**Experiment: 3.1**

# Student Name: Vivek UID:

Branch: CSE Section/Group:

# Semester: 3rd Subject Name: Data Structure Subject Code: 21CSH-211

# **Aim/Overview of the practical:**

# Program to implement Queue using Linked list.

# **Objective:**

To study the implementation of various operation on Queue using Linked list.

**Theory:**

**Queue:**

1. A queue can be defined as an ordered list which enables insert operations to be performed at one end called **REAR** and delete operations to be performed at another end called **FRONT**.

2. Queue is referred to be as First In First Out list.

3. For example, people waiting in line for a rail ticket form a queue.



## Applications of Queue:

Due to the fact that queue performs actions on first in first out basis which is quite fair for the ordering of actions. There are various applications of queues discussed as below.

1. Queues are widely used as waiting lists for a single shared resource like printer, disk, CPU.
2. Queues are used in asynchronous transfer of data (where data is not being transferred at the same rate between two processes) for eg. pipes, file IO, sockets.
3. Queues are used as buffers in most of the applications like MP3 media player, CD player, etc.
4. Queue are used to maintain the play list in media players in order to add and remove the songs from the play-list.
5. Queues are used in operating systems for handling interrupts.

# **Linked List implementation of Queue:**

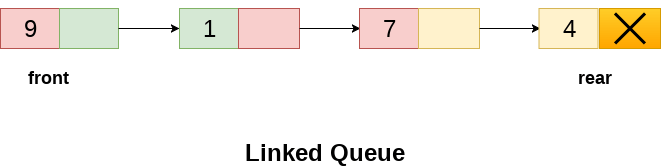
Due to the drawbacks discussed in the previous section of this tutorial, the array implementation can not be used for the large scale applications where the queues are implemented. One of the alternative of array implementation is linked list implementation of queue.

The storage requirement of linked representation of a queue with n elements is o(n) while the time requirement for operations is o(1).

In a linked queue, each node of the queue consists of two parts i.e. data part and the link part. Each element of the queue points to its immediate next element in the memory.

In the linked queue, there are two pointers maintained in the memory i.e. front pointer and rear pointer. The front pointer contains the address of the starting element of the queue while the rear pointer contains the address of the last element of the queue.

Insertion and deletions are performed at rear and front end respectively. If front and rear both are NULL, it indicates that the queue is empty.



## Insert operation

The insert operation append the queue by adding an element to the end of the queue. The new element will be the last element of the queue.

## Deletion

Deletion operation removes the element that is first inserted among all the queue elements. Firstly, we need to check either the list is empty or not. The condition front == NULL becomes true if the list is empty, in this case , we simply write underflow on the console and make exit.

Otherwise, we will delete the element that is pointed by the pointer front. For this purpose, copy the node pointed by the front pointer into the pointer ptr. Now, shift the front pointer, point to its next node and free the node pointed by the node ptr.

# **ALGORITHM:**

## Insert operation:

* **Step 1:** Allocate the space for the new node PTR
* **Step 2:** SET PTR -> DATA = VAL
* **Step 3:** IF FRONT = NULL  
  SET FRONT = REAR = PTR  
  SET FRONT -> NEXT = REAR -> NEXT = NULL  
  ELSE  
  SET REAR -> NEXT = PTR  
  SET REAR = PTR  
  SET REAR -> NEXT = NULL  
  [END OF IF]
* **Step 4:** END

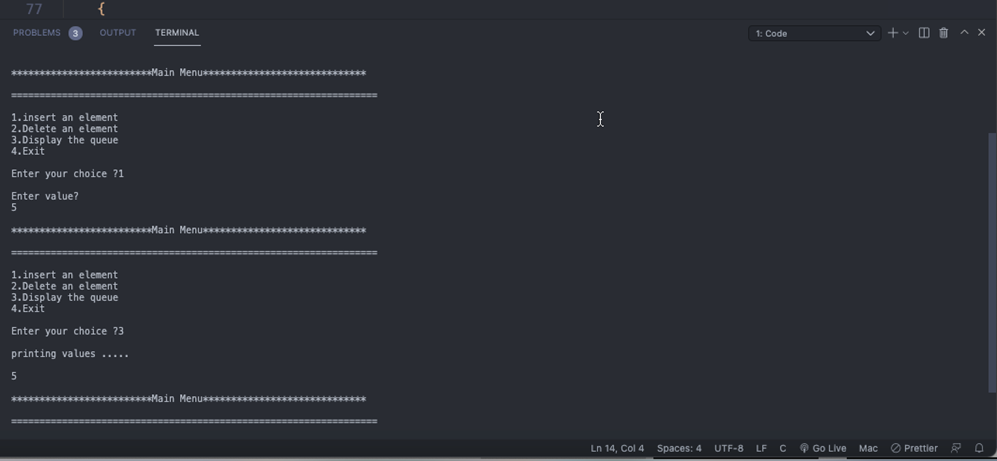
## Deletion:

* **Step 1:** IF FRONT = NULL  
  Write " Underflow "  
  Go to Step 5  
  [END OF IF]
* **Step 2:** SET PTR = FRONT
* **Step 3:** SET FRONT = FRONT -> NEXT
* **Step 4:** FREE PTR
* **Step 5:** END

# **PROGRAM CODE:**

1. #include<stdio.h>
2. #include<stdlib.h>
3. struct node
4. {
5. int data;
6. struct node \*next;
7. };
8. struct node \*front;
9. struct node \*rear;
10. void insert();
11. void delete();
12. void display();
13. void main ()
14. {
15. int choice;
16. while(choice != 4)
17. {
18. printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");
19. printf("\n=================================================================\n");
20. printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");
21. printf("\nEnter your choice ?");
22. scanf("%d",& choice);
23. switch(choice)
24. {
25. case 1:
26. insert();
27. break;
28. case 2:
29. delete();
30. break;
31. case 3:
32. display();
33. break;
34. case 4:
35. exit(0);
36. break;
37. default:
38. printf("\nEnter valid choice??\n");
39. }
40. }
41. }
42. void insert()
43. {
44. struct node \*ptr;
45. int item;
47. ptr = (struct node \*) malloc (sizeof(struct node));
48. if(ptr == NULL)
49. {
50. printf("\nOVERFLOW\n");
51. return;
52. }
53. else
54. {
55. printf("\nEnter value?\n");
56. scanf("%d",&item);
57. ptr -> data = item;
58. if(front == NULL)
59. {
60. front = ptr;
61. rear = ptr;
62. front -> next = NULL;
63. rear -> next = NULL;
64. }
65. else
66. {
67. rear -> next = ptr;
68. rear = ptr;
69. rear->next = NULL;
70. }
71. }
72. }
73. void delete ()
74. {
75. struct node \*ptr;
76. if(front == NULL)
77. {
78. printf("\nUNDERFLOW\n");
79. return;
80. }
81. else
82. {
83. ptr = front;
84. front = front -> next;
85. free(ptr);
86. }
87. }
88. void display()
89. {
90. struct node \*ptr;
91. ptr = front;
92. if(front == NULL)
93. {
94. printf("\nEmpty queue\n");
95. }
96. else
97. {   printf("\nprinting values .....\n");
98. while(ptr != NULL)
99. {
100. printf("\n%d\n",ptr -> data);
101. ptr = ptr -> next;
102. }
103. }
104. }

# **Result/Output:**



# **LEARNING OUTCOMES (What I have learnt):**

1. I have learnt about Queue.
2. I have learnt how to perform different operations on Queue.
3. I have learnt about Circular Queue.