Lecture11

This lecture we learn about
- Queue
- Representation of Queue using Arrays
- Representation of Queue using Linked List

Queue

We come across queues in out everyday life, when was the last time you were standing in queue and waiting for your turn to come?

Queue is a datastructure, which is based on the principle of FIFO. FIFO means First In First Out, i.e. Element first inserted is the first one to come out.

Queue is extensively used in Compuer Science. If you have studied Operating System, we use queue there for scheduling of processes. Like this, there are many instances where we can use queue.

Definition of Queue

Queue is a datastructure, which is based on the principle of FIFO. FIFO means First In First Out.
There are 2 variables front and rear, front points to the first element in the queue and rear points to the last element in the queue.

We define 2 operations on Queue, Insert and delete.
-> Insert means to insert element in the end of the queue. To insert an element we increment rear. Then insert an element at the position rear.
-> Delete means to delete an element from the beginning of the queue. Get the element from the position front. Then increment front.

Representation of Queue using Array

Watch the presentation working of queue. After watching presentation, we can summarize it as follows:

- In an empty queue, front=-1 and rear-1. Or front=rear+1
- If there is only one element in the queue, both front and rear point to that element. i.e. front=rear.
- In a Full queue, rear is equal to the size of array.

We assume that, we have an array 'a' of integer of size 10. We will represent 'a' as a queue.

```
int a[10]; // array a which we will implement as queue
int data; // variable data for storing the value entered by user.
```

Step1: Declare varaibles, front, rear, flag.
Since queue is empty, i.e. There are no elements in the linked list, therefore

```
int front = -1;
int rear = -1;
```

We use a variable flag.
flag = 1, means queue is empty.
flag = 0, means queue is not empty.

```
int flag = 0;
```

Step2: Write function insert(). In function insert, Check if queue is full or not.

```
        boolean queue_full()
        {
                if (rear == 9)
                        return 1;
                else
                        return 0;
        }

        void insert()
        {
                If (queue_full())
                {
                        printf("cannot insert, queue is full");
                }
```

Step3: if flag=1, that means we are inserting for the first time.
If we are inserting for the first time, then increment both front and rear.
Ask user to enter a data value.
Insert the data value entered by user at position rear.
Set flag to 0, because queue is no longer empty.

```
        else // queue is not full
        {
                if (flag == 0) // means queue is empty,
                {
                        front = front + 1;
                        rear = rear + 1;
                        flag = 1; // set flag = 1, which means queue is not empty anymore.
                }
                else  // queue is not empty,
                        rear = rear + 1;

                printf("Enter value");
                scanf("%d", &data);
                a[rear]=data;
```

```
    } // end of outer if

}// end of function insert()
```

Step4: Write the code for function delete(), But before deletion,we need to check if queue is empty because we cannot delete from an empty queue.

```
        boolean empty_queue()
        {
            if (front==-1 && rear==-1 || front == rear + 1)
                return 1;
            else
                return 0;
        }

        void delete()
        {
            if (empty_queue())
                printf(“cannot delete, queue empty”);
```

Step5: If queue is not empty, we can delete an element. To delete an element, get an element from the position front and then increment front.

```
            else // queue is not empty
            {
                x = a[front];
                front = front + 1;
                printf(“Element deleted is %d”, x);
            }

        } // end of function delete
```

Step6: The last function of this program, code for function print().
We know that front points to the first element in the queue and rear points to the last element in the queue.
To print all the elements of the queue, we need to start a loop starting from front until rear.

```
        If (queue_empty())
            printf(“queue is empty”);
        else
        {
            for(i=front;i<=rear;i++)
                    printf(“%d”, a[i]);
        }
```

Q. What is the major disadvantage of linear queue?
A. Once rear reaches the end of the array, we can no longer insert an element in queue.

There is workaround for the above drawback, we can treat a queue as a circular queue, means, once rear reaches the end of array, next increment makes rear 0.
Same for front, once front reaches the end of array, next increment will makes front 0.

Watch the presentation for circular queue.

We also make some more changes in insert and delete function.

When we insert an element, we increment rear by 1. Now, we simply donot increment, we use rear = (rear+1) % size_of_array.
In this ex, rear = (rear+1) % 10. This ensures rear takes valur between 0 and 9.

if rear=0, Then rear = (rear + 1) % 10, makes rear=1.
If rear=1, Then rear = (rear + 1) % 10, makes rear=2.
If rear=8, Then rear = (rear + 1) % 10, makes rear=9.
If rear=9, Then rear = (rear + 1) % 10, makes rear=0.

After incrementing rear, we insert data at the position rear.

When we delete an element from queue, we check if we are deleting last element.
How can we check if we are deleting last element in the queue?
If (front == rear), that means queue only consist of one element.

So, if we are deleting the last element, get the value from position rear,then set front=-1 and rear=-1.

If we are not deleting last element, display the value from position front, and increment front by 1.
To increment front use front = (front + 1) % 10.

Watch the presentation for working of circular queue. After watching presentation, we can summarize it as follows

- Condition for empty circular queue
  (front=-1 and rear=-1)

- Condition for circular queue full
  front == (rear + 1) % size_of_array

- Condition for only one element in the queue
  front == queue.

Representation of Circular queue using array

We assume that, we have an array 'a' of integer of size 10. We will represent 'a' as a queue.

```
        int a[10]; // array a which we will implement as queue
        int data; // variable data for storing the value entered by user.
```

Step1: Declare varaibles, front, rear, flag.
Since queue is empty, i.e. There are no elements in the linked list, therefore

```
int front = -1;
int rear = -1;
```

We use a variable flag.
flag = 1, means queue is empty.
flag = 0, means queue is not empty.

```
int flag = 0;
```

Step2: Write function insert(). In function insert, Check if queue is full or not. If queue is full, we display message, cannot insert. If queue is not full, we insert an element.

```
        boolean queue_full()
        {
                if (rear == 9)
                        return 1;
                else
                        return 0;
        }

        void insert()
        {
                If (queue_full())
                {
                        printf("cannot insert, queue is full");
                }
```

Step3: if flag=1, that means we are inserting for the first time.
If we are inserting for the first time, then increment both front and rear.
Ask user to enter a data value.
Insert the data value entered by user at position rear.
Set flag to 0, because queue is no longer empty.

```
        else // queue is not full
        {
                if (flag == 0) // means queue is empty,
                {
                        front = (front + 1) % 10;
                        rear = (rear + 1) % 10;
                        flag = 1; // set flag = 1, which means queue is not empty anymore.
                }
                else  // queue is not empty,
                        rear = (rear + 1) % 10;

                printf("Enter value");
                scanf("%d", &data);
                a[rear]=data;
```

Step4: Write the code for function delete(), But before deletion,we need to check if queue is empty because we cannot delete from an empty queue.

```
        boolean empty_queue()
        {
                if (front==-1 && rear==-1)
                        return 1;
                else
                        return 0;
        }

        void delete()
        {
                if (empty_queue())
                        printf("cannot delete, queue empty");
```

Step5: If queue is not empty, then check if we are deleting the last element. If we are deleting the last element, then after deletion, set front=-1 and rear=-1.

If there are more than one element in the queue, then get the value from position front and increment front.

```
                else // queue is not empty
                {
                        if (front == rear) // there is only one element
                        {
                                x = a[front];
                                front = -1;
                                rear = -1;
                                flag = 0;
                        }
                        else // there are more than one element
                        {
                                x = a[front];
                                front = (front + 1) % 10;

                        } // end of inner if-else
                } // end of outer if-else
        } // end of function
```

Step6: Last step is printing all the elements of the queue,

```
void display()
{
        if(front==-1&&rear==-1)
                printf("Queue is empty");
        else
        {
                if(front>rear)
                        for(i=rear;i<=front;i=(i+1)%9)
                                printf("%d", a[i]);
                else
                        for(i=front;i<=rear;i=(i+1)%9)
                                printf("%d", a[i]);
        }
}
```