Programming and Data Structures
Prof. N.S. Narayanaswamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

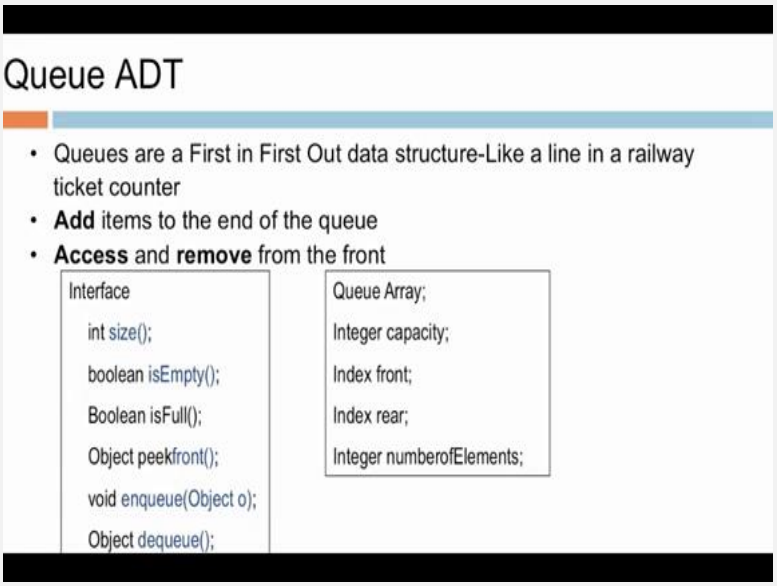
Lecture - 12
Queue ADT

Welcome to this 5th week lecture on the Queue Abstract Data Type. So, recall that so far we have looked at list, doubly link list, we implementing stacks, we done examples of how useful stacks are and in today's lecture we are going to look at the queue data type. And we will see as we go through this course, multiple applications of the queue data type. But in the sequence of lectures this week which are three short lectures, we will essentially explore the queue data type.

We will do one programming application, we will look at the implementation of the queue data type methods, and then we will have another short presentation on the different types of queues that we typically implement. The reason for the last presentation will be that we will set up a programming exercise which will be, in which you will be expected to implement certain features of the different types of the queues that we talk about.

So, that is, therefore today's presentation is going to be about the queue abstract data type, and we will also look at the implementation of the queue abstract data type; that is the focus of today's lecture. It is going to be a short above 20 minutes lecture.

(Refer Slide Time: 01:33)



So, what is the queue abstract data type? So, a queue is a first in first out data structure. It is like a line in a railway ticket counter and are those of you have not seen railway ticket counters in this generation, it is like a line in a movie counter, where you buy tickets and if you really have not seen need to movie counters. And if you got tickets online, then you can imagine a queue to get into a metro or a queue to get into a bus or queue to get to the examination hall or something like that.

So, it is essentially a first in first out data structure, the person who gets into the queue first get service first and leaves the data structure or leaves the queue first. So, this is quite an obvious name for this kind of a data structure. We are going to see what kind of data we would like to manipulate in a queue like fashion in this lecture. So, let us just look at the methods that are associated with the queue abstract data type. So, sure enough the main operations that we would want to or we would like to add elements to the end of the queue.

We would like to access the element at the head of the queue and also we may want to remove the elements from the front of the queue, these are three main operations that immediately come to mind, when one thinks of a queue and in particular, we will also organize data in a queue like fashion and service that. We will also see many applications of the queue abstract data type in these three lectures this week. Let us look at the data items in the different methods.

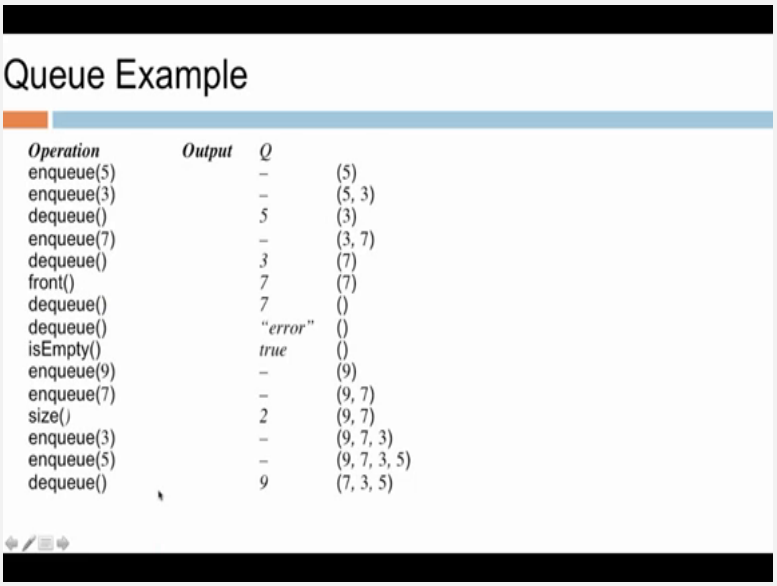
So, first of all if you think of a queue in which data is organized in a queue like fashion, then one thinks an array which contains the whole queue. The array has a certain capacity, this is an integer value variable, then we have an index into the array which we refer to as the front and an index into the array which is the rear. Now, depending on the implementation, index could either be a pointer or it can be an integer value.

We also keep track of how many elements are there in the queue by this data item which is the number of elements. So, clearly these are the kind of queries, these are the kind of data item that we would be interested, we would be interested into the value of the queue or the set of people or the set of data items in the queue, we will be interested then how large queue we can maintain, we are interested in accessing the front of the queue and to insert element into the queue, we would like to have access to the rear end of the queue and we have an integer which represents the total number of elements.

Now, the interface methods are the following, the size method is a query which returns an integer value. Of course, it takes as an argument queue and many of these functions and in these methods have deliberately left out the fact that the queue will always be an argument. So, the argument to this particular method is the queue and the return value is the total number of elements which are there in the queue.

The next query is whether the queue is empty, whether it is full, the names immediately suggest that these are two Boolean value function that return whether the queue which is passes an argument is empty or not. Peek front is an interesting function, you just look at the element at the top of the queue and we obtain the value, this has it is own purposes and we will see what we can do with this particular function.

(Refer Slide Time: 05:16)



This is very much like the peek function which was there with the stack abstract data type. Enqueue is an interesting function, this is very fundamental central functions to the queue data type, we insert an object into the queue and the argument is missing, so in a particular queue the object o has been inserted into the rear end of the queue and of course, this operation can fail and we will see how to implement that, so that we get appropriate error messages.

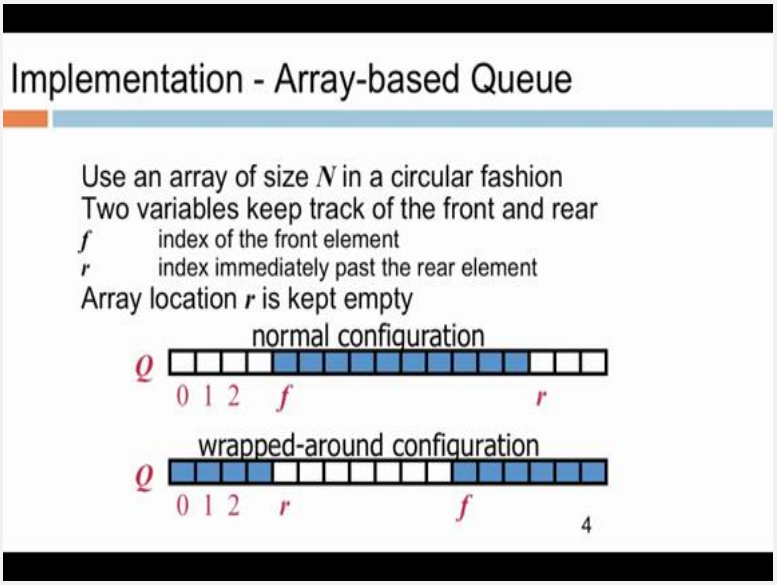
Object dequeue, this method returns the object from the head of the queue and the end result is that it also removes the element from the queue. So, this is initially describes the interface to the queue data type and these are the data items that we maintained. Let us run an example, queue is a queue in question and the operation that we perform or the method that is called is enqueue 5, then 5 is inserted into the queue. When you enqueue 3, then 3 is inserted the rear end of the queue.

Now, if you dequeue from queue when you get the value 5 and what is left at the queue is only 3. If you enqueue 7 now, then the queue now has the elements 3 and 7, 3 is at the front and 7 as the rear. Now, when you dequeue 3 is value that is returned and 7 is what is on the queue, front. So, there is a small error here, this is peek front. When you ask for the value 7, when you look at the front of the queue you, the return value is 7, but the queue does not change. This is the peek front function, there is small missing word here, it must be peek front.

Now, dequeue will return the value 7 and remove it from the queue itself. Now, if you dequeue an error messages printed, if you ask the question is queue empty, then the answer is true. Now, when enqueue is 9, 9 enters the queue, then 7 enters the queue, now when you ask for the size of the queue the return value is 2. Now, when you enqueue 3, 3 enters the queue, then 5 enters the queue, now when you dequeue, 9 is return which is the head of the queue and 7, 3, 5 is a condensed queue.

So, essentially these are the kind of transactions that are enable by the interface functions that we have for the queue. The one function that we really did not talk about was the constructor function or the function that sets of the queue and we will see it in the implementation.

(Refer Slide Time: 07:57)



So, now let us look at the implementation of an array based queue. So, let us assume that the queue has n elements, we are going to maintain the queue in a circular fashion and the circular fashion is very important, we will see this again in a couple of lectures time. But, conceptually it is very easy. We have two variables that keep track of the front and rear. As you can see, the array is indexed from 0, 1, 2 onwards up to n, the shaded portion is the relevant part of the queue. That is where the data items are and f is the front element to the queue and r is the rear element of the queue.

Now, you can see that f is not necessarily 0, now you can imagine why this has happened. So, the reason of elements 0, 1, 2 onwards up to the current f have been

dequeued or eliminated from the queue. Now, to be able to check whether the queue is empty or not, because it is a circular queue, we use one location to check that r is empty. So, therefore there is one location which is a marker which will tell us whether the queue is empty or not.

Now, let us just look at the wrapped around configuration of the queue. So, the normal configuration is that the queue is present inside the array, in the wrapped around configuration the front element of the queue is here, then the next element is just behind it and so on. The last element of the array is here and the next element is here and the last element of the queue is at the location indexed by 3 in this array queue and as you can see r is kept empty.

So, essentially you can imagine the queue to be a circular queue and there is a well defined marker us to, what the front of the queue is and what the rear of the queue is. Innovate this is an efficient utilization of this space, so for example, if you did not have this wrapped around configuration, observe that after you dequeue sum of the elements, so much spaces either wasted or the front of the queue has to be moved to the 0th element, in either case there is a certain loss.

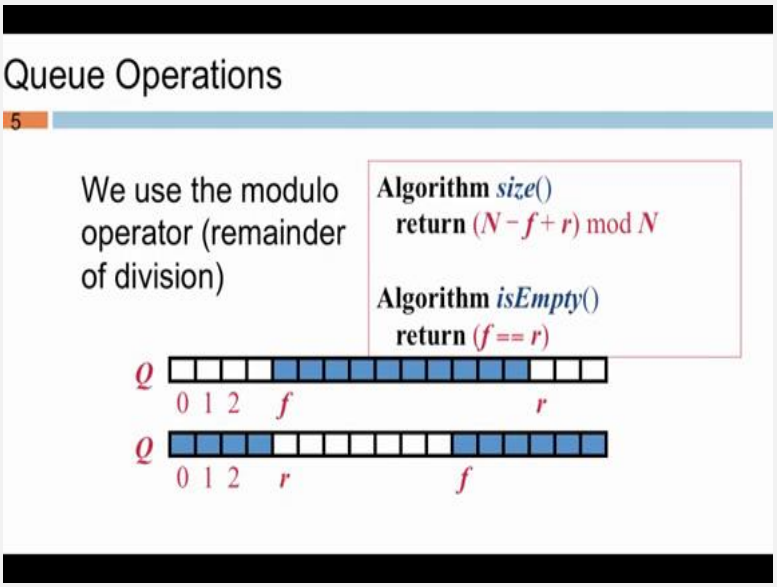
If you have to adjust the queue, so that the front of the queue comes to the 0th location, then the program has to do a lot of reorganization work of the queue. This is what happens in real queues or physical queues in the world where people standard in lines, whenever the front of the queue, an element leaves a front of the queue, the element just behind it comes to the head and everybody else moves forward.

But, in a programming environment when a program is executing such an automatic feature does not happen, the elements have to be moved to the front. The other way of dealing with it is to keep a circular queue. Now for example, observe that if in the normal configuration if three more elements come, then they will be inserted into these three locations and the 0th location will become the rear of the queue.

And the meanwhile, if a few elements are dequeued like you see here and then four elements come to the rear of the queue that is in a wrapped around configuration, this is the kind of picture that you get. So, a circular queue actually is a better utilization of the queue data structure, space is not wasted just the array index arithmetic is carefully done to get a lot of benefit from the fact that the queue is organized in an array.

The natural limitation that you can see is that the size of the queue is limited by the size of the array, unless you can add a few more locations as and when necessary that depends upon the programming language in which you are implementing the queue.

(Refer Slide Time: 12:08)



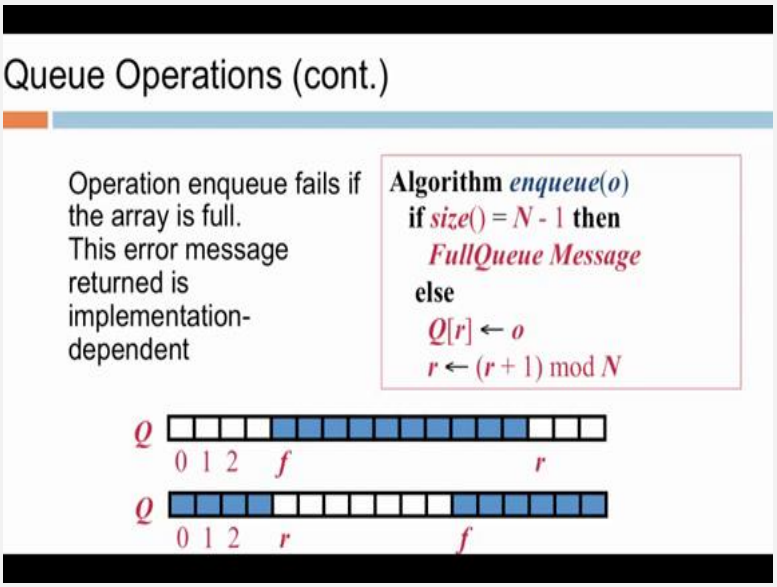
So, now let us look at the queue operations, the queue operations are to return the size of the... Let us look at the function which returns the size of the queue. So, this is n minus the front of the queue plus reverse of the queue modulo n. What do I mean by modulo? Modulo is the remainder of the division that is you take the values the total n. Now, you look at r minus f, now you look at N minus r minus f, in other words I written as N minus f plus r and take the remainder modulo n.

So, this will essentially tell you how many elements are there in the array. Observe that this is very important, the modulo arithmetic is very important. In the normal case, it is very clear, the size of the array in this case is r minus f, there is no doubt about it. So, on the other hand if it is in a circular fashion, you have to take r minus f, then N minus r minus f and observe that when you do r minus f you will actually get a negative term.

Because, f is larger in this case than the value of r, then you subtract out n from it, then to modulo n that will tell you the remainder will tell you, how many elements are there in the queue that is the size. I encourage you to write, you will also look at the program which has done this and I will also argument to that program, when I make it available to you. Here is a function which checks if the queue is empty, if f is equal to r then the

queue is empty, definitely this is what the condition is when the whole array itself is empty. But, subsequently if all these elements the blue elements are all dequeued, then f will point to r and that will be an indicator that the queue is empty.

(Refer Slide Time: 14:16)



Let us look at the enqueue operation, enqueue operation says that if there should have been a double equal to, but it does not matter. If the size of the array is n minus 1, then you say that the queue is full. Otherwise, you look at the rear element, insert the object into the rear element that is in this location and then increment the value of r by r plus 1 modulo n. This is the crucial thing that is when you do modulo, when you come here and then increment it to 2 n plus 1 modulo n the remainder will point to the first location.

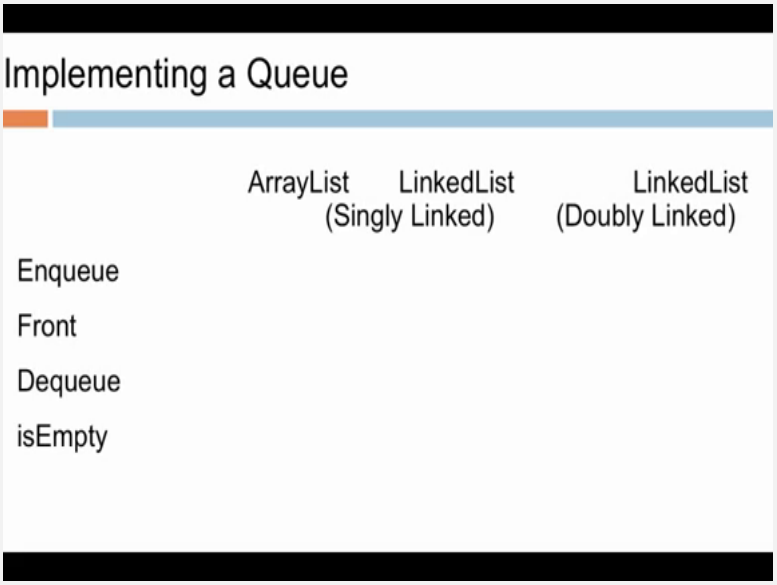
So, this is the enqueue algorithm, the dequeue algorithm is to update the value of f. So, when you remove an element from the queue, you return the value queue of f that is the fth location that is whatever is at the front and you increment f to f plus 1 modulo n, again modulo means remainder mod n remainder after dividing by f. So, before you write programs please ensure that you perform this array arithmetic by hand on a sheet of paper, so that you become comfortable with this way of indexing into a circular array.

You look at the array is a linear array, but as a programmer you are using the power of division to actually treat it as it was a circular object. Therefore, you should not think that the array is a circular array kind of a thing, it is a meaningless. The array is essentially a sequence of contiguous locations that is a sequence of locations in memory

which have consecutive addresses and it is the arithmetic by the program which treats that linear array in a circular fashion.

So, those were really the methods which are important to us. So, we looked at the implementation of enqueue and dequeue to find out the size, what the size of the queue is and whether the queue is empty or not.

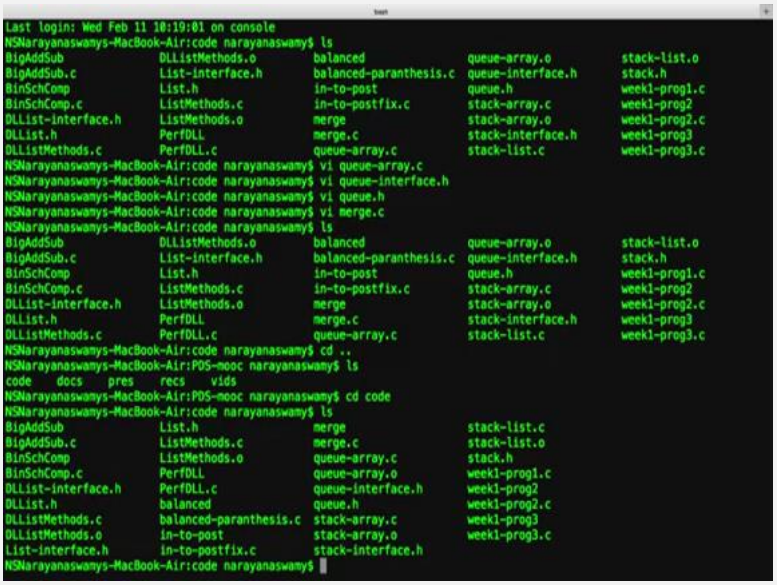
(Refer Slide Time: 16:30)



It is very important for us to do this small homework of trying to understand, how many operations are taken when the queue is maintained as a singly link list, as an array and a doubly link list. So, this is left as a small exercise for you to experiment and fill up the entries in this matrix. Of course, the entries of the matrix are not shown to you, but you can see there are three columns here, classified into singly link list and doubly link list and the functions are enqueue, front, dequeue and isEmpty.

And you can calculate how many operations are performed by the enqueue, dequeue, front and isEmpty functions before answering this questions when the queue is implemented in the array, when the queue is implemented in singly link list and as a doubly link list, this is left as a small exercise. Let us go to our program now.

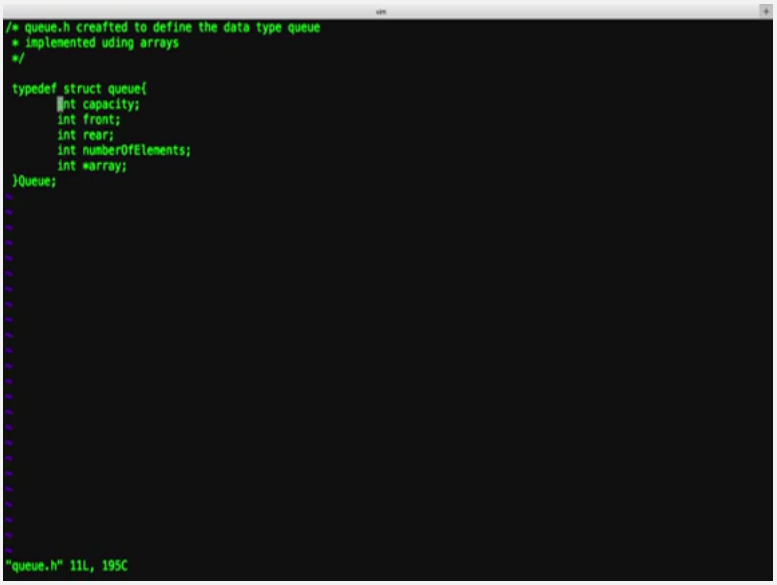
(Refer Slide Time: 17:34)



As you can see, I am also followed some additional discipline, you would notice that the directories are well better organized than the earlier lectures. So, I have organize the presentation here into multiple directories, this contains all the code, this contains all the documents which are important, this contains a presentation that I making, this contains video recordings, this contains videos which are uploaded onto you tube, so that you can see them.

And the code is present inside this directory called code and the new additions here are the functions queue, queue array dot c, queue interface dot h, queue dot h and I have written a merge function. In this lecture, let us just look at queue dot h and queue array dot c and queue interface dot h.

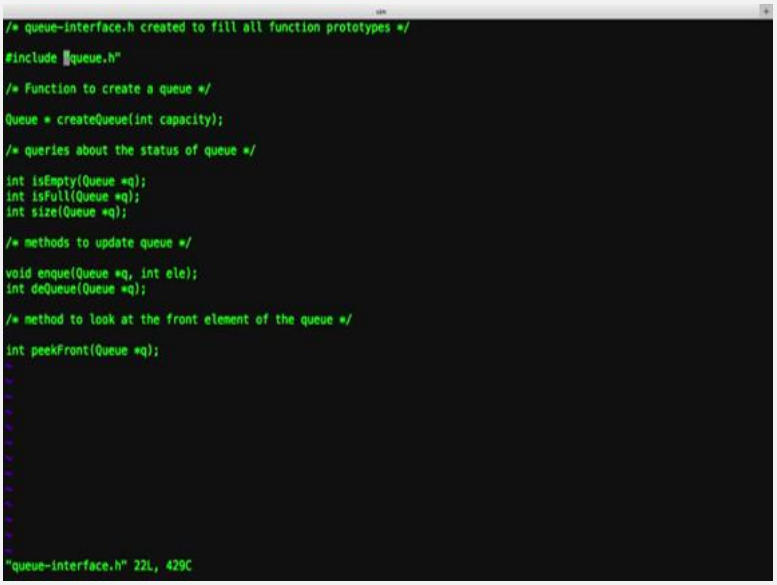
(Refer Slide Time: 18:34)



As usual queue dot h contains the basic object, basic queue object observe that it has a variable for the capacity of the queue, a pointer to the front which is an index pointer to the rear, which is an index, there is an integer. The number of elements which are present in the queue and capacity tells you how many elements you can store on the queue and this is an array.

This is a pointer to the actual queue actual array, the data items are present inside the array pointer to by this variable array and this object is called or this data type is called the queue data type. We will see how to instantiate this queue data type, whenever we need it.

(Refer Slide Time: 19:37)



But, now we need to look at the functions, which we have implemented for that let us look at the queue interface dot h, that we tell us what all the different methods, that we have implemented. So, this is the interface, the interface function includes the queue dot h header file, here is a constructed functions, the one that create a the queue, look at the argument, it is a method called create queue, that takes as an argument a certain capacity and it returns a pointer to an object of data type queue.

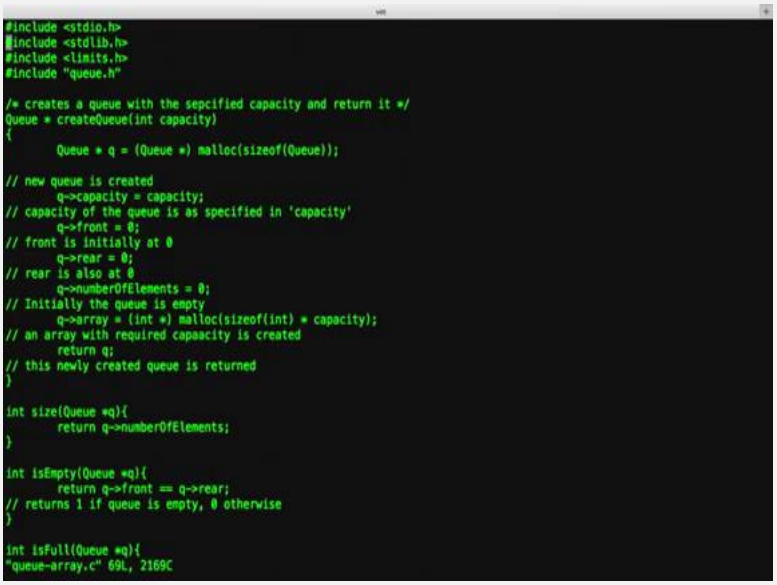
The function is empty queue, returns of Boolean value it takes pointer to queue and returns 1 or 0, if it is a depending on whether it is empty or full, this one returns 1 or 0 depending on whether it is full. Size of the queue, it was an integer value regarding the size of the queue, then we have a methods update the queue, like you said when I show to the presentation, the pointer to the queue is taken, the argument to be insert into the queue is taken and it is enqueue into the queue.

The dequeue returns an integer and it takes a pointer queue from the integer must be return, but this also removes the element from the head of the queue, observe the both is functions really do not give error messages and this is a drawback. Somehow, we can for example, use global variables to actually signal, when errors have happened, but that is not something we are going to do now.

But, I will already mention and I put the thought into let say how will actually print or communicate error messages to calling functions, when these function calls to either add

an element into the queue and delete an element from the queue fairy, we will see the implementation. Now, here is method to look at the front element to the queue, we just peek into the front of the queue, it returns the value, it will not remove this. Now, let us look at the implementation of this functions, it is going to be a very crisp look at the implementation of this functions.

(Refer Slide Time: 21:41)



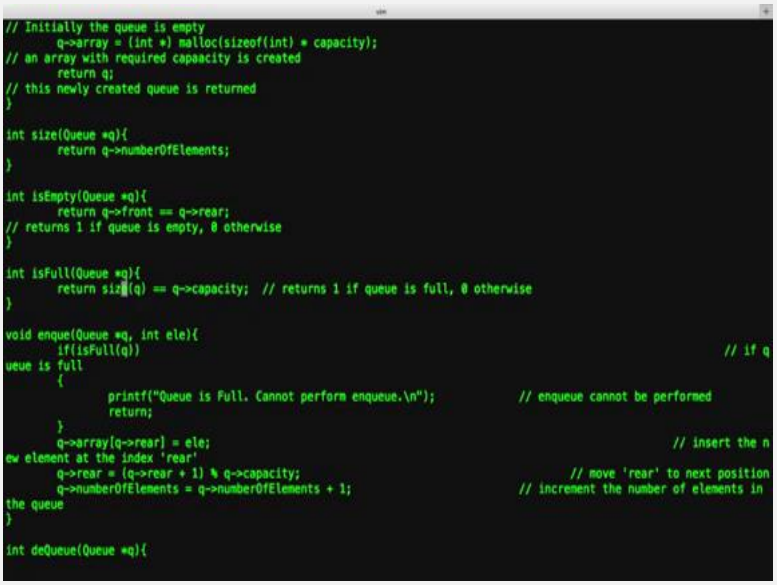
Almost a Pseudo code that we saw in the presentation is what you are going to see here, this is in the file queue array dot c, as usual stdio, stdlib and queue dot h has been inserted here. Let us just look at first method it create a queue with this specific capacity and returns a pointed to it, look at create the queue first thing that it does is, it creates a, it is allocates size of queue number of locations returns a queue.

Therefore, observe there it will create a single queue and inside the queue is an array and what does do it is queue is do it is key point is capacity is equal to capacity. And I say queue pointer dot front is 0, queue pointed dot real is also 0, it is front is equal to rear that the queue is actually empty, then queue pointer of number of element is 0 that is a number of element in the queue is 0. Now, what it do is you get the array to point to an array of integers. How many integers? Capacity number of integers.

Therefore, now after this instruction, after this statement queue pointed dot array points to in the queue the element array points to an array of capacity number of integers. Now, the array has been created, now you return the pointer that is it queue. So, now this queue

is created this is the method, now here is the size of queue, you just return queue pointer number of elements is empty, you check if queue point of front is equal to queue pointer return, if that expression is true you return 1 otherwise you return 0. When is the queue full you check when the size of the queue is exactly queue pointer dot capacity, this returns when the queue is full; otherwise, it returns the 0 note the slide def...

(Refer Slide Time: 23:48)



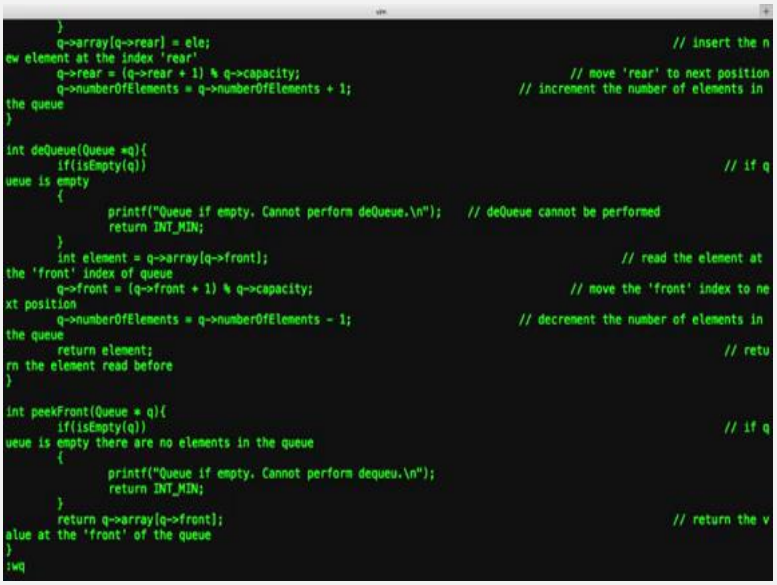
So, here we check if the if created the queue in this function and here we check if the queue is empty, we check here of the queue is full. And let us just look at the enqueue and dequeue functions, you take an element as an argument, you take a pointer to the queue you check if the queue is full, the queue is full, then we actually print the error message here remember that we just discuss this little while back and then you return.

How will in a typical programming environment printing a messages is useless exercise. So, we may want to use some global variables which will be use for other function to check if the queue is full or not, this is a very important programming practice such printf statement are very useful only for people who are looking at the execution. A good programmer or a professional programmer would actually find another way of communicating to other function that the enqueue into this particular queue has fail.

One of the ways of doing it is to have a global variable, which will store a certain value to record whether the enqueue and dequeue failed a succeeded, in this case whether the enqueue failed a succeeded. So, if the queue is not full then the control comes here, so

now, what you do is you copy the element into the rear element, then you increment the value of rear by 1. Modulo the capacity that is we take the remainder by dividing with the capacity, then you increment the number elements by 1. So, this is the enqueue procedure dequeue procedure is very similar.

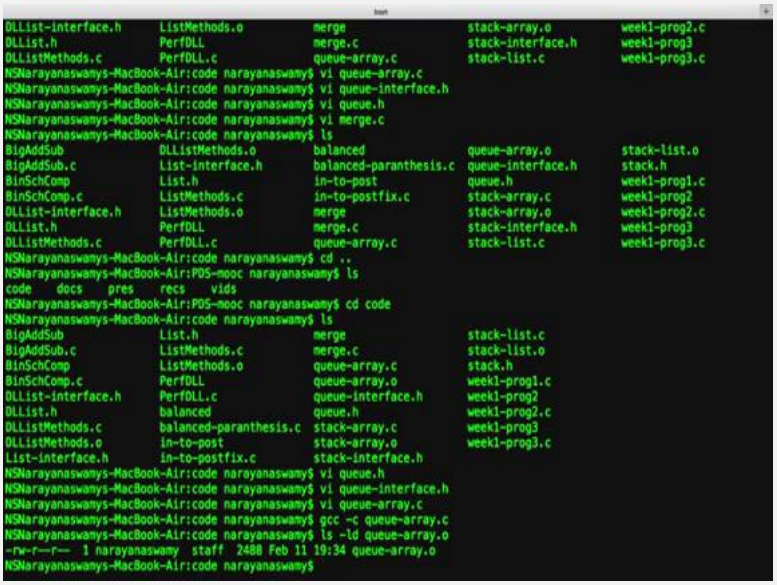
(Refer Slide Time: 25:34)



So, if the queue is empty then we printed the error message, but it is better programming practice actually set a global variable. So, the other methods will note that this particular function call has failed, then we return this particular value in element, we increment the value of front by want and take the remainder with capacity. Then, we decrement the total number of elements then you return the element.

Peek front is almost same, you check if the queue is empty, if it is not empty return the front element to the queue do not change any other parameter associated with the queue. So, these are really the functions that we are going to use in the next programming exercise and we have seen the implementation of this.

(Refer Slide Time: 26:22)



So, this are the already been compile and I generated the object. So, this has been compiled and the object file has been generated, when we know use it in our merging program, we will and read take a look at how to compile this object file we have already seems. So, in this short lecture you have we looked at all the methods associated with the queue abstract data type, the kind of data items we have again on into the good programming practice of defining the queue object appropriately, the queue class appropriately.

Then, we look at the interface functions, then we look at the appropriate methods and we have compiled it. So, now our queue implementation or the queue abstract data type is ready for use and other words we have the dot o file, we have the header file and we are ready to use this queue object in our programs, which is what we will do in the next file, next lecture. So, this basically brings to end today's lecture.

Thank you.