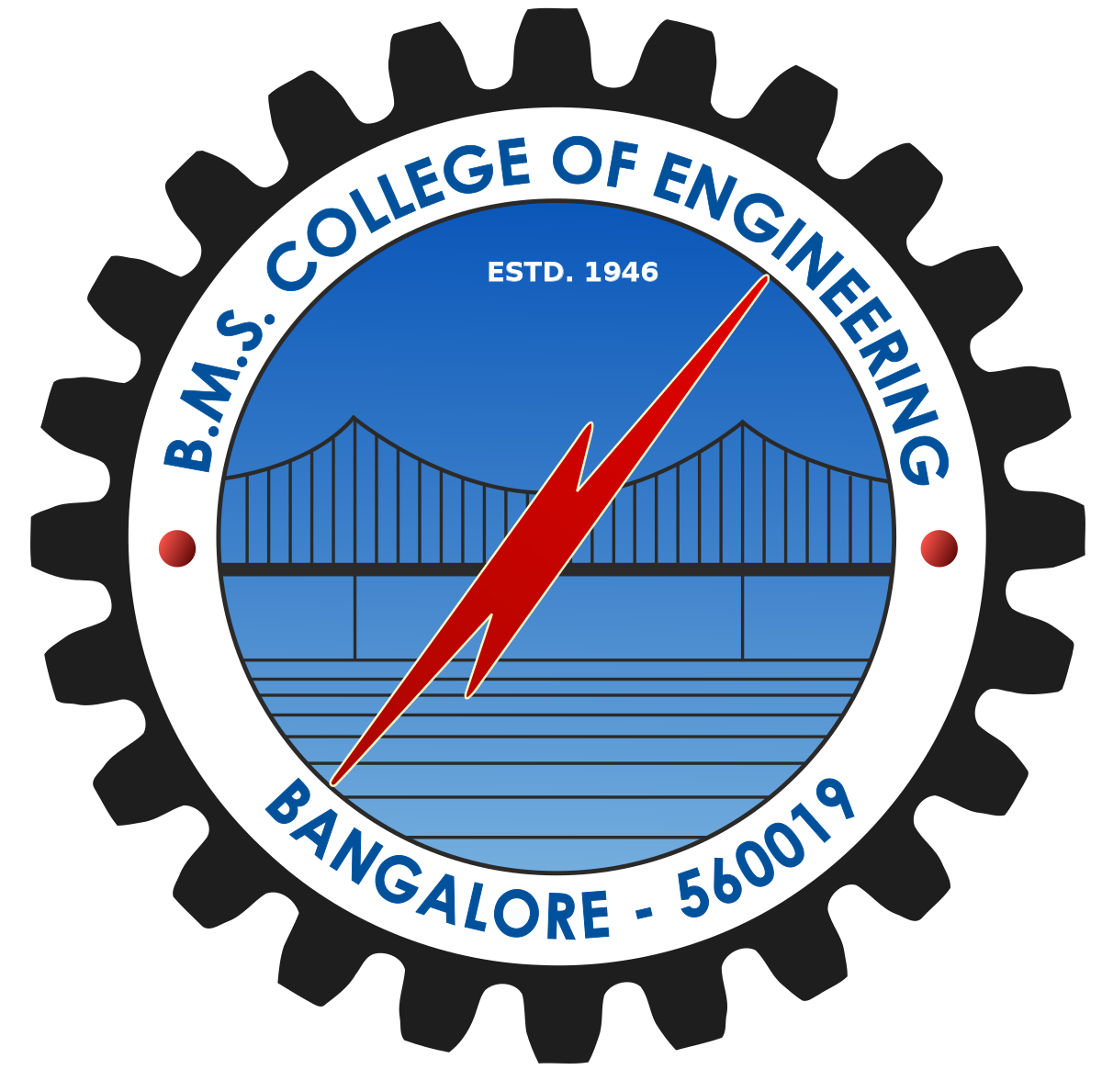
**B.M.S. COLLEGE OF ENGINEERING**

**(AUTONOMOUS COLLEGE UNDER VTU)**

**BENGALURU-19**

****

**DATA STRUCTURE**

**LAB REPORT**

**ACADEMIC YEAR : SEPT-DEC 2020**

**NAME : VAISHNAVI JAGDALE**

**USN : 1BM19CS215**

**SEM : 3**

**PROGRAM-1**

**Write a program 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 the 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 student

{

int id;

int age;

int marks;

};

int main()

{

struct student s[20];

int i,n;

printf("Please enter number of students \n");

scanf("%d",&n);

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

{

printf("Enter student ID: \n");

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

printf("Enter student AGE: \n");

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

printf("Enter student MARKS: \n");

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

}

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

{

if((s[i].age>20)&&((s[i].marks>=0)&&(s[i].marks<=100)))

{

printf("Student has qualified for the exam \n");

printf("ID-- %d \n",s[i].id);

printf("AGE-- %d \n",s[i].age);

printf("MARKS-- %d \n",s[i].marks);

}

else

{

printf("Invalid data! \n");

}

if(s[i].marks>=65)

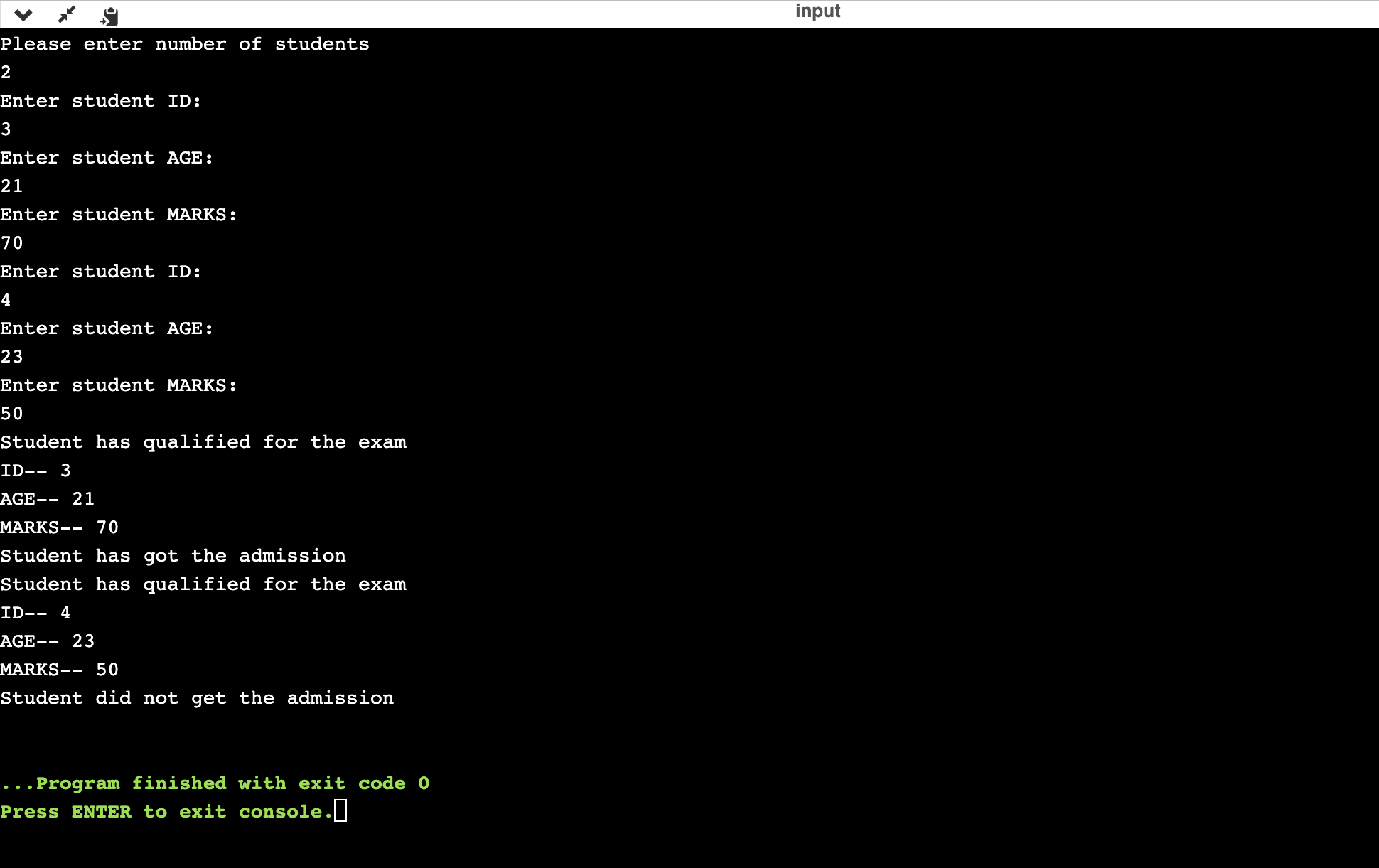
printf("Student has got the admission \n");

else

printf("Student did not get the admission \n");

}

***OUTPUT:***



**PROGRAM-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**

#include<stdio.h>

#include<conio.h>

#define SIZE 10

void push();

void pop();

void display();

int p[SIZE];

top=-1;

void push(int x)

{

if(top==SIZE-1)

printf("\n Stack is FULL! \n");

else

{

top++;

p[top]=x;

printf("\n Insertion is successful \n");

}

}

void pop()

{

if(top==-1)

printf("\n Stack is EMPTY! \n");

else

{

printf("\n Deleted : %d", p[top]);

top--;

}

}

void display()

{

if(top==-1)

printf("\n Stack is empty! \n");

else

{

int i;

printf("\n Stack elements are: \n");

for(i=top;i>=0;i--)

printf("%d \n", p[i]);

}

}

void main()

{

int x,choice;

clrscr;

while(1)

{

printf("\n\n \*\*MENU\*\* \n\n");

printf("1.PUSH\n");

printf("2.POP\n");

printf("3.DISPLAY\n");

printf("4.EXIT!\n");

printf("\nEnter your choice: \n");

scanf("%d",&choice);

switch(choice)

{

case 1: printf("Enter values to be inserted: \n");

scanf("%d",&x);

push(x);

break;

case 2: pop();

break;

case 3: display();

break;

case 4: exit(0);

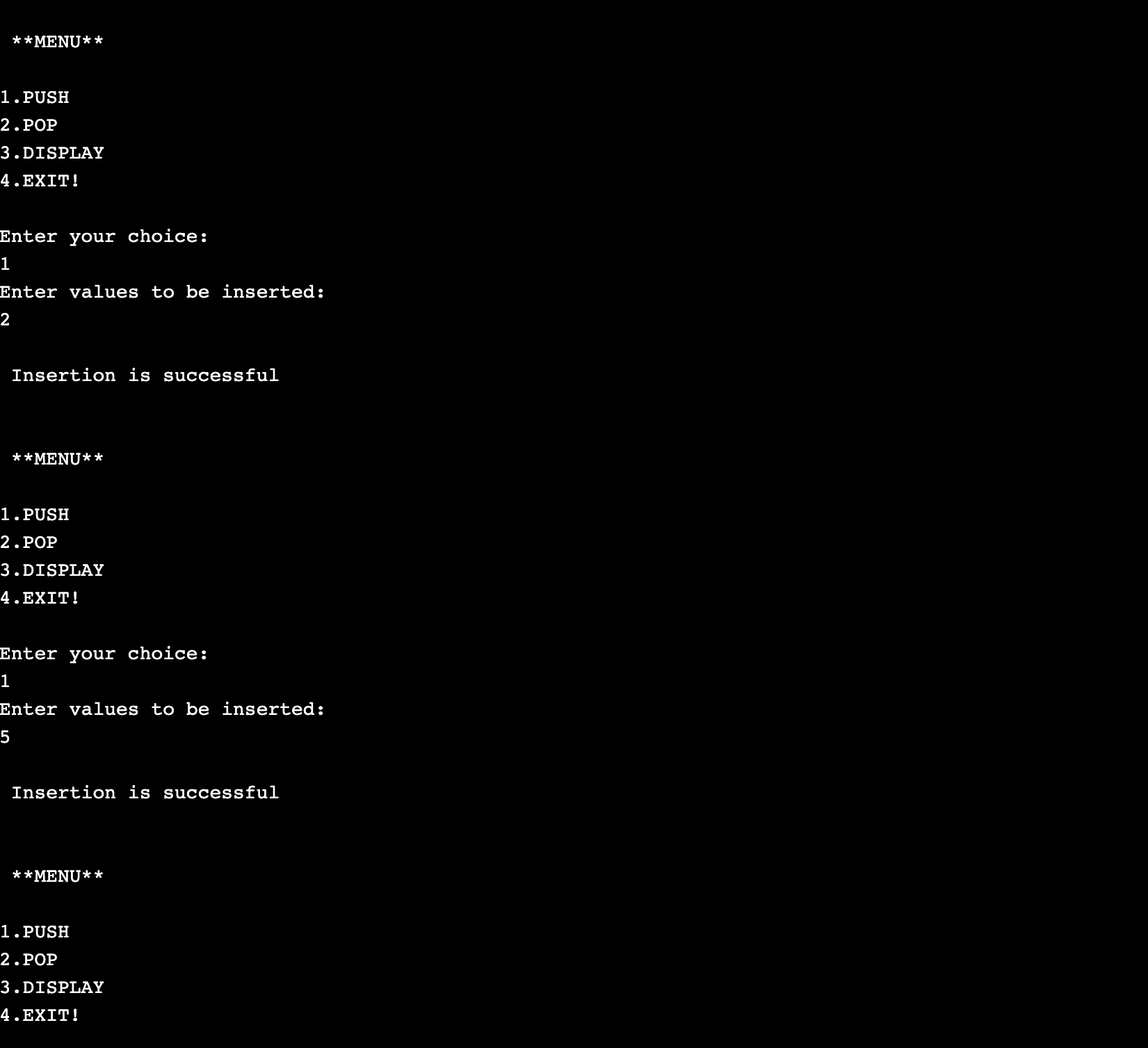
default: printf("\n WRONG SELECTION! TRY AGAIN. \n ");

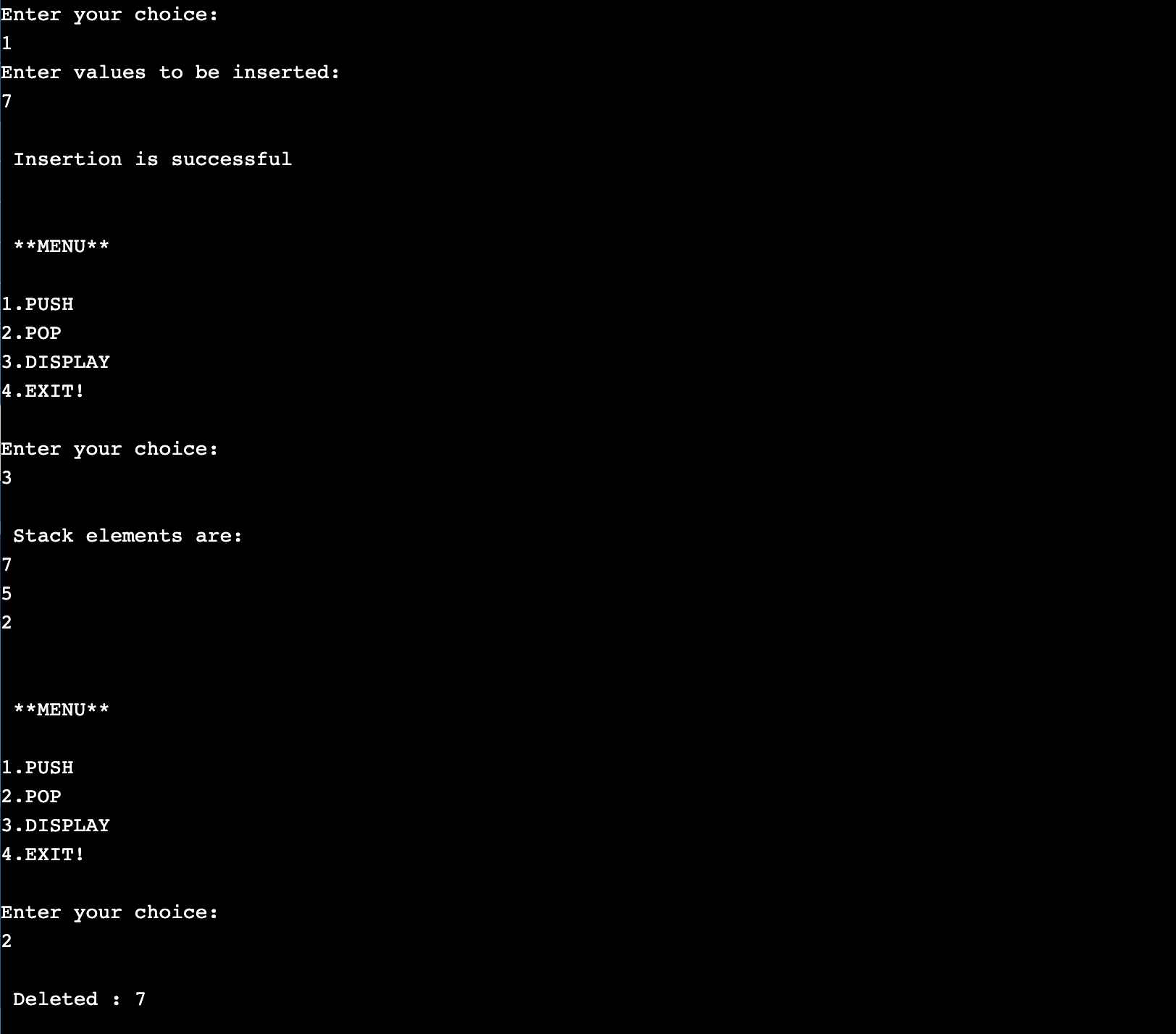
}

}

}

***OUTPUT:***





**PROGRAM-3**

**Write a program 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>

#define SIZE 50

char s[SIZE];

int top=-1;

push(char elem)

{

s[++top]=elem;

}

char pop()

{

return(s[top--]);

}

int pr(char symbol){

if(symbol =='^'){

return(3);

}

else if (symbol=='\*'|| symbol=='/'){

return(2);

}

else if(symbol=='+'|| symbol=='-'){

return(1);

}

else{

return(0);

}

}

void main()

{

char infix[50],postfix[50],ch,elem;

int i=0,k=0;

printf("Enter the INFIX expression \n");

scanf("%s",infix);

push('#');

while((ch=infix[i++]) != '\0')

{

if(ch=='(') push(ch);

else

if(isalnum(ch)) postfix[k++]=ch;

else

if (ch==')')

{

while(s[top] != '(')

postfix[k++]=pop();

elem=pop();

}

else

{

while(pr(s[top])>=pr(ch))

postfix[k++]=pop();

push(ch);

}

}

while (s[top]!='#')

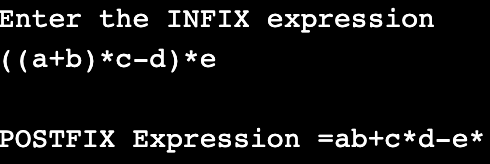
postfix[k++]=pop();

postfix[k]='\0';

printf("\nPOSTFIX Expression =%s\n",postfix);

}

***OUTPUT:***

******

**PROGRAM-4**

**Write a program 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>

#define MAX 50

void insert\_arr();

void delete\_arr();

void display();

int queue\_a[MAX];

int rear=-1;

int front=-1;

int main()

{

int choice;

while(1)

{

printf("1.INSERT \n");

printf("2.DELETE \n");

printf("3.DISPLAY \n");

printf("4.EXIT \n");

printf("\n\tEnter your choice: \n ");

scanf("%d",&choice);

switch(choice)

{

case 1: insert\_arr();

break;

case 2: delete\_arr();

break;

case 3: display();

break;

case 4: exit(1);

default: printf("Invalid Choice! \n");

}

}

}

void display()

{

int i;

if (front == - 1)

printf("Queue is empty \n");

else

{

printf("Queue is : \n");

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

printf("%d ", queue\_a[i]);

printf("\n");

}

}

void insert\_arr()

{

int add\_item;

if (rear == MAX - 1)

printf("Queue Overflow. \n");

else

{

if (front == - 1)

/\*If queue is initially empty \*/

front = 0;

printf("Inset the element in queue : ");

scanf("%d", &add\_item);

rear = rear + 1;

queue\_a[rear] = add\_item;

}

}

void delete\_arr()

{

if (front==-1 || front>rear)

{

printf("Queue Underflow \n");

return;

}

else

{

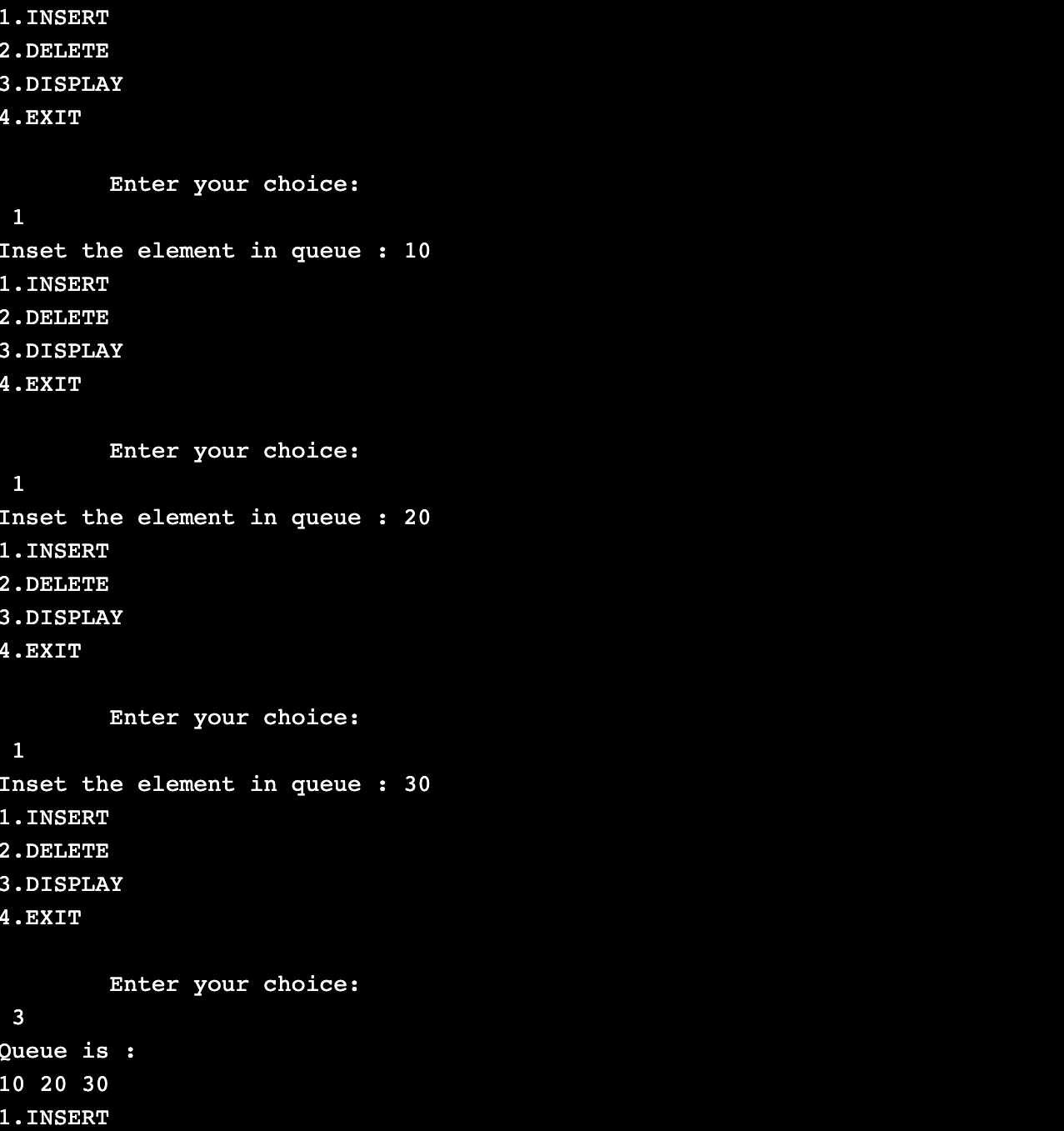
printf("Deleted Element is: %d \n ", queue\_a[front]);

front=front+1;

}

}

***OUTPUT:***

******

**PROGRAM-5&6**

**Write a program 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 <stdlib.h>

#include <stdio.h>

void create();

void display();

void insert\_begin();

void insert\_end();

void insert\_pos();

void delete\_begin();

void delete\_end();

void delete\_pos();

struct node

{

int data;

struct node \*next;

};

struct node \*next=NULL;

int main()

{

int choice;

while(1){

printf("\n ~\*\*\*\* MENU \*\*\*\*~ \n");

printf("\n 1.Create");

printf("\n 2.Display");

printf("\n 3.Insert at the beginning");

printf("\n 4.Insert at the end");

printf("\n 5.Insert at specified position");

printf("\n 6.Delete from beginning");

printf("\n 7.Delete from the end");

printf("\n 8.Delete from specified position");

printf("\n 9.Exit");

printf("\n--------------------------------------\n");

printf("\nEnter your choice:");

scanf("%d",&choice);

switch(choice)

{

case 1: create();

break;

case 2: display();

break;

case 3: insert\_begin();

break;

case 4: insert\_end();

break;

case 5: insert\_pos();

break;

case 6: delete\_begin();

break;

case 7: delete\_end();

break;

case 8: delete\_pos();

break;

case 9: exit(0);

break;

default: printf("\n Wrong Choice:n");

break;

}

}

return 0;

}

void create()

{

struct node \*temp,\*ptr;

temp=(struct node \*)malloc(sizeof(struct node));

if(temp==NULL)

{

printf("\n Overflow!");

exit(0);

}

printf("\nEnter the data value for the node:");

scanf("%d",&temp->data);

temp->next=NULL;

if(next==NULL)

{

next=temp;

}

else

{

ptr=next;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=temp;

}

}

void display()

{

struct node \*ptr;

if(next==NULL)

{

printf("\nList is empty");

return;

}

else

{

ptr=next;

printf("\nThe List elements are:");

while(ptr!=NULL)

{

printf("%d\t",ptr->data );

ptr=ptr->next ;

}

}

}

void insert\_begin()

{

struct node \*temp;

temp=(struct node \*)malloc(sizeof(struct node));

if(temp==NULL)

{

printf("\nOverflow!");

return;

}

printf("\nEnter the data value for the node:" );

scanf("%d",&temp->data);

temp->next =NULL;

if(next==NULL)

{

next=temp;

}

else

{

temp->next=next;

next=temp;

}

}

void insert\_end()

{

struct node \*temp,\*ptr;

temp=(struct node \*)malloc(sizeof(struct node));

if(temp==NULL)

{

printf("\nOverflow!");

return;

}

printf("\nEnter the data value for the node:" );

scanf("%d",&temp->data );

temp->next =NULL;

if(next==NULL)

{

next=temp;

}

else

{

ptr=next;

while(ptr->next !=NULL)

{

ptr=ptr->next ;

}

ptr->next =temp;

}

}

void insert\_pos()

{

struct node \*ptr,\*temp;

int i,pos;

temp=(struct node \*)malloc(sizeof(struct node));

if(temp==NULL)

{

printf("\nOverflow!");

return;

}

printf("\nEnter the position for the new node to be inserted:");

scanf("%d",&pos);

printf("\nEnter the data value of the node:");

scanf("%d",&temp->data) ;

temp->next=NULL;

if(pos==0)

{

temp->next=next;

next=temp;

}

else

{

for(i=0,ptr=next;i<pos-1;i++) { ptr=ptr->next;

if(ptr==NULL)

{

printf("\nPosition not found.");

return;

}

}

temp->next =ptr->next ;

ptr->next=temp;

}

}

void delete\_begin()

{

struct node \*ptr;

if(ptr==NULL)

{

printf("\nList is Empty:n");

return;

}

else

{

ptr=next;

next=next->next ;

printf("\nThe deleted element is :%d",ptr->data);

free(ptr);

}

}

void delete\_end()

{

struct node \*temp,\*ptr;

if(next==NULL)

{

printf("\nList is Empty:");

exit(0);

}

else if(next->next ==NULL)

{

ptr=next;

next=NULL;

printf("\nThe Deleted element is:%d",ptr->data);

free(ptr);

}

else

{

ptr=next;

while(ptr->next!=NULL)

{

temp=ptr;

ptr=ptr->next;

}

temp->next=NULL;

printf("\nThe Deleted element is:%d",ptr->data);

free(ptr);

}

}

void delete\_pos()

{

int i,pos;

struct node \*temp,\*ptr;

if(next==NULL)

{

printf("\nThe List is Empty:");

exit(0);

}

else

{

printf("\nEnter the position of the node to be deleted:");

scanf("%d",&pos);

if(pos==0)

{

ptr=next;

next=next->next ;

printf("\nThe Deleted element is:%d",ptr->data );

free(ptr);

}

else

{

ptr=next;

for(i=0;i<pos;i++) { temp=ptr; ptr=ptr->next ;

if(ptr==NULL)

{

printf("\nPosition not Found:");

return;

}

}

temp->next =ptr->next ;

printf("\nThe deleted element is:%d",ptr->data );

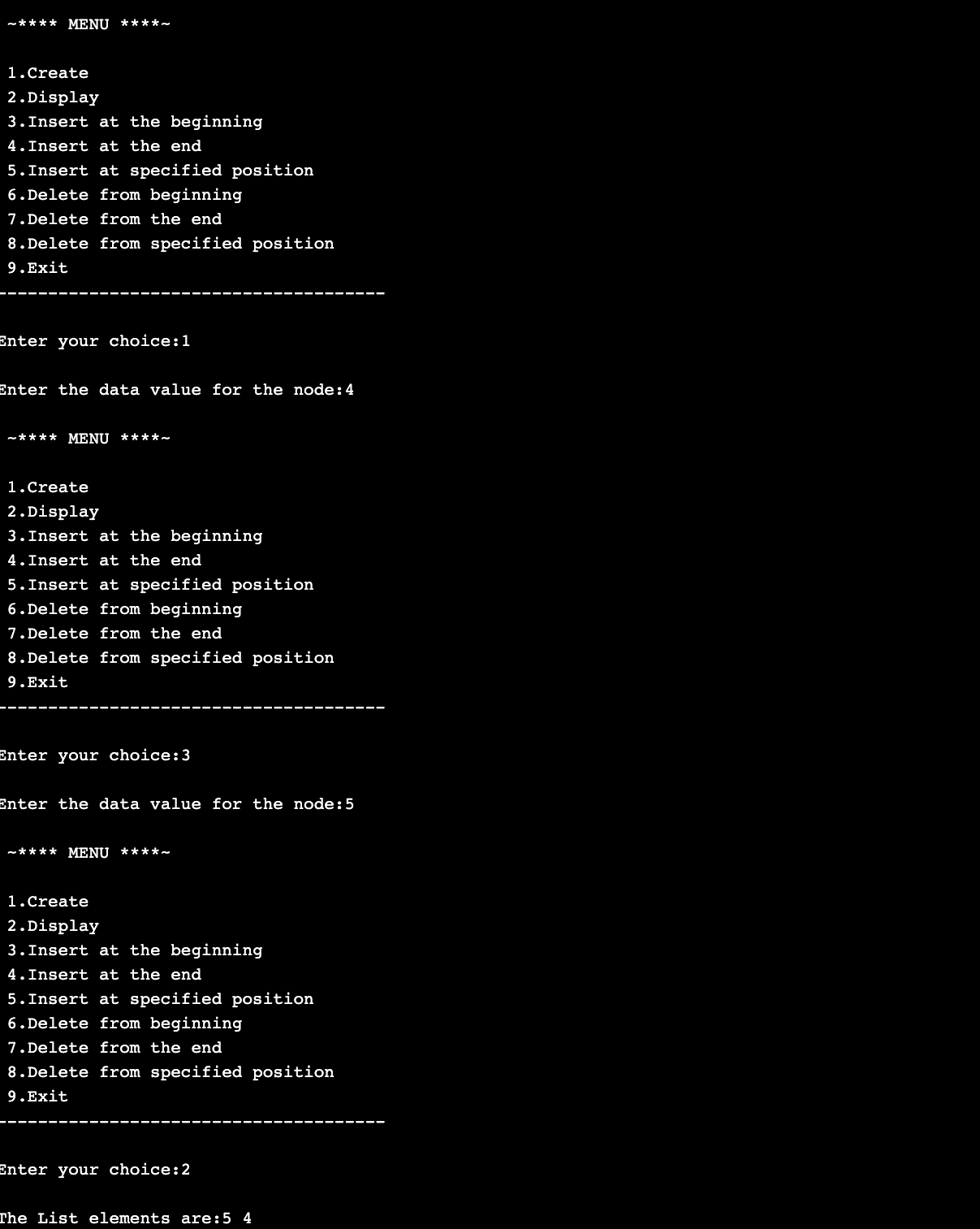
free(ptr);

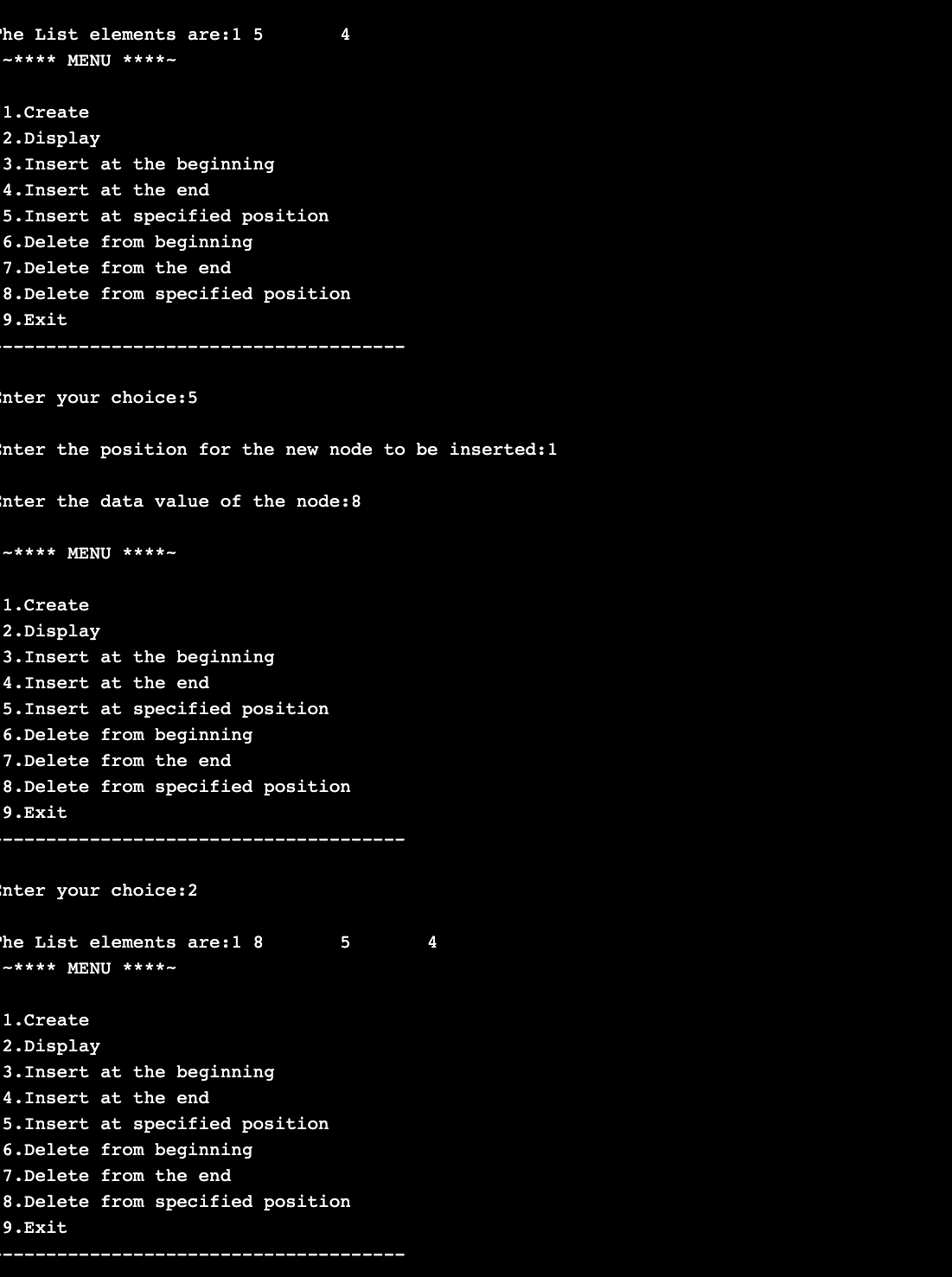
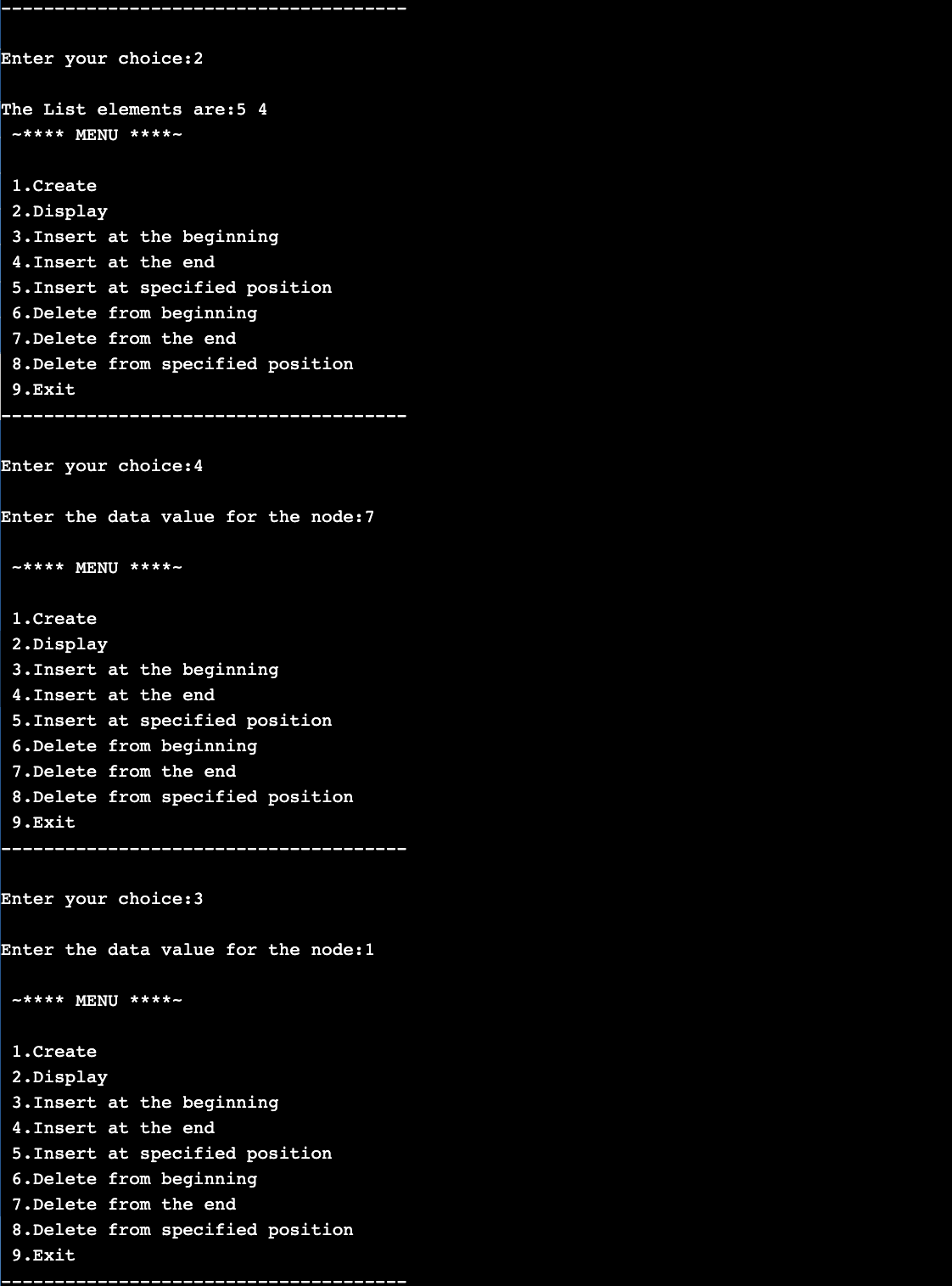
}

}

}

***OUTPUT:***

******

******

**PROGRAM-7**

**Write a program to implement Single Link List with following operations**

**a) Sort the linked list.**

**b) Reverse the linked list.**

**c) Concatenation of two linked lists**

#include <stdlib.h>

#include <string.h>

struct node

{

int sem;

struct node \*next;

};

struct node \*head= NULL;

struct node \*head2= NULL;

int c=0;

void Insert()

{

struct node \*newnode;

struct node \*temp;

int s;

printf("Enter integer : ");

scanf("%d",&s);

newnode=(struct node\*)malloc(sizeof(struct node));

newnode->sem =s;

if (head==NULL)

{

newnode->next=NULL;

head=newnode;

printf("First node created.\n");

c++;

}

else

{

temp=head;

while(temp->next!=NULL)

{

temp=temp->next;

}

temp->next=newnode;

newnode->next=NULL;

c++;

printf("Node created.\n");

}

}

void Insert2()

{

struct node \*newnode;

struct node \*temp;

int s,y;

printf("Enter elements to create list. \n");

do

{

printf("Enter integer : \n");

scanf("%d",&s);

newnode=(struct node\*)malloc(sizeof(struct node));

newnode->sem =s;

if (head2==NULL)

{

newnode->next=NULL;

head2=newnode;

printf("First node created.\n");

c++;

}

else

{

temp=head2;

while(temp->next!=NULL)

{

temp=temp->next;

}

temp->next=newnode;

newnode->next=NULL;

c++;

printf("Node created\n");

}

printf("Add more? Click:0 or 1\n");

scanf("%d",&y);

}while(y!=0);

}

void bubSort()

{

int swapped, i;

struct node \*ptr1;

struct node \*lptr = NULL;

if (head == NULL)

return;

do

{

swapped = 0;

ptr1 = head;

while (ptr1->next != lptr)

{

if (ptr1->sem > ptr1->next->sem)

{

int temp = ptr1->sem;

ptr1->sem = ptr1->next->sem;

ptr1->next->sem = temp;

swapped = 1;

}

ptr1 = ptr1->next;

}

lptr = ptr1;

}

while (swapped);

}

void reverse()

{

struct node\* prev = NULL;

struct node\* current = head;

struct node\* next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

head= prev;

}

void concat()

{

struct node \*ptr;

if(head==NULL)

{

head=head2;

}

if(head2==NULL)

{

head2=head;

}

ptr=head;

while(ptr->next!=NULL)

ptr=ptr->next;

ptr->next=head2;

}

void display1()

{

struct node \*ptr;

ptr=head;

int i=1;

if(ptr==NULL)

{

printf("Linked list is empty!\n");

}

else

{

while(ptr!= NULL)

{

printf(" %d",ptr->sem);

i++;

ptr=ptr->next;

}

}

}

void display2()

{

struct node \*ptr;

ptr=head2;

int i=1;

if(ptr==NULL)

{

printf("Linked list is empty!\n");

}

else

{

while(ptr!= NULL)

{

printf(" %d",ptr->sem);

printf("\n");

i++;

ptr=ptr->next;

}

}

}

int main()

{

int choice,pos;

do

{

printf("\n1. Insert node \n2.Sort node \n3.Reverse node \n4.Concatenate two lists \n5.exit \n");

printf("\nEnter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

Insert();

break;

case 2:

bubSort();

display1();

break;

case 3:

reverse();

display1();

break;

case 4:

Insert2();

concat();

display1();

break;

case 5:

break;

default:

printf("Wrong choice!\n");

break;

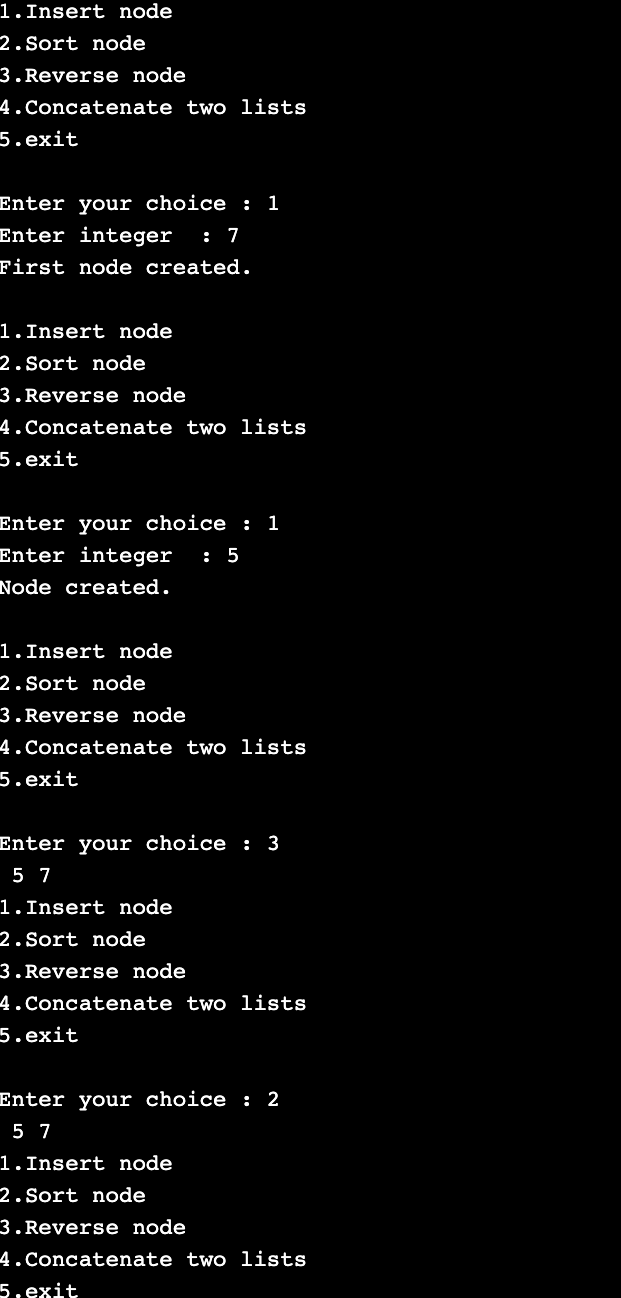
}

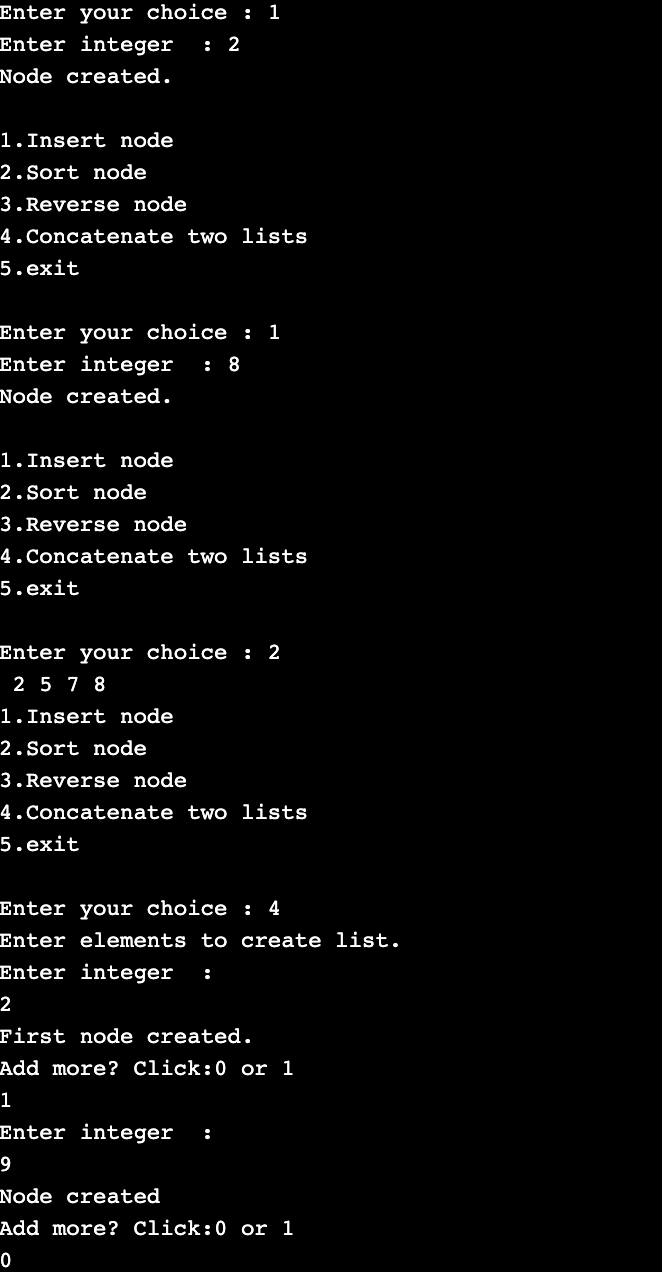
}while(choice!=5);

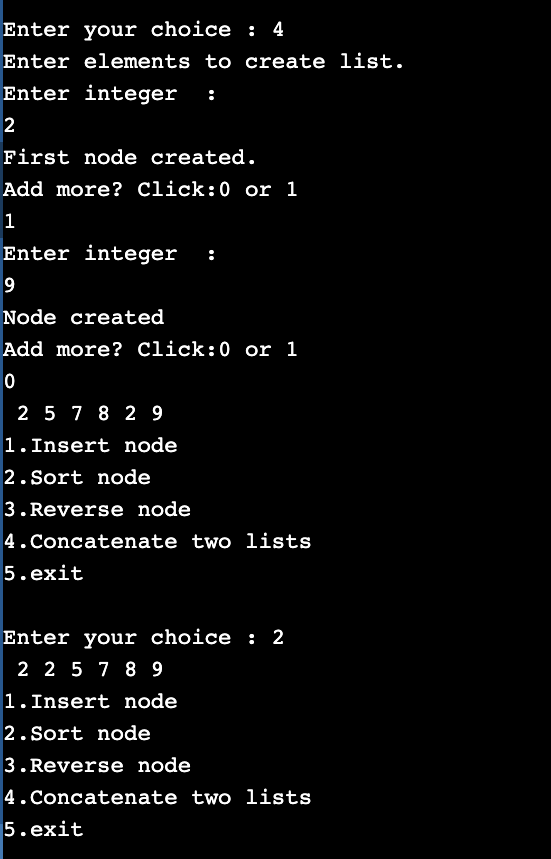
return 0;

}

***OUTPUT:***

******

******

******

**PROGRAM-8**

**Write a program to implement Stack & Queue using Linked Representation.**

Queue using linked list

#include <stdio.h>

#include <stdlib.h>

typedef struct node{

int data;

struct node \*link;

}node;

node \*root=NULL;

void enqueue()

{ //insert at end (rear)

node \*temp;

temp=(node \*)malloc(sizeof(node));

printf("Enter the node element\n");

scanf("%d",&temp->data);

temp->link=NULL;

if(root==NULL)

{

root=temp;

}

else

{

node \*p=root;

while(p->link!=NULL)

{

p=p->link;

}

p->link=temp;

}

}

void dequeue()

{

node \*temp;

if(root==NULL) //delete from front

{

printf("Queue is empty\n");

}

else

{

temp=root;

root=temp->link;

temp->link=NULL;

free(temp);

}

}

void display()

{

node \*temp=root;

if(temp==NULL)

{

printf("Queue is empty\n");

}

else

{

while(temp!=NULL)

{

printf("%d\n",temp->data);

temp=temp->link;

}

}

}

int main()

{

printf("Enter the operation\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");

int op,len;

while(1)

{ printf("Enter the operation: ");

scanf("%d",&op);

switch (op)

{

case 1:enqueue();

break;

case 2: dequeue();

break;

case 3: display();

break;

case 4: exit(0);

break;

default: printf("No such operation\n");

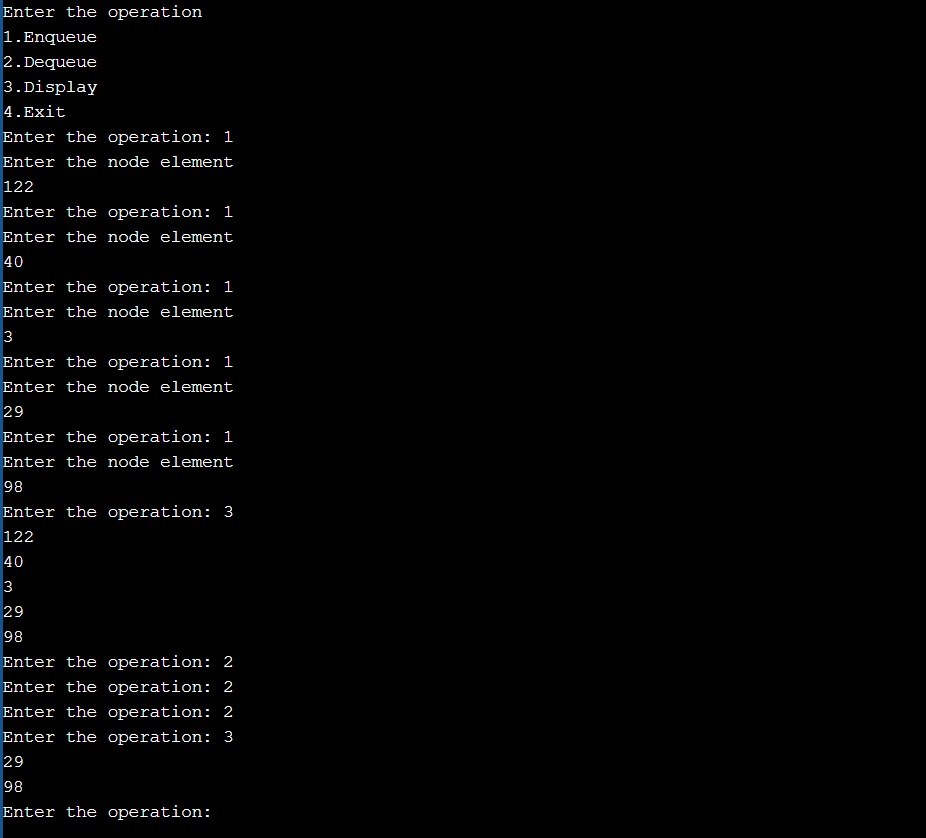
}

}

return 0;

}

***OUTPUT:***



Stack using linked list

#include <stdio.h>

#include <stdlib.h>

typedef struct node{

int data;

struct node \*link;

}node;

node \*top=NULL;

void push()

{

node \*temp;

temp=(node \*)malloc(sizeof(node));

printf("Enter node element\n");

scanf("%d",&temp->data);

temp->link=NULL;

if(top==NULL)

{

top=temp;

}

else

{

temp->link=top;

top=temp;

}

}

void pop()

{

node \*temp;

if(top==NULL)

{

printf("Stack is empty\n");

}

else

{

temp=top;

top=temp->link;

temp->link=NULL;

free(temp);

}

}

void display()

{

node \*temp=top;

if(temp==NULL)

{

printf("Stack is empty\n");

}

else

{

while(temp!=NULL)

{

printf("%d\n",temp->data);

temp=temp->link;

}

}

}

int main()

{

printf("Enter the operation\n1.Push\n");

printf("2.Pop\n3.Display\n4.Exit\n");

int op,len;

while(1)

{ printf("choose your option : ");

scanf("%d",&op);

switch (op)

{

case 1:push();

break;

case 2: pop();

break;

case 3: display();

break;

case 4: exit(0);

break;

default: printf("No such operation\n");

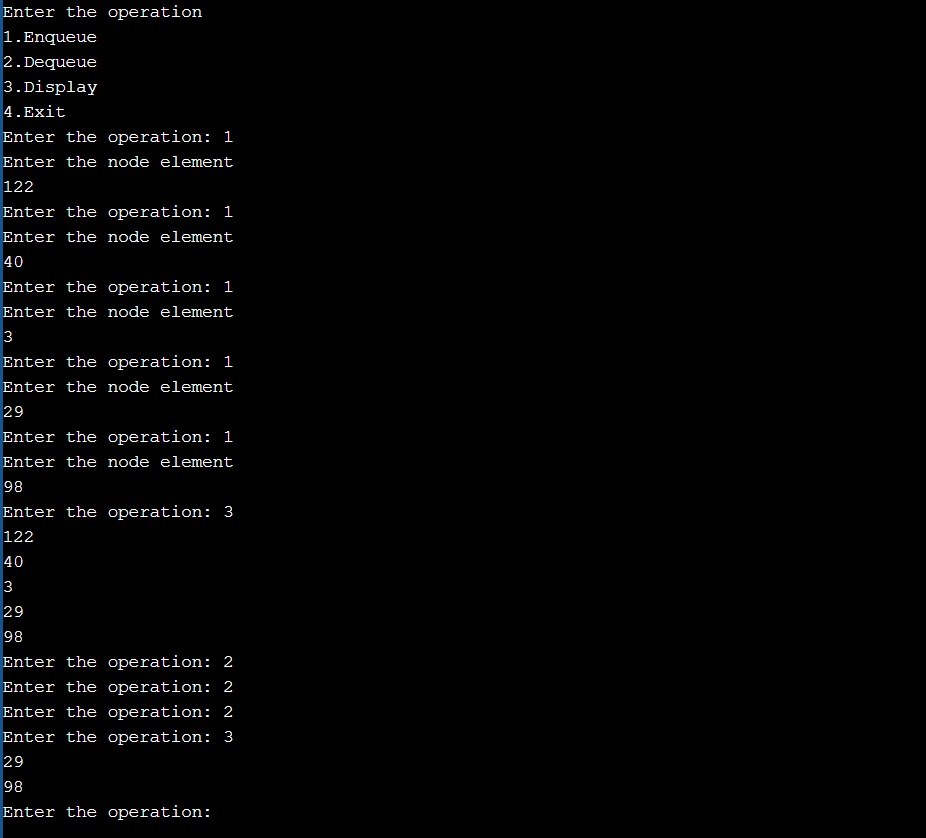
}

}

return 0;

}

***OUTPUT:***



**PROGRAM-9**

**Write a program to 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 data;

struct node \*next;

};

struct node \*head=NULL;

void add\_b( )

{

struct node \*ptr = NULL;

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

printf("Enter the node data :");

scanf("%d",& ptr->data);

ptr->prev=NULL;

ptr->next=NULL;

if(head==NULL){

head=ptr;

}

else{

ptr->next=head;

head->prev=ptr;

head=ptr;

}

}

void delete\_pos( )

{

struct node \*ptr, \*temp;

int val;

printf("\nEnter the position\n");

scanf("%d",&val);

temp = head;

while(temp -> data != val)

temp = temp -> next;

if(temp -> next == NULL)

{

printf("\nCan't delete\n");

}

else if(temp -> next -> next == NULL)

{

temp ->next = NULL;

printf("\nNode Deleted\n");

} else

{

ptr = temp -> next;

temp -> next = ptr -> next;

ptr -> next -> prev = temp;

free(ptr);

printf("\nNode Deleted\n");

}

}

void display( ){

if(head==NULL){

printf("List is empty\n");

}

else{

struct node \*temp=head;

while(temp!=NULL){

printf("%d\t",temp->data);

temp=temp->next;

}

printf("\n");

}

}

int main(int argc, const char \* argv[]) {

int opt=0;

while(1)

{

printf("DOUBLY LINKED LIST\n");

printf("1.Insert at the beginning\n");

printf("2.Delete at a position\n");

printf("3.Display\n");

printf("Enter choice :");

scanf("%d",& opt);

switch(opt){

case 1:

add\_b();

break;

case 2:

delete\_pos();

break;

case 3:

display();

break;

default:

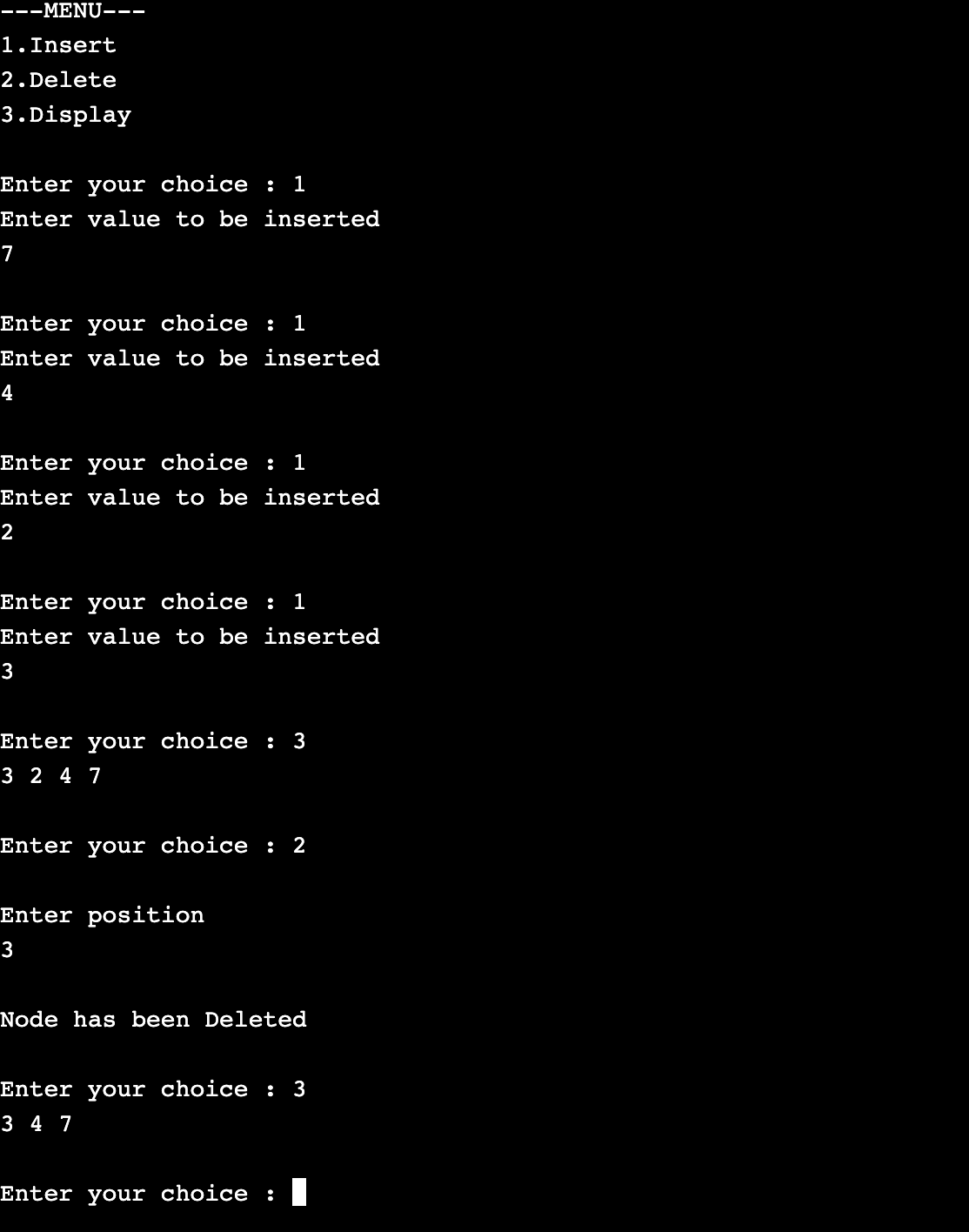
printf("Wrong Choice!\n");

}

}

return 0;

}

***OUTPUT:***

**PROGRAM-10**

**Write a program to**

1. **To construct a binary search tree.**
2. **To transverse the tree using all the methods ie, in-order, pre-order, post-order**
3. **Display contents of the list.**

#include <stdio.h>

#include <stdlib.h>

struct btnode

{

int value;

struct btnode \*l;

struct btnode \*r;

}

\*root = NULL, \*temp = NULL, \*t2, \*t1;

void insert();

void inorder(struct btnode \*t);

void create();

void search(struct btnode \*t);

void preorder(struct btnode \*t);

void postorder(struct btnode \*t);

int flag = 1;

void main()

{

int ch;

printf("\n-\*\*OPERATIONS\*\*-");

printf("\n1-Insert an element into tree\n");

printf("2-Inorder Traversal\n");

printf("3-Preorder Traversal\n");

printf("4-Postorder Traversal\n");

printf("5-Exit\n");

while(1)

{

printf("\nEnter your choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

insert();

break;

case 2:

inorder(root);

break;

case 3:

preorder(root);

break;

case 4:

postorder(root);

break;

case 5:

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)) /\* value more than root node value insert at right \*/

search(t->r);

else if ((temp->value > t->value) && (t->r == NULL))

t->r = temp;

else if ((temp->value < t->value) && (t->l != NULL)) /\* value less than root node value insert at left \*/

search(t->l);

else if ((temp->value < t->value) && (t->l == NULL))

t->l = temp;

}

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 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);

}

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);

}

***OUTPUT:***

