**Queue Implementation Using Linked List**

**Done By:** Rohit Karunakaran **Roll no:** 58

**Date :** 12-11-2020

**Aim:** To implement a Queue using Linked List

**Data Structure used :** Queue, Linked List

**Algorithms**

**1. Algorithm for Enqueue**

**Input:** An Array implementation of Queue (Q), with Front pointing to the first element and Rear pointing to the last element in and an element ITEM to be inserted into the queue.

**Output:** The Queue with the element ITEM inserted at the rear

**Data Structure:** Queue, Linked List

**Steps:**

Step 1: Start

Step 2: new = GetNode(Node)

Step 3: if(new == NULL)

Step 1: Print(“Can nont Insert a new node”)

Step 2: Exit(1)

Step 4: else

Step 1: new→data = ITEM

Step 2: new→ Link = NULL

Step 3: if(Front==NULL) then

Step 1: Front = new

Step 4: else

Step 1: Rear→ link = new

Step 5: endif

Step 6: Rear = new

Step 5: endif

Step 6: Stop

**2. Algorithm for dequeue**

**Input:** An Array implementation of Queue (Q), with Front pointing to the first element and Rear pointing to the last element in the queue.

**Output:** The element ITEMwhich is removed form the Front of the queue

Steps

Step 1: if(front == NULL) then

Step 1: print(“The Queue is empty”)

Step 2: exit(1)

Step 2: else

Step 1: ITEM = Front→data

Step 2: rem = Front

Step 3: if(Front==Rear)then

Step 1:Rear =NULL

Step 2: Front = NULL

Step 4:else

Step 1: Front = Front→link

Step 5:endif

Step 6: ReturnNode(rem)

Step 7: return ITEM

Step 3: endif

Step 4: Stop

**Program code:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Queue Implementation Using Linked List

\* Done By: Rohit Karunakaran

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct Linked\_List\_Node

{

struct Linked\_List\_Node \*link;

int data;

}Node;

typedef struct Linked\_Queue

{

Node\* Front;

Node\* Rear;

}Queue;

Queue\* initQueue()

{

Queue \*q = (Queue\*) malloc (sizeof(Queue));

q->Front = NULL;

q->Rear = NULL;

return q;

}

//Insertion Algorithm

void enQueue(Queue \*q,int val)

{

Node \*new\_node = (Node\*) malloc(sizeof(Node));

if(new\_node!=NULL)

{

new\_node->link=NULL;

new\_node->data = val;

if(q->Rear == NULL)

{

q->Front = new\_node;

}

else

{

q->Rear->link = new\_node;

}

q->Rear = new\_node;

}

else

{

printf("Queue Is Full");

exit(1);

}

return ;

}

//Deletion Algorithm

int deQueue(Queue \*q)

{

if(q->Front == NULL)

{

printf("Queue Is Empty");

exit(0);

return 0;

}

else

{

Node\* ptr = q->Front;

q->Front = q->Front->link;

int elem = ptr->data;

free(ptr);

return elem;

}

}

void displayQueue(Queue \*q)

{

Node\* ptr = q->Front;

if(ptr!=NULL)

{

printf("The Queue is: ");

while(ptr!=NULL)

{

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

ptr=ptr->link;

}

printf("\n");

}

else

{

printf("The Queue is empty\n");

}

}

int menu(Queue\* q)

{

int RUN = 1;

while(RUN)

{

printf("\n");

printf("=============================\n");

printf(" MENU \n");

printf("=============================\n");

printf("1.Enqueue\n");

printf("2.Dequeue\n");

printf("3.Display the Queue\n");

printf("4.Exit\n");

printf("Enter Choice: ");

int choice;

int elem;

scanf("%d%\*c",&choice);

switch(choice)

{

case 1: printf("Enter the element to be inserted: ");

scanf("%d%\*c",&elem);

enQueue(q,elem);

printf("\n");

break;

case 2: elem = deQueue(q);

printf("The Element removed is %d",elem);

printf("\n");

break;

case 3: displayQueue(q);

break;

case 4: RUN=0;

break;

default: printf("Enter a valid choice\n");

printf("\n");

break;

}

}

printf("Exiting........");

return RUN;

}

int main()

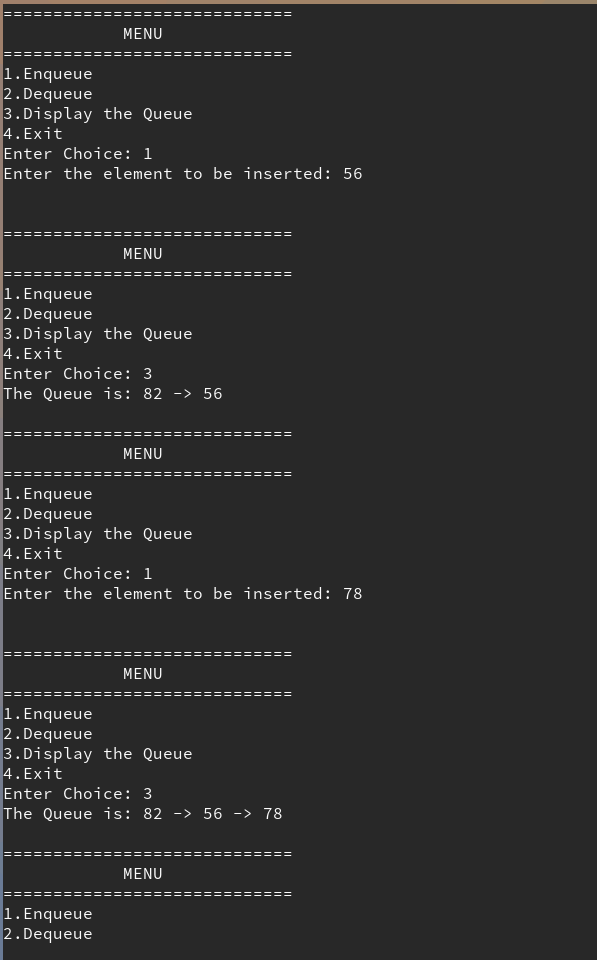
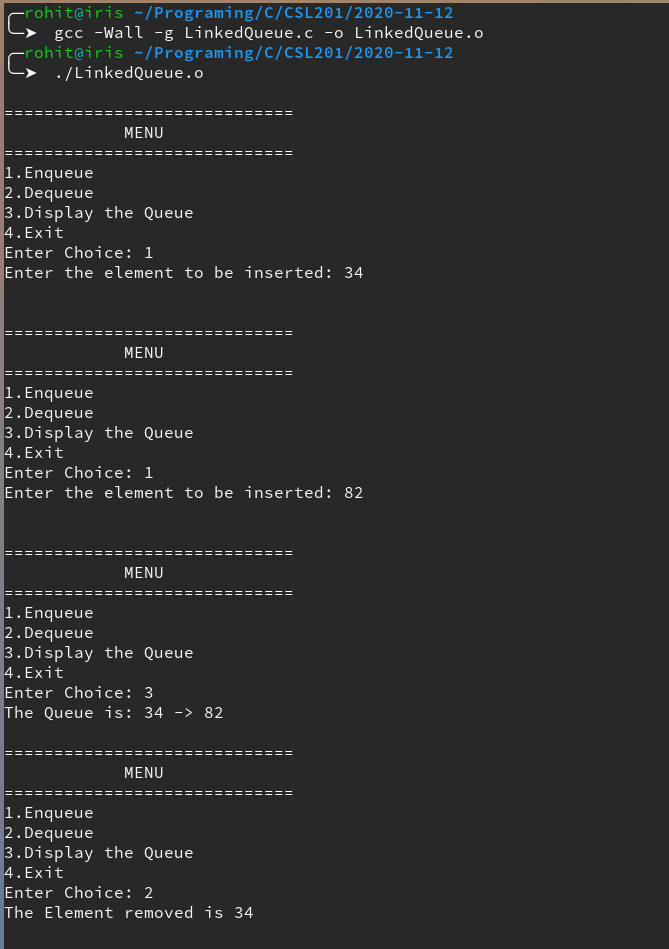
{

Queue \*q = initQueue();

return menu(q);

}

**Result:** the Program compiled successfully and the desired output was obtained.

**Sample Input/Output**

