**EXPERIMENT NO. 05**

|  |  |
| --- | --- |
| **DATE OF PERFORMANCE:** | **GRADE:** |
| **DATE OF ASSESSMENT:** | **SIGNATURE OF LECTURER/ TTA:** |

**AIM: IMPLEMENTATION OF QUEUE.**

**THEORY:**

**QUEUES:**

**A queue is simply a waiting line that grows by adding elements to its end and shrinks by removing elements from the front. Compared to stack, it reflects the more commonly used maxim in real-world, namely, “first come, first served”. Waiting lines in supermarkets, banks, food counters are common examples of queues**.

**A formal definition of queue as a data structure: It is a list from which items may be deleted at one end (front) and into which items may be inserted at the other end (rear). It is also referred to as a first-in-first-out (FIFO) data structure.**

**APPLICATIONS OF QUEUE:**

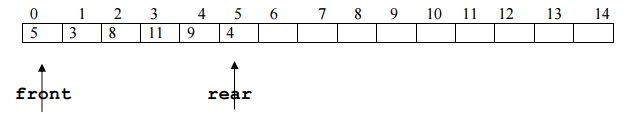
**1. It is used to schedule the jobs to be processed by the CPU.**

**2. When multiple users send print jobs to a printer, each printing job is kept in the printing queue. Then the printer prints those jobs according to first in first out (FIFO) basis.**

**3. Breadth first search uses a queue data structure to find an element from a graph.**

**ARRAY IMPLEMENTATION:**

**The array to implement the queue would need two variables (indices) called front and rear to point to the first and the last elements of the queue.**

****

**Initially: q->rear = -1;**

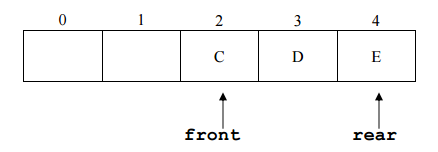
**q->front = -1;**

**For each enqueue operation rear is incremented by one, and for each dequeue operation , front is incremented by one.**

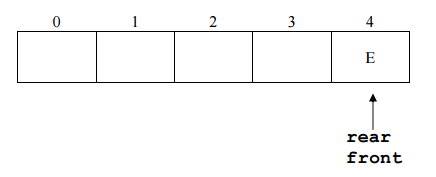
**While the enqueue and dequeue operations are easy to implement, there is a big disadvantage in this set up. The size of the array needs to be huge, as the number of slots would go on increasing as long as there are items to be added to the list (irrespective of how many items are deleted, as these two are independent operations.)**

**PROBLEMS WITH THIS REPRESENTATION:**

**Although there is space in the following queue, we may not be able to add a new item. An attempt will cause an overflow.**

****

**It is possible to have an empty queue yet no new item can be inserted.( when front moves to the point of rear, and the last item is deleted.)**

****

**A SOLUTION: CIRCULAR QUEUE**

**Let us now imagine that the above array is wrapped around a cylinder, such that the first and last elements of the array are next to each other. When the queue gets apparently full, it can continue to store elements in the empty spaces in the beginning of the array. This makes efficient use of the array slots. It is also referred to as a circular array. This enables us to utilize the unavailable slots, provided the indices front and rear are handled carefully. Here again front refers to the index of the element to be next removed from the queue, and rear refers to the index of the last element added to the queue.**

**QUEUE SIZE :**

**The front and rear indices can be used to find out the current size of the queue, that is, the number of elements currently in the queue. The value rear – front gives us the size, and when this value is negative, we simply add the capacity to this to give us the size. Thus in general we have,**

**size = ( rear – front + capacity) % capacity.**

**PRIORITY QUEUE:**

**a priority queue is an abstract data type which is like a regular queue or stack data structure, but where additionally each element has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority.**

**PROGRAM-1: C PROGRAM TO IMPLEMENT A QUEUE USING AN ARRAY.**

**#include <stdio.h>**

**#define MAX 50**

**int queue\_array[MAX];**

**int rear = - 1;**

**int front = - 1;**

**main()**

**{**

**int choice;**

**while (1)**

**{**

**printf("1.Insert element to queue \n");**

**printf("2.Delete element from queue \n");**

**printf("3.Display all elements of queue \n");**

**printf("4.Quit \n");**

**printf("Enter your choice : ");**

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

**switch (choice)**

**{**

**case 1:**

**insert();**

**break;**

**case 2:**

**delete();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**exit(1);**

**default:**

**printf("Wrong choice \n");**

**} /\*End of switch\*/**

**} /\*End of while\*/**

**} /\*End of main()\*/**

**insert()**

**{**

**int add\_item;**

**if (rear == MAX - 1)**

**printf("Queue Overflow \n");**

**else**

**{**

**if (front == - 1)**

**/\*If queue is initially empty \*/**

**front = 0;**

**printf("Inset the element in queue : ");**

**scanf("%d", &add\_item);**

**rear = rear + 1;**

**queue\_array[rear] = add\_item;**

**}**

**} /\*End of insert()\*/**

**delete()**

**{**

**if (front == - 1 || front > rear)**

**{**

**printf("Queue Underflow \n");**

**return ;**

**}**

**else**

**{**

**printf("Element deleted from queue is : %d\n", queue\_array[front]);**

**front = front + 1;**

**}**

**} /\*End of delete() \*/**

**display()**

**{**

**int i;**

**if (front == - 1 || front > rear)**

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

**else**

**{**

**printf("Queue is : \n");**

**for (i = front; i <= rear; i++)**

**printf("%d ", queue\_array[i]);**

**printf("\n");**

**}**

**} /\*End of display() \*/**

**OUTPUT:**

**PROGRAM-2: C PROGRAM TO IMPLEMENT A CIRCULAR QUEUE USING AN ARRAY.**

**#include<stdio.h>**

**#include<conio.h>**

**#define SIZE 5**

**void insert();**

**void delet();**

**void display();**

**int queue[SIZE], rear=-1, front=-1, item;**

**void main()**

**{**

**int ch;**

**do**

**{**

**printf("\n\n1.\tInsert\n2.\tDelete\n3.\tDisplay\n4.\tExit\n");**

**printf("\nEnter your choice: ");**

**scanf("%d", &ch);**

**switch(ch)**

**{**

**case 1:**

**insert();**

**break;**

**case 2:**

**delet();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**exit(0);**

**default:**

**printf("\n\nInvalid choice. Pleasr try again...\n");**

**}**

**} while(1);**

**// getch();**

**}**

**void insert()**

**{**

**if((front==0 && rear==SIZE-1) || (front==rear+1))**

**printf("\n\nQueue is full.");**

**else**

**{**

**printf("\n\nEnter ITEM: ");**

**scanf("%d", &item);**

**if(rear == -1)**

**{**

**rear = 0;**

**front = 0;**

**}**

**else if(rear == SIZE-1)**

**rear = (rear+1)%SIZE;**

**else**

**rear++;**

**queue[rear] = item;**

**printf("\n\nItem inserted: %d\n", item);**

**}**

**}**

**void delet()**

**{**

**if(front == -1)**

**printf("\n\nQueue is empty.\n");**

**else**

**{**

**item = queue[front];**

**if(front == rear)**

**{**

**front = -1;**

**rear = -1;**

**}**

**else if(front == SIZE-1)**

**front = (front+1)%SIZE;**

**else**

**front++;**

**printf("\n\nITEM deleted: %d", item);**

**}**

**}**

**void display()**

**{**

**int i;**

**if((front == -1) || (front==rear+1))**

**printf("\n\nQueue is empty.\n");**

**else**

**{**

**printf("\n\n");**

**for(i=front; i<=rear; i++)**

**printf("\t%d",queue[i]);**

**}**

**}**

**OUTPUT:**

**PROGRAM-3: C PROGRAM TO IMPLEMENT A PRIORITY QUEUE USING AN ARRAY.**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX 5**

**void insert\_by\_priority(int);**

**void delete\_by\_priority(int);**

**void create();**

**void check(int);**

**void display\_pqueue();**

**int pri\_que[MAX];**

**int front, rear;**

**void main()**

**{**

**int n, ch;**

**printf("\n1 - Insert an element into queue");**

**printf("\n2 - Delete an element from queue");**

**printf("\n3 - Display queue elements");**

**printf("\n4 - Exit");**

**create();**

**while (1)**

**{**

**printf("\nEnter your choice : ");**

**scanf("%d", &ch);**

**switch (ch)**

**{**

**case 1:**

**printf("\nEnter value to be inserted : ");**

**scanf("%d",&n);**

**insert\_by\_priority(n);**

**break;**

**case 2:**

**printf("\nEnter value to delete : ");**

**scanf("%d",&n);**

**delete\_by\_priority(n);**

**break;**

**case 3:**

**display\_pqueue();**

**break;**

**case 4:**

**exit(0);**

**default:**

**printf("\nChoice is incorrect, Enter a correct choice");**

**}**

**}**

**}**

**/\* Function to create an empty priority queue \*/**

**void create()**

**{**

**front = rear = -1;**

**}**

**/\* Function to insert value into priority queue \*/**

**void insert\_by\_priority(int data)**

**{**

**if (rear >= MAX - 1)**

**{**

**printf("\nQueue overflow no more elements can be inserted");**

**return;**

**}**

**if ((front == -1) && (rear == -1))**

**{**

**front++;**

**rear++;**

**pri\_que[rear] = data;**

**return;**

**}**

**else**

**check(data);**

**rear++;**

**}**

**/\* Function to check priority and place element \*/**

**void check(int data)**

**{**

**int i,j;**

**for (i = 0; i <= rear; i++)**

**{**

**if (data >= pri\_que[i])**

**{**

**for (j = rear + 1; j > i; j--)**

**{**

**pri\_que[j] = pri\_que[j - 1];**

**}**

**pri\_que[i] = data;**

**return;**

**}**

**}**

**pri\_que[i] = data;**

**}**

**/\* Function to delete an element from queue \*/**

**void delete\_by\_priority(int data)**

**{**

**int i;**

**if ((front==-1) && (rear==-1))**

**{**

**printf("\nQueue is empty no elements to delete");**

**return;**

**}**

**for (i = 0; i <= rear; i++)**

**{**

**if (data == pri\_que[i])**

**{**

**for (; i < rear; i++)**

**{**

**pri\_que[i] = pri\_que[i + 1];**

**}**

**pri\_que[i] = -99;**

**rear--;**

**if (rear == -1)**

**front = -1;**

**return;**

**}**

**}**

**printf("\n%d not found in queue to delete", data);**

**}**

**/\* Function to display queue elements \*/**

**void display\_pqueue()**

**{**

**if ((front == -1) && (rear == -1))**

**{**

**printf("\nQueue is empty");**

**return;**

**}**

**for (; front <= rear; front++)**

**{**

**printf(" %d ", pri\_que[front]);**

**}**

**front = 0;**

**}**

**OUTPUT:**