Experiment No.8
Implement Linear Queue ADT using Linked list
Name: Sharvari Anand Bhondekar
Roll No: 06
Date of Performance:
Date of Submission:
Marks:
Sign:

Experiment No. 8: Linear Queue using Linked list

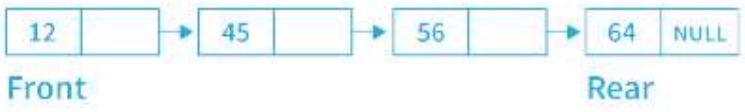
Aim: Implement Linear Queue ADT using Linked list

Objective:

Linear queue can be implemented using a linked list for dynamic allocation. Linked list implementation gives flexibility and better performance to the linear queue.

Theory:

A linear queue implemented using an array has a limitation in that it can only handle a fixed number of data values, and this size must be defined at the outset. This limitation makes it unsuitable for cases where the data size is unknown. On the other hand, linear queue implemented using a linked list is more flexible and can accommodate an unlimited number of data values, making it suitable for variable-sized data. A queue that is implemented using a linked list will continue to operate in accordance with the FIFO principle. Front and rare pointers are used to insert and delete the elements from linear queue implemented using array.



Linear queue Operations using Linked List

To implement a Linear queue using a linked list, we need to set the following things before implementing actual operations.

Step 1 - Include all the header files which are used in the program. And declare all the user defined functions.

Step 2 - Define a 'Node' structure with two members data and next.

Step 3 - Define a two Node pointers 'front' and ‘rare’ and set it to NULL.

Step 4 - Implement the main method by displaying Menu with list of operations and make suitable function calls in the main method.

enqueue(value) - Inserting an element into the Queue

Step 1 - Create a newNode with given value.

Step 2 - Check for two conditions one is whether queue is Empty (front ==rear == NULL) or the queue contains at least one element.

Step 3 – If it is Empty, then the new node added will be both front and rear, and the next pointer of front and rear will point to NULL.

Step 4 –If the queue contains at least one element, then the condition front == NULL becomes false. So, make the next pointer of rear point to new node ptr and point the rear pointer to the newly created node ptr

```
rear -> next = ptr;
 rear = ptr;
```

dequeue() - Deleting an Element from a Queue

Step 1 - Check whether queue is Empty or not (top == NULL).

Step 2 - If the queue is empty, i.e., front==NULL, so we just print 'underflow' on the screen and exit.

Step 3 - If the queue is not empty, delete the element at which the front pointer is pointing. For deleting a node, copy the node which is pointed by the front pointer into the pointer ptr and make the front pointer point to the front's next node and free the node pointed by the node ptr. This can be done using the following statement:.

```
*ptr = front;
 front = front -> next;
 free(ptr);
```

display() - Displaying queue of elements

Step 1 - Check whether queue is Empty (top == NULL).

Step 2 - If it is Empty, then display 'Queue is Empty!!!' and terminate the function.

Step 3 - If it is Not Empty, then define a Node pointer 'temp' and initialize with front.

Step 4 – Display 'temp → data --->' and move it to the next node. Repeat the same until temp reaches to the last node in the queue. (temp → next != NULL).

Step 5 – Finally! Display 'temp → data ---> NULL'.

Code:

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
```

```
struct node{
int data;
struct node *next;
};
struct node *front = NULL;
struct node *rear = NULL;
void enqueue(){
int num;
struct node *temp;
temp = (struct node *)malloc(sizeof(struct node));
printf("Enter Number:");
scanf("%d",&num);
temp->data = num;
if(front == NULL && rear == NULL){
temp->next=NULL;
rear=temp;
front=temp;
}
else{
rear->next = temp;
rear = temp;
temp->next = NULL;
}
}

void dequeue(){
int num;
struct node *p;
if(front == NULL && rear == NULL){
printf("Queue is Empty");
}
```

```
else if(front == rear){
printf("%d is deleted",front->data);
front=NULL;
rear= NULL;
}
else{
p=front;
printf("%d is deleted\n",front->data);
front = front->next;
p->next = NULL;
}
}

void display(){
struct node *display = front;
if(front == NULL && rear == NULL){
printf("Queue is Empty");
}
else{
printf("Contents:\n");
while(display!=NULL){
printf("%d\n",display->data);
display = display->next;
}
}
}

void main()
{
enqueue();
enqueue();
enqueue();
display();
```

```
dequeue();
display();
getch();
}
```

Output:

```
Enter Number:1
Enter Number:2
Enter Number:3
Contents:
1
2
3
1 is deleted
Contents:
2
3
```

Conclusion:

Write in detail about an application where Queue is implemented as a linked list?

1.Print Queue Management: In a multi-user environment, such as an office or a school, multiple users often send print jobs to a shared printer. Managing these jobs efficiently is crucial to ensure that documents are printed in the order they were received or according to specific priority rules.In offices where multiple employees submit documents for printing, the queue helps maintain order and efficiency, preventing printer bottlenecks.

2.Network Packet Scheduling: In computer networking, packets of data are transmitted across various paths from source to destination. To manage this data flow efficiently, especially in congested networks, packet scheduling is essential. Queues can be used to prioritize packets based on several criteria, such as type of service or time of arrival.For platforms like Netflix or YouTube, ensuring smooth streaming requires prioritizing data packets related to video and audio streams.