Date : Page No :

Assignment Number

Problem Statement

Program in C to implement Queue using Singly Linked List.

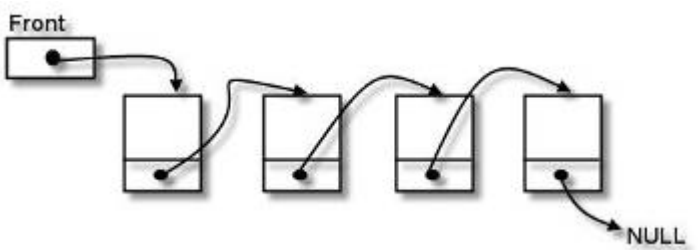
Theory

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 :

1. A pointer to the front of the queue implemented by a linked list, say FRONT
2. A pointer to the end of the queue implemented by a linked list, say REAR

Data Structure Used : A singly linked list, where each node contains two members, a data member, say DATA, and a pointer which holds the address of the next node in the list, say NEXT

Algorithm_Enqueue() :

Step 1 : temp=getnode() //getnode() is a function which dynamically
// allocates memory for a node of the list
Step 2 : If(temp = Null)
a. Print “Insufficient Memory”
b. Exit
Step 3 : Print "Enter the element: "
Step 4 : Input temp->data
Step 5 : temp->next=ptr
Step 6 : ptr=temp
Step 7 : Return ptr

Algorithm_Deque() :

Step 1 : temp=rear
Step 2 : If(temp = Null) Then
a. Print "No element to delete! The queue is empty!"
b. Return Null
Step 3 : Else If(temp = front) Then
a. Print "The deleted element is: "temp->data
b. Free(temp) // Free is a procedure which deallocates memory
// pointed by the argument pointer
c. Return Null
Step 4 : Else

a. Repeat step 3.a.i While(temp→next→next is not Null)
    i. temp=temp->next
b. temp1=temp->next
c. temp->next=NULL
d. Print "The deleted element is: " temp1->data
e. Free(temp1)

Step 5 : Return temp

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 and Output

1.Insert
2.Delete
3.Traverse
4.Exit
Enter your choice: 1
Enter the no. of elements to insert: 6
Enter the element: 12
Enter the element: 35
Enter the element: 291
Enter the element: 39
Enter the element: 46
Enter the element: 83

1.Insert
2.Delete
3.Traverse
4.Exit
Enter your choice: 3
83->46->39->291->35->12

1.Insert
2.Delete
3.Traverse
4.Exit
Enter your choice: 2
The deleted element is: 12

1.Insert
2.Delete
3.Traverse
4.Exit
Enter your choice: 2
The deleted element is: 35

1.Insert
2.Delete
3.Traverse
4.Exit
Enter your choice: 2
The deleted element is: 291

1.Insert
2.Delete
3.Traverse
4.Exit
Enter your choice: 4

Discussion

1. Queue does not provide sequential access, this may be advantageous in some algorithms but in others it is a disaster.
2. 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.
3. Nonlinear data structures are particularly bad for processor caches as they tend to be fragmented all over the memory, hence performance is affected.
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.