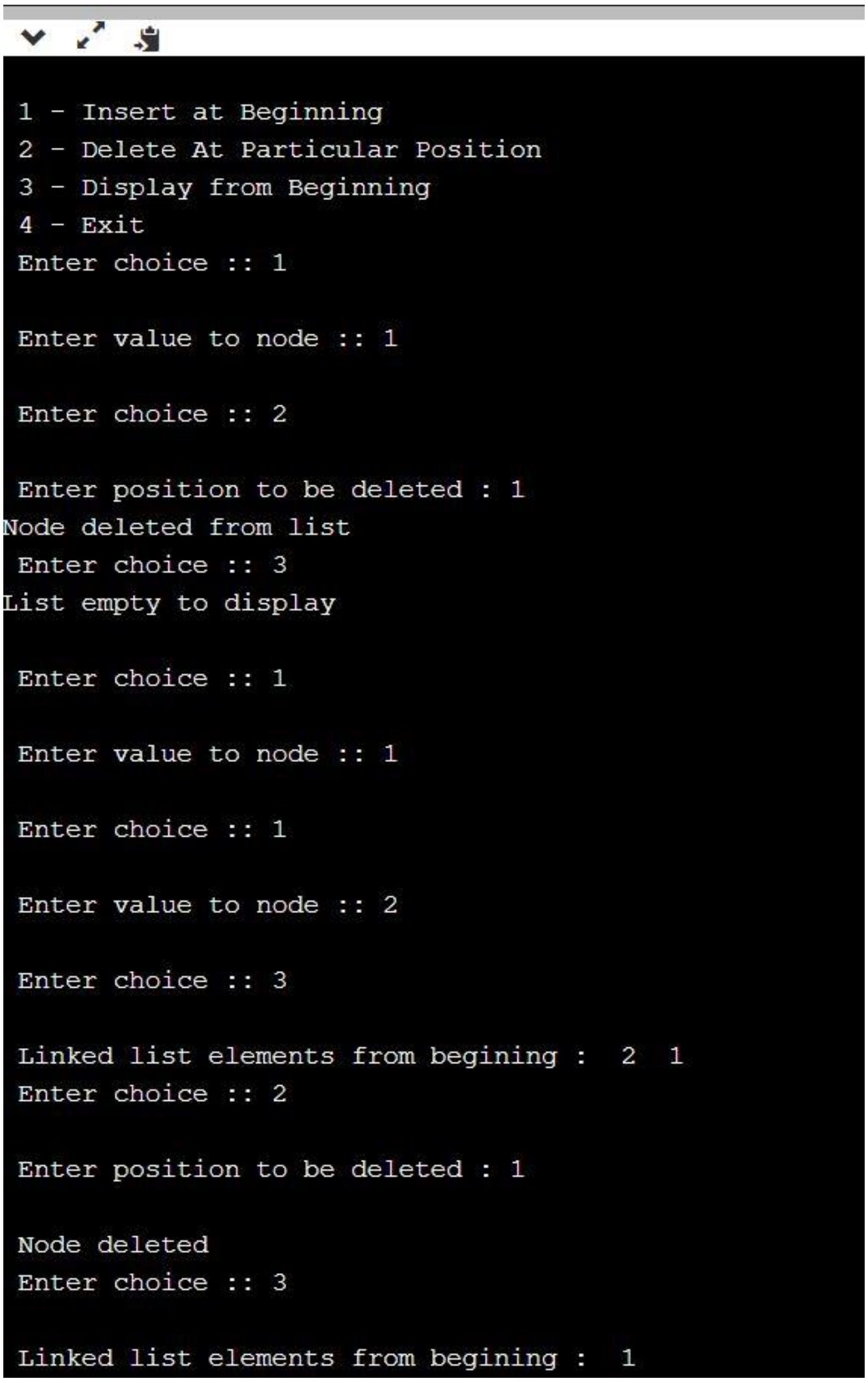
c) Delete the node based on a specific value

d) Display the contents of the list:

```
#include <stdio.h>

#include <stdlib.h>
```

```
struct node
{
    struct node *prev;
    int n;
    struct node *next;
}*h,*temp,*temp1,*temp2,*temp4;

void insert_beg();


void insert_atpos();
void display_beg();
void delete_atpos();
int count = 0;
int main()
{
    int ch;

    h = NULL;
    temp = temp1 = NULL;

    printf("\n 1 - Insert at Beginning");
    printf("\n 2 - Delete At Particular Position");
    printf("\n 3 - Display from Beginning");
    printf("\n 4 - Exit");

    while (1)
    {
        printf("\n Enter choice :: ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            insert_beg();
            break;
        case 2:
            delete_atpos();
            break;
        case 3:
            display_beg();
            break;
        case 4:
            exit(0);
        default:
            printf("\n  Entered the Wrong choice from the menu!!!!!!!!!!!");
        }
    }
}

/* TO create an empty node */
void create()
{
    int data;
    temp =(struct node *)malloc(1*sizeof(struct node));
    temp->prev = NULL;
    temp->next = NULL;
    printf("\n Enter value to node :: ");
    scanf("%d", &data);
    temp->n = data;
    count++;
}

/*  TO insert at beginning */
void insert_beg()
{
    if (h == NULL)
    {
        create();
        h = temp;
        temp1 = h;
    }
    else
    {
        create();
        temp->next = h;
        h->prev = temp;
        h = temp;
    }
```

```
}



/* To delete an element */
void delete_atpos()
{
    int i = 1, pos;

    printf("\n Enter position to be deleted : ");
    scanf("%d", &pos);
    temp2 = h;

    if ((pos < 1) || (pos >= count + 1))
    {
        printf("\n Error : Position out of range to delete!!!!!!");
        return;
    }
    if (h == NULL)
    {
        printf("\n Error : Empty list no elements to delete!!!!!!");
        return;
    }
    else
    {
        while (i < pos)
        {
            temp2 = temp2->next;
            i++;
        }
        if (i == 1)
        {
            if (temp2->next == NULL)
            {
                printf("Node deleted from list");
                free(temp2);
                temp2 = h = NULL;
                return;
            }
        }
        if (temp2->next == NULL)
        {
            temp2->prev->next = NULL;
            free(temp2);
            printf("Node deleted from list");
            return;
        }
        temp2->next->prev = temp2->prev;
        if (i != 1)
            temp2->prev->next = temp2->next;    /* Might not need this statement
if i == 1 check */
        if (i == 1)
            h = temp2->next;
        printf("\n Node deleted");
        free(temp2);
    }
    count--;
}

/* display from beginning */
void display_beg()
{
    temp2 = h;

    if (temp2 == NULL)
    {
        printf("List empty to display \n");
        return;
    }
    printf("\n Linked list elements from begining : ");

    while (temp2->next != NULL)
    {
        printf(" %d ", temp2->n);
        temp2 = temp2->next;
    }
    printf(" %d ", temp2->n);
}
```



```
 1 - Insert at Beginning
 2 - Delete At Particular Position
 3 - Display from Beginning
 4 - Exit
 Enter choice :: 1

 Enter value to node :: 1

 Enter choice :: 2

 Enter position to be deleted : 1
Node deleted from list
 Enter choice :: 3
List empty to display

 Enter choice :: 1

 Enter value to node :: 1

 Enter choice :: 1

 Enter value to node :: 2

 Enter choice :: 3

 Linked list elements from begining :  2  1
 Enter choice :: 2

 Enter position to be deleted : 1

 Node deleted
 Enter choice :: 3

 Linked list elements from begining :  1
```

LAB 10:

Binary Search Tree:

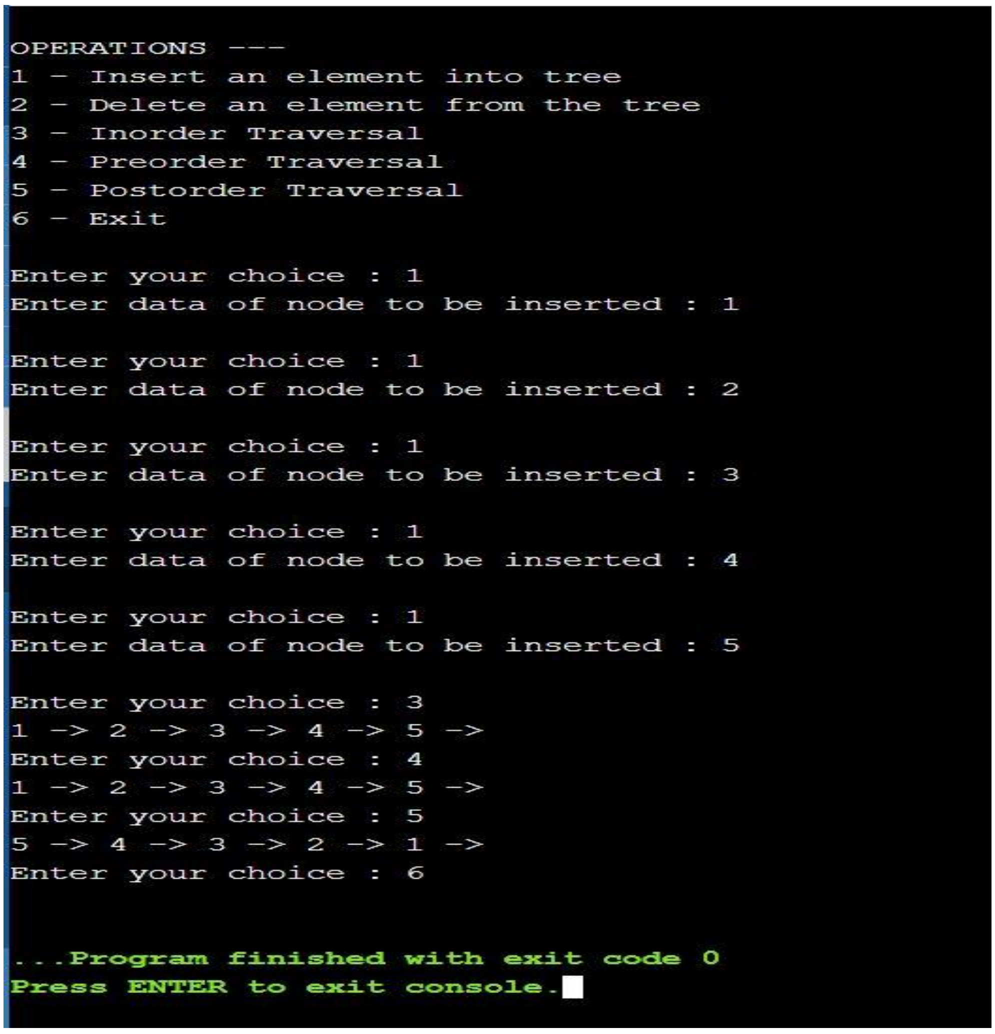
```
#include <stdio.h>

#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;
      }
    }
}

void insert()
{
    create();
    if (root == NULL)
        root = temp;
    else
        search(root);
}
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;
}
```

```
void search(struct btnode *t)
{
   if ((temp->value > t->value) && (t->r != NULL))
      search(t->r);
   else if ((temp->value > t->value) && (t->r == NULL))
      t->r = temp;
   else if ((temp->value < t->value) && (t->l != NULL))
      search(t->l);
   else if ((temp->value < t->value) && (t->l == NULL))
      t->l = temp;
}


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);
}
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);
}

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);
}

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);
    }
}

void delete1(struct btnode *t)
{
    int k;

    if ((t->l == NULL) && (t->r == NULL))
    {
        if (t1->l == t)
        {
            t1->l = NULL;
        }
        else
        {
            t1->r = NULL;
        }
        t = NULL;
        free(t);
        return;
    }



    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;
    }

    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;
    }

    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;
    }

}
int smallest(struct btnode *t)
{
    t2 = t;
    if (t->l != NULL)
    {
        t2 = t;
        return(smallest(t->l));
    }
    else
```

```
        return (t->value);
}

int largest(struct btnode *t)
{
    if (t->r != NULL)
    {
        t2 = t;
        return(largest(t->r));
    }
    else
        return(t->value);
}
```



```
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 : 1

Enter your choice : 1
Enter data of node to be inserted : 2

Enter your choice : 1
Enter data of node to be inserted : 3

Enter your choice : 1
Enter data of node to be inserted : 4

Enter your choice : 1
Enter data of node to be inserted : 5

Enter your choice : 3
1 -> 2 -> 3 -> 4 -> 5 ->
Enter your choice : 4
1 -> 2 -> 3 -> 4 -> 5 ->
Enter your choice : 5
5 -> 4 -> 3 -> 2 -> 1 ->
Enter your choice : 6


...Program finished with exit code 0
Press ENTER to exit console.
```