PROGRAM 5

Objective: Program to implement Linked list using dynamic memory allocation

Linked list is a data structure with the following specifics::

1.Data is dynamically added or removed.

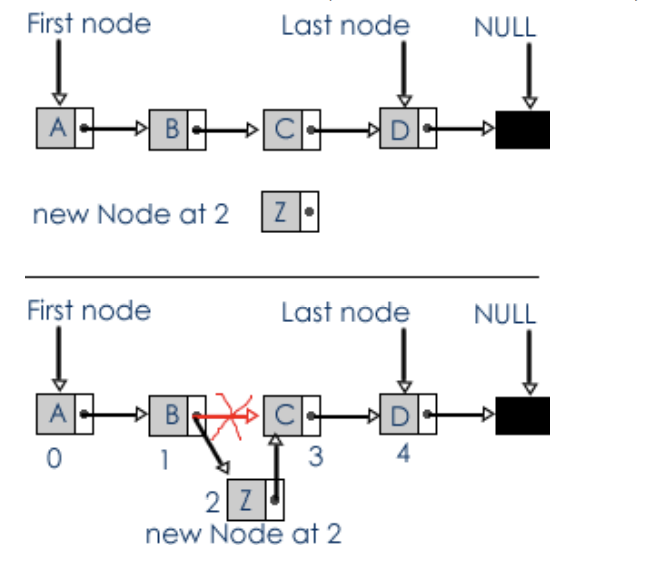
2. Every data object has two parts-a data part and a link part. Together they constitute a node.

3. The list can be traversed only through pointers.

4. Every node is an important constituent of the data.

5. The end of the data is always a leaf end.

6. Tree structures (branched link lists) are used to store data into disks.



# Insertion of a new NODE at beginning –

1.)Create a new Node

NODE \*ptr;

ptr = malloc(sizeof(NODE));

2.) Field Initialization

ptr->INFO = value;

ptr->NEXT = NULL;

3.) Actual Insertion (Link Establishment)

ptr->NEXT = head;

head = ptr;

# Insertion of a new NODE at end -

1.)Memory Allocation – create a new NODE

NODE \*ptr;

ptr = malloc(sizeof(NODE));

2.) Field Initialization

ptr->INFO = value;

ptr->NEXT = NULL;

3.)Searching for Last Node

NODE \*loc = head;

while(loc->NEXT != NULL)

loc = loc->NEXT;

4.)Actual Insertion(Link Establishment)

loc->NEXT = ptr;

# Insertion after a given NODE -

1.)Memory Allocation

NODE ptr;

ptr = malloc(sizeof(NODE));

2.)Field Initialization

ptr->INFO = value;

ptr->VALUE = NULL;

3.)Searching

NODE \*loc = head;

while(loc->INFO != after)

loc = loc->NEXT;

4.)Insertion

ptr->NEXT = loc->NEXT;

loc->NEXT = ptr;

# Delete first NODE –

int y;

NODE \*temp;

temp = head;

head = head->NEXT;

y = temp->INFO;

free(temp);

return y;

# Delete any given NODE

NODE \*temp,\*prev,\*loc;

if(\*head == NULL)

printf("\nLinked List is empty.");

return -1;

else if((\*head)->NEXT != NULL && (\*head)->INFO == item)

temp = \*head;

\*head = (\*head)->NEXT;

free(temp);

else if( (\*head)->NEXT == NULL && (\*head)->INFO == item)

temp = \*head;

\*head = NULL;

free(temp);

else

loc = \*head;

while(loc->INFO != item) {

prev = loc;

loc = loc->NEXT;

}

temp = loc;

prev->NEXT = loc->NEXT;

free(temp);

# PROGRAM CODE –

#include <stdio.h>

#include <stdlib.h>

typedef struct link\_list {

int INFO;

struct link\_list \*NEXT;

}NODE;

void insert\_at\_beg(NODE \*\*head,int value) {

NODE \*ptr;

ptr = malloc(sizeof(NODE));

ptr->INFO = value;

ptr->NEXT = NULL;

if(\*head == NULL) {

\*head = ptr;

}

else {

ptr->NEXT = \*head;

\*head = ptr;

}

}

void insert\_at\_end(NODE \*\*head,int value) {

NODE \*ptr1,\*ptr2;

ptr1 = malloc(sizeof(NODE));

ptr2 = \*head;

ptr1->INFO = value;

ptr1->NEXT = NULL;

if(ptr2 == NULL) {

\*head = ptr1;

return;

}

while(ptr2->NEXT != NULL) {

ptr2 = ptr2->NEXT;

}

ptr2->NEXT = ptr1;

}

void insert\_after(NODE \*\*head,int after,int value) {

NODE \*ptr,\*loc;

ptr = (NODE \*)malloc(sizeof(NODE));

ptr->INFO = value;

ptr->NEXT = NULL;

if(\*head == NULL) {

\*head = ptr;

}

else {

loc = \*head;

while(loc->INFO != after)

loc = loc->NEXT;

ptr->NEXT = loc->NEXT;

loc->NEXT = ptr;

}

}

int del\_first(NODE \*\*head) {

NODE \*ptr,\*temp;

int y;

if(\*head == NULL) {

return -1;

}

y = (\*head)->INFO;

temp = \*head;

\*head = (\*head)->NEXT;

free(temp);

return y;

}

int delete\_any(NODE \*\*head,int item) {

NODE \*temp,\*loc,\*prev;

if(\*head == NULL) {

printf("\nLinked List is empty.");

return -1;

}

else if((\*head)->NEXT != NULL && (\*head)->INFO == item) {

temp = \*head;

\*head = (\*head)->NEXT;

free(temp);

}

else if( (\*head)->NEXT == NULL && (\*head)->INFO == item){

temp = \*head;

\*head = NULL;

free(temp);

}

else {

loc = \*head;

while(loc->INFO != item) {

prev = loc;

loc = loc->NEXT;

}

temp = loc;

prev->NEXT = loc->NEXT;

free(temp);

}

}

void display(NODE \*\*head) {

NODE \*ptr;

ptr = \*head;

if(ptr == NULL) {

printf("\nLINKED LIST is empty.");

return;

}

printf("\nLINKED LIST is : ");

while(ptr != NULL) {

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

ptr = ptr->NEXT;

}

}

void main() {

NODE \*head;

head = NULL;

int ch,value,del,after,x;

while(1) {

printf("\n\n1.Insert a node at front in LINKED LIST.");

printf("\n2.Insert a node at end of LINKED LIST.");

printf("\n3.Insert after a given node in LINKED LIST.");

printf("\n4.Delete first NODE in LINKED LIST.");

printf("\n5.Delete any given NODE in LINKED LIST.");

printf("\n6.Display LINKED LIST.");

printf("\n7.Exit.");

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

scanf("%d",&ch);

switch(ch) {

case 1 : printf("\nenter value to add at begining : ");

scanf("%d",&value);

insert\_at\_beg(&head,value);

break;

case 2 : printf("\nenter value to add at begining : ");

scanf("%d",&value);

insert\_at\_end(&head,value);

break;

case 3 : printf("\nenter node to enter after : ");

scanf("%d",&after);

printf("\nenter value to enter : ");

scanf("%d",&value);

insert\_after(&head,after,value);

break;

case 4 : del = del\_first(&head);

if(del != -1)

printf("Deleted Item is is : %d",del);

break;

case 5 : printf("\nEnter NODE to delete : ");

scanf("%d",&x);

del = delete\_any(&head,x);

if(del != -1)

printf("\ndelete item is : %d",del);

break;

case 6 : display(&head);

break;

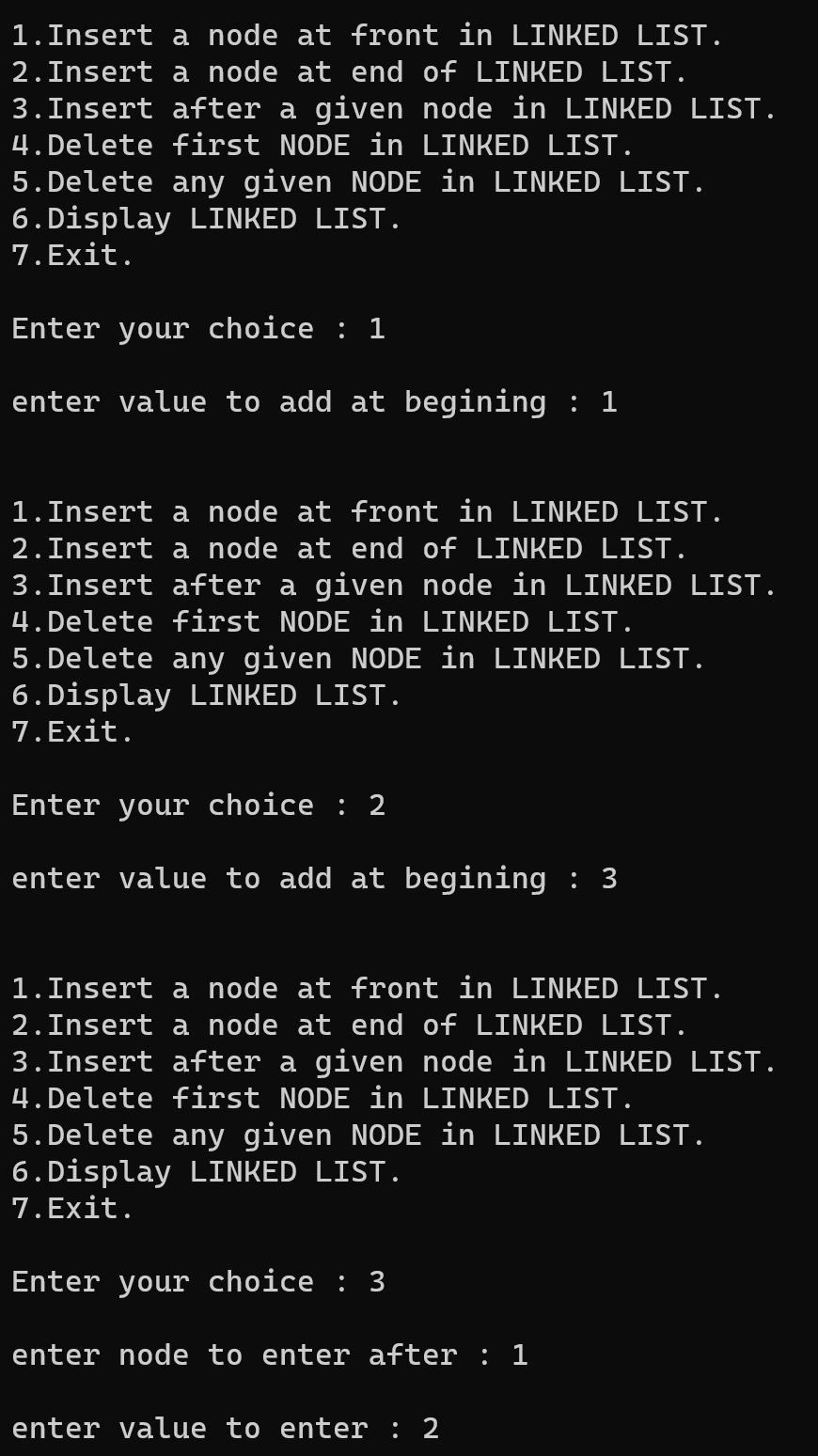
case 7 : exit(0);

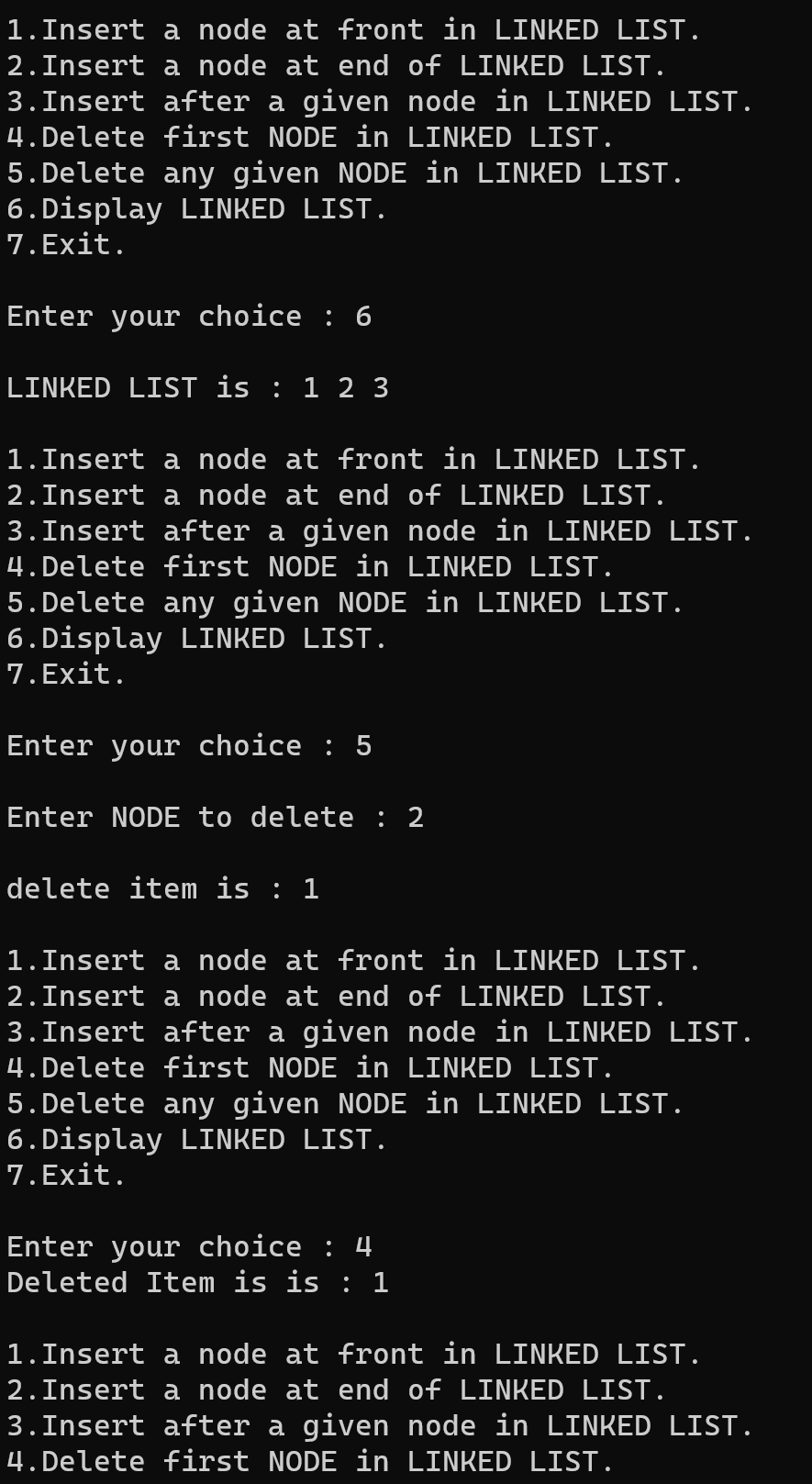
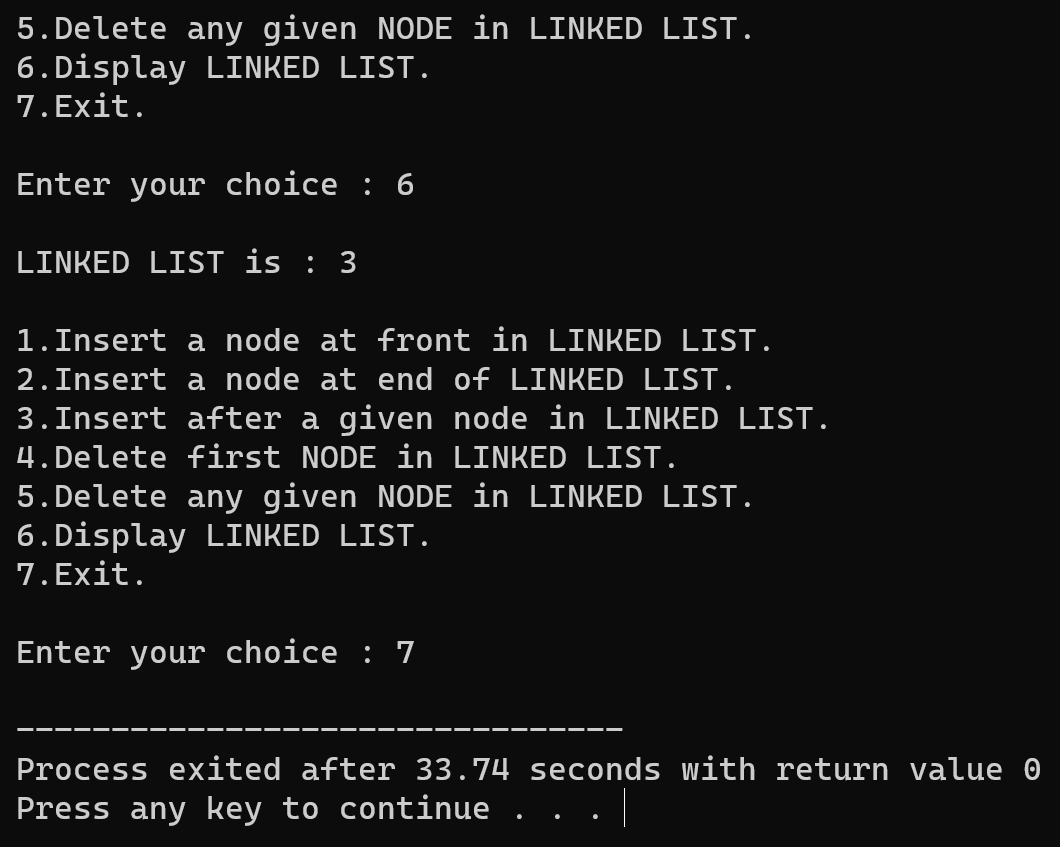
}

}

}

# OUTPUT -



**** ****

PROGRAM 6

Title: Program for Implementation of Stack using dynamic memory allocation.

A stack is an ordered list where insertion and deletion take place at one end called top of the stack (stack pointer).

A stack is a basic data structure that is used all throughout programming. The idea is to think of your data as a stack of plates or books where you can only take the top item off the stack in order to remove things from it.

A stack is also called a LIFO (Last In First Out) to demonstrate the way it accesses data.

Stack<item-type > Operations

push(new-item:item-type) - Adds an item onto the stack.

pop() - Removes the most-recently-pushed item from the stack.

# PSEUDO CODE –

1.)PUSH operation -

PUSH(NODE \*\*TOP,int value)

{

NODE \*ptr;

ptr = malloc(sizeof(NODE));

ptr->INFO = value;

ptr->NEXT = NULL;

if(\*TOP == NULL) {

\*TOP = ptr;

}

else {

ptr->NEXT = \*TOP;

\*TOP = ptr;

}

2.)POP Operation -

int POP(NODE \*\*TOP) {

NODE \*ptr,\*temp;

int y;

if(\*TOP == NULL) {

printf("\n\nSTACK is empty.");

return -1;

}

y = (\*TOP)->INFO;

temp = \*TOP;

\*TOP = (\*TOP)->NEXT;

free(temp);

return y;

}

3.)Display function -

void display(NODE \*\*TOP) {

NODE \*ptr;

ptr = \*TOP;

if(ptr == NULL) {

printf("\n\nSTACK is empty.");

return;

}

printf("\n\nSTACK is : ");

while(ptr != NULL) {

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

ptr = ptr->NEXT;

}

}

# PROGRAM CODE –

#include <stdio.h>

#include <stdlib.h>

typedef struct stack {

int INFO;

struct stack \*NEXT;

}NODE;

void PUSH(NODE \*\*TOP,int value) {

NODE \*ptr;

ptr = malloc(sizeof(NODE));

ptr->INFO = value;

ptr->NEXT = NULL;

if(\*TOP == NULL) {

\*TOP = ptr;

}

else {

ptr->NEXT = \*TOP;

\*TOP = ptr;

}

}

int POP(NODE \*\*TOP) {

NODE \*ptr,\*temp;

int y;

if(\*TOP == NULL) {

printf("\n\nSTACK is empty.");

return -1;

}

y = (\*TOP)->INFO;

temp = \*TOP;

\*TOP = (\*TOP)->NEXT;

free(temp);

return y;

}

void display(NODE \*\*TOP) {

NODE \*ptr;

ptr = \*TOP;

if(ptr == NULL) {

printf("\n\nSTACK is empty.");

return;

}

printf("\n\nSTACK is : ");

while(ptr != NULL) {

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

ptr = ptr->NEXT;

}

}

void main() {

NODE \*TOP;

int ch,value,del;

TOP = NULL;

while(1) {

printf("\n\n1.Insert in STACK.");

printf("\n2.Delete from STACK.");

printf("\n3.Display STACK.");

printf("\n4.Exit.");

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

scanf("%d",&ch);

switch(ch) {

case 1 : printf("\nenter value to add : ");

scanf("%d",&value);

PUSH(&TOP,value);

break;

case 2 : del = POP(&TOP);

if(del != -1)

printf("Deleted Item is is : %d",del);

break;

case 3 : display(&TOP);

break;

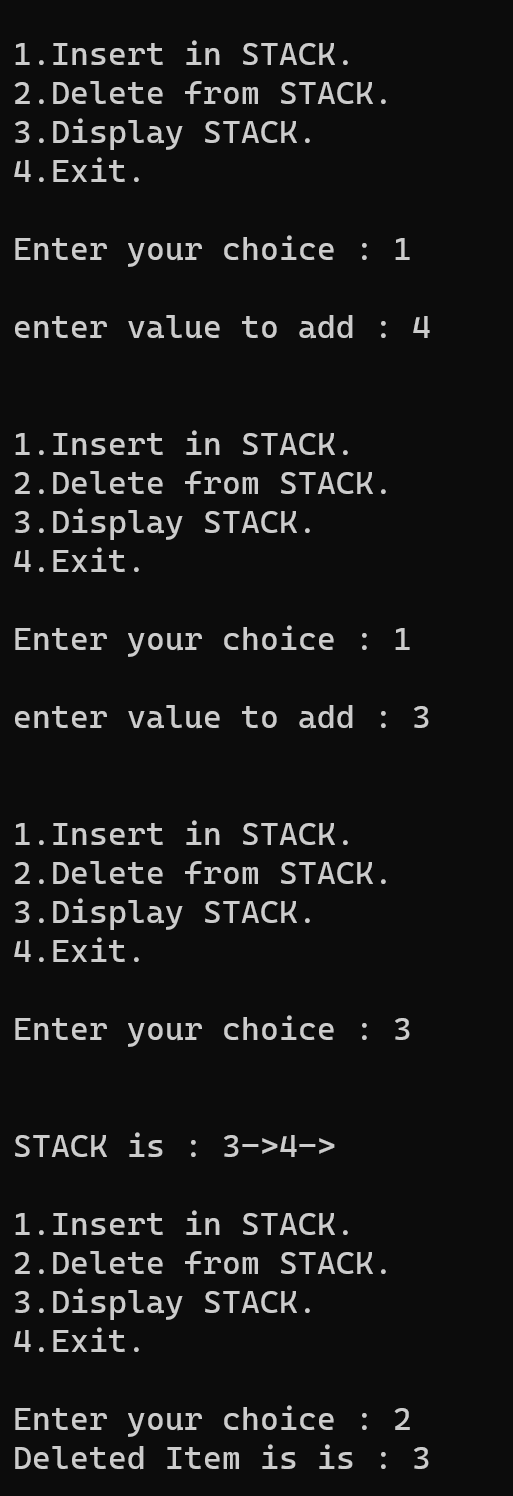
case 4 : exit(0);

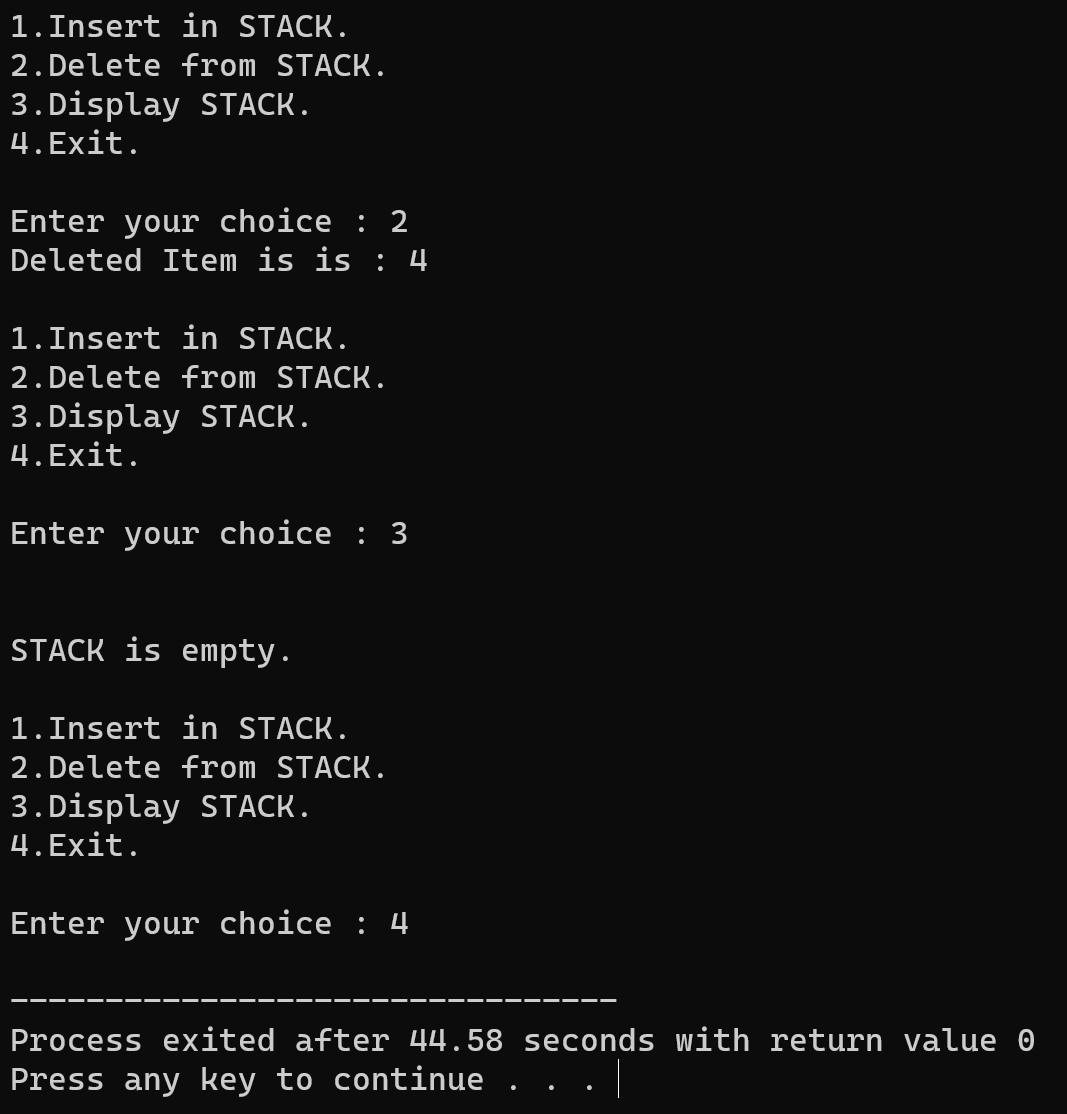
}

}

}

# OUTPUT





# PROGRAM 7

Objective : Program for Queue implementation Using Dynamic Memoryallocation.

For Queue implementation using dynamic memory allocation operations, we can maintain two pointers – qfront and qback as we had done for the case of array implementation of queues.

For the Enqueue operation, the data is first loaded on a new node. If the queue is empty, then after insertion of the first node, both qfront and qback are made to point to this node, otherwise, the new node is simply appended and qback updated.

In Dequeue function, first of all check if at all there is any element. If there is none, we would have \*qfront as NULL, and so report queue to be empty, otherwise return the data element, update the \*qfront pointer and free the node. Special care has to be taken if it was the only node in the queue.

# PSEUDO CODE –

1.)ADD Operation

insert\_LQ(NODE \*\*head,int value) {

NODE \*ptr1,\*ptr2;

ptr1 = malloc(sizeof(NODE));

ptr2 = \*head;

ptr1->INFO = value;

ptr1->NEXT = NULL;

if(ptr2 == NULL) {

\*head = ptr1;

return;

}

while(ptr2->NEXT != NULL) {

ptr2 = ptr2->NEXT;

}

ptr2->NEXT = ptr1;

}

}

2.)Delete Operation –

int DEL\_LQ(NODE \*\*head) {

NODE \*ptr,\*temp;

int y;

if(\*head == NULL) {

printf("\nQUEUE is empty.");

return -1;

}

y = (\*head)->INFO;

temp = \*head;

\*head = (\*head)->NEXT;

free(temp);

return y;

}

3.)DISPLAY Operation

void display(NODE \*\*head) {

NODE \*ptr;

ptr = \*head;

if(ptr == NULL) {

printf("\nQUEUE is empty.");

return;

}

printf("\n\nQUEUE is : ");

while(ptr != NULL) {

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

ptr = ptr->NEXT;

}

}

# PROGRAM CODE –

#include <stdio.h>

#include <stdlib.h>

typedef struct queue {

int INFO;

struct queue \*NEXT;

}NODE;

void insert\_LQ(NODE \*\*head,int value) {

NODE \*ptr1,\*ptr2;

ptr1 = malloc(sizeof(NODE));

ptr2 = \*head;

ptr1->INFO = value;

ptr1->NEXT = NULL;

if(ptr2 == NULL) {

\*head = ptr1;

return;

}

while(ptr2->NEXT != NULL) {

ptr2 = ptr2->NEXT;

}

ptr2->NEXT = ptr1;

}

int DEL\_LQ(NODE \*\*head) {

NODE \*ptr,\*temp;

int y;

if(\*head == NULL) {

printf("\nQUEUE is empty.");

return -1;

}

y = (\*head)->INFO;

temp = \*head;

\*head = (\*head)->NEXT;

free(temp);

return y;

}

void display(NODE \*\*head) {

NODE \*ptr;

ptr = \*head;

if(ptr == NULL) {

printf("\nQUEUE is empty.");

return;

}

printf("\n\nQUEUE is : ");

while(ptr != NULL) {

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

ptr = ptr->NEXT;

}

}

void main() {

NODE \*head;

int ch,value,del;

while(1) {

printf("\n1.Insert in QUEUE.");

printf("\n2.Delete From QUEUE.");

printf("\n3.Display.");

printf("\n4.Exit.");

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

scanf("%d",&ch);

switch(ch) {

case 1 : printf("\nenter value to add in QUEUE : ");

scanf("%d",&value);

insert\_LQ(&head,value);

break;

case 2 : del = DEL\_LQ(&head);

if(del != -1)

printf("Deleted ITEM is : %d",del);

break;

case 3 : display(&head);

break;

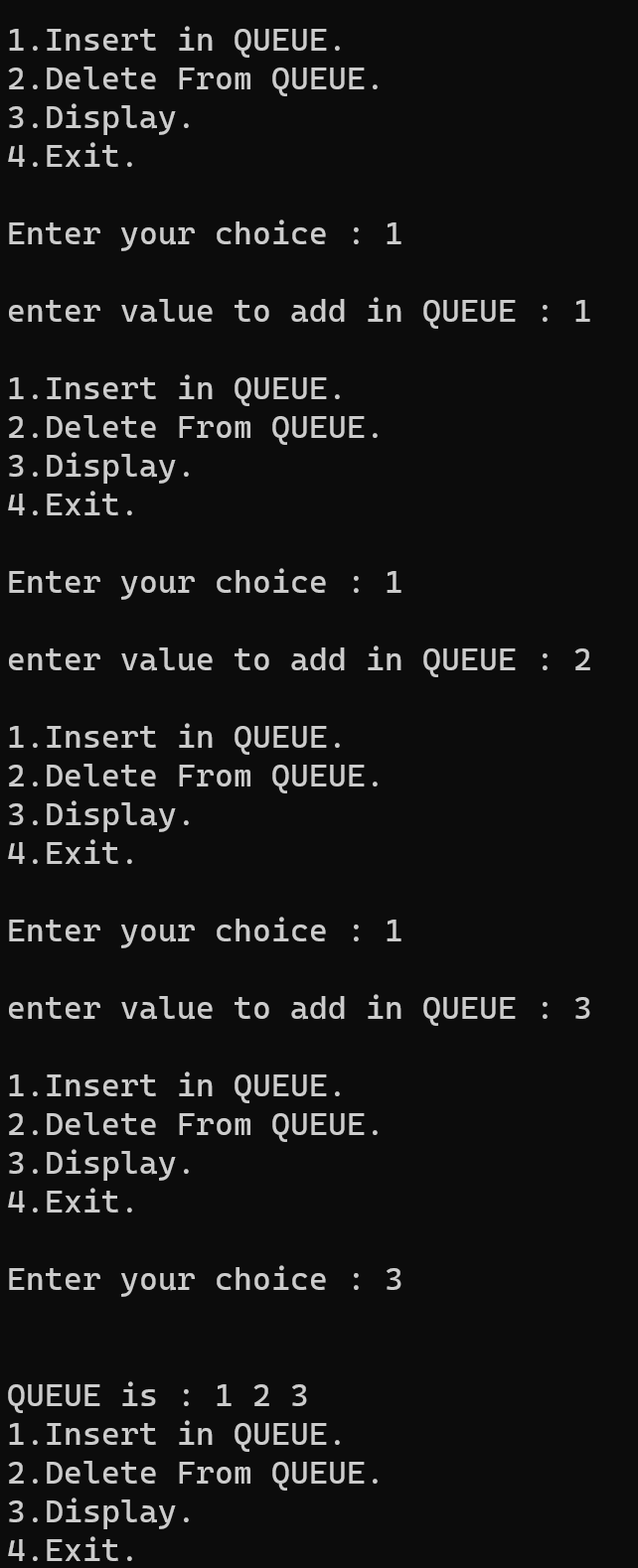
case 4 : exit(0);

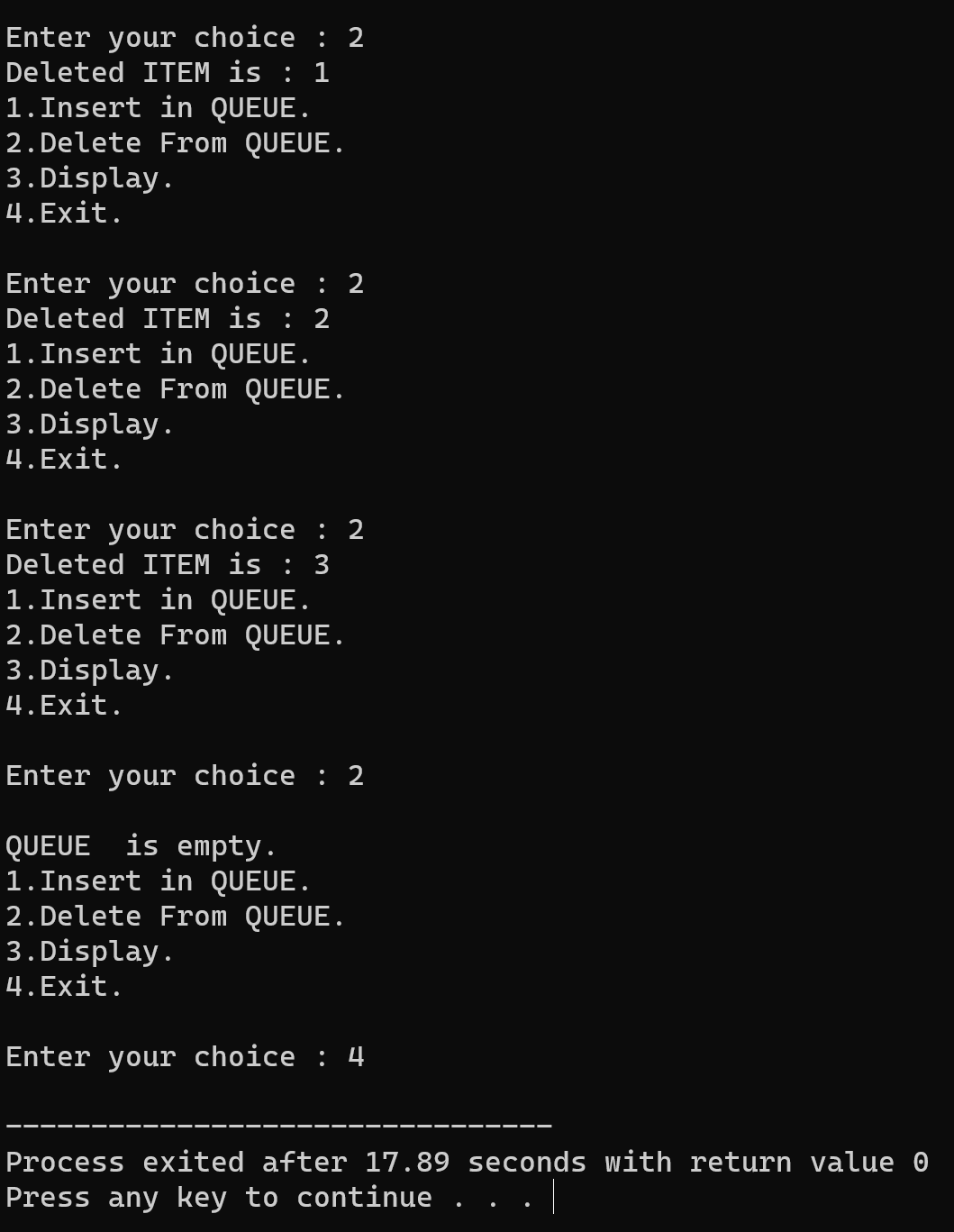
}

}

}

# OUTUPUT





PROGRAM 8

Title: Program for Circular Queue implementation through Array.

A circular queue is a Queue but a particular implementation of a queue. It is very efficient. It is also quite useful in low level code, because insert ion and deletion are totally independent. Which means that you don’t have to worry about an interrupt handler trying to do an insertion at the same time as your main code is doing deletion.

# PSEUDO CODE –

1.)ADD Operation –

void add\_CQ(QUEUE \*q,int value) {

if(q->front == (q->rear+1)%MAX) {

printf("\nCircualar queue is full.");

}

else {

if(q->rear == -1)

q->rear = q->front = 0;

else

q->rear = (q->rear+1)%MAX;

q->Q[q->rear] = value;

}

}

2.)DELETE Operation –

int del\_CQ(QUEUE \*q) {

int y;

if(q->front == -1) {

printf("CIRCULAR QUEUE is empty.");

return -1;

}

y = q->Q[q->front];

if(q->front == q->rear)

q->front = q->rear = -1;

else

q->front = (q->front+1)%MAX;

return y;

}

3.)DISPLAY Operation

void display(QUEUE \*q) {

int x;

if(q->rear == -1) {

printf("\nCIRCULAR QUEUE is empty.");

return;

}

printf("\nQUEUE is : ");

for(x = q->front;x <= (q->rear)%MAX;x++) {

printf("%d ",q->Q[x]);

}

}

# PROGRAM CODE –

#include <stdio.h>

# define MAX 5

#include <stdlib.h>

typedef struct cq {

int front;

int rear;

int Q[MAX];

}QUEUE;

void Create\_CQ(QUEUE \*q) {

q->front = -1;

q->rear = -1;

}

void add\_CQ(QUEUE \*q,int value) {

if(q->front == (q->rear+1)%MAX) {

printf("\nCircualar queue is full.");

}

else {

if(q->rear == -1)

q->rear = q->front = 0;

else

q->rear = (q->rear+1)%MAX;

q->Q[q->rear] = value;

}

}

int del\_CQ(QUEUE \*q) {

int y;

if(q->front == -1) {

printf("CIRCULAR QUEUE is empty.");

return -1;

}

y = q->Q[q->front];

if(q->front == q->rear)

q->front = q->rear = -1;

else

q->front = (q->front+1)%MAX;

return y;

}

void display(QUEUE \*q) {

int x;

if(q->rear == -1) {

printf("\nCIRCULAR QUEUE is empty.");

return;

}

printf("\nQUEUE is : ");

for(x = q->front;x <= (q->rear)%MAX;x++) {

printf("%d ",q->Q[x]);

}

}

void main() {

QUEUE q;

int y,ch,value;

Create\_CQ(&q);

while (1)

{

printf("\n1 : ADD");

printf("\n2 : DElETE");

printf("\n3 : Display");

printf("\n4 : Exit");

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

scanf("%d",&ch);

switch (ch)

{

case 1:

printf("Enter value : ");

scanf("%d",&value);

add\_CQ(&q,value);

break;

case 2:

y = del\_CQ(&q);

if (y != -1)

printf("Deleted item is %d", y);

break;

case 3:

display(&q);

break;

case 4:

exit(0);

}

}

}

# OUTPUT –

