ICT-2101
Data Structure

Lecture 07

QUEUE

INTRODUCTION

- A queue is a linear list of elements in which deletion can take place only at one end, called the front, and insertions can take place only at the other end, called the rear.

- The term “front” and “rear” are used in describing a linear list only when it is implemented as a queue.

Implement a queue

- **There are main two ways to implement a queue :**
  - 1. Circular queue using array
  - 2. Linked Structures (Pointers)

- When a queue is implemented using array, that queue can organize only limited number of elements.

- When a queue is implemented using linked list, that queue can organize unlimited number of elements

Operations

- **Primary queue operations:**
  - Enqueue: insert an element at the rear of the queue
  - Dequeue: remove an element from the front of the queue

Insertion in Queue

```
Algorithm:  ENQUEUE(QUEUE, MAXSIZE, FRONT, REAR,COUNT, ITEM)
              This algorithm inserts an element ITEM into a circular queue.
       1.  [QUEUE already filled?]
           If COUNT = MAXSIZE then:   [ COUNT is number of values in the QUEUE]
              Write: OVERFLOW, and Return.
       2.  [Find new value of REAR.]
           If COUNT= 0, then:       [Queue initially empty.]
              Set FRONT= 0  and REAR = 0
           Else: if REAR = MAXSIZE - 1, then:
              Set REAR = 0
           Else:
              Set REAR = REAR+1.
           [End of If Structure.]
       3.  Set QUEUE[REAR] = ITEM.     [This insert new element.]
       4.  COUNT=COUNT+1          [ Increment to Counter.   ]
       5.  Return.
```

Deletion in Queue

```
Algorithm:  DEQUEUE(QUEUE, MAXSIZE, FRONT,  REAR,COUNT, ITEM)
            This procedure deletes an element from a queue and assigns it to the
            variable ITEM.
    1. [QUEUE already empty?]
       If COUNT= 0, then: Write: UNDERFLOW, and Return.
    2. Set ITEM = QUEUE[FRONT].
    3. Set COUNT = COUNT -1
    4. [Find new value of FRONT.]
            If COUNT = 0, then: [There was one element and has been deleted ]
                    Set FRONT= -1, and REAR = -1.
            Else if FRONT= MAXSIZE, then: [Circular, so set Front = 0 ]
                    Set FRONT = 0
            Else:
                    Set FRONT:=FRONT+1.
            [End of If structure.]

    5. Return ITEM
```

Queue Example

- Following Figure shows that how a queue may be maintained by a circular array with **MAXSIZE = 6 (Six memory locations).**
- **Observe that queue always occupies** consecutive locations except when it occupies locations at the beginning and at the end of the array.
- If the queue is viewed as a circular array, this means that it still occupies consecutive locations. Also, as indicated by **Fig(k), the queue will be empty** only when **Count = 0 or (Front = Rear but not null) and an element is deleted.**
- **For** this reason, **-1 (null) is assigned to Front and Rear.**

MaxSize = 6

(a) Initially QUEUE is Empty

Front = -1
Rear = -1
Count = 0

0	1	2	3	4	5

(b) A, B, C are Enqueued / Inserted

Front = 0
Rear = 2
Count = 3

A	B	C			
0	1	2	3	4	5

(c) A is Deleted / Dequeue

Front = 1
Rear = 2
Count = 2

	B	C			
0	1	2	3	4	5

(d) D, E, F are Enqueued / Inserted

Front = 1
Rear = 5
Count = 5

	B	C	D	E	F
0	1	2	3	4	5

(e) B and C are Deleted / Dequeue

Front = 3
Rear = 5
Count = 3

			D	E	F
0	1	2	3	4	5

(f) G is Enqueued / Inserted

Front = 3
Rear = 0
Count = 4

G			D	E	F
0	1	2	3	4	5

(g) D and E are Deleted / Dequeue

Front = 5
Rear = 0
Count = 2

G					F
0	1	2	3	4	5

(h) H and I are Enqueued / Inserted

Front = 5
Rear = 2
Count = 4

G	H	I			F
0	1	2	3	4	5

(i) F is Deleted / Dequeue

Front = 0
Rear = 2
Count = 3

G	H	I			
0	1	2	3	4	5

(j) G and H are Deleted / Dequeue

Front = 2
Rear = 2
Count = 1

		I			
0	1	2	3	4	5

(k) I is Deleted. Queue is *Empty*

Front = -1
Rear = -1
Count = 0

0	1	2	3	4	5

Implement Circular QUEUE using array

```
#include<iostream.h>
#include <process.h>
#define MAXSIZE 10    // int const  MAXSIZE = 10;

//   Global  declarations and available to every
      int Queue[MAXSIZE];
      int front = -1;
      int rear  = -1;
      int count =0;

bool IsEmpty(){if(count==0)return true; else return false; }

bool IsFull() { if( count== MAXSIZE) return true; else return false;}

void Enqueue(int ITEM)
{     if(IsFull()) { cout<< "\n QUEUE is full\n"; return;}

      if(count == 0) rear = front= 0;  //  first item to enqueue
      else
      if(rear == MAXSIZE -1) rear=0 ;  // Circular, rear set to zero
      else rear++;

      Queue[rear]=ITEM;
      count++;
}
```

Implement Circular QUEUE using array

```
int Dequeue()
{
        if(IsEmpty()) { cout<<"\n\nQUEUE is empty\n"; return -1; }

        int ITEM= Queue[front];
        count--;

        if(count == 0 ) front = rear = -1;
        else if(front == MAXSIZE -1) front=0;
        else front++;

        return ITEM;
}

void Traverse()
        { int i;
          if(IsEmpty()) cout<<"\n\nQUEUE is empty\n";
          else
            { i = front;
              While(1)
               { cout<< Queue[i]<<"\t";
                 if (i == rear) break;
                  else if(i == MAXSIZE -1) i = 0;
                  else i++;
               }
            }
        }
```

Iimplement Circular QUEUE using array

```
int main()
{
  int choice,ITEM;
  while(1)
      {
        cout<<"\n\n\n\n QUEUE operation\n\n";
        cout<<"1-insert value \n 2-deleted value\n";
        cout<<"3-Traverse QUEUE \n 4-exit\n\n";
        cout<<"\t\t your choice:"; cin>>choice;

      switch(choice)
            {
      case 1:
                          cout"\n put a value:";
                          cin>>ITEM);
                          Enqueue(ITEM);break;
      case 2:
                    ITEM=Dequeue();
                    if(ITEM!=-1)cout<<t<<  " deleted \n";
                    break;
      case 3:
                    cout<<"\n queue state\n";
                    Traverse(); break;

      case 4:exit(0);
             }
      }
            return 0;
      }
```

Queue using Linked List

- In linked list implementation of a queue, the last inserted node is always pointed by '**rear**' and the first node is always pointed by '**front**'.
- To implement queue using 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 two **Node** pointers '**front**' and '**rear**' and set both to **NULL.**

Step 4: Implement the **main** method by displaying Menu of list of operations and make suitable function calls in the **main** method to perform user selected operation.

Queue using Linked List

- **enQueue(value) - Inserting an element into the Queue:**

Step 1: Create a **newNode** with given value and set '**newNode** → **next**' to **NULL**.

Step 2: Check whether queue is **Empty** (**rear** == **NULL**)

Step 3: If it is **Empty** then, set **front** = **newNode** and **rear** = **newNode**.

Step 4: If it is **Not Empty** then, set **rear** → **next** = **newNode** and **rear** = **newNode**.

Queue using Linked List

- **deQueue() - Deleting an Element from Queue:**

Step 1: Check whether **queue** is **Empty** (**front == NULL**).

Step 2: If it is **Empty**, then display "**Queue is Empty!!! Deletion is not possible!!!**" and terminate from the function

Step 3: If it is **Not Empty** then, define a Node pointer '**temp**' and set it to '**front**'.

Step 4: Then set '**front** = **front** → **next**' and delete '**temp**' (**free(temp)**).

Queue using Linked List

- **display() - Displaying the elements of Queue:**

Step 1: Check whether queue is **Empty (front == 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 **'rear'** (**temp → next** != **NULL**).

Step 4: Finally! Display **'temp → data** ---> **NULL'**.

Program for Queue Using Linked List

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

struct Node
{
    int data;
    struct Node *next;
}*front = NULL,*rear = NULL;

void insert(int);
void delete();
void display();
void main()
{
   int choice, value;
   clrscr();
   printf("\n:: Queue Implementation using Linked List ::\n");
   while(1){
      printf("\n****** MENU ******\n");
      printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d",&choice);
      switch(choice){
         case 1: printf("Enter the value to be insert: ");
                 scanf("%d", &value);
                 insert(value);
                 break;
         case 2: delete(); break;
         case 3: display(); break;
         case 4: exit(0);
         default: printf("\nWrong selection!!! Please try again!!!\n");
      }
   }
}
```

Program for Queue Using Linked List

```c
void insert(int value)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode -> next = NULL;
   if(front == NULL)
      front = rear = newNode;
   else{
      rear -> next = newNode;
      rear = newNode;
   }
   printf("\nInsertion is Success!!!\n");
}
void delete()
{
   if(front == NULL)
      printf("\nQueue is Empty!!!\n");
   else{
      struct Node *temp = front;
      front = front -> next;
      printf("\nDeleted element: %d\n", temp->data);
      free(temp);
   }
}
```

Program for Queue Using Linked List

```c
void display()
{
    if(front == NULL)
        printf("\nQueue is Empty!!!\n");
    else{
        struct Node *temp = front;
        while(temp->next != NULL){
            printf("%d--->",temp->data);
            temp = temp -> next;
        }
        printf("%d--->NULL\n",temp->data);
    }
}
```

Thank you