**Date:-**

**ASSIGNMENT NUMBER :-**

**PROBLEM STATEMENT:-**

Program in C to implement Queue using Singly Linked List.

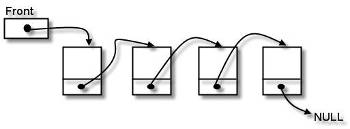
**THEORY:-**

A queue is like a line of people. The first person to join a line is the first person served and is thus the first to leave the line. Queues are appropriate for many real-world situations. For instance, we often wait in a queue to buy a movie ticket, or to use an automatic ticket machine. The person at the front of the queue is served, while new people join the queue at its rear. A queue is then a form of list. The difference from the previous forms of list is in its access procedures.

Whereas a stack can be seen as a list with only one end, because all the operations are performed at the top of the stack, the queue can be seen, in contrast, as a list with two ends, the front and the rear.

Queues have the main property of allowing new items to be added only at the rear of the list, and the item first inserted in the list (i.e. the item at the front of the list) is the first to be removed. This behaviour is commonly referred to as “first-in-first-out” or simply FIFO. So the first item in a queue is the “front” item, whereas the last item in a queue is the “rear”item.

The access procedures for a queue include operations such as examining whether the queue is empty, inspecting the item at the front of the queue but not others, placing an item at the rear of the queue, but at no other position, and removing an item from the front of the queue, but from no other position.



**ALGORITHM:-**

**Input specification:** An array (Sorted for linear/Binary search or Unsorted for linear search) say **a[]**, where the search will be done,

The element which need to be searched, say **find** and the number of elements in the array a[], say **n**.

**Output specification:** Success message of the search with the position of the element or appropriate failure message.

**Steps:**

Algorithm for method enqueue(\*ptr):

1. temp=getnode()//getnode() is a function to dynamically allocate memory
2. Print "Enter the element: "
3. Input temp->data
4. temp->next=ptr
5. ptr=temp
6. Return ptr

Algorithm for method dequeue(\*front,\*rear):

1. temp=rear
2. If(temp==NULL) Then
   1. Print "No element to delete! The queue is empty!"
   2. Return NULL
3. Else If(temp==front) Then
   1. Print "The deleted element is: "temp->data
   2. free(temp)
   3. Return NULL
4. Else
   1. Repeat step 3.a.i While(temp->next->next!=NULL)
      1. temp=temp->next
   2. temp1=temp->next
   3. temp->next=NULL
   4. Print "The deleted element is: "temp1->data
   5. free(temp1)
5. Return temp

Algorithm for method binary\_search(a[], n, find):

1. If(ptr==NULL) Then
   1. Print "No element to show! The queue is empty!"
2. Else
   1. Repeat step 2.a.i to 2.a.ii While(temp->next!=NULL)
      1. Print temp->data
      2. temp=temp->next
   2. Print temp->data

**SOURCE CODE:-**

#include<stdio.h>

#include<stdlib.h>

typedef struct sll{

int data;

struct sll \*next;

}sll;

sll \*enqueue(sll \*ptr){

sll \*temp;

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

printf("Enter the element: ");

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

temp->next=ptr;

ptr=temp;

return ptr;

}

sll \*dequeue(sll \*temp, sll \*front){

sll \*temp1;

if(temp==NULL){

printf("No element to delete! The queue is empty!");

return NULL;

}

else if(temp==front){

printf("The deleted element is: %d\n",temp->data);

free(temp);

return NULL;

}

else{

while(temp->next->next!=NULL){

temp=temp->next;

}

temp1=temp->next;

temp->next=NULL;

printf("The deleted element is: %d\n",temp1->data);

free(temp1);

}

return temp;

}

void display(sll \*ptr){

sll \*temp=ptr;

if(ptr==NULL)

printf("No element to show! The queue is empty!\n");

else{

while(temp->next!=NULL){

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

temp=temp->next;

}

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

}

}

int main(){

sll \*rear=NULL,\*front=NULL,\*temp;

int ch,i,n;

while(1){

printf("\n1.Insert\n2.Delete\n3.Traverse\n4.Exit\nEnter your choice: ");

scanf("%d",&ch);

switch(ch){

case 1: printf("Enter the no. of elements to insert: ");

scanf("%d",&n);

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

rear=enqueue(rear);

temp=rear;

while(temp->next!=NULL)

temp=temp->next;

front=temp;

break;

case 2: front=dequeue(rear,front);

if(front==NULL)

rear=NULL;

break;

case 3: display(rear);

break;

case 4: return 1;

default: printf("Wrong choice!");

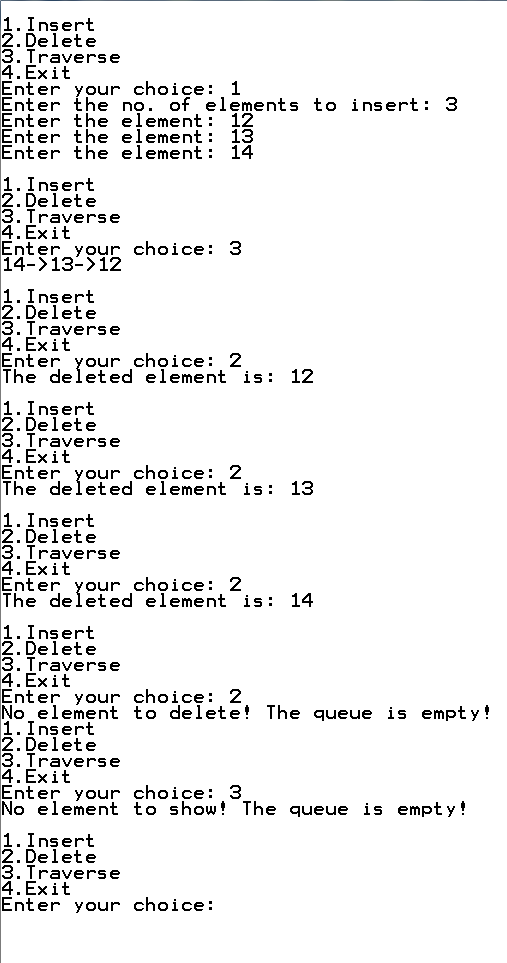
}

}

return 0;

}

**Input & Output:**



**DISCUSSION:-**

1. They are difficult to create, maintain and manipulate as many of them are implemented using pointers.
2. They do not provide sequential access, this may be advantageous in some algorithms but in others it is a disaster.
3. Nonlinear data structure takes up more memory as compared to linear data structure. This is mainly because non-linear Data Structures require pointers or adjacency matrix or some other technique to logically represent it.
4. The algorithm in this program is a basic implementation of queue using SLL and can be developed further by decreasing the lines of code in the main function.

--------------------------------